Arnaud Duhamel
Ghais Dahdouh
Sergei Johansen

User Collaboration in Specialized Softwares

Bachelor's thesis in Programming
Supervisor: Frode Haug
May 2024

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Arnaud Duhamel
Ghais Dahdouh
Sergei Johansen

User Collaboration in Specialized Softwares

Bachelor's thesis in Programming
Supervisor: Frode Haug
May 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Summary

Norkart AS is a company providing, among other things, products aimed at digitalizing the various processes of municipalities. Their software Komtek has this purpose. A part of the Komtek software is dedicated to handling jobs, where case handlers can go in a job, fill the relevant job fields, and submit the job. At the time of this thesis, this part of the Komtek software was not adapted to multiple case handlers working in the same job. Case handlers were not aware that other case handlers were working on the same job, the work of each case handler were not synchronized among each other, and only the last submitted version was saved. This created inefficiencies where the same work would needlessly be done multiple times and other problems.

Multiple features were developed to inform case handlers if many of them are working on the same job, to ensure that every case handler use the same version of the job, to only allow one case handler at a time to edit a job and finally, that if many case handlers edit the job at the same time, that the changes are sent to every case handlers in real-time and that version conflicts are also solved in real-time.

Sammendrag

Norkart AS er et selskap som tilbyr, blant annet, produkter som tar sikte på å digitalisere ulike prosesser i kommunene. Deres programvare Komtek har dette formålet. En del av Komtek-programvaren er dedikert til å håndtere jobb, der saksbehandlere kan gå inn i en jobb, fylle ut relevante jobbfelter og sende inn jobben. På tidspunktet for denne avhandlingen var ikke denne delen av Komtek-programvaren tilpasset for flere saksbehandlere som jobber med samme jobb. Saksbehandlerne var ikke klar over at andre saksbehandlere jobbet med samme jobb, arbeidet til hver saksbehandler var ikke synkronisert med hverandre, og bare den siste innsendte versjonen ville bli lagret. Dette skapte ineffektivitet der samme arbeid unødvendig ville bli gjort flere ganger sammen med andre problemer.

Flere funksjonaliteter ble utviklet for å informere saksbehandlere hvis mange av dem jobber med samme jobb, for å sikre at hver saksbehandler bruker samme versjon av jobben, for å bare tillate én saksbehandler om gangen å redigere en jobb, og til slutt, hvis mange saksbehandlere redigerer jobben samtidig, at endringene sendes til hver saksbehandler i sanntid, og at versjonskonflikter også blir løst i sanntid.

Preface

We would like to thank our contact person at Norkart, Vebjørn Fonstad Leiros, who met with us every single week for the duration of the project. His guidance and feedback was invaluable. We would also like to thank our supervisor, Frode Haug, for his availability, his diligence, his numerous feedbacks in the writing of this thesis and helpfulness for the whole duration of the thesis. Lastly, we woud like to thank Norkart for having invested time and resources in this thesis.

Contents

List of Figures

Chapter 1

Introduction

Software like Teams, Google docs and GitHub have made collaboration among teams much easier than it used to be.

With those tools, many users can modify in real-time the same document. More precisely, each user's version of the document is synchronized with other versions such that all users work with identical documents. This does not remove the need to handle potential version conflicts among users, but they are handled immediately, in real-time. With such tools, users can avoid unwillingly overriding each other's work.

1.1 Project

1.1.1 Fields

Collaboration features

These tools draw, among others, from the following two fields:

Collaboration features provide information in real-time to users about other users. This allows users to collaborate with each other effectively and above all, to avoid unwillingly overriding each other's work. For example, notifications could be used, warnings, having document statuses being updated in real-time and knowing when other users are inside a document are all collaboration features.

This is wider than collaborative editing.

Collaborative editing is a feature that allows multiple users to edit the same document in real-time. It is one collaboration feature among many to address the issue of having multiple users working together and potentially overriding each other's work.

conflict solving

In the case where multiple users edit the same document and end up with two different versions, a way to decide what will be the final version is necessary.

Git is the best example of that. With git, the user can decide what will be the final version for each part of a file with two different versions. The user can choose the current version, the incoming version, or a whole new version.

This is something that can be implemented with or without collaboration features. Git does not provide collaboration features.

1.1.2 Project limitation

Our project focused on implementing features related to the fields in the job manager of one of Norkart's products called Komtek.

We did not work in Norkart's systems directly. We created our own web application that recreated the features that were relevant for the project. That way, this thesis could be public and it reduced complexity.

The web application that we developed also did not reproduce Norkart's system in all its complexities because it would have required more work to implement the collaboration features. We prioritized implementing more features with less complexity. Because of that, some features developed would have to be adapted to be implemented in Norkart's systems.

1.1.3 Project description

Our project was divided into multiple incremental features, ranging from the easiest to the hardest. They can be divided in 3 main categories:

- Features to inform users in real-time that multiple users are working on the same job
- Features to prevent users from editing the same job at the same time
- Features to allow users to edit the same job at the same time, including proper version conflict solving.

1.2 Target groups and goals

1.2.1 Target groups

For the thesis

This thesis is for those interested in the implementation of real-time exchange of data between multiple users in a web application, and the implementation of version conflict resolution in the case of the collaborative editing of text.

For the project

The project is for those interested in the implementation of real-time exchange of data between multiple users in a job manager web application with the Websocket protocol, and the implementation of version conflict resolution in the case of the collaborative editing of text.

1.2.2 Goals

The goal of the thesis was to find, and implement in a project, ways for users to know in real-time when many users are in the same job, to prevent users from needlessly completing a job and to synchronize users' versions in real-time with appropriate version conflict solving, also in real-time. The last feature was for the case where collaborative editing was implemented.

Deliverables to Norkart

What Norkart wished to obtain at the end of this project was a prototype that replicates the job handling processes of the Komtek systems and that includes various collaboration and conflict solving features, all the way from simple checks to collaborative editing with version conflict solving in real-time.

They also wished, to the extent feasible, for our prototype to be developed with a generic architecture, so that the features we developed could easily be implemented in other products.

Benefits to Norkart

Based on this prototype, Norkart wishes to integrate similar features in the job handler of Komtek and potentially other systems.

Learning objectives

The learning objectives were the following:

- Learn about web technologies used in the Norwegian industry(React, Dotnet)
- Learn about real-time technologies(WebSocket, Redis)
- Learn about API and frontend testing
- Learn about continuous deployment and integration
- Learn about developing generic architecture designs
- Further develop teamwork skills
- Learn more about writing documentation
- Learn to program and to develop with a sustainable development perspective

These were wider then learning about collaboration features because a job handler prototype had to be developed, and this required all of the knowledge mentioned above.

1.3 Background

Through studies at NTNU, the team acquired a lot of usefull knowledge that were needed for this thesis.

The backend part of the project being in Dotnet 8, the basic programming course was useful because it taught the basics of algorithm structure and the programming language C. The object-oriented programming course was also useful because it taught the object-oriented programming paradigm and taught the programming language C++. Although Dotnet is used with C#, a solid basis in C and C++ is helpful to learn both Dotnet and C#. We learned about the notion of programming patterns in the advanced programming course. We learned about network protocols such as HTTP and HTTPS in the cybersecurity and computer networks course. we learned about the characteristic of restful APIs in the cloud technologies course. We also learned how to implement unit tests and to perform Git actions in the cloud technologies course. The group also learned about HTML, CSS and Javascript in the web technologies course. We also learned about documentation both in the software development course and in the cloud technologies course.

What was missing was to learn about the Dotnet framework, about Dotnet web APIs, to learn about the React framework, to learn how to join a Dotnet API with a React frontend, to learn to implement a test suite in both a Dotnet API and a React frontend. Git actions needed to be studied more in depth to be able to have a proper continuous integration and continuous deployment pipeline. We also had to learn how to use Git actions to run tests. The Websocket protocol also needed to be learned, as well as how to use it in a web application. Learning all of the above was part of the goals of the thesis.

1.4 Responsibilities and roles

- Arnaud was the group leader and also worked on the backend part of the project.
- Sergei was responsible for the backend part of the project
- Ghais was responsible for the frontend part of the project

- Vebjørn Fonstad Leiros was our contact person at Norkart. He was directing our project and providing us feedback.
- Frode Haug was our thesis supervisor.

1.5 Thesis structure

The introduction above presented the thesis, its context, its logistics and how its project was managed.

Here are the other sections of this thesis:

Theory: this section explores the main issue that forms the basis of this thesis.

Requirements: this section details the requirements of the project.

Design: this section describes the design of the project.

Implementation: this section describes how the project was implemented.

Development process: this section describes the development process that was followed during the project.

Testing: this section describes the various tests that were implemented in the project.

Deployment: this sections describes how our solution was deployed.

Conclusion: the conclusion includes a discussion of the results achieved in the project in relations to the different objectives outlined in the introduction, a critic of the thesis, possibilities for future thesis, an evaluation of the group's work and a closing statement.

Chapter 2

Theory

This chapter gives a thorough presentation of the the problem that lead to this thesis with potential solutions to address it. It also presents the objectives of the thesis in light of that problem.

2.1 Problem description

The typical use-case that the thesis aimed to address is the following:

Two case handlers see the same unassigned job on the job list. Both of them click on the job. They do not know that someone has been assigned to the job unless they refresh their page after the job has been assigned. Because there is no collaboration feature, they do not know that there are two case handlers working on the same job. Both complete the job. The first case handler submits the completed job. After, the second case handler also submits the completed job. Completing a job that is already completed leads to various problems. Among other things, it replaces the previously completed job.

This is undesirable for multiple reasons. In the normal workflow of the Komtek's job handler, a case handler normally does not enter a job that is already assigned to another case handler. A case handler is not supposed to work on a job in which another case handler is already working. However, in a perspective of flexibility, he has the ability to do so. For example, if the assigned case handler cannot complete the job himself for any reason, another case handler should be able to easily go into the job and complete it.

Currently, when a case handler opens a new job, and therefore gets assigned to it, other case handlers do not receive that information. The data of the job is updated in the server, but this is not sent to other case handlers. Other case handlers will receive the updated information only when they fetch the list of jobs from the server. This can be done, for example, by refreshing the job list page.

Case handlers are not expected to refresh the job list page before they open a job. And even there, a case handler could theoretically click on a new job less then one second after another case handler.

This is an edge case. However, when multiplied with hundreds of users in hundreds of municipalities, this edge case inevitably happens more frequently. That makes it worth addressing this issue in this thesis.

The most effective solution would be that when a case handler is assigned to a new job, this information makes its way to other users right away. Latency here is determinant. The fastest other users are made aware that a job is now assigned, the less the risk that they open the job thinking it is unassigned.

As a continuation of the typical use-case described above, when a case handler opens a job, he is now assigned to it. If the same job appears unassigned to another case handler and he clicks on it, he will now be assigned as case handler. His assignation will override the assignation of the previous case handler. The case handler that was assigned to the job first would not know that he is no longer assigned unless he refreshes the job page.

Insuring that a job can only be assigned once would already be beneficial, because the second user opening the job would see that he is not the assigned case handler even if the job was unassigned when he opened it. This would indirectly alert him that he opened an already assigned job.

However, in the Komtek job handler, who is assigned to a job does not have any impact. As mentioned above, even after a job is assigned, any case handler can enter the job and complete it. It is like that because of a flexibility consideration.

What would be more valuable is to know if a case handler is already working inside a job when another case handler enters the job. Because the problem itself comes from the fact that two or more case handlers complete the same job. And this requires multiple case handlers being in the same job before any one of them completes it. Otherwise, after the job is completed, it will no longer be in the job list. But if a case handler entered the job before another case handler completed it, the case handler remaining inside what is now a completed job will still have the opportunity to complete it.

A potential solution could be that, when a case handler enters a job, he is made aware if another case handler is already inside the job. The risk of multiple case handlers completing the same job would then be clear and the case handler entering the job could leave.

The case handler that is already inside the job would gain to also be made aware of the other case handler entering the job. That way, every case handler involved would have an opportunity to correct the situation. This is where the whole notion of collaboration comes in. Informing case handlers about other case handlers' actions gives them the opportunity to collaborate with each other. But this would also require a way for the server to send information to case handlers without them asking for it, without them taking any action. The more information is shared, the better. And the faster it is shared, the better.

The ability for multiple case handlers to coordinate to avoid working on the same job is a form of collaboration.

One more step ahead in the process, when two case handlers are in the same job, they both have the ability to complete the job, even after one of them completes it.

Just like for job assignation, one way to address this is to only allow a job to be completed one time. This could be done through a check when a job is completed. If a job is already completed, it cannot be completed again. However, a check when the job is completed would come late in the process. For example, the first case handler completes the job. At that moment, the other case handler completed half of the job. He will then keep working to complete the job, only to be told later that the job is already completed. This is time wasted.

What would be more desirable is for a case handler that is inside a job to be notified immediately when another case handler completes the same job. This would avoid unnecessary work.

In fact, currently, each case handler inside a job works on their own version of the job. The first completed version becomes the 'final' version. That in and of itself is inefficient. If a case handler filled a part of the job, another case handler entering the job should not have to refill that same part of the job, especially given the fact that the other case handler could fill his version with something completely different.

This is where collaboration to allow multiple case handlers to work on the same job becomes relevant. By same job, it is rather meant the same version of a job. This requires sending the work made by a case handler in a job to the server and then to other users in the same job, preferably in real-time. This would remove the inefficiencies of multiple case handlers each filling the same part of a job as well as removing the possibility of diverging versions. In this case, it becomes one version for everyone.

The notion of one common version for every case handler would however raise the question of allowing them to edit the common version at the same time or not. Because in the case of one common version for every case handler, if many can edit at the same time, there will come a time where the changes of one case handler will conflict with the changes of another case handler.

On a technical standpoint, there is actually no such thing as a common version. Each case handler has its own version of the job and each of their versions are synchronized in a way that the content is the same for each version.

It is the synchronization aspect that becomes challenging if multiple case handlers can edit a job at the same. If two changes interfere with each other, what should be the final result? If a case handler changes a word, but another case handler removes it completely, should the final version be the modified word, deleting the word or just the modifications the first case handler made? Those possible outcomes are the conflict itself. Choosing a final result in a way that always lead to one version for every case handler and in a way that preserves as much as possible the changes made by each case handler is difficult.

This difficulty can be avoided with locking mechanisms so that only one case handler at a time can edit a job or a part of the job. But this would remove some flexibility to the process, especially if the case handler with editing rights stays in the job without working on it.

It is possible to see that, from a relatively short process: opening a job out of a job list and completing it, a lot of features can be implemented at various steps to handle the possibility of multiple case handlers opening and working on the same job.

2.2 Objectives

Objectives from a user perspective

From a user perspective, the main objective is to implement various collaboration features that would make it easier for case handlers to avoid working on the same job and, above all, that would prevent a job from being completed more then one time.

The last objective would be to implement various collaboration features that would make it easier for case handlers to work on the same job. Even though this is not the normal process, it is desirable to keep the process as flexible as possible.

Although part of the second objective, a third objective would be to allow multiple case handlers to *edit* a job at the same time, in a way that properly handle conflicting versions.

Objectives from a technical perspective

The most easily achievable objective would be to implement a check when a job is completed to avoid it being completed more then one time. This is something that would require little effort and eliminate what is the biggest problem in the process.

The next objective would be to come up with a way to send data about users to the server, to store that user data in the server's memory or a cache and have the server communicate that data to other users, all of that in real-time.

The last objective would be to implement the ability for multiple case handlers to edit the same job. Two approaches could be explored for that: locking all or part of the job or implementing version conflict resolution.

Chapter 3

Requirements

This chapter discusses the requirements for our project.

The requirements elicitation for the upcoming functionality was continuously conducted through an iterative analysis of the assignment description and multiple interactions with the client.

The requirements specification is divided in two main sections: user requirements and system requirements. They are described in natural language with the use of simple diagrams to facilitate additional clarity. Some of the requirements contain rationale behind their specification.

3.1 User requirements

We described the problem that we wanted to solve in the theory chapter. Based on the description, we derived user requirements for our application. The specification of user requirements is written in a little detailed fashion.

Concurrent job collaboration

Concurrent job collaboration was a large requirement, that consisted of multiple parts that we decided to describe in more detail.

Conflict solving

The primary objective of this project was to establish a collaborative platform for users to collectively engage with a singular resource, specifically focusing on job-related tasks. This necessitated the implementation of robust conflict resolution mechanisms that maintain a seamless user experience.

As mentioned previously, multiple users collaborating on a resource could stumble on conflicts when writing simultaneously. The diagram 3.1 illustrates the process behind collaboration of two case handlers. There is nothing dangerous that happens when one user does writes at a time. The update is sent to the collaboration system, validated and then sent further to the other user, updating their user interface. When two users write simultaneously, which change to preserve?

Imagine the following: User A writes "banana", user B writes "apple", and user C writes "collaboration". The writes are sent to the system that has to decide which version to send back. This is something that we had to find out and implement in this project.

The use case diagram 3.2 shows a higher level modeling of the conflict involving more than two users.

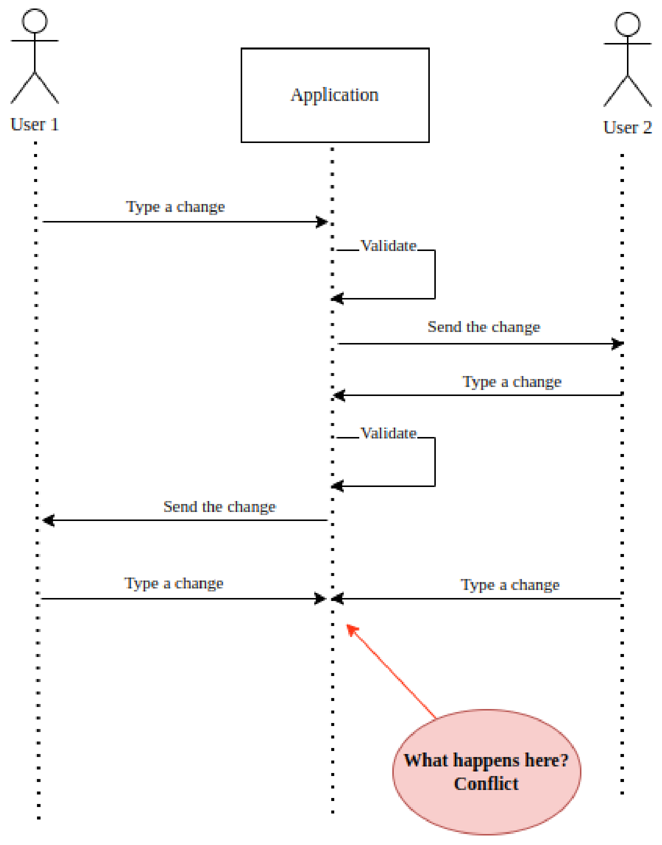


Figure 3.1: Sequence diagram showcasing where the conflict happens

List of concurrent users

In order to address the problem where users do not know about each other being on the same page, we decided on implementing a list of users. The list would provide the following functionality:

- Showing collaborators
- Being updated when a user enters or leaves

This requirement alone would make the user experience quite satisfactory for the case handlers. When they would enter a job and see that someone is already on the page, they could immediately decide whether to continue on the job, or go and work with another one. The diagrammatic representation of the problem and a solution to the problem is shown on the diagram 3.3.

Notification system

In order to further enhance the user experience, we decided to have some form of notification system in the application. The system would make sure that concurrent users are aware of what is going on beyond editing a job. Users should be notified based on the following events:

- A user leaving the job page
- A user entering the job page
- A user submitting the job

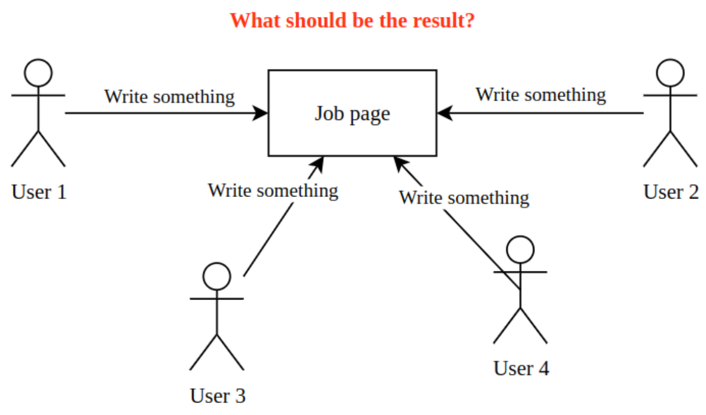


Figure 3.2: Use case of multiple users writing to a single job resource.

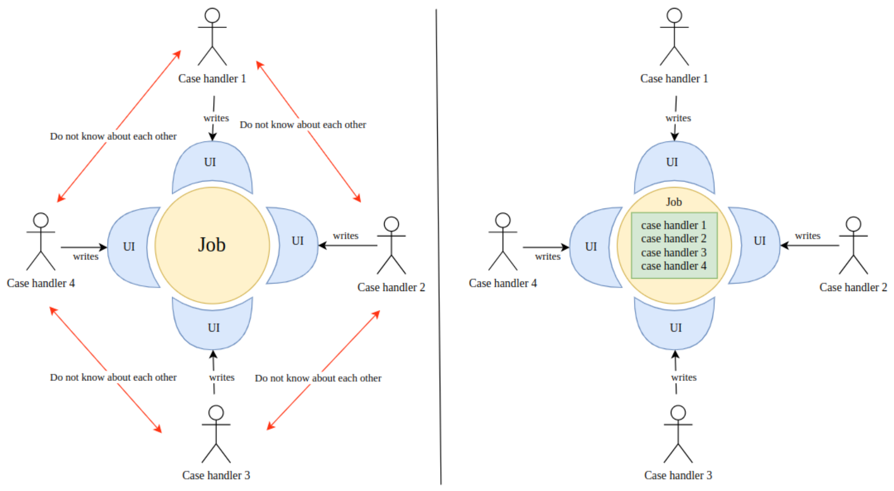


Figure 3.3: Job editing without a user list (to the left) and with user list (to the right)

Web interface - Komtek replica

As our project aimed to develop a solution compatible with the Komtek system, it was imperative to create a partial replica of Komtek comprising two key pages: one housing active and completed jobs, while the other centers around a specific job. Because Komtek is a web based system, we should implement a web based interface.

Accessibility

Norkart mentioned that the average case handler age is approximately 50 years old. Considering that, significant emphasis had to be placed on designing a user interface that prioritizes ease of use and intuitive navigation.

API server

Norkart demanded from us including an API server in our project to facilitate seamless integration with the existing Komtek system. This requirement stems from Komtek's heavy reliance on backend API servers in its existing system.

Job creation

For testing purposes, we decided to include a job creation functionality to the interface, even though it was not important for our client.

3.2 System requirements

Persistent storage

One of the requirements we identified was the integration of a persistent storage system. While options such as a file system or in-memory storage were plausible in theory, we decided to opt for a database to mimic real-world scenarios. This choice prioritizes data durability, consistency, and availability, essential for the reliability and functionality of our software. The selection of a specific database is flexible, with both NoSQL and SQL databases being viable considerations.

Caching mechanism

Since the application involved frequent data changes while multiple users are interacting with a resource, the data had to be stored somewhere temporarily before going to the persistent storage. This is because persistent storage that focuses on availability, durability and consistency is usually not designed for very frequent writes. Caching of temporary changes should be implemented either in-memory or with a service like Redis.

Security

Our software would potentially be deployed to a public cloud. We had to take security measures in order to avoid the disclosure of our application to unwanted users.

Authorization and authentication with JWT

The software must implement a robust authorization mechanism to ensure secure access control to its resources. The client explicitly wanted us to use the Json Web Token for authentication and authorization.

Performance

The application pages should load within 3 seconds to ensure a positive user experience. Regarding concurrent editing of a job page, we should minimize the delay between one user typing and the appearance of changes on other users' client.

Stateful connection

Certain connection handling should take place in order to provide real-time communication between concurrent users. There were multiple technologies to choose from, and the requirement was to research and choose an appropriate solution for the task.

Browser compatibility

Our application should be able to run on all the major browsers supporting their older versions (Chrome, Safari, Firefox, Edge and IE).

Chapter 4

Design

This chapter examines the design aspects of our project. It describes the sources from which we took inspiration for both the technical and graphic user interface design of the application as well as the designs that were ultimately implemented.

4.1 Inspiration sources

Section summary

This section outlines the core concepts, methods, technologies and algorithms from which we took inspiration to address the problem raised in this thesis.

It presents:

1. An overview of the collaboration features found today
2. The underlying user data on which those features are based
3. The protocols and methods with which this data is exchanged between multiple users
4. In the case of multiple users editing the same text, the architectures and algorithms with which users' version are synchronized among each other based on the changes made by each user
5. The algorithm used to extract the insert and delete operations out of the changes made by users to their versions, something necessary for synchronization

Komtek's current system architecture

Komtek is a web application. It's an application that is used through a web browser with an internet connection.

This contrasts with an on-premises application where the application is hosted and operated on the user's local network.

Komtek also has a server-client architecture where the whole application and its data is stored on servers controlled by Norkart. Users access the application by making requests to that server. This is in contrast to a peer to peer architecture where there is no central server and where every user's computer is both a server and a client.

Komtek's rendering paradigm

When a request is made by a client to use Komtek, the application is rendered to the user under the single page application paradigm. Under this paradigm, when the browser makes the first request to access the application, no HTML is returned. The browser starts with an empty DOM. Then the entire

HTML content of the application is loaded at the root of the DOM by Javascript. Inserting a different route in the navigation bar of the browser does not generate a request to the server. The route is instead used by the application to decide what content is rendered. In this paradigm, the data needed by the application is fetched in separate HTTP requests [41].

There are other rendering paradigms such as the static website where the application is a series of HTML files stored in the server. When a user wants to navigate to a page, a new request is sent to the server and a corresponding HTML file is returned to the browser. There is also the multi-page application paradigm. In this paradigm, the web pages returned to the user are dynamic, but they are assembled in the server with the use of the necessary data and HTML templates. Entering a different route in the navigation bar creates a request to the server for the new page. There are many more rendering paradigms [41].

Technologies used for Komtek

For the HTML content of the application, React is used. In React, the different elements of an application are divided into reusable modules called components [81]. The elements of a page are components. An entire page is a component and, ultimately, the entire application is one big component. React was first released in 2013 by Facebook [33].

For the required data, the application fetches them from the server through an API. An API is a 'set of rules or protocols that enables software applications to communicate with each other to exchange data, features and functionality'[44].

In terms of technology, the API is written with the Dotnet development platform using C# as the programming language.

Dotnet is presented as a 'free, open source & cross-platform development platform'. Free because it costs absolutely nothing to use it. Open source because its source code is publicly available and everyone can contribute to it. Cross platform because it can be used with Windows, Mac and Linux. Lastly, it is a development platform because it includes programming languages such as F# and C# and libraries [29].

C# is the object-oriented language of Dotnet [29] and it has features that makes it easier to use then C++ to develop web applications such as automatic memory management and integrated concurrency [108].

The API used for Komtek is a REST API. The REST standard was introduced by Roy Thomas Fielding in 2000 in his PhD thesis titled 'Architectural Styles and the Design of Network-based Software Architectures' [36]. There are two aspects of a REST API that is relevant to mention for this thesis.

First, it is stateless. This means that the server does not store any information about its users [37]. For Komtek, this means that the server does not know if a user is inside a particular job or not. One of the consequences of statelessness is that a REST API works on a request basis. The server only performs actions such as sending data to a user or storing data in a database after receiving a request from a user. For Komtek, this means that a user will only know if a job has been completed by another user if he refreshes the job list.

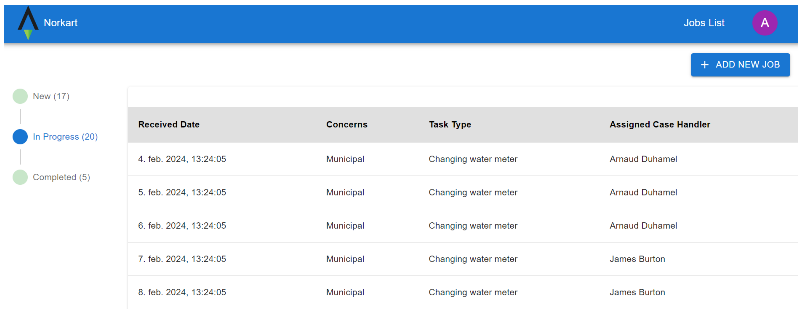
Second, it is uniform. This means that an identical request will always return the same response. For Komtek, this means that clicking on the button to complete a job will send a request to the server to store the completed job in the database, even if the job has already been completed by another user. This allows case handlers to override each other's work.

Background

It is the two characteristics mentioned above that are the source of the problem faced by Komtek's users that Norkart wanted to address with this thesis.

As part of the project description, it was mentioned that this problem had to be addressed in a way that is adapted to Komtek's processes.

In Komtek, processing a job begins with a job list such as this one:



Received Date	Concerns	Task Type	Assigned Case Handler
4. feb. 2024, 13:24:05	Municipal	Changing water meter	Arnaud Duhamel
5. feb. 2024, 13:24:05	Municipal	Changing water meter	Arnaud Duhamel
6. feb. 2024, 13:24:05	Municipal	Changing water meter	Arnaud Duhamel
7. feb. 2024, 13:24:05	Municipal	Changing water meter	James Burton
8. feb. 2024, 13:24:05	Municipal	Changing water meter	James Burton

Figure 4.1: Job list similar to Komtek's job list

The job list is the root of the problem. Because it is where a user risks clicking on a job in which another case handler is already working.

The problem could be solved by removing the use of a job list and have the software distribute jobs automatically to case handlers after they complete a job. However, this would not be adapted to Komtek's processes. It was therefore discarded.

Based on that, the natural starting point for developing various real-time features would be the list page. However, at the start of the project, we were notified that Norkart had implemented an automatic refresh feature. We therefore did not work on the list page and instead focused our effort on the job page. And since the project was directed towards collaborative editing, it was more relevant to focus our efforts on the job page.

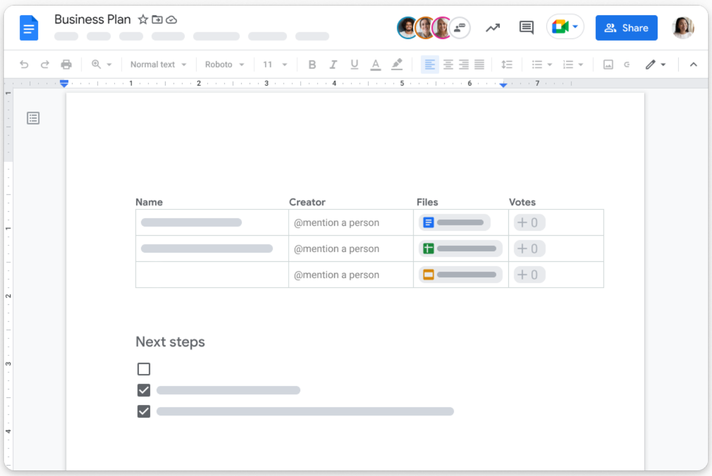
The other key aspect to which we had to adapt our solutions is that, in Komtek, collaborative editing is not something desirable. The normal process is that a job is completed by only one case handler. Collaborative editing would therefore be in place only to prevent inefficiencies and problems in the case that multiple case handlers do end up working on the same job.

real-time collaboration features

A non-exhaustive inventory of softwares providing collaboration features to their users have been made.

Google Docs

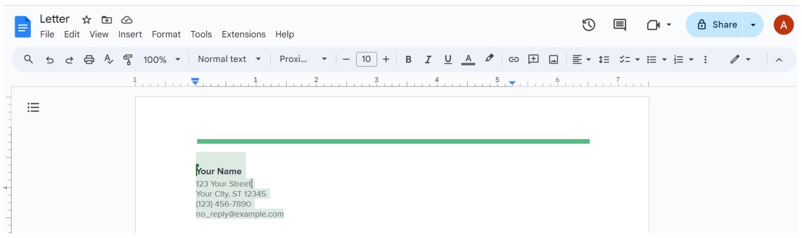
Google Docs is a known and free collaborative text editor available to everyone with a Google account. Its collaboration features include, among others, avatars being shown for every user currently inside the document, collaborative editing, showing the cursor of every user inside the document with the same color as the color surrounding the user's avatar, seeing the name of the user when hovering over his cursor, seeing the selection of characters made by other users, the possibility to focus on another user's cursor by clicking on his avatar, a version history, the possibility to leave comments in the text, a possibility to share documents, a chat feature and even an online meeting feature.



Name	Creator	Files	Votes
	@mention a person		+ 0
	@mention a person		+ 0
	@mention a person		+ 0

Next steps

Figure 4.2: Image of Google docs showing collaboration features



Your Name
123 Your Street
Your City, ST 12345
(123) 456-7890
no_reply@example.com

Figure 4.3: Image of Google docs showing a character selection made by another user

Microsoft Words

In it's most recent version at the time of this thesis, Microsoft Word offered multiple collaboration features. It includes, among others, avatars being shown for every user currently inside the document, collaborative editing, showing the cursor of every user inside the document with the color being the same as the color surrounding the user's avatar, showing the avatar of a user next to his cursor when he makes a change, seeing the name of the user when hovering over his cursor, the possibility to focus on another user's cursor by clicking on his avatar, a version history, the possibility to leave comments in the text and a possibility to share documents.

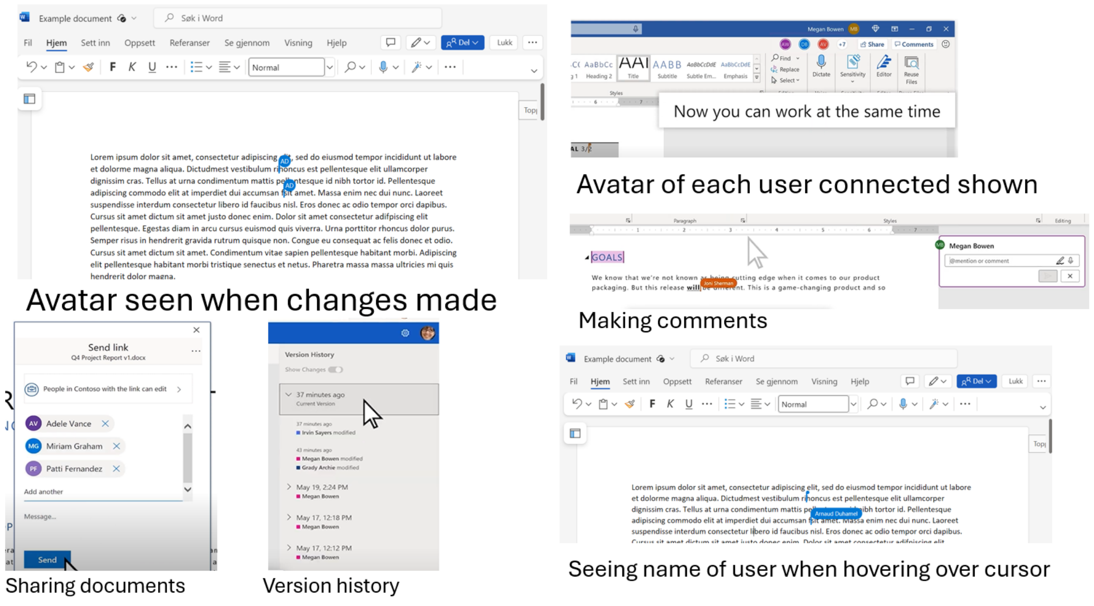

Avatar seen when changes made
Now you can work at the same time
Avatar of each user connected shown
Making comments
Sharing documents
Version history
Seeing name of user when hovering over cursor

Figure 4.4: Collaboration features of Microsoft Word

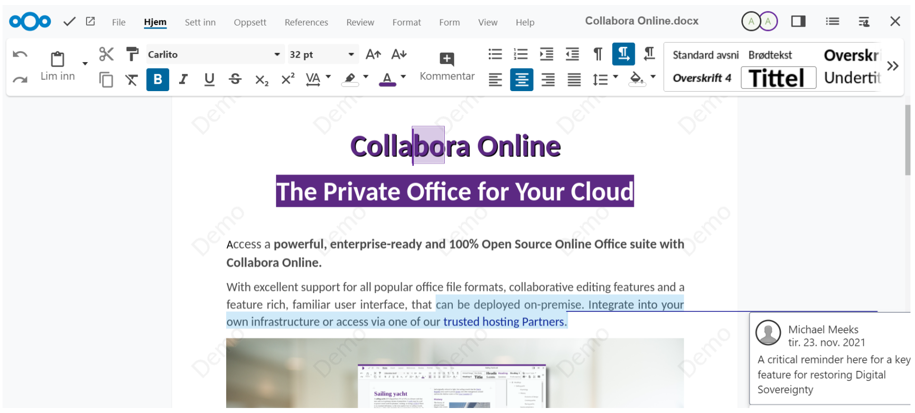
Source: [13]

It is also worth mentioning that, at the time of this thesis, the Microsoft Word editor is the editor used in other softwares that are part of the Microsoft Office 365 software suite such as Teams and SharePoint.

Collabora Online

Collabora is a consultant company specialized in open source development [109]. One of the technologies it works with is the LibreOffice suite. LibreOffice is an open source document editing suite established in 2010 that originated from the OpenOffice suite [47]. Collabora created the product Collabora Online, which essentially took LibreOffice to the Cloud [72]. This product contains multiple collaboration features. One version of that product is Collabora Online Development Edition [17].

The collaboration features of the latest version are avatars of users connected to the document along with a color assigned to them, cursor indicators of the colors assigned to the users, selection indicators, the name of the users next to the cursor when it makes an action and leaving comments:



Access a **powerful, enterprise-ready and 100% Open Source Online Office suite with Collabora Online.**

With excellent support for all popular office file formats, collaborative editing features and a feature rich, familiar user interface, that can be deployed on-premise. Integrate into your own infrastructure or access via one of our trusted hosting Partners.

Michael Meeks
tir. 23. nov. 2021
A critical reminder here for a key feature for restoring Digital Sovereignty

Figure 4.5: Collaboration features of Collabora Online Development Edition

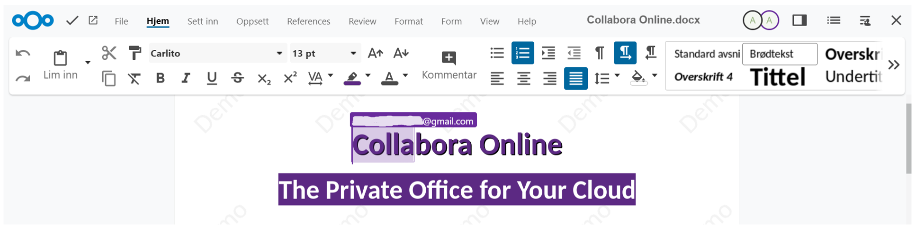


Figure 4.6: Cursor and selection indicators of Collabora Online Development Edition

Figma Design

Figma Design [38] is an application created by Figma [50] to design prototypes of user interfaces. It contains multiple collaboration features:

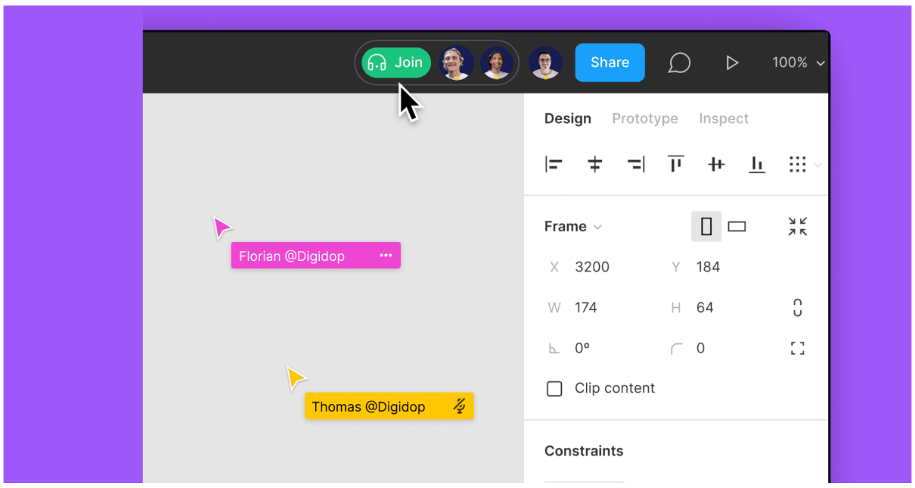


Figure 4.7: Avatar of connected users to a document, mouse tracking and online meeting features of Figma Design

Source: [59]

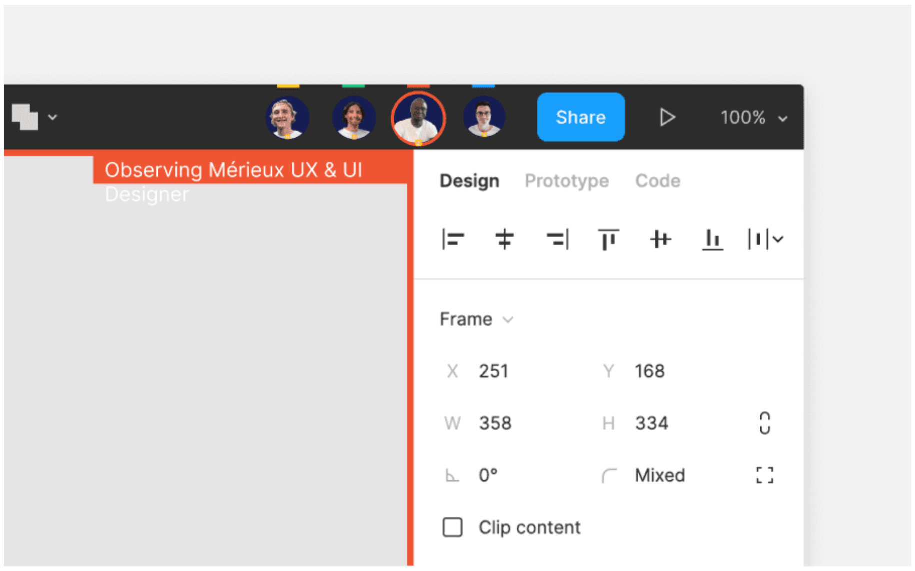


Figure 4.8: Observing the screen of another user in Figma Design

Source: [59]

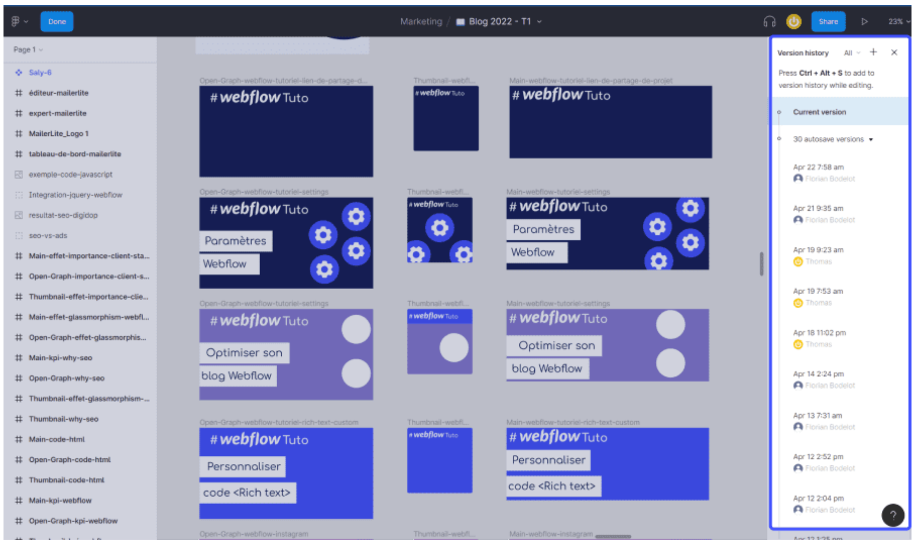


Figure 4.9: Document version history feature of Figma Design

Source: [59]

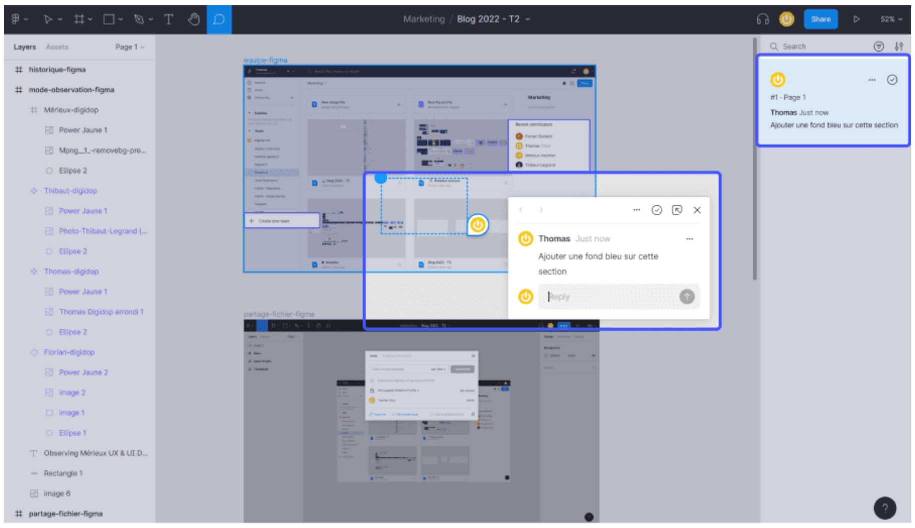


Figure 4.10: Making comments in Figma Design

Source: [59]

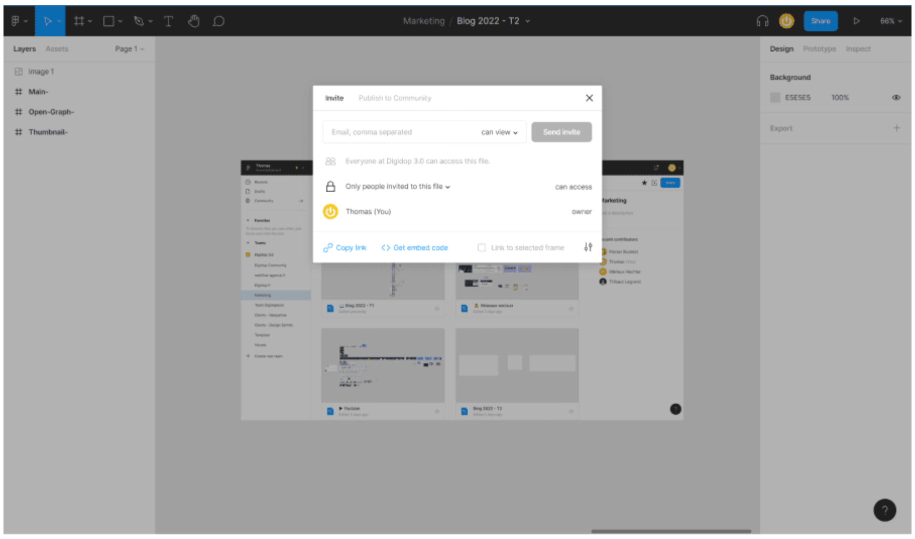


Figure 4.11: Sharing documents in Figma Design

Source: [59]

Jotform

Jotform is a software company based in San Francisco. It offers, among other products, an online form builder. The builder offers collaboration features. It shows avatars of users connected to the form and it highlights the element on which the user is working with the color of its avatar along with showing the avatar of the user in the top right corner of the element:

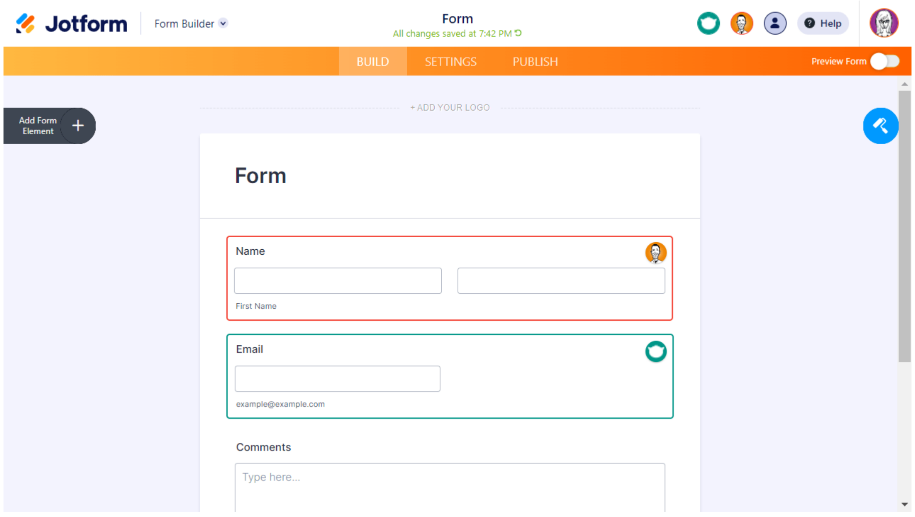


Figure 4.12: User avatars and element highlighting in JotForm

Source: [57]

User data used for collaboration features

Without being complete, this list of softwares offering collaboration features show thoroughly what data can be used to track other users' activity in an interface:

1. The users that are connected to a document
2. Their usernames
3. The position of their cursor

4. The position of their mouse
5. The changes they make to the text
6. The text they select
7. The buttons they click
8. In the case of a form, the element they select

Based on this data, an endless amount of different interfaces with collaboration features can be developed.

Discussion on the collaboration softwares explored

The team mainly used Google Docs and Microsoft Word through Teams, and one aspect that stood out is that when a user connects to a document, users that are already inside the document will only see the avatar of the incoming user appear on their interface, without any other kind of notification. Such that sometimes we do not notice immediately when another user connects to the document.

Technologies and method used to implement collaboration features

As mentioned above, collaboration features are based on the data generated by the actions of the users. In order for the features to be collaborative, this user data must make its way to other users. In the context of a server-client architecture, this user data must make its way from a user to the server, and from the server to other users. For those features to provide a real-time experience, there must be no latency in the transmission of data.

We surveyed the following ways and technologies to achieve these two characteristics:

Short polling

Short polling involves automatic and repeated HTTP requests to fetch data from the server.

This solution was implemented by Norkart to keep the job list updated.

This solution has the advantage of being very easy to implement in the context of a rest API because it is based on the same technology and methods: one time, self-contained get requests made through the HTTP protocol. The connection lasts only for the duration of the request and all the needed information is in the request itself.

In a React frontend, this can be very easily achieved with the revalidate on interval method [7] of the SWR package [95].

This method is valuable in the case that frequent changes by multiple users are expected in an interface. In the case of Norkart, for example, it is expected that case handlers will often make modifications to the common job list as they open new jobs and complete them.

However, it becomes ineffective in cases where it is uncertain if multiple users will use the interface. For example, if only one user is in the document and requests are made every 5 seconds to see if another user entered the document, a lot of resource is wasted.

With this method, user data is sent to the server through an HTTP request. And the server is no longer stateless because it now stores data about users. It has to store user data because in this method, the server does not act as a relay.

Long polling

Long polling involves making an HTTP request to the server and keeping it open until the server sends a response or until the request times out. The idea is not to have a connection that persists beyond multiple requests. The idea is rather that a client makes a request and the server waits for the data before returning a response. This is what keeps the connection open. When a response is either received by the client or the request times out, the client immediately makes a new request to the server. This cycle is repeated indefinitely. This allows the client to make HTTP requests that act as listeners [55].

Compared to short polling, this method reduces the amount of requests made to the server. However, it still involves multiple requests.

In this method, the user also sends data to the server through HTTP requests and the server must store that data because it does not act as a relay.

In the case of both short and long polling, the server cannot act as a relay because the connections between the client and the server are not persistent.

Server-sent events

Server-sent events [104] is a way for a server to send information to a client through a persistent HTTP connection.

The standard for this mechanism has been established in the HTML Living Standard, section 9.2: 'Using this API [server-sent events] consists of creating an EventSource object and registering an event listener.' [3]. This EventSource object is created in the client. This is what establishes the connection. This persistent connection can be established by using any version of the HTTP protocol. Connections with HTTP 1.1 and above are persistent by default, not with HTTP 1.0 [**HTTP1.1_above_persistent**]. With HTTP 1.0, both the request and the response must set the connection header to 'keep-alive' [**HTTP1.0_keep_alive**]. Those connections are then stored by the server by storing the response objects. The data that needs to be broadcasted is then sent to the clients through those response objects.

Server-sent events does not establish bidirectional communication between the server and the client. A separate HTTP request must be made by a client to send data to the server. Only then can the sent data be broadcasted to other users. It is also important to mention that, with this method, each HTTP request establishes a separate TCP connection with the server.

This method is more effective then short and long polling because the connection between the server and the client is persistent. It is only one connection that always remains open.

With this method, and every other below, the server can act as a relay, because the connection is persistent.

Forever Frame

This method is a mix between server-sent events and long polling.

The method creates a hidden iframe element in the client that makes an HTTP request to the server at the designated endpoint. This connection is meant to never complete. Through this connection, the server sends scripts to the client that are immediately executed when they are received [55].

It draws on long polling because the way the connection is established is through an HTTP request that does not complete. It draws on the server-sent events because, in effect, there is only one HTTP connection made by the client to the server that always stays open and through which the server can send data to the client at any time. And just like with server-sent events, the client must send data to the server in separate HTTP requests and each request establishes a separate TCP connection.

Bidirectional communication

The main disadvantage of all the methods mentioned above is the need for multiple and repeated HTTP requests that each open its own TCP connection, either to send or to receive data, or both.

And in all three methods, there must be a separate request to receive and send data [97].

The Websocket protocol

The Websocket protocol aims to address those disadvantages by providing a single, persistent TCP connection between the client and the server in which data can travel in both directions.

To establish a Websocket connection, the client sends an HTTP request to the server indicating a wish to establish a Websocket connection. The server will then send an HTTP response saying that it accepts the Websocket connection. This is the handshake part of the protocol.

Once the handshake is successful, the two-way communication channel over the TCP protocol is established. At that point, both the server and the client can send data independently from each other

[97].

The protocol has been standardized in 2011 in the RFC 6455 [97].

The HTML living standard also provides a standardized Websocket interface to 'enable web applications to maintain bidirectional communications with server-side processes' [107]. It is also referred to as the Websocket API [97].

One of the main disadvantages of using of the Websocket protocol is that the application must manage the connection through the Websocket API. With the HTTP protocol, connections are handled by the browser and the application only needs to make requests.

Another disadvantage is that the only endpoint available to both send and receive data with a Websocket connection is the connection object itself. This means that there can only be one listener and sender on both the client and the server for all of the data that will travel through the connection. This can make handling the exchange of data between the server and the client more challenging in contrast to using HTTP requests where the use of URIs allows for a modular management of data communication.

A solution to that could be to establish multiple Websocket connections between the server and the client, something that would provide more communication endpoints and therefore allow for a more modular management of data communication. However, each Websocket would establish a separate TCP connection between the client and the server. And this is not optimal because multiple TCP connections take more computing resources than only one.

Bidirectional streaming with HTTP 2

The HTTP 2 protocol addresses both of those challenges. This version of the HTTP protocol was released in the RFC 7540 on May 2015 [52].

It allows 'interleaving of request and response messages on the same connection' [52]. In this version, multiple requests and responses can now travel on the same connection. This 'is achieved by having each HTTP request/response exchange associated with its own stream [...]. Streams are largely independent of each other' [52]. This architecture is referred to as 'multiplexing of requests' [52].

In practice, bidirectional streaming with HTTP 2 is implemented in the same way as server-sent events, except that the requests must be made with the HTTP 2 protocol.

This way, the client can listen to multiple EventSource objects and send every piece of data to the server through separate HTTP requests, all on one TCP connection. Because of this single TCP connection, using HTTP 2 provides the same real-time experience as the Websocket protocol, while also allowing for a modular management of data communication both by the server and the client.

HTTP 3, released on June 2022 in the RFC 9114, provides an improvement of this multiplexing feature [51].

Unfortunately, the technology is not widely used. As of April 23, 2024, 35.4% of all websites used HTTP 2 and 29.5% used HTTP 3 [22]. One website may support both protocols.

Version conflict solving

In collaborative editing, each user works on their own version of the text. Through the exchange of text data in the ways mentioned above, those different versions are synchronized so that changes made by a user are incorporated in every other version.

In the course of that process, version conflicts may arise if there are multiple possible final versions resulting from the changes made by multiple users.

Here is an example:

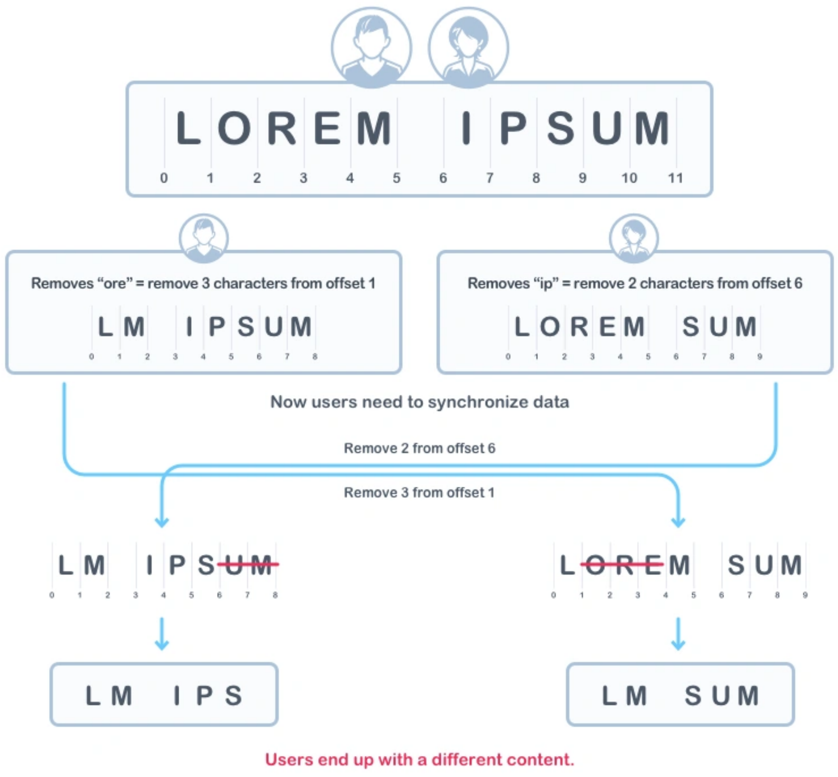

LOREM IPSUM
0 1 2 3 4 5 6 7 8 9 10 11
Removes "ore" = remove 3 characters from offset 1
LM IPSUM
Removes "ip" = remove 2 characters from offset 6
LOREM SUM
Now users need to synchronize data
Remove 2 from offset 6
Remove 3 from offset 1
LM IPS~~UM~~
L~~ORE~~M SUM
LM IPS
LM SUM
Users end up with a different content.

Figure 4.13: Diagram showing the outcome of text collaboration without version conflict solving

Source: [21]

Below are some methods to handle those conflicts.

operational transformations

This method is based on the premise that changes to a text document is done through operations. For pure text, those operations are inserting and deleting characters.

Those operations are then sent to other users working on the same document.

The second core principle of this method is that when a user receives an operation made by another user, it will be transformed to be adapted to the changes made locally. For example, two users start with the string 'abcd'. One user deletes 'c' on his local version. The other user deletes 'd' on his local version. This operation is a deletion at index 3. This operation is broadcasted to the other user, but because the other user deleted 'c' on his local version, there is no more character at index 3. Applying the operation without transformation would have no effect whereas the intention was to remove 'd'. The delete operation will therefore be transformed to a deletion at index 2.

Those transformations must meet the following requirements.

First, they must achieve convergence. The transformations must be made in such a way that the version of each user becomes the same.

This is why, at its core, operational transformations can be summarized by the following equation:

$$O1'(O2(X)) = O2'(O1(X))$$

Second, the transformation of an operation should preserve the initial intention of that operation. For example, two users start with the string 'hello world', if one user deletes 'world' but the other users changes the string to 'hello wooorld', the final result should be 'hello oo' so that the intended changes of both users are applied.

lastly, operations must be performed on a document in the same order for every user. This matters for convergence because operations are index-based. For example, two users start with the string ‘a’. There are two operations in the queue: to insert ‘b’ at index 0 and to delete the character at index 0. Doing the insertion first and the deletion second gives ‘a’ whereas if it is done in the opposite order, the final result is ‘b’.

This is a drawback of operational transformations. Because of this, extra controls must be put in place to insure that the changes are applied in the same order for every user. One way to address this issue is to have a central server maintain a list of operations made on the document. Only one operation made by a user can be sent to the server at a time. The server must keep a list of every operation made on the document. This list must be ordered based on the moment the operation is received. And the server only broadcasts one change at a time to other users [111].

Conflict-free Replicated Data Types

This kind of data types was introduced in July 2011 in a research report titled ‘Conflict-free Replicated Data Types’ [85].

This system is as the name says: based on special data types. Those types must only be modifiable through operations. For example, for an integer, it is done through increase and decrease operations. This is important to note: an operation is always added, even to reduce the value.

Under that method, the operations performed on the data types must have the three following traits [85]:

1. They must be idempotent. This means that repeating the same operation multiple times must produce the same result.
2. They must be associative. This means that the same operations grouped in different ways must produce the same result.
3. They must be commutative. This means the the same operations performed in different orders must produce the same result.

Let’s take the example of an integer that keeps a count value of an item in inventory in an e-commerce website, without conflict-free replicated data types.

Three users open the page of the product. It is indicated that there are 2 items left. One of the users decides to buy one product. The count is then reduced to 1, and this value is broadcasted to other users. Everyone now see that there is 1 item left. After that, a second user decides to buy the last product. The count is reduced to 0 and this value is broadcasted to the every user.

The problem will come if, for any reason, a user will buy a product, then another user will buy a product before receiving the updated count. Both users will reduce their count from 2 to 1, and broadcast that value. After the broadcasts, the count is at 1, when in fact two items have been sold.

With a conflict-free integer, what determines the value of the count is the set of operations from which the integer is composed. Every operation performed on the integer is stored in the data type with a unique id.

The research report above outlines two ways to update a conflict-free replicated data type. The first one is by broadcasting the entire conflict-free replicated data type after an update. This means that the conflict-free replicated data types on other clients will receive the entire set of operations of the updated data type. The two sets of operations will be combined. With this update method, the receiving data type will now have in its set operations that have already been performed. This is simply addressed by removing from the set duplicate operation ids. This is how idempotence is achieved. The new operations will then be performed on the count. The other possible way to update a conflict-free replicated data type is to broadcast only the new operations. In the case of an integer, its easy to see that the increase and decrease operations are both associative and commutative, because additions and subtractions are both associative and commutative [85][64].

The greatest advantage of conflict-free replicated data types is that the server only needs to relay the updates to other users for synchronization. It does not need to ensure that only one operation is broadcasted at a time. Multiple operations can be sent at the same time, and it does not need to ensure that updates are sent in the same order to every user; updates can now be received in different order by each user. As long as every operation is received, the result will be the same. This is especially suited for distributed systems, where the order in which operations are received by each node in the network varies.

When it comes to storing the conflict-free replicated data type on the server, it is the set of operations that must be stored. When a new client retrieves the set of operations, he will declare a new conflict-free data type and update it with the retrieved set.

In the case of strings, the implementation is more complex because a string is sequential data: the order of the characters matter. 'hello' is not the same string as 'olleh'.

There are multiple algorithms that implement conflict-free strings. One of them is the LSEQ algorithm [67]. It is explained very well in [26]. The explanations of the algorithm below are drawn from this resource.

In this algorithm, the characters of the string are represented by a tree. The string is obtained out of the tree through a depth-first search starting from the left, like in this example:

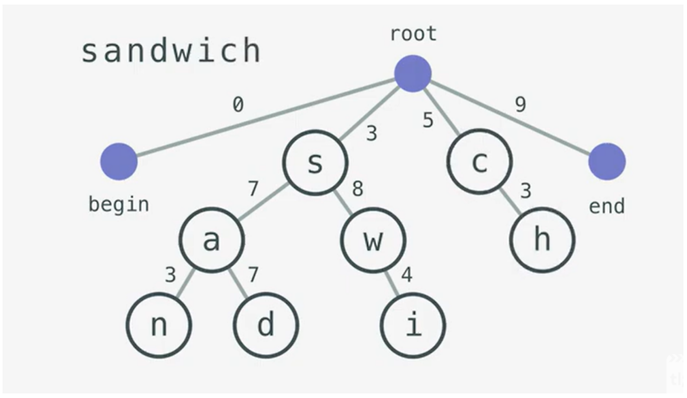


Figure 4.14: Example of a tree representation of a string

Source: [26]

Each character in the string is attributed a unique id based on its position in the tree:

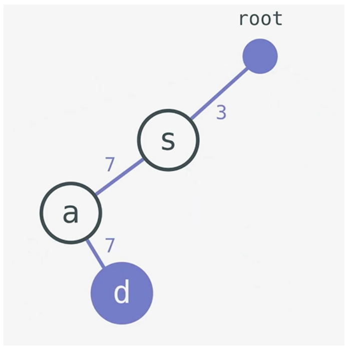


Figure 4.15: Example of a node path determining its id

Source: [26]

In this case, the id of the node would be [3.7.7].

When a character is to be inserted at a specified index of the string, the index will be mapped to character ids between which the character must be inserted.

For example, in the string above, to insert the letter 'y' at index 4, the algorithm would map that index to the space between the letter ‘d’ and ‘w’, meaning the space between id [3.7.7] and [3.8]:

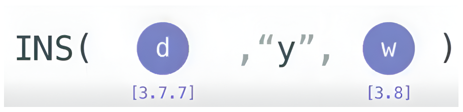


Figure 4.16: Example of an insertion operation for the LSEQ algorithm

Source: [26]

The inserted character can then be given any unique id between [3.7.7] and [3.8] such as [3.7.9] for example:

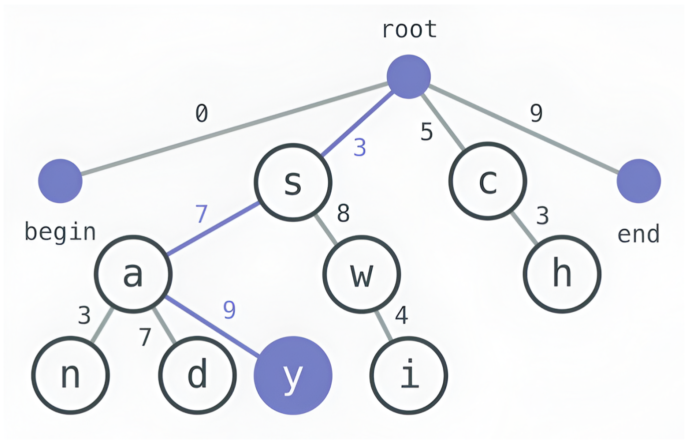


Figure 4.17: Example of an inserted character in the tree model of the LSEQ algorithm

Source: [26]

If there are no more numbers available between two characters, for example if the lower bound had id [3.7.9] and the upper bound had id [3.8], the tree would create a new layer and the id of the inserted character would be [3.7.9.1]. This is bound to happen because, since ids must be unique, they can also never be reused. A deleted character will be removed from memory, but his id will not become available again and if the node of the character in the tree still has sub nodes, the node will also stay in the tree. There can theoretically be an infinite amount of layers in the tree.

Ids must also be immutable. The unique id of each node in the tree must never change, because the algorithm relies on this immutability to be consistent and allow for synchronization between versions: all operations must always lead to only one possible outcome for all versions.

Through this system of unique ids, the algorithm achieves the three requirements of the conflict-free replicated data types. It achieves idempotence because when merging two sets of operations, duplicate ids are removed.

How commutativity and associativity is achieved is well demonstrated by the following figure:

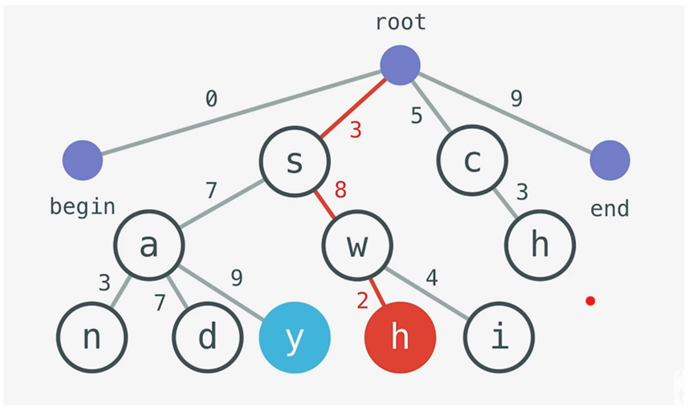


Figure 4.18: Example of a concurrent modification on different indexes with the LSEQ algorithm

Source: [26]

In this example, one user inserts the letter 'y' at index 4 of his copy and another user inserts the letter 'h' at index 5 of his copy. These changes are then broadcasted to the other user.

What is sent to the other user is not the index operation. Otherwise, the order of the operations would matter like it does in operational transformations. Instead, it is the character with the assigned unique id that is sent to the other user:

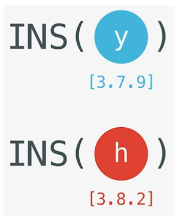


Figure 4.19: Examples of operation broadcasts with the LSEQ algorithm

Source: [26]

Because those character ids are unique and immutable, they always end up at the same place in the tree. Their potential grouping or the order in which they are received will not have any effect on their final destination in the tree.

There is however one edge case that is worth addressing. It is the case in which two users concurrently insert characters at the same index. This is addressed by the unique character ids generated by the algorithm. Those unique ids also distinguish between different trees. So if a change made by another tree is received at the same time as a local change is made to the same index, the algorithm will be able to detect that it comes from another tree. And the ids are crafted in such a way that their concurrent insertion will lead to the same outcome on both trees. Like this:

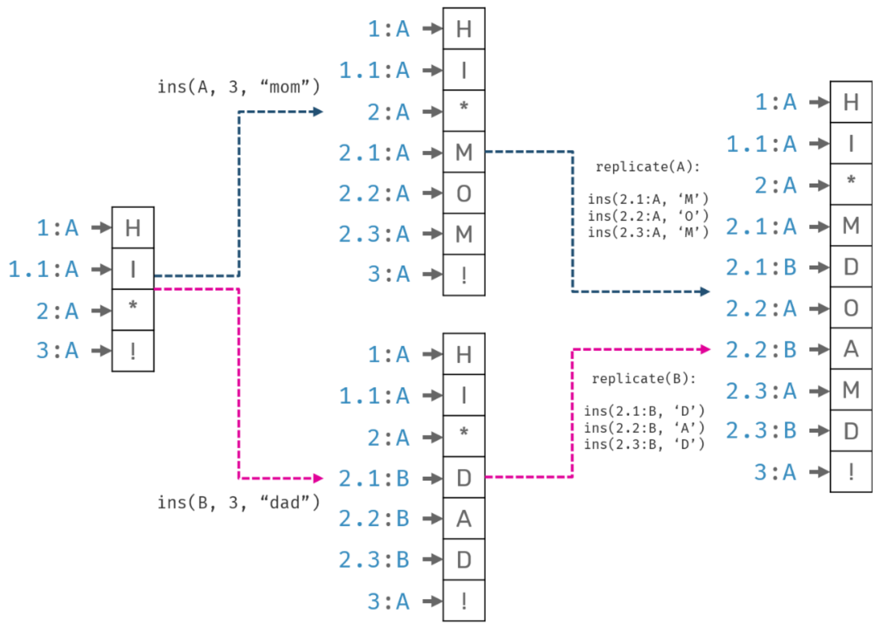


Figure 4.20: Outcome of a concurrent insertion at the same index of a string with the LSEQ algorithm

Source: [26]

The drawback is that the concurrent operations will be interlaced, and that does not respect the intention of the users. In this case, an outcome that would respect the intention of the users would be something like 'HI*MOMDAD!'. The convergence principle is however respected.

The conflict-free replicated data type is the most effective algorithm today to provide real-time version conflict solving.

Differential synchronization

This method was released in 2009 by Neil Fraser from Google [43]. It was released before both operational transformations and conflict-free replicated data types, but it is worth mentioning it last to be able to compare it to the two previous approaches.

In this approach, each client keeps a shadow copy of the document. When changes are made to the main document, a diff operation is made between the shadow copy and the main copy to extract the least amount of insert and delete operations that are required to go from the shadow document to the main document.

This series of operations is then sent to the server as a group. This group of operations is called a patch. This is why the algorithm is also referred to as diff patch algorithm.

The shadow copy of the client is also updated with the changes made on the main copy.

On the server, there is a main document. Also, for each client, a shadow copy that reflects the shadow copy of the client's document and a backup copy. This backup copy reflects the last change of the client's document that was successfully received by the server. A patch made by the client is sent to the server. On the server, it is inserted in the shadow copy. After that, there is a check between the shadow copy of the server and the shadow copy of the client. This is done to verify that the server received the patch of the client. If the patch is successfully received, it is applied to the backup copy. If there is a difference between the shadow copy of the client and the server, there is an error, because the two versions must always be identical after a patch. In that case, all the changes made by the server are rolled back to the version of the backup copy. The client is made aware by the server that there was a rollback, and he resends the patch. This operation is repeated until the patch is successfully received and applied by the server.

This means that the client stores his latest patch in case it needs to be resent. The client is also informed by the server in case that his patch was successfully received and applied. Only then does it flush his latest patch.

In case the patch is successfully received and applied by the server, the patch is applied to the main copy of the server. This main copy is the copy that is common to every client. So when the patch is applied to the main copy of the server, it is highly likely that some changes from other clients are present. The presence of those changes may nullify the incoming patch.

Here is how this situation is handled:

a. Client Text, Common Shadow and Server Text start out with the same string: 'Macs had the original point and click UI.'

b. Client Text is edited (by the user) to say: 'Mac<u>intoshes</u> had the original point and click <u>interface</u>.' (edits underlined)

c. The Diff in step 1 returns the following two edits:

```
@@ -1,11 +1,18 @@
Mac
+intoshe
s had th
@@ -35,7 +42,14 @@
ick
-UI
+interface
.
```

d. Common Shadow is updated to also say: 'Macintoshes had the original point and click interface.'

e. Meanwhile Server Text has been edited (by another user) to say: '<u>Smith & Wesson</u> had the original point and click UI.' (edits underlined)

f. In step 4 both edits are patched onto Server Text. The first edit fails since the context has changed too much to insert 'intoshe' anywhere meaningful. The second edit succeeds perfectly since the context matches.

g. Step 5 results in a Server Text which says: 'Smith & Wesson had the original point and click interface.'

h. Now the reverse process starts. First the Diff compares Server Text with Common Shadow and returns the following edit:

```
@@ -1,15 +1,18 @@
-Macintoshes
+Smith & Wesson
had
```

i. Finally this patch is applied to Client Text, thus backing out the failed 'Macs' → 'Macintoshes' edit and replacing it with 'Smith & Wesson'. The 'UI' → 'interface' edit is left untouched. Any changes which have been made to Client Text in the mean time will be patched around and incorporated into the next synchronization cycle [43].

This is what an architecture under this algorithm looks like:

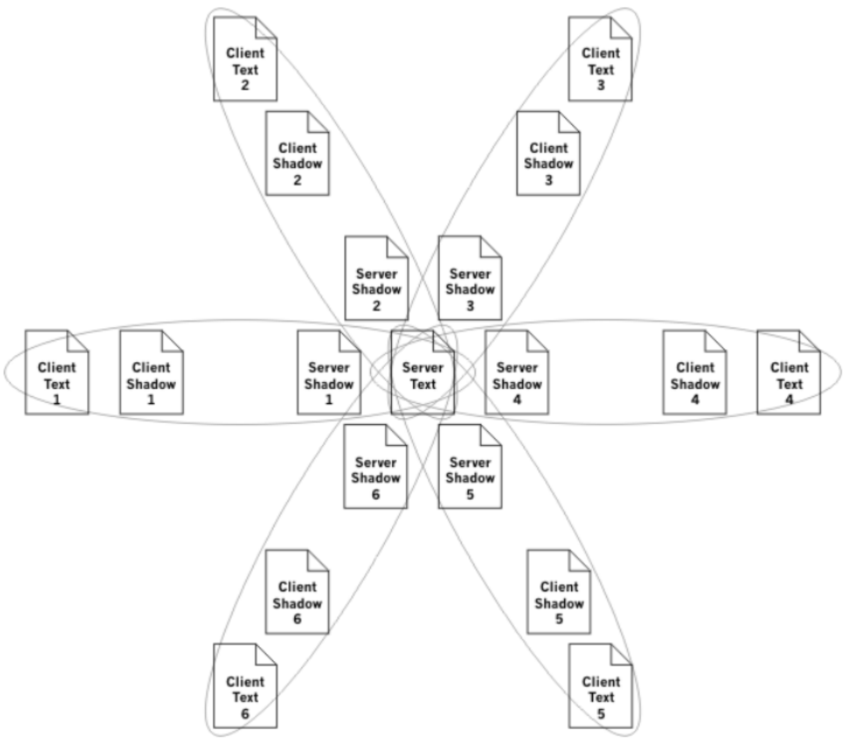


Figure 4.21: Six client, one server synchronization network.

Source: [43]

This figure omits the backup copies.

Because operations in this algorithm are also index-based, only one patch can be applied at a time.

The way conflicts are handled, as quoted above, can also be quite detrimental to the users' intention. But it does have the advantage of ensuring convergence. There is another advantage with algorithm working with patches; a group of operations. This avoids the kind of operation interlacing that can happen with conflict-free replicated data types.

Another important disadvantage lies in the fact that the server stores string values on which diff operations are performed. These diff operations could be avoided if the string operations would be stored instead like in operational transformations. The use of a shadow version and a backup versions on the server for every client also makes the algorithm heavy for the server. In operational transformations, the server sends an acknowledgement to the client when it successfully receives an operation and in conflict-free replicated data types, a failed send has no impact on the final result because the change can be sent later, after other changes, and the final result will be the same.

The diff algorithm

This algorithm is used to extract the shortest amount of insert and delete operations to go from string A to string B. It is required to implement any of the version conflict solving methods mentioned above because they all work with string operations.

In a web interface using Javascript, when a change is made in a text element, what is returned by the listener is the new value. A diff operation is then needed to extract the required insert and delete operations.

The standard algorithm to achieve this is the one developed by Eugene W. Myers [65].

The following algorithm's rundown is drawn from [24].

It is done by putting the two strings in a table. The starting string is at the top of the table and the final string is on the left side of the table.

Every cell for which the two corresponding characters are equal are marked with a diagonal.

The next step is to draw a graph where the nodes are at the intersection of the cells. The graph starts at the top left corner of the table.

The table is traversed in turn. At every turn, the tree can add one node by going down or right, except if the created node is at the top left corner of a marked cell. In that case, another node is immediately added at the bottom right of the marked cell. The traversing is complete as soon as at least one tree reaches the bottom right corner of the table. If multiple tree reaches the end at the same turn, they are all equally valid solutions.

A vertex moving down is an insertion of the character to the left of the vertex behind the character at the right of the vertex. A vertex moving right is a delete operation of the character above the vertex. A vertex from the top left to the bottom right of a cell means that the character above the vertex is kept.

Here is an example of going from string 'CBABA' to 'ACABB':

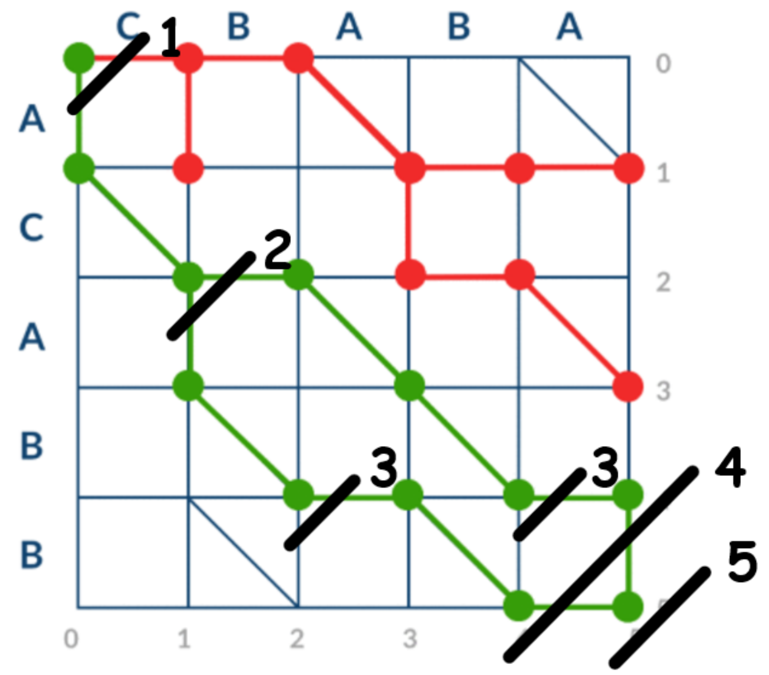
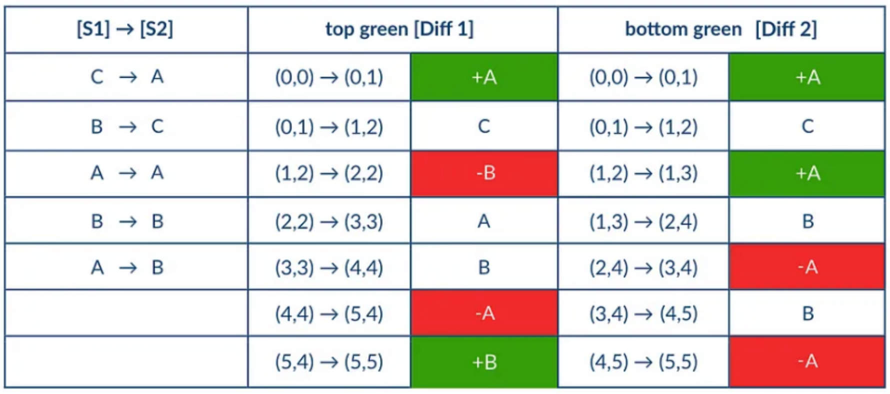


Figure 4.22: A graph made according to the Myers' algorithm

Source: [24]

In this graph, there are two valid paths that both reach the bottom right of the table in 5 turns.

They map to the following string operations:



[S1] → [S2]	top green [Diff 1]		bottom green [Diff 2]	
C → A	(0,0) → (0,1)	+A	(0,0) → (0,1)	+A
B → C	(0,1) → (1,2)	C	(0,1) → (1,2)	C
A → A	(1,2) → (2,2)	-B	(1,2) → (1,3)	+A
B → B	(2,2) → (3,3)	A	(1,3) → (2,4)	B
A → B	(3,3) → (4,4)	B	(2,4) → (3,4)	-A
	(4,4) → (5,4)	-A	(3,4) → (4,5)	B
	(5,4) → (5,5)	+B	(4,5) → (5,5)	-A

Figure 4.23: Graph operations mapped to string operations for the two most efficient solutions when going from string ‘CBABA’ to ‘ACABB’

Source: [24]

4.2 Technical design

System modeling

Once the core requirement specification was established, we proceeded to model the system.

With the help of UML diagrams we illustrated the system design with various models providing different perspectives on the system.

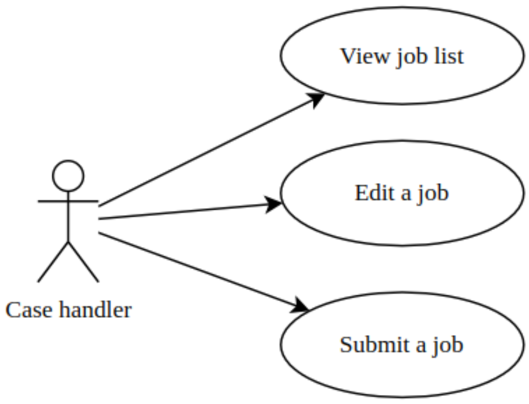


Figure 4.24: Use case displaying the role of the case handler in the system

The figure 4.24 shows the role of a case handler within the boundaries of the system.

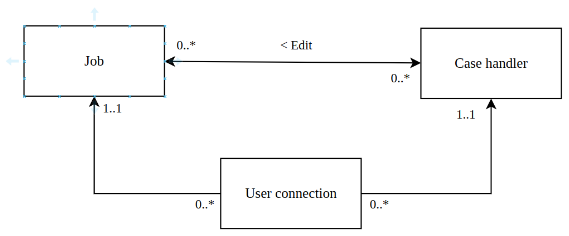


Figure 4.25: Domain model

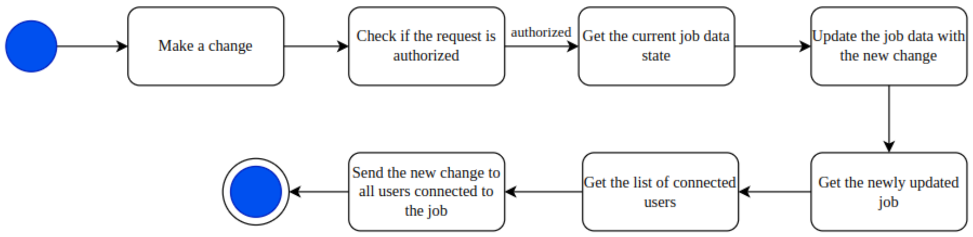

Make a change
Check if the request is authorized
authorized
Get the current job data state
Update the job data with the new change
Get the newly updated job
Get the list of connected users
Send the new change to all users connected to the job

Figure 4.26: Data flow of a job update request

On 4.26 we visualized the journey of data as it travels through the pipeline, starting from the moment a user initiates a change and ending at its destination on other users' web interfaces.

System architecture

The requirement specification was crucial in guiding the architectural decisions for our system. Certain requirements held considerable importance in shaping our choices. These factors encompassed security, performance and the essential need for Websocket connections.

Ensuring the responsiveness of our system was paramount for delivering a satisfactory user experience, particularly to justify the investment in real-time, two-way communication. To optimize performance, we looked at several factors:

- Network delays
- Locality of system components
- Overhead associated with Websocket connections
- Overhead related to data parsing between the components

To sustain the application's deployment on the public cloud while iteratively introducing updates to the live environment. This approach enabled us to evaluate the application's performance post-deployment and this would impact our further development decisions.

Due to our desire to keep the application publicly deployed, our client enforced a security mandate that required the concealment of the application from unauthorized access when the application was temporarily deployed on the public cloud.

Because of these factors, we chose to implement a layered architecture.

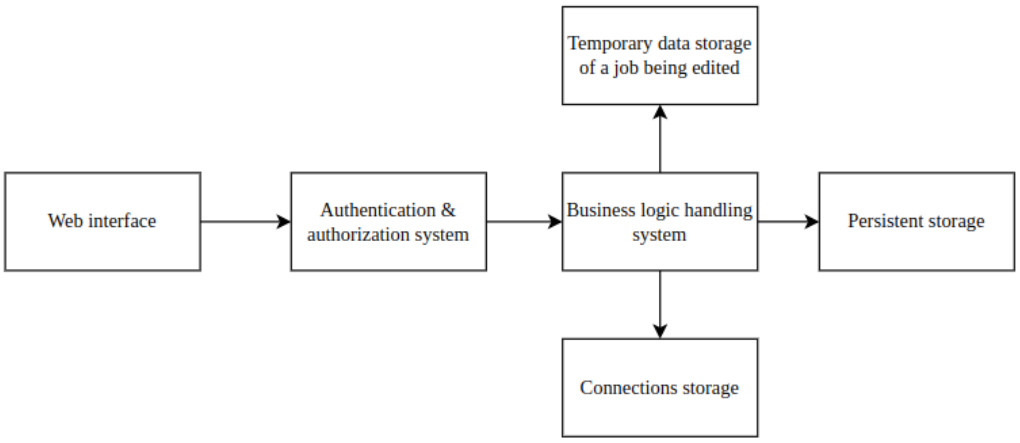

Temporary data storage of a job being edited
Web interface
Authentication & authorization system
Business logic handling system
Persistent storage
Connections storage

Figure 4.27: High level architecture overview

REST API

Our client requested the implementation of an API, making it a mandatory aspect of our project. While the choice of technology was left to us, we considered various languages and frameworks, including Go, NodeJS, Python, Java, and more. Ultimately, we opted for the Dotnet technology stack with the C# language. Several factors influenced this decision:

Integration with Norkart's Existing Software: Dotnet and C# were already prevalent technologies within Norkart, facilitating seamless integration with their existing software systems.

Demand in the Norwegian Market: C# holds significant demand in the Norwegian job market, enhancing our employability after graduation.

Prior Familiarity: Our team had some familiarity with C# due to previous coursework, particularly in game development. Additionally, one team member had prior experience with backend technologies, easing the learning curve for the Dotnet framework.

Had circumstances been different, Go would have been our alternative choice due to its simplicity and familiarity.

Websocket API

To enable the collaborative feature in our application, we chose to implement Websocket. While .NET offers the robust SignalR library, built on top of the Websocket API, we opted to handle the connection using the base Websocket API provided by the .NET platform for learning purposes.

Websocket messages are sent in plain text. We structured the messages into JSON format for consistency and ease of parsing.

Persistent storage

Regarding persistent storage, we had the flexibility to choose any database technology. After considering both SQL and NoSQL options, we selected MongoDB. MongoDB is a document-based database that can be hosted locally or on the MongoDB cloud platform. To streamline deployment, we opted for the cloud-based solution.

Cache

Caching data enhances the application's performance and reduces unnecessary requests to persistent storage during data retrieval. We chose a caching mechanism using simple dictionary-based data structures provided by the programming language (a singleton class with a dictionary field).

Although we considered using Redis as a standalone cache, we decided against it due to the additional overhead involved in deploying and managing the cache database.

4.3 Graphical user interface (GUI)

We made a GUI based on the Norkart website but slightly simplified and with a few needed features customized. For instance, a card 'viewer' points out the number of users viewing the same page as the user which increases interactivity and gives an alert to the user. Moreover, we integrated a feature of adding new jobs directly through the GUI. This enabled data input in various forms without the need to access the backend. This improved the process of streamlining, thus affecting efficiency.

We deliberately decided to not use pre-designed templates so that the interface is simple and put more focus on the functionality behind it. This option allowed much tailoring and a better level of tuning with our project goals. The GUI is iteratively refined, taking the testers' feedback into account, in such a manner that it becomes user-friendly and effective for its purposes.

4.3.1 Unique GUI Components

The following are the components that are developed uniquely for our GUI and have no counterpart in the existing Norkart client system. The features were designed to give a very efficient, lean, and user-friendly interface based on the needs of our specific project, thus eliminating all the features that render the system complex today.

4.3.2 Add New Job Dialog

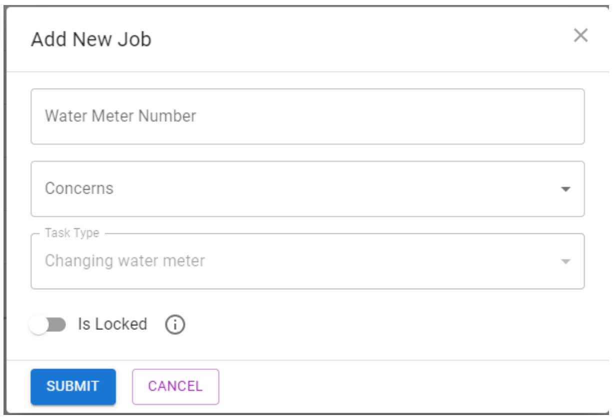


Figure 4.28: The Add New Job dialog for Norkart is to provide the option to create new jobs directly through the GUI in a much faster way; hence, a great number of forms available directly to put in data will result in better interactiveness.

4.3.3 Warning Dialog

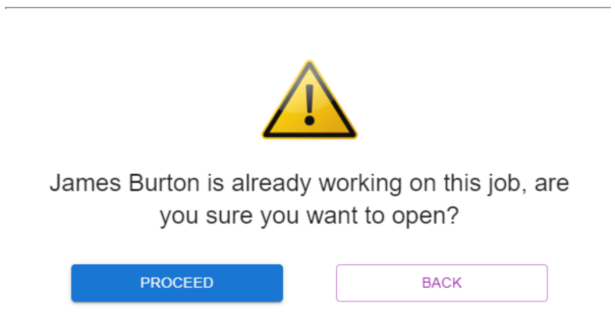

James Burton is already working on this job, are you sure you want to open?

Figure 4.29: Alert dialog warning the user that the job is currently active and being worked on by a different user, that is, James Burton. This helps create accountability for collaboration on the platform.

4.3.4 Viewer Card

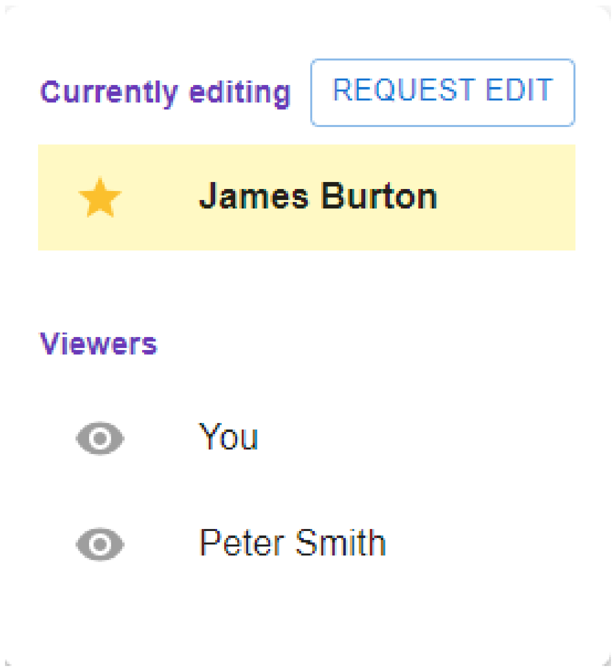


Figure 4.30: Viewer card displaying now active users for the job. It thus fosters transparency. Users can know who is viewing and who is editing the job.

4.3.5 Job Completion Notification

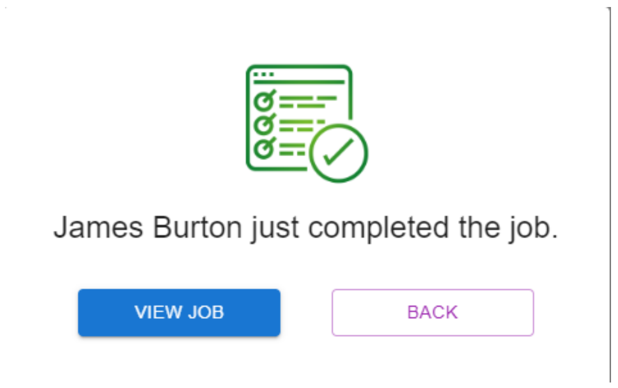

James Burton just completed the job.

Figure 4.31: Notification dialogue popping up stating that James Burton completed the task. This ensures that all current viewers of the job are updated on the completion of a task.

Chapter 5

Implementation

This chapter encapsulates the translation of our design into executable code. Initially, we describe the project as a whole. Then we discuss the primary components of the project: frontend and backend implementations. Finally, we explore a domain-specific implementation regarding job handling, authorization, notification systems, and connection management.

5.1 The 3-tier architecture

Our application follows the classic three-tier architecture model, comprising distinct layers for presentation, business logic, and data management.

The chief benefit of three-tier architecture is that because each tier runs on its own infrastructure, each tier can be developed simultaneously by a separate development team. It can also be updated or scaled as needed without impacting the other tiers [2]. The figure 5.1 shows the idea.

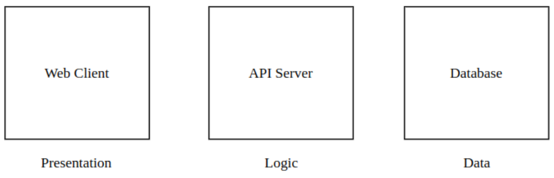


Figure 5.1: Illustration of the three-tier architecture

5.1.1 Presentation tier

It acts as the bridge between the user and the system, providing the interface through which users interact with the application. This is the frontend part of our application. The presentation layer communicates with the business logic layer to perform operations based on user input.

5.1.2 Business logic

The core business logic resides within the backend API server. A set of functionalities is implemented to handle tasks, including security enforcement, data validation, management of concurrent user connections and doing operations on jobs.

The business logic layer handles the interaction with the data access layer, so the presentation layer indirectly accesses data stored in databases or other storage systems.

For example, after the business logic layer processes a user's request, it retrieves data from the data access layer and sends it back to the presentation layer to be displayed to the user.

5.1.3 Data tier

In a three-tier architecture, the data tier (also known as the data access layer or database layer) is the bottom layer responsible for managing the application's data. This tier handles the storage, retrieval, and manipulation of data and is typically where the database is [2].

Our data tier is powered by MongoDB, a cloud-based NoSQL database solution.

5.2 Frontend

5.1 Technology Stack

We deliberately designed our frontend technology stack with the same components that are already in use at Norkart: this is a way to make a product that is not only performant and responsive but is also harmonized with a proven technology stack both for maintainability and further extensions. This has been picked because it assures the solution to be valuable to Norkart and to give them the same positive results.

The best of these technologies is integrated into our application: React, Vite, TypeScript, and Material UI.

React: We chose to work with React because it provides a dynamic backbone for the user interface. Its component-based nature enables the creation of reusable user interface components, and therefore developers can ultimately scale and maintain them easily. In React, we tapped into a massive ecosystem with high quality libraries that enable feature-rich responsive designs with reflected efficiency and elegance.

- **Vite** [105]: We arrived at Vite as our build tool because of its truly impressive speed and unparalleled development experience. Vite does away with the normal bundling process while developing and chooses to spin up the server instantly and change the modules hot. This decision helps to have a productive and fast development cycle, with rapid feedback and iteration for us.
- **TypeScript** [99]: We hereby introduce static typing for JavaScript, which helps a lot in reducing runtime errors and making the quality of the code better. It is really good for documenting the shape of data and the behavior that comes with it in ways that make the code clear and maintainable.
- **Material UI** [63]: We use Material UI because it has a huge library of built-in components that are all Google Material Design compatible. This allows for a consistent and intuitive user interface for easy crafting of polished designs with the modern standards of an interface.

Norkart allowed us to use the technology stack we wanted, although we decided to go for the very same technologies they are using mainly to ensure that the solution we develop is compatible with their environment and because the technologies they use have big advantages.

Websockets

We used Websockets to apply the functionality of simultaneous editing which allows us to push the data from the server to the client in real-time. For learning purposes, we used raw Websockets instead of the SignalR package, which is the standard and the best practice when having a Dotnet backend.

Version conflict solving

At the ending stage of the project, we looked at ways to possibly implement version conflict solving. We found two packages: the Yjs package [**yjs**], a conflict-free replicated datatype and the jsdiff package [58], a package implementing the diff algorithm presented in the design chapter.

When a change is made to the text field, the jsdiff extracts the string operations that lead to the change. Those operations are applied to the Yjs string variable. The updated Yjs string is then sent through Websocket to the server that then relays the change to every other user connected to the job. The received Yjs string is then applied to the local Yjs string.

A package based on conflict-free replicated datatypes is the best choice because it is the best available algorithm for version conflict solving so far. After finding this package, we did not explore other options. Instead, we focused our time on learning how to use the package because this was something new for us. It was not easy because there was not a lot of community support for it.

For the jsdiff package, it implements the diff algorithm from Eugene W. Myers [65]. Out of our theoretical research, this algorithm is the standard. Therefore, once we found a package that implements it, we looked no further.

5.3 Backend

This section describes the structure of the backend part.

Our backend server consists of two types: stateless and stateful. The stateless part is the one that uses REST principles and the stateful part serves over Websocket connections and keeps track of the socket connections in memory.

For the backend, as mentioned previously, we used the .NET Core framework for building APIs in C# specifically. It was possible to use C# without any framework, however, we would have to implement the networking logic from scratch, and that would lead to significant delays. The .NET API framework provides a way of implementing simple web-based APIs in a secure, testable and decoupled manner. It provides the dependency injection container that serves as a central registry for managing dependencies in an application. Its purpose is to facilitate the decoupling of components by dynamically providing instances of required objects (dependencies) to the classes that need them.

Project structure

Our backend project was divided into the following directories:

- Controllers
- Services
- Repositories
- Models
- Data

This structure was inspired by the Robert C. Martin's book Clean Architecture [61]. The clean architecture proposed by Robert is a software architectural pattern that emphasizes separation of concerns and independence of frameworks and tools. Even though we used the .NET Core framework, the majority of code is written in pure language constructs with its standard library. The parts where we relied on the framework were the provided dependency injection container 5.2 that would compose all our classes in a centralized manner and the endpoint routing mechanism that is abstracted in the .NET Core controllers that made us more focused on the business logic of the application.

```
builder.Services.AddScoped<IJobRepository, MongoDBJobRepository>(...);
builder.Services.AddScoped<IUserRepository, MongoDBUserRepository>(...);
builder.Services.AddScoped<IWebSocketRepository, WebSocketRepository>();
builder.Services.AddScoped<IWebSocketService, WebSocketService>();
builder.Services.AddScoped<ICacheService, CacheServiceMock>();
builder.Services.AddScoped<IJobService, JobService>();
builder.Services.AddScoped<IUserService, UserService>();
```

Figure 5.2: Program.cs - dependency injection container in action

In our API, the use of controllers, services, and repositories did help to organize the code in a way that promoted testability and maintainability.

Models

This is the place where we defined the domain of our application. Models are business domain entities such as Job, User, WaterMeter. By defining the models, we clearly state what the application is about.

Controllers

Controllers are the adapter classes that are responsible for handling incoming HTTP requests, interpreting them, and orchestrating the appropriate response. They act as the entry point for the API endpoints, defining the routes and actions that correspond to different HTTP methods (GET, POST, PUT, DELETE, etc.). By separating the handling of HTTP requests from business logic, controllers promote separation of concerns and make the codebase easier to understand and manage [83].

Services

Services contain the majority of the business logic of the application. They encapsulate the core functionality of the API, such as data manipulation, validation, and business rules. By abstracting this logic into service classes, we achieved modularity and maintainability. Services can be reused across multiple controllers, promoting code reuse and minimizing duplication. Figure 5.3 shows an example of how a service interface looks like in our application. Notice that the naming of the interface methods include the domain specifics.

```
public interface IWebSocketService {
    public Task<List<ConnectedUserDTO>> GetConnectedUsersByJobId(Guid jobId);
    public void AddSocket(WebSocket socket, Guid jobId, string userId);
    public Task NotifyCurrentUsersAboutNewUser(Guid jobId, string userId, string username);
    public Task SendToAllExceptThisSocket(WebSocket socket, Guid jobId, string data);
    public Task NotifyEditorsRequestEditPermission(Guid jobId, string userId);
    public Task NotifyEditorsDenyEditPermission(Guid jobId);
    public Task NotifyAllGrantEditPermission(Guid jobId, string granteeUserId);
    public Task SendToAll(Guid jobId, string data);
    public Task RemoveSocket(WebSocket socket, Guid jobId);
    public Task<bool> IsAllowedToSendMsg(Guid jobId, string userId);
}
```

Figure 5.3: IWebSocketService.cs - Websocket service interface.

Repositories

Repositories are responsible for interacting with the data layer. They encapsulate database access logic, abstracting away the details of data storage and retrieval. Figure 5.4 shows a code snippet of a repository used in our application. The naming of the methods of the repository indicates retrieving from or adding to some kind of storage.

```
public interface IWebSocketRepository
{
    public List<UserConnectionsDTO> GetAllByJobId(Guid id);
    public UserConnectionsDTO? GetUserSocketsByUserAndJobId(Guid jobId, string userId);
    public string? RemoveSocket(WebSocket socket, Guid jobId);
    public void AddSocket(WebSocket socket, Guid jobId, string userId);
}
```

Figure 5.4: IWebSocketRepository.cs - Websocket repository interface.

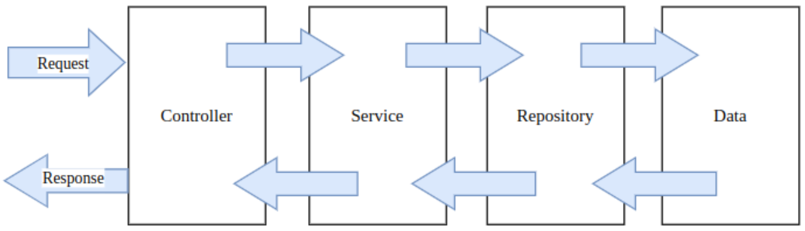


Figure 5.5: Data flow through the layers of abstractions in the backend

REST API

In order to retrieve, insert and update data that did not require real-time connectivity, we used a REST API. We chose the method because it is what Norkart uses. It is quick and simple to implement with the .NET Web API framework.

Websockets

The main problem that we had to solve was real-time working on a job page by multiple case handlers. We decided to implement Web Sockets for this, and this approach would allow for pushing data from the server to the clients in real-time. One alternative was to use long polling, however Web Socket implementation leads to higher performance and less memory usage due to HTTP long polling causing additional HTTP headers to be included with each request and response [5].

After the project was implemented, we learned about the HTTP 2 standard, which would have been an even better alternative.

Interfaces instead of classes

In our implementation of the backend, we relied heavily on using classes through interfaces that they implemented or inherited. It was enforced by the ASP.NET framework, but soon we found out that it had a good reason to do so. It made us produce a more decoupled and maintainable code. See figure 5.6.

```
// Instantiates MongoDBUserRepository where IUserRepository is passed as an argument
builder.Services.AddScoped<IUserRepository, MongoDBUserRepository>(...);

// If we decide to switch to a PostgreSQL database in the future,
// we can create a PostgreSQLDBUserRepository that implements IUserRepository.
// We would simply replace MongoDBUserRepository with PostgreSQLDBUserRepository,
// without needing to change any of the classes that use the repository.
```

Figure 5.6: Program.cs - interfaces allow for safer class substitution

5.4 Domain

Having talked about the structure of our backend and frontend implementations, we describe below the specifics of the domain that comprise jobs, users and connection handling.

5.4.1 Jobs

In this section, we describe the implementation of simple job operations as well as maintaining a temporary job data in a cache while it is being edited.

CRUD

We set up basic actions for dealing with job information in our application: creating new jobs, reading existing ones, updating job details, and deleting jobs that are no longer needed. These actions make it easy for users to manage job-related data. Whether they're adding new job listings, checking out job details, editing information or getting rid of old entries, these actions help users get things done quickly and easily.

Even though we defined the actions in the backend, not all of them were accessible to the end users. Deleting jobs was added only for development purposes. This is indicated by the 'DevOnly' attribute that is placed above the method definition that can be seen in 5.7.

```
[Route([controller])]  // <-- all actions within this class expect endpoint
                       // <-- starting by the name of this class without the word Controller
                       // <-- example.com/Jobs
[ApiController]
public class JobsController(...) : ControllerBase
{
    [AllowAnonymous]      // <-- the action does NOT require an authorization token
    [DevOnly]             // <-- the action is only accessible in development mode (e.g testing)
    [HttpDelete]          // <-- expects DELETE HTTP method
    [Route("{id:guid}")]  // <-- endpoint expects a path argument (e.g example.com/Jobs/123123)
    public Task<IActionResult> DeleteAsync([FromBody] Guid id) {...}

    // ... snip other methods/actions
}
```

Figure 5.7: JobController.cs - delete a job by id action. Showcasing the controller and attributes

Job being edited

When a user enters a job page, the job fields are prefilled with up-to-date data from the last write. This ensures synchronization of the job fields across all users currently on the job page. It happens in the following steps:

1. An HTTP request is sent to server to get the job by its id
2. The corresponding job controller endpoint action handles the request
3. The job cache is checked first. In case of a cache hit, it sends the data back to the client
4. In case of a cache miss, the request proceeds to retrieve the job data from the persistent storage, inserts it in the cache and then it sends it back to the client

Before the job data ends up in the persistent storage, a user has to type something in the fields the first time the job is opened. After a write to a field, a message is sent to the server. The figure 5.8 shows how the message is handled by the controller.

```
public async Task ProcessWSMessageAsync(...) {
    -- snip --

    if (isWebSocketRequest) {
       async void HandleMessage(...) {

          -- snip --
          if (message == "meterModel"
            || message == "meterNumber"
            || message == "shouldRegisterNewMeter"
            || message == "isHandledManually"
          ) {
            cache.SetData(...); // Save temporarily to the cache
            await webSocketService.SendToAllExceptThisSocket(...);
          }

          -- snip --
       }
       -- snip --
    }
}
```

Figure 5.8: Backend: WebSocketController.cs - job field is updated. Simplified for an easier understanding

So the data is stored in the job cache while a job is being edited. Since the cache is just an in-memory dictionary static field of a class, it has a potential to get accumulated endlessly. It could lead to a couple of problems.

- When the server is re-started, all data contained in the cache is lost
- Memory is not endless

We solved the first problem by occasionally flushing the data to the persistent storage. It happens when all users have left a job page.

```
// Websocket controller class
public async Task ProcessWSMessageAsync(...) {
  async void HandleMessage(...) {
    -- snip --
    case WebSocketMessageType.Close:
    await webSocketService.RemoveSocket(ws, jobId);
  }
}

// Websocket service class
public async Task RemoveSocket() {
  -- snip --
  await cache.FlushOne(jobId); // Remove the job data from the cache, and save in the database
}

// cache class
public bool RemoveData(...)
{
    -- snip --
    JobDataCache.CacheData.Remove(jobId);
}

public async Task FlushOne(...) {
  RemoveData(jobIdKey);
}
```

Figure 5.9: Backend - Process of closing a Websocket connection. Simplified

Completed job

When a user completes a job, an HTTP request to that effect is sent to the backend. The jobs controller then notifies the completion to every other user inside the job through their Websocket connection. This is done by inserting the Websocket service class in the jobs controller:

```
// Websocket controller class
public class JobsController(IJobService jobService,
                            IWebSocketService webSocketManager,
                            ICacheService<Job> cache) : ControllerBase
-- snip --
```

Figure 5.10: Backend - Inserting the Websocket service class in the jobs controller.

The Websocket service class is then used to send the completion notification to other users:

```
 [HttpPut]
 [Route("{id:guid}")]
 public async Task<IActionResult> UpdateAsync([FromRoute] Guid id,
                                              [FromBody] UpdateJobDTO updateJobDTO)
{
-- snip --
 /* For when the job is completed */
 if (updateJobDTO is { Status: JobStatus.Completed, CompletedBy: not null })
 {
     -- snip --
     var completedUserObject = new
     {
         jobGotCompleted = "true",
         userCompleted = updateJobDTO.CompletedBy
     };
     var jsonObject = JsonSerializer.Serialize(completedUserObject);

     /* a message is sent to all open web socket connections to announce in
     real time that the job got completed */
     await webSocketManager.SendToAll(id, jsonObject);

     return NoContent();
 }
}
```

Figure 5.11: Backend - Notifying a job completion through Websocket.

This process can be modelled as follows:

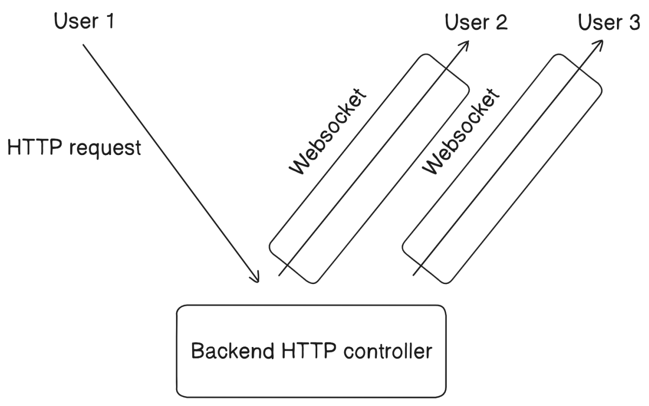


Figure 5.12: Triggering a Websocket broadcast through an HTTP request

5.4.2 Users and authorization

Similarly to jobs, we defined the CRUD-endpoints for management of users. The 5.13 showcases the summary of user controller endpoints.

Authorization

In addition the CRUD endpoints, there is a login-endpoint that handles incoming credentials and returns a generated JWT token. The token is sent to the client, and it will then be stored as a cookie on the clients browser. Each subsequent request from the client will then contain the token in the cookie in order to get access to the endpoints that require the token.

```
public class UsersController(...) : ControllerBase {
    public Task<IActionResult> GetAllAsync()
    public Task<IActionResult> GetByIdAsync(...)
    public Task<IActionResult> CreateAsync(...)
    public Task<IActionResult> UpdateAsync(...)
    public async Task<IActionResult> DeleteAsync(...)
    public async Task<IActionResult> LoginAsync(...)
}
```

Figure 5.13: UsersController.cs - user endpoint actions

5.4.3 Notification system

One of the critical parts in our application was the notification system that would provide the necessary functionality for case handler collaboration on a job resource. In this section, we describe the implementation of the system.

- Notify users about incoming user
- Notify users about leaving user
- Notify editors about pending editor permission from another user
- Notify user when granted editor permission

  For those notifications, data is sent by a user to the server through Websocket and the server then relays the data, also through Websocket, to other users inside the job. It can be modelled as follows:

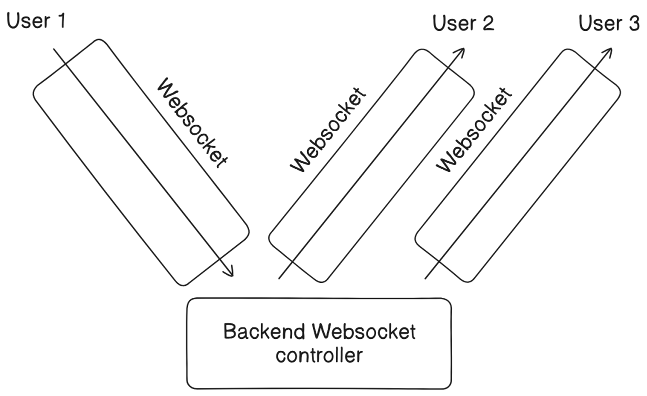


Figure 5.14: Server acting as a relay through the use of the Websocket protocol

- Notify incoming user about users on the page

Incoming user

In order to notify other users on a particular page about an incoming user, we expect a Websocket message being sent to the server when a new user enters the job page. See 5.15.

The message is sent when the frontend client has established the Websocket connection with the server.

When the user enters the job page, they are also notified about other users that are there from before.

```
public async Task ProcessWSMessageAsync(...) {
    -- snip --

    if (isWebSocketRequest) {
       async void HandleMessage(...) {

          -- snip --
          if (message == "usernameConnected") {
            await webSocketService.NotifyCurrentUsersAboutNewUser(...);
          }

          -- snip --
       }
       -- snip --
    }
}
```

Figure 5.15: Backend: WebSocketController.cs - expect userConnected to process. Simplified for an easier perception

Leaving user

When a user leaves the page, the Websocket connection is closed. A message of type ‘close’ is sent to the server, where it is handled. We remove the stored Websocket connection that is associated with the user from the connection storage, which is just an in-memory dictionary, and then we send a message to other users that that particular user has left the job page.

On the frontend we implemented the corresponding appearence of a snackbar that we mentioned in the design chapter.

Editor permission

We implemented a concept called ‘closed job’ where the first user entering a job page will have rights to edit the job. Other newcomers have only viewing rights. They can only see the job being edited. They are restricted from editing.

In order to gain editing rights, the viewers may request the access from users that already have editing rights. In this case, a message containing the editor permission request is sent to editors. See 5.16

```
    async void HandleMessage(...) {
      -- snip --
      if (message == "editPermission") {
        await webSocketService.NotifyEditorsRequestEditPermission(...)
      }
      if (message == "denyPermission") {
        await webSocketService.NotifyEditorsDenyEditPermission(...)
      }
      if (message == "granteeId") { // user id of a user who is being granted the editor rights
        await webSocketService.NotifyAllGrantEditPermission(...);
      }
      -- snip --
    }
    -- snip --
  }
```

Figure 5.16: Backend: WebSocketController.cs - message handling regarding editor permissions

When the editors receive the editing permission request from the server, a dialogue is opened on their clients containing two buttons: deny or accept.

In case of acceptance, a message 'granteeId' containing user id is sent back to the server and the user connection is updated allowing the user to edit. Then a message is sent back to every user being on the job updating their front end clients.

Chapter 6

Development process

This chapter describes the process that we put in place for the conduct of the project and the project management tools that we used.

6.1 First team meeting and philosophy

Our first group meeting took place in the first week of January.

During this meeting, we presented ourselves to each other and took the time to know each other better.

We also discussed our expectations and ambitions for the thesis. That's when we laid down our main guiding principle for the project: the 80/20 principle.

The way we understood this principle is that 20% of our efforts would produce an outcome worth 80%, and to obtain the remaining 20% would required 80% of the effort. In other words, the idea is that there is a point where a disproportionate amount of effort is required to produce value. And at this point, the result is no longer worth the effort. Our objective in the course of the thesis was to never cross this threshold.

Another core philosophy that we laid down during that first meeting was to work with several small short term objectives that would build on each other throughout the project instead of immediately setting an ambitious objective. We would also proceed in order of difficulty. The easiest objectives would be set first. This was chosen as a form of insurance policy. If a more advanced, ambitious or difficult objective fails, we have something to fall back on.

Those two principles served us well. They made us focused and effective.

Also during that first meeting, we laid down the logistics of our process.

6.2 Meetings

We first decided to have meetings on Mondays to plan the week, another meeting on Wednesdays to follow-up with the work done, and a meeting on Saturdays to both follow-up on the work done and to have a retrospective meeting.

Soon after it was decide to combine the Saturday meeting with the Monday meeting. This meeting would be held on Sundays instead.

It was agreed with our supervisor to have meetings on Tuesdays, as we needed. During the development part, we met our supervisor almost every week to get guidance on the thesis process. After finishing the project and moving to writing this report, we met less frequently and only when we had questions about the writing of this report.

It was later agreed with our contact person at Norkart to meet every Thursday at their office. After the meeting, we could spend the day there to work. Norkart provided us with a meeting room. For the development of the project, we met almost every week. Sometimes, the whole team was present, some other times, one team member would join online.

A typical day at Norkart would start with a meeting with our contact person. We would present the work done in the previous week, get feedback on it, discuss the next step and discuss alternative solutions. We would then pause for lunch. After lunch, we talked at length about what we understood from the meeting, how to integrate the feedback into our work and the next steps we would take. Extensive notes of both meetings would be written down, and then we would work for the rest of the work day.

The meeting on Thursday naturally replaced the meeting that we had planned to have on Wednesdays. However, at around the middle of the project, we added a meeting on Wednesday evening to prepare the next day's meeting with Norkart.

Outside of meetings, the team frequently communicated on the instant-messaging platform Discord.

In our retrospective meetings, we did, in fact, discuss about the process. We did dare to raise issues such as taking detailed notes to track important information, prioritizing merge requests when they are opened so that further changes can always be made against the latest versions, encouraging team members to come to Norkart in person, etc. The meeting notes in appendix attest to that.

Decisions regarding features were all taken as a group, even though responsibilities for the frontend and backend were divided. Because real-time features required work on both the backend and frontend, team members working on the real-time aspect of the project became familiar with both the frontend and the backend. Merge requests were all reviewed by a team member. This made it such that every team member had an overview of the whole project. And members working on real-time features were familiar with the codebase of the whole project. There were no silos in the project. The enforcement of the 80/20 also contributed to this aspect. It allowed us to take the time to have an overview of the whole project.

In short, by communicating often, daring to take up more uncomfortable issues, and having team members have an overview of more then just their areas of responsibilities, we managed to truly work as a team.

6.3 Collaboration framework

Our meeting logistics described above is closest to the SCRUM collaboration framework. However, we did not have daily meetings, either at the start or at the end of the day.

We also made the voluntary decision to not have a SCRUM master. However, still on the first meeting that we had, all of us committed to a proper project management system, and we remained committed to following it throughout the whole project, along with having proper issue tracking.

Just the existence of a proper process made a huge difference in the quality of the teamwork.

We also had only a partial product backlog. We knew that we had to create a prototype replicating the job handling part of the Komtek system. So for this part of the project, we clearly knew which tasks had to be accomplished. However, for the real-time features, it was uncertain which features we would developing and what it would take to develop them. This is something that we established in close consultation with Norkart later on in the project, and that will be detailed below.

Following the 80/20 principle, we made sure to not be over ambitious in the course of the project. So we gave ourselves a good amount of time to implement the various features. Because of that, we managed to deliver to Norkart what we committed to. And, overall, we managed to deliver our work within the schedule that we set in the project plan. This will be detailed below.

We had one-week sprints. A period of one week for the sprints was also helpful because it allowed for a tighter follow-up of the project. We established a sprint backlog every Sunday. At about the middle of the project, we started updating our sprint backlog after the meeting that we had with Norkart. Updating it freshly after having received feedback from Norkart is something that was helpful for us.

6.4 Issue tracking

To track those sprint backlogs and other issues throughout the project, we created an issue board on our GitLab projects. We had two GitLab projects. One for the backend and one for the backend. We therefore had one issue board for each. Issues that were not related to either the backend or the frontend were inserted in the issue board of the backend. It was the default issue board.

Each issue board had 4 columns. One for the opened issues, but that were not part of a weekly sprint. At the end of the project, in the backend, out of a total of 118 issues, 17 remained opened without having been part of a sprint. In the frontend, there are 9 such issues out of a total of 74 issues. This represents a total of 28 issues out of 192. This is about 15% of all issues. Most of those issues, if not all, are issues that we decided to leave aside and work on only if we had free time during the project. This demonstrates our commitment to the 80/20 principle and how we managed to keep our focus on issues that were at the core of the thesis.

The second column was the ‘in-sprint’ column. This column contained issues that were part of a sprint. When sprints backlog were created on Sundays, issues were created in this column.

The third column was the ‘status::in-process’ column. Issues were moved in this column when we started working on them.

The last column was for the closed issues.

We kept the issue board up to date throughout the whole project. However, were did not diligently move issues from the ‘in-sprint’ column to the ‘status::in-process’ column. More often than not, we would move them from the ‘in-sprint’ column directly to the closed column when they were done. We considered an issue done when the associated merge request was merged, after review from another team member.

We were also not too diligent to close an issue right after it was done. Issue boards were updated every Sunday in any case. If there were issues related to finished tasks still opened, they would be closed then.

The comment section of the issues was used to keep track of resources used, challenged faced and solutions found.

6.5 Version control

For our version control tool, we used GitLab.

Specific to our project was to have one GitLab project for the backend, and one project for the frontend. This is because we believed it made it generally easier for us to manage the project in two different repositories. It might just as well have been just as easy with only one repository for both projects.

As detailed in the testing chapter, each merge request was reviewed by another team member. This is also one of the reasons why team members managed to keep an overview of the whole project.

6.6 Project milestones

The following milestones were not explicitly planned at the beginning of the project. However, after the project started, in collaboration with Norkart, we defined 3 different versions. Each version built on each other and each version contained one or more collaboration features.

We have been fortunate that, throughout the project, Norkart shared our core philosophy of developing the project through small objectives that would build on each other, starting from the easiest to the hardest, along with the 80/20

Komtek's job handler replica

Before implementing any 'version', we created a replica of the Komtek's job handler.

This part of the project took about 4 weeks to implement. It was presented to Norkart on February 8.

It was implemented 4 days beyond the planned schedule.

Version 1

After discussion with Norkart, version 1 would include two features. The first feature was to ensure that a job could only be assigned one time. The second feature was that a job could only be completed one time.

For this part of the project, 3 weeks were planned. 2 for development and 1 for user testing.

Only 1 week was used to implement the two checks, along with the authentication interface. The version was presented to Norkart on February 15. Because of that, we were able to get back on our schedule and even get ahead of schedule.

Version 2

This version included the implementation of real-time collaboration features. This was the version with the most features. This version built on version 1. The features of version 1 remained in version 2.

This version included the following features:

1. When a user completes a job, every other user in the same job are interrupted by a warning telling them that the job has just been completed.
2. When a user enters a job, every other user immediately receives a snack bar, non-interrupting notification telling them that a new user entered the job.
3. The first user to enter a job has editing rights. All the changes that he makes to a field are sent in real-time to other users in the job. Every other users in the job do not have editing rights.
4. A users card was created. The card displays the username of users that are present in a job. The user with editing rights is on top. This list is updated in real-time as users enter and leave a job.
5. If a user enters a job and there are already other users in the job, the user entering a job receives a flow-interrupting notification warning that other users are already in the job.
6. When the user with editing rights had focus on an element in the job page, a colored frame around this element appears in the job page of other users inside the page.

This version was completed on March 6. It was presented to Norkart on March 7. The version also included feedback on our user interface provided by our contact person at Norkart on March 1.

At this point in the project, we were ahead of schedule by about two weeks. It was the case partly because no formal user tests had been arranged.

Version 3

The features to implement in this version were more open-ended.

We had been advised by our supervisor that we should stop developing and focus on writing the thesis report at the start of April.

So we had 3 weeks to get as much done as possible.

This is the list of features that we set out, in order of priority:

1. The ability for a user without editing right to ask for it. This feature also included version conflict solving for a text field of the job page because multiple users would be allowed to edit the text field at the same time.

2. Caching of changes made to jobs. That way, a new user entering the job would have in front of him the same version as the users currently inside the job instead of an empty job.
3. The ability for a user to create a new job.
4. Implement tasks inside a job. The idea of this feature was to change the structure of a job page to contain multiple subgroups of elements, called tasks. Those tasks would be completed one by one and when all tasks would be completed, the job would be considered completed. Also included in this feature was a request from Norkart to look into insuring the type safety of the job object in the frontend. That is making sure a job object only had fields that belonged to its type.

This list was shared with Norkart on March 14.

Caching of the job data was implemented on March 12.

The ability to create a new job was implemented on March 16.

Version conflict solving of the text field of the job page was implemented on March 17. A Conflict-free replicated data type was implemented in the frontend.

The ability to request editing right was implemented on March 18.

This gave us time to work on implementing tasks in the job page. However, we felt we did not have enough time to fully implement the feature. So instead, we implemented dynamic rendering of the fields of the job page.

Another thorough review was conducted by our contact person at Norkart with a colleague and shared with us on March 21. All of the shared feedback regarding the user interface was implemented by March 27.

Type safety of the job data object was implemented on March 29.

All those features were built on top of the two previous versions.

We finished developing the product two weeks in advance of what was scheduled. What mainly gave us extra time was that no formal user tests were conducted.

After that, we commented our code and implemented automated quality assurance tests until April 19. This put us five days beyond the final development date that we planned.

Overall, following the 80/20 principle, we did not overload ourselves, we did not over-promise and we managed to stay on schedule.

6.7 Meeting notes

Throughout the whole project. Meetings notes were taken for every team meeting, meeting with our supervisor and meeting with our contact person at Norkart. The notes we took in our meetings with our contact person at Norkart were especially thorough, given how important those meetings were for the project.

Those meeting notes, along with the issues and the commit history of each repository were crucial for the redaction of the timeline above.

Those meeting notes proved useful as well in the redaction of this thesis. Our contact person at Norkart advised us to take good notes along the way to make it easier to write this report. We heeded this advice and it did prove helpful.

Chapter 7

Testing

This chapter describes our work related to testing in the course of the project.

At the outset, we did not use test-driven development. However, we did implement testing processes right at the start of the project, and implemented automated tests at various points throughout the project.

One of the things we learned about testing through the course of the project is that testing is a complete under the umbrella of software development [92][100][88].

7.1 Requirements

The main requirements that we had regarding testing was that we had to integrate testing in the thesis one way or another. This was a requirement from the faculty.

Norkart also encouraged us to conduct performance testing on the different data transmission protocol and methods explored in the theory section to support the choice that we made.

Review requirement

We agreed that each merge requests would be both reviewed and tested by another team member.

This is a requirement that we all followed strictly and consistently throughout the whole project.

Automated tests requirements

We set an open-eded objective to implement unit tests, integrated tests and end-to-end tests. As will be detailed below, we did not manage to implement end-to-end testing, but we did manage to implement multiple unit tests and some integration tests.

Unfortunately, we confused code coverage for a testing strategy when it is in fact a metric. As a result of that, we did not get all of the value that we could have got out of the time that we invested in automated quality assurance testing.

User testing

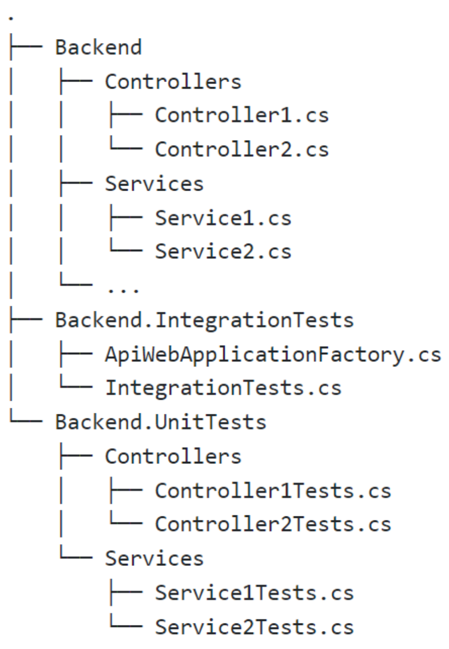
We planned to conduct user testing, in partnership with Norkart if our prototype had a user interface that had all of the core features of the user interface of Komtek's job handler. Otherwise the investment would not be worth it because it would not be comparable enough.

We did not reach that point in the development of our prototype. But it did not prevent Vebjørn, our contact person at Norkart to give us frequent feedback in the course of the whole project. He also conducted two thorough review of our user interface in the course of the project, which will be detailed below.

7.2 Design

Backend

We implemented the following testing architecture in the backend project:



```
.
├── Backend
│   ├── Controllers
│   │   ├── Controller1.cs
│   │   └── Controller2.cs
│   ├── Services
│   │   ├── Service1.cs
│   │   └── Service2.cs
│   └── ...
├── Backend.IntegrationTests
│   ├── ApiWebApplicationFactory.cs
│   └── IntegrationTests.cs
└── Backend.UnitTests
    ├── Controllers
    │   ├── Controller1Tests.cs
    │   └── Controller2Tests.cs
    └── Services
        ├── Service1Tests.cs
        └── Service2Tests.cs
```

Figure 7.1: 2nd backend test architecture design

In this architecture, each type of tests are in their own project. For the unit tests, the project mimics the same architecture as the main project. For integration tests, due to technical limitations that is detailed in the implementation part, every tests had to be kept together.

In this architecture, the design of the tests themselves followed an object-oriented approach. A class was created for each group of tests, for example for each controller. Attributes were created for data that was used by multiple tests. Each method of the class was one specific test.

Each test followed the 3 following steps:

- Arrange
- Act
- Assert

In the arrange phase, all the necessary variables and mocks are created.

In the act phase, what is to be tested is run. Namely the relevant method of the tested class.

In the assert phase, various assertions are run on the output of the test. These assertions are the tests themselves. One test method can contain multiple assertions.

For the integration tests, the tests had the same architecture, although it was only one test class. So the class contained tests for multiple controllers and services.

This project architecture is recommended by the Microsoft community for testing object-oriented web API in Dotnet [101].

Frontend

For the frontend, the various components were tested. Each component had their own separate file, and each component had a separate test file that contained one or many tests for that component. The test file had the same name as the file of component itself, with '.test' added at the end.

In one instance, a file containing constant variables was mocked to bypass the call to an environment variable. This was a technical requirement, but a '__mock__' folder was created next to the relevant file, and a mocked file with the same name was inserted in the '__mock__' folder.

It looked as follows:

```
Frontend
├── Component1.tsx
├── Component1.test.tsx
├── Component2.tsx
├── Component2.test.tsx
└── Api
    ├── consts.ts
    └── __mocks__
        └── consts.ts
```

Figure 7.2: Frontend test architecture design

This architecture matches what has been taught in the Cloud Technologies course. This architecture was also recommended by this ressource: [19]

The tests followed the same 'arrange, act & assert' structure as for the backend.

7.3 Use in CI/CD pipeline

Build test

A CI/CD pipeline was setup at the start of the project for both the backend and the frontend. In the backend, it was running on the development branches before being merged. In the frontend, it was running on every branch on both versions. The project was built before being deployed. Although this was most likely not needed. Everytime the pipeline was triggered, a build test was performed.

Implementing the build test in the pipeline was relatively easy, and once implemented, it operated for the whole duration of the project at every push to any branch, both in the backend and the frontend.

Once the use of the Firebase Authentication service was implemented, a challenge came up. For the application to be able to use this service, it requires a JSON file that acts as an authentication key. This file cannot be in the repository because it is a secret. However, it is too big to be inserted as a CI/CD variable through the GitLab user interface. The solution was to insert en encrypted file in the repository, insert the decryption phrase as a CI/CD variable through the GitLab user interface, and have the pipeline decrypt the file [103] before building the application.

Running automated tests

Unit tests were implemented early in the backend. They were run by the CI/CD pipeline in the same way as the build test.

Tests failed when we made changes to the code. This helped us to insure that the changes we made were intentional. Testing parts of a software that often change is a good practic [87].

For the frontend, once implemented, automated tests were run in the CI/CD pipeline in the same way as the build test.

7.4 Test processes

Even though we had a CI/CD pipeline that ran automated tests, we all did extensive manual testing before opening a merge request right from the start of the project. Another team member would do the same when reviewing a merge request.

This process was the main quality assurance system in place throughout the project. The use of automated tests never changed that process.

Manual testing became longer as new features were added.

Because we aimed to have a high amount of tests, we thought the best was to implement multiple unit tests. The implemented unit tests only insured that the controllers returned the appropriate response code and that the methods of the different services returned the proper type. It did not reduce in any way our need to do manual testing when new changes were made.

What would have been the right approach would have been to identify a use-case and test it thoroughly.

The most frequent use-case was to log in, click on a job in the job page, fill the form and submit it. When the real-time features were implemented, each feature was also tested with multiple users.

What would have been better would have been to have a some unit tests for that process, some integrated tests and one end-to-end test. This would have reduced our need for manual testing.

What added to this mistake is that unit tests on the Websocket controller and in the frontend components were especially challenging. They took a lot of time, for little value.

At the end of the project, because we favored volume and coverage over use-cases, we had a lot of unit tests for a lot of the backend and for every component in the frontend, some integrated tests in the backend and no end-to-end tests.

As a general approach, the priority was to produce the collaboration features. Automated tests were worked on when we were either on or ahead of schedule. Because of that, most of the tests were implemented at the end of the project. This also reduced the value that we got out of the tests.

However, it definitely had a significant learning value because it made us venture deeper into what is a complete field of software development.

We did not really assess different tools and technologies in the field. We spent most of our time learning how it was done. We followed these two resources that were comprehensive and beginner friendly: [87][18]. We used the design and tools they recommended.

7.5 Implementation

Use of ChatGPT

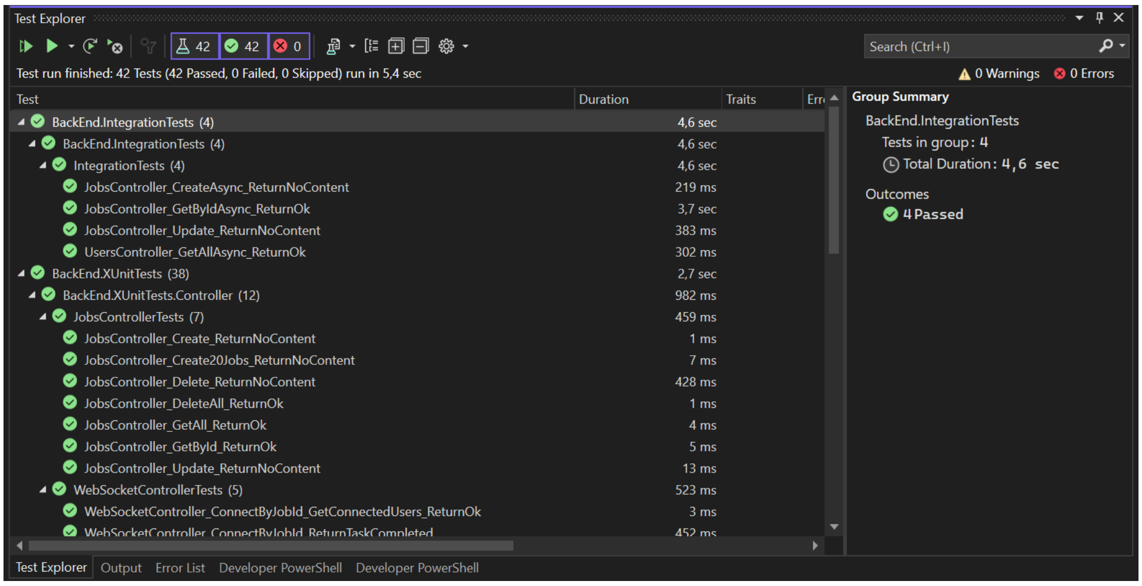
Because testing was new and challenging, we used ChatGPT tools to generate code. A significant amount of code for testing is code that has been obtained by ChatGPT, either modified or not.

For the Websocket controller, it was used both for syntax and logic. What was different for the Websocket controller was that, for effectiveness and limiting the length of the code, one of the tests tested multiple messages received by the controller. The test acted as a loop where each iteration meant one different message received. This loop logic was implemented with the help of ChatGPT, but it is not ChatGPT that gave the idea.

We made sure to understand all of the code produced by ChatGPT.

Tools used for testing in the backend

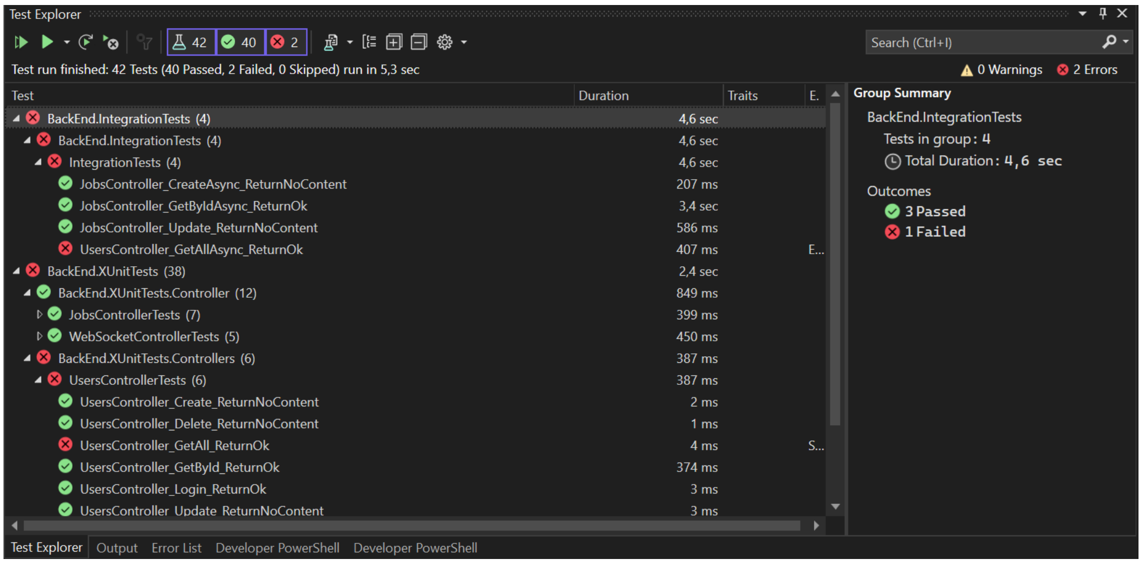
As recommended by [87], we used the test explorer provided by Visual Studio. Its interface is of very good quality:



```
Test Explorer
42  42  0                                                        Search (Ctrl+I)
Test run finished: 42 Tests (42 Passed, 0 Failed, 0 Skipped) run in 5,4 sec    0 Warnings   0 Errors
Test                                                          Duration   Traits   Err   Group Summary
BackEnd.IntegrationTests (4)                                  4,6 sec                   BackEnd.IntegrationTests
  BackEnd.IntegrationTests (4)                                4,6 sec                     Tests in group: 4
    IntegrationTests (4)                                      4,6 sec                     Total Duration: 4,6 sec
      JobsController_CreateAsync_ReturnNoContent              219 ms
      JobsController_GetByIdAsync_ReturnOk                    3,7 sec                   Outcomes
      JobsController_Update_ReturnNoContent                   383 ms                      4 Passed
      UsersController_GetAllAsync_ReturnOk                    302 ms
BackEnd.XUnitTests (38)                                       2,7 sec
  BackEnd.XUnitTests.Controller (12)                          982 ms
    JobsControllerTests (7)                                   459 ms
      JobsController_Create_ReturnNoContent                     1 ms
      JobsController_Create20Jobs_ReturnNoContent               7 ms
      JobsController_Delete_ReturnNoContent                   428 ms
      JobsController_DeleteAll_ReturnOk                         1 ms
      JobsController_GetAll_ReturnOk                            4 ms
      JobsController_GetById_ReturnOk                           5 ms
      JobsController_Update_ReturnNoContent                    13 ms
    WebSocketControllerTests (5)                              523 ms
      WebSocketController_ConnectByJobId_GetConnectedUsers_ReturnOk   3 ms
      WebSocketController_ConnectByJobId_ReturnTaskCompleted  452 ms
Test Explorer   Output   Error List   Developer PowerShell   Developer PowerShell
```

Figure 7.3: Test explorer of Visual Studio

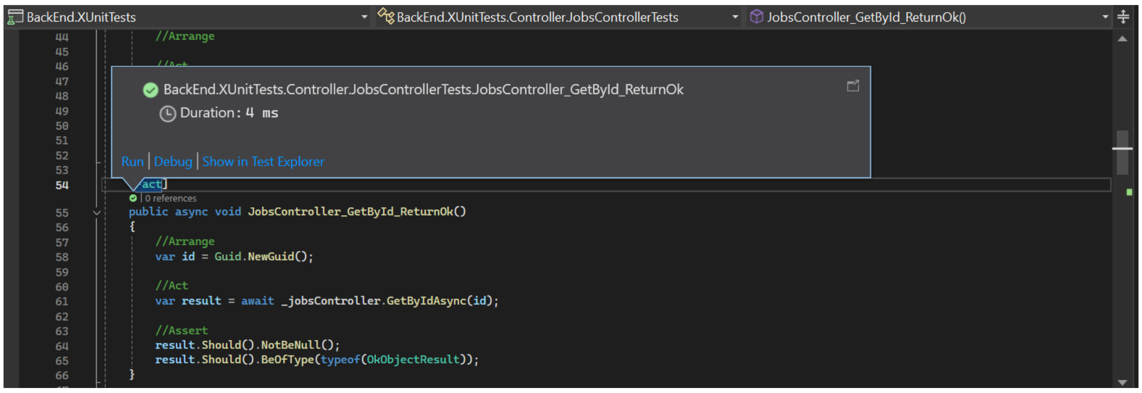
The explorer can identify every test and organize them by the folders in which they are located in a project. Failed tests are also very easy to locate:



```
Test Explorer
42  40  2                                                        Search (Ctrl+I)
Test run finished: 42 Tests (40 Passed, 2 Failed, 0 Skipped) run in 5,3 sec    0 Warnings   2 Errors
Test                                                          Duration   Traits   E.    Group Summary
BackEnd.IntegrationTests (4)                                  4,6 sec                   BackEnd.IntegrationTests
  BackEnd.IntegrationTests (4)                                4,6 sec                     Tests in group: 4
    IntegrationTests (4)                                      4,6 sec                     Total Duration: 4,6 sec
      JobsController_CreateAsync_ReturnNoContent              207 ms
      JobsController_GetByIdAsync_ReturnOk                    3,4 sec                   Outcomes
      JobsController_Update_ReturnNoContent                   586 ms                      3 Passed
      UsersController_GetAllAsync_ReturnOk                    407 ms             E...     1 Failed
BackEnd.XUnitTests (38)                                       2,4 sec
  BackEnd.XUnitTests.Controller (12)                          849 ms
    JobsControllerTests (7)                                   399 ms
    WebSocketControllerTests (5)                              450 ms
  BackEnd.XUnitTests.Controllers (6)                          387 ms
    UsersControllerTests (6)                                  387 ms
      UsersController_Create_ReturnNoContent                    2 ms
      UsersController_Delete_ReturnNoContent                    1 ms
      UsersController_GetAll_ReturnOk                           4 ms             S...
      UsersController_GetById_ReturnOk                        374 ms
      UsersController_Login_ReturnOk                            3 ms
      UsersController_Update_ReturnNoContent                    3 ms
Test Explorer   Output   Error List   Developer PowerShell   Developer PowerShell
```

Figure 7.4: Test explorer of Visual Studio with failed tests

Another interesting feature that the tests can be run individually:


BackEnd.XUnitTests
BackEnd.XUnitTests.Controller.JobsControllerTests
JobsController_GetById_ReturnOk()
Run | Debug | Show in Test Explorer
0 references

```
//Arrange

//Act
```

BackEnd.XUnitTests.Controller.JobsControllerTests.JobsController_GetById_ReturnOk
Duration: 4 ms

```
[Fact]
public async void JobsController_GetById_ReturnOk()
{
    //Arrange
    var id = Guid.NewGuid();

    //Act
    var result = await _jobsController.GetByIdAsync(id);

    //Assert
    result.Should().NotBeNull();
    result.Should().BeOfType(typeof(OkObjectResult));
}
```

Figure 7.5: Inline test interface of Visual Studio

The Xunit package

All of the packages that we used for testing in the backend were the ones recommended by [87]. We did not look further, as we were mostly learning something completely new.

The main package for unit testing that was used is the Xunit package [110]. With this package, the user only needs to insert the attribute '[Fact]' above a test method of the test class for it to be detected by the test explorer and the 'dotnet test' command.

When it comes to the classes containing the unit tests, variables that are used in multiple tests are inserted in the class as attributes and initialized in the constructor. Others are created in each tests.

The FakeItEasy package

This package is used to mock classes for unit testing[34].

This package allows to mock only the required methods of an interface instead of create a full mock class. The return values of the relevant methods are set in advance and then inserted in the class to be tested.

A lot of the interfaces that had to be mocked have asynchronous methods. Those methods returned an object called 'Task' or a templated 'Task' such as 'Task<Job>'. ChatGPT was used to find out which syntax had to be used to return either one. The return value is respectively 'Task.CompletedTask' and 'Task.FromResult(job)'.

As mentioned earlier, the unit testing of the Websocket controller was especially challenging because one of the tests acted as a loop where different outcomes were tested. Mocking was especially challenging in this case.

For the looping Websocket test we used ChatGPT to find out which object to mock and how to insert it in the controller.

First, a WebSocket object had to be mocked. Then an object called a WebSocketManager had to be mocked as well. One method and one attribute from this object had to be mocked: the 'AcceptWebSocketAsync()', which had to return the mocked WebSocket object and the 'IsWebSocketRequest' attribute, which had to return true.

An HttpContext object [102] then had to be mocked.

The 'WebSockets' attribute of the HttpContext class had to be mocked to return the mocked WebSocketManager.

Lastly, the http context had to be inserted in an instance of the controller that had to be tested. This instance could not be a mock because it was the tested class.

ChatGPT provided an example that used an object initializer[71] and assigning the 'ControllerContext' attribute.

This setup was common with other unit tests of the Web socket controller. All those 3 objects had to be mocked. However, some unit tests had different configurations.

When it came to the testing part, the Websocket endpoint contains a loop that listens for incoming messages as long as the connection is opened. So we had to close the connection to end the test.

For the solution to that, ChatGPT pointed to a method from the FakeItEasy package called 'ReturnsNext-FromSequence()' that returned the provided values in sequence such as 'ReturnsNextFromSequence(Open,Closed)'.

This also had to be done for the different mock messages received by the controller. For that, the 'ReceiveAsync' method had to be mocked in sequence.

For that, ChatGPT proposed using the 'ReturnsLazily()' method of the FakeItEasy package. We created a list of message to received. An enumerator for the list was fetched. The enumerator begins before the first element in the list. Everytime the 'ReturnsLazily()' function was invoked, the enumerator would take a step, and therefore provide the next message in the list.

The last challenge faced when mocking the Websocket controller came from testing a closing Websocket connection. The 'WebSocketReceiveResult' object returned by 'ReceiveAsync' method needs to return a 'WebSocketCloseStatus'. We looked in the source code and found one constructor in which such parameter could be directly inserted.

Fluent assertions

Fluent assertions is a package that provides extension methods that allow to run assertions in one line of code [42]. A multitude of different assertions are offered.

We did not research for other potential assertion frameworks, we instead focused on learning how to use this framework.

Microsoft.AspNetCore.TestHost & Microsoft.AspNetCore.Mvc.Testing package

The Microsoft.AspNetCore.TestHost package is used to create an in-memory instance of the application to be tested as well as a test server that will be used to send requests to the tested application [53].

Those are for integration tests.

With those packages, integration tests can be performed with the 'dotnet test' command without needing to run the application before running the tests.

The other main advantages of those packages is that the test server can be customized. For example, it can redirect to another database [53].

The basic class for creating an instance of the main application and of the test server is the 'WebApplicationFactory'. This class is templated with the class that contains the application that is tested. In our case, it is the 'Progam' class.

Our integration tests are mainly based on those two resources [53][25]. The second resource recommended using a dedicated database for integration tests in order to provide more predictability to the tests. This requires customization of the main application for integration testing.

The way to achieve this is to create a new class that inherits the 'WebApplicationFactory<Program>' class, and have the class override the 'ConfigureWebHost' method.

In our case we created a new collection in our database to which we pointed to for the test instance.

To be able to change the configuration of a class only for test purposes, the 'builder.ConfigureTestServices()' method needs to be used. This method comes from the 'Microsoft.AspNetCore.TestHost' package.

The test class for the integration tests is implemented in the same way as for unit tests except that the common attributes differ. There is an application variable, which is the in-memory application server and the test server. Out of this application variable, an HTTP client is created. It is with this client that the various endpoints of the application are called. The other step to take in our case is to fetch a bearer token to access the secured endpoints. At the same time, it is an opportunity to test the token fetching endpoint.

In implementing the above, ChatGPT was used to help find the proper syntax.

Some of the tests called more then only one endpoints. For example, to test the endpoint fetching a single job, a job id was needed. For that, the endpoint to fetch all the jobs in the database was used, and then the id of the first job in the list was used to test the endpoint fetching only one job by its id.

For all of the tests, some job data needed to be in the database. To insure that, a helper function calling the endpoint creating 20 jobs was used. At the end of all the tests, a destructor method was implemented with the 'IDisposable' interface. This method called the endpoint deleting all the jobs in the database.

Integration testing was easier to implement because of the absence of any mocking.

Tools used for testing in the frontend

Jest

Jest is a package that comes with every React project. It is the package that locates tests and runs them when running the 'npm run test' command.

The tests run quite slowly compared to the tests being run in the backend.

We used the Vite as the build tool of the frontend. One testing framework that can be used with Vite is the Vitest framework [106]. Vitest is reportedly more performant than Jest [35]. This is the tool that we should have used instead of Jest. But we instead focused on learning how to implement automated tests and therefore followed the recommendations of the resource that we found on the topic [18].

Because we used Vite, the setup presented in our main resource had to be adapted to the use of Vite. For that, we used this resource: [6].

The first major challenge was the presence of style files such as css files, svg files and other media files in the codebase. This was causing errors when running the tests. The solution to this was to use what is called a module mapper in the Jest configuration file. What it does is that, when Jest encounters a file that is mapped, it will ignore the file and instead use the module to which it is mapped [6] [4].

The second issue was that, when running tests, a typescript error was returned. The issue came from the fact that the Typescript configuration file needed to include a typescript file that imported the 'jest-dom' module [6].

React Testing Library

This is the library used to implement the tests themselves.

We also took help from ChatGPT for mocking.

Hooks and other function that were used inside a component had to be mocked at the top of the file with the 'jest.mock()' with the path to the hook as parameter and a function that would return an object containing the mocked values.

To mock a function that had to be inserted in a component as a prop, we learned, through ChatGPT, to use the 'jest.fn()' function. If the mock function had to return a value, we learned to use, also through ChatGPT, 'jest.fn().mockReturnValue()'.

The last major challenge we faced with mocking was that, in our file storing constants variables, one of them was fetching an environment variable. This was causing errors with Jest. We took help from ChatGPT and this resource: [84]. The solution was to mock the whole file, and remove the use of the environment variable in the mocked file. The way to implement is solution is to insert a folder nameed '__mocks__', at the same location in the project as the file that needs to be mocked. Inside the '__mocks__' folder, the mock file must have the same name as the mocked file.

To be able to redirect to this mock file in a test when needed, the 'jest.mock(<path to the file>);' function must be inserted at the top of the test file.

A lot of the unit tests are to ensure that component simply renders. The way to achieve that is to render the component and assert the presence of an element in the rendered code. There are multiple ways to query those elements:

1. getByRole - the roles attributed to HTML elements to help screen readers

2. getByLabelText
3. getByPlaceholderText
4. getByText
5. getByDisplayValue
6. getByAltText
7. getByTitle
8. getByTestId [20]

The priority of queries follow the same order. Queries methods are prioritized this way because the goal is to test the component from a user perspective.

It can be argued that we implemented integration tests in the frontend because we unit tested page components. However, even those tests involved mocking.

7.6 User Testing

As described in the development process chapter, two thorough review were performed by our contact person at Norkart. The first on March 1 and the second on March 21. The feedback mainly concerned the user interface, which weere all implemented.

7.7 Refactoring

One major refactoring of the Websocket controller was attempted to make it more readable and less monolithic. It failed. The changes introduced errors. And most of the errors were not errors that Intellisense could detect.

This experience underscored how important a test suite is to ensure that changes to not introduce unintended changes.

Chapter 8

Deployment

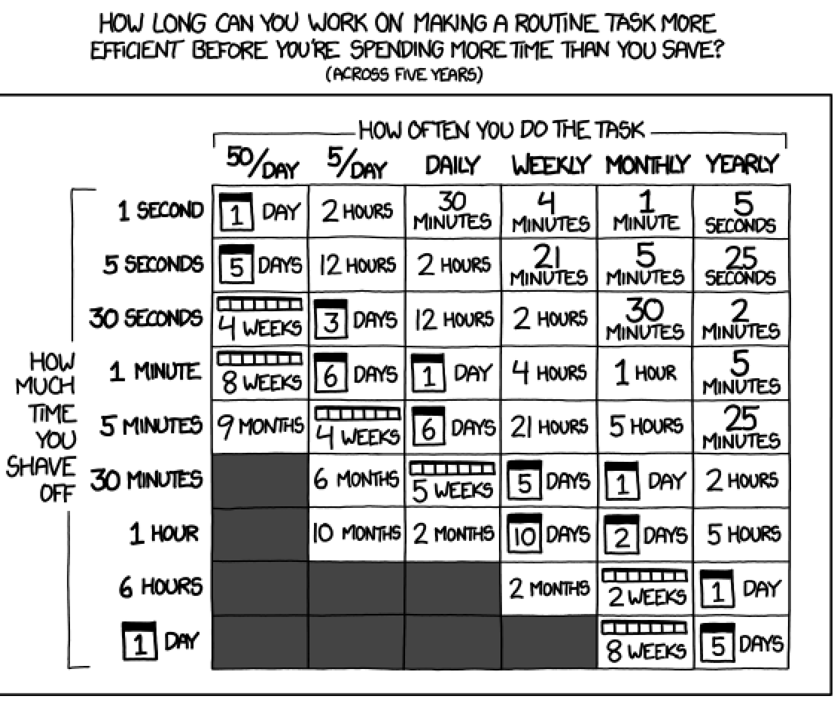
This chapter describes how our application was deployed, both for the backend and the frontend.

Deployment has been very successful because it has been automated, right at the start of the project. Right at the beginning of the project, we completed the last step of a project.

Through automation, the deployment has been continuous. As a result, the first deployed version was very rudimentary. It was basically the shell of a project. However, as the project evolved, so did the deployed version.

It was also a good opportunity right at the start of the project, because the development of the application itself was not yet at full pace. We were still in the period of setting up the project. It would have been much more difficult and less justifiable to set up automatic deployment later in the project when the priority was the development of collaboration features.

At the outset, not every process is worth automating. If a process takes more time to automate then the time saved through the automation, it is not worth it. The table below shows the time a repetitive tasks takes over a period of 5 years:



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

HOW MUCH TIME YOU SHAVE OFF	HOW OFTEN YOU DO THE TASK: 50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES	■	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR	■	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS	■	■	■	2 MONTHS	2 WEEKS	1 DAY
1 DAY	■	■	■	■	8 WEEKS	5 DAYS

Figure 8.1: Accumulated time of a repetitive task over 5 years

Source: [48]

Once the automatic deployment had been implement, everytime the main branch in both the backend and frontend repository would be modified, there would be a deployment of the main branch. There has only been 1 manual deployment in the course of the project. It was the deployment of the frontend to Firebase.

In our case, throughout the project 60 merges were made in the backend and 61 merges were made in the frontend. The amount of deployment is a little bit less because there has been some merges before done before the automated deployment was implemented. In the backend and the frontend, 5 merges were not automatically deployed.

There has therefore been around 111 automatic deployments. Manual deployments would have involved, for the backend, pushing the project to a GitHub repository. This would have been done by running a couple of commands in the terminal. In the frontend, it would have been done by pushing the project to Firebase. This would have been done by running one command in the terminal. 1 minute per deployment, both in the backend and frontend, would be a conservative estimate. This would mean 1 hour and 51 minutes. However, it is important to not that, if we did not have automatic deployment, we would most likely have deployed less often.

Looking at the history data of our Gitlab instances for both the backend and the frontend, it took a total of approximately 6 hours to set up the automatic deploy of the backend and frontend.

It was therefore not worth it for us to automate the process.

However, it is worth noting that this was our second time implementing an automatic deployment process, having done it before only in the Cloud Technologies course. We had to do research and we had to do debugging as well.

Another aspect to consider is that, an automatic deploy was triggered at every change of the main branch of each repository. This is not desirable, because if a small change is done, or perhaps even just comments are added, it is not worth to trigger a deploy.

Releases should have been used instead [60]. However, given the fact that we only developed a prototype over only 4 months, the extra tasks that come with managing releases would not have been worth it. It would have been too much for the size of our project.

Creating a release is also usually a manual process, although the deployment of a release can be automated. So implementing a release system would have essentially brought us back to manual deployment.

And this is another area where we got value out of an automated process. We had the assurance that the latest version of our main branches was always deployed. Once the automated deployment was implemented, we did not have to think about it again throughout the whole project.

Overall, even though automating the deployment process did not give a lot of value for this project, it was a good learning experience and it was a relief to have the assurance that our changes would be deployed reliably.

8.1 Design

The backend project is deployed to Render [1] through a GitHub repository and Docker.

The deployment takes place as follows:

1. The main branch of the GitLab instance is mirrored to the main branch of a GitHub repository
2. The main branch of the GitHub repository is pushed to Render
3. Render dockerizes the application with the help of the Docker file in the project. The API is then deployed.

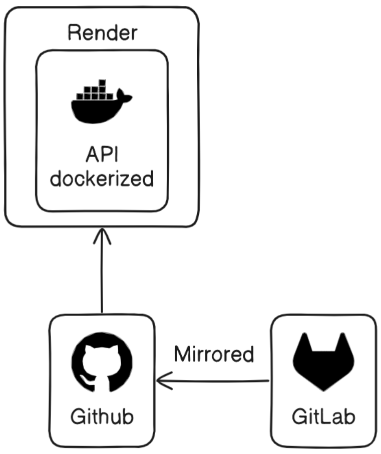


Figure 8.2: Backend deployment diagram

The frontend project is deployed to Firebase Hosting [40] through a CI/CD pipeline.

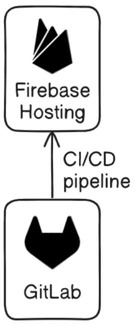


Figure 8.3: Frontend deployment diagram

8.2 Backend implementation

The reason why Render was chosen was because this platform was used in the Cloud Technologies course. We were already familiar with it.

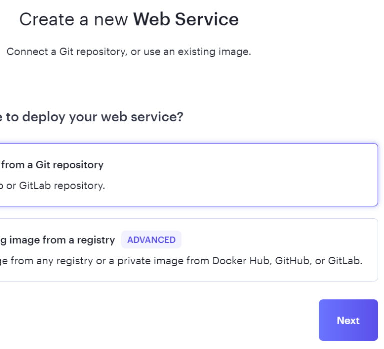
However to be able to deploy the API to Render, we had to use a GitHub account. So we decided to mirror the GitLab Backend repository to a GitHub repository. We learned this method in the Cloud Technologies course. We choosed this method thinking it was going to be easy and quick. It turned to be more difficult then expected.

We proceeded with a push mirror. This means that the repository performing the mirror operation is the mirrored repository. In our case, this is the GitLab repository. The GitLab repository pushes the main branch to the GitHub repository.

In the GitLab interface to mirror a repository, there is an option to 'keep divergent refs'. Understanding what this option meant was actually not straightforward. The default mechanism of a mirror operation is to delete in the mirroring repository all commits that are not part of the mirrored repository. When divergent refs are kept, if there is a commit in the mirroring repository that is not part of the mirrored repository, the diverging commit will not be deleted, the mirror operation will instead fail [79]. We did not chose this option.

The next step was to chose an authentication method to give the GitLab repository access to the GitHub repository. The simplest was to use the https address of the GitHub repository, the username and password. However, it turned out that GitHub no longer allows passwords to be used for this. A personal access token must be used [80]. We had never created such token before, so creating it was actually a challenge. As of the writing of this thesis, it is done by clicking on the profile picture, going to the settings menu, and then to 'Developper settings' which is the last option of the vertical navigation bar, then clicking 'Personnal access tokens' . There was two options of tokens: 'Fine-grained tokens' and 'Tokens (classic)'. We chosed 'Tokens (classic)'.

Once this was completed, the next step was to create a Render project. When setting up the project, using a GitHub or GitLab repository is the default option:



Create a new **Web Service**

Connect a Git repository, or use an existing image.

How would you like to deploy your web service?

[✓ Build and deploy from a Git repository
Connect a GitHub or GitLab repository.]

Deploy an existing image from a registry ADVANCED
Pull a public image from any registry or a private image from Docker Hub, GitHub, or GitLab.

Figure 8.4: Render new project menu

Whenever the selected branch of the repository changes, a redeployment is triggered.

However, Render does not natively support C#. So a Docker image has been used [93]. We chosed to have Render build the image by inserting a Dockerfile in the repository [28]. We used the Dockerfile provided by this resource [73], changed the Dotnet version to version 8 and adapted the file to our application.

The use of secrets was very easy with Render. Because both files and variables can be inserted as variables through Render's user interface. At this point, what was important was that the variable names be the same in both GitLab and Render.

A MongoDB database was also set up on MongoDB Atlas [62].

A Redis database was also deployed to Render to be used as a cache by the backend. It was setup entirely by following the instructions of the user interface of Render. The provided link was then used in the backend to access the database.

8.3 Frontend implementation

To deploy the frontend, it was decided to use Firebase Hosting because one team member had prior experience with it.

The main task was to push the project to Firebase through the CI/CD pipeline. The push had to be done only on a merge request to the main branch. For other cases, only the build test was running.

This is the first version of the '.gitlab-ci.yml' file:

```
image: node

build-job:
  script:
    - npm install
    - npm run build

prod-hosting:
  stage: deploy
  script:
    - npm i -g firebase-tools
    - npm install
    - npm run build
    - firebase deploy --only hosting --token $FIREBASE_TOKEN
  only:
    refs:
      - main
    changes:
      - src/**/*
```

Figure 8.5: First version of the CI/CD pipeline file of the frontend

The image to use for a React application is Node: [66].

The commands ‘npm install’ and ‘npm run build’ are provided by this resource: [56].

The command ‘firebase deploy only hosting token $FIREBASE_TOKEN’ can be crafted with the help of the official Firebase documentation: [39].

For the second action, which is the deployment action, the notation ‘only:refs:main’ is used. This triggers the action only when the event involves the main branch, in our case, a merge to the main branch [**gitlab_ci_onlyrefs**]. The notation ‘only:changes:src//’ triggers the action only when there is a change to the content inside the ‘src’ folder [15]. Unfortunately, this notation is deprecated[**gitlab_ci_onlyrefs**][15]. In this version, when a branch is merged to the main branch, both actions were running as well. But it was unnecessary to run the first action.

The proper notation is now to use ‘rules:if:’ and ‘rules:changes:’ [16].

As a result, this became the second version of the ‘.gitlab-ci.yml’ file:

```
image: node

build-test:
  script:
    - npm install
    - npm run build
  rules:
    - if: $CI_COMMIT_BRANCH != $CI_DEFAULT_BRANCH &&
    $CI_PIPELINE_SOURCE != "merge_request_event"

firebase-deploy:
  stage: deploy
  script:
    - npm i -g firebase-tools
    - npm install
    - npm run build
    - firebase deploy --only hosting --token $FIREBASE_TOKEN
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      changes:
        - src/**/*
```

Figure 8.6: Second version of the CI/CD pipeline file of the frontend

As can be seen, multiple CI/CD variables were also used [77].

Finding the proper if statements to insure that the first action is triggered at every push to any branch and not when merging with the main branch, and vice-versa for the second action, proved to be challenging. Because of that, ChatGPT was used. The mixture of ChatGPT and the official document worked because those rules do achieve the intended result mentioned above.

.gitlab-ci.yml file of the backend

This was the first version of the .gitlab-ci.yml file of the backend:

```yaml
build-job:
  image: mcr.microsoft.com/dotnet/sdk:8.0
  script:
    - dotnet build
```

Figure 8.7: First version of the CI/CD pipeline file of the backend

It was a test that was running at every event, at every push to any branch and also when a branch was merged with the main branch, which was not necessary.

This was the second version:

```yaml
before_script:
   - cd BackEnd
   - chmod +x decrypt_firebase_credential.sh
   - ./decrypt_firebase_credential.sh

default:
  image: mcr.microsoft.com/dotnet/sdk:8.0

build-test:
  script:
    - dotnet build
    - cd ..
    - dotnet test
  rules:
    - if: $CI_COMMIT_BRANCH != $CI_DEFAULT_BRANCH &&
    $CI_PIPELINE_SOURCE != "merge_request_event"
```

Figure 8.8: Second version of the CI/CD pipeline file of the backend

This version includes a decryption of the Firbase credentials file and the use of if statements along with the use the CI/CD variables. The if statement in the file prevents the action from running when a branch is merged to the main branch. This file only builds the application and runs automated tests.

8.4 Choice of tools

For the deployment, we did not thoroughly assess various options. This aspect was not an area of focus in our thesis. What mattered was to have a deployed version that Norkart could easily access. We therefore went with the tools with which we had prior experience and with which we were comfortable.

The end result was nonetheless an effective and reliable automatic deployment process.

Chapter 9

Conclusion

This chapter reviews if our objectives were achieved, summarizes the reasons behind our choices of technologies and reviews our project critically under various criteria such as the use of artificial intelligence and sustainable development.

9.1 Results

This assesses if the objectives outlined in the project plan were achieved or not.

Deliverables to Norkart

What was to be delivered to Norkart was a prototype of the job handler of the Komtek system in which there would be various collaboration features to both help case handlers avoid working on the same job and to help them working on the same job. The prototype would also include a feature of collaborative editing and a version conflict solving system. We achieved that.

If possible, our prototype would be a generic implementation what could easily be usable by systems other then the job handler of the Komtek system.

When it comes to the collaboration feature, those objectives were achieved and we provided Norkart with multiple possible features that they could implement themselves, including collaborative editing and version conflict solving.

Our prototype did not replicate exactly every feature of the job handler of the Komtek system. It was a simplified version. It could therefore be argued that we did not fully achieve our objective there.

The project would have also been more generic if it would have used the HTTP 2 protocol with a server-sent events architecture. One could therefore also argue there that the generic implementation objective was not fully achieved.

Benefits for Norkart

The value for Norkart in this thesis is to have a demonstration of how various collaboration feature can be implemented in the context of a Rest API backend and a single-page application in the frontend, including features of collaborative editing and version conflict solving.

This objective was achieved with the Websocket tool. The demonstration of those features has been made with Websockets only. One may argue that having presented solutions with only one technology is a shortcoming, and that it would have been more valuable to demonstrate other technologies such as long polling and server-sent events with the use of the HTTP 2 protocol.

The fact that our prototype was a simplified version of the job handler of the Komtek system makes it such that the features that we developed cannot directly be inserted in the Komtek system. Some changes would have to be made and the final solution would be more complex then what we developed. Because of that, one could argue that the demonstration was only partially useful to Norkart.

Learning objectives

Our learning objectives have been achieved for the most part. When it comes to the core technologies: Dotnet and React, we definitely know more now then we did before this thesis. In React, we know more about conditional rendering depending on whether or not a user is authenticated, the use of bearer tokens for authentication with Firebase, the use of Websockets, dynamic rendering of components and so on. For Dotnet, we learned about some of its proprietary features such as top-level statements and attributes above classes, some related packages used for testing, mocking and asserting, asynchronous programming and so on.

This acquired knowledge was used to build various real-time features using Websockets on top of an already existing Rest API in the backend and a single-page application in the frontend. It was also used to develop collaboration features that both help to avoid multiple case handlers working on the same job and other features to help multiple case handlers working on the same job.

We learned about testing by having both a manual testing process that we strictly followed and by implementing automated quality assurance tests both in the backend and the frontend. Those automated tests were continuously used inside a CI/CD pipeline that we implemented in both the frontend and backend project.

Those features and those automated tests were implemented in a prototype that imitated the job handling part of the Komtek software of Norkart. This prototype was continuously deployed online with the help of both GitLab mirroring and GitLab actions.

In our implementation of the real-time features, we tried to have an approach that was as generic as possible. Throughout our project, we showed that collaboration features based on Websockets can be integrated in an existing Rest API. Using the HTTP 2 protocol with a server-sent events architecture would have been even more generic because it would have been even easier to implement on existing Rest APIs and other servers using the HTTP protocol.

Throughout the whole project, we managed to truly work as a team: we communicated often, team members had an overview of the whole project, team members reviewed the work of other team members and we dared to address questions that were uncomfortable. This was a learning experience as valuable as the technical achievements, if not more.

The whole process has also been documented thoroughly through issue tracking, version control and notes for every meeting, among other things.

The objective on which we came short was to develop the project through a sustainable development perspective. Unfortunately, when working on the nuts and bolts of developing a project, the notion of sustainable development is a concept that appears to be quite far away from daily tasks. In our case, we worked on a very narrow and specific process. So integrating notions of sustainable development into our project appeared to be something quite distant.

9.2 Reasons for our choices of technologies

As outlined in the implementation chapter, for both Dotnet and React, the main reason why we picked those technologies and their related packages is that it is used by Norkart. Norkart gave us a free choice but we nonetheless decided to go with the technology they use to maximize the value of our thesis for them.

For the choice of real-time technology, we did a thorough assessment, as shown by the first section of the design chapter, because this was the core of the thesis. In the end, we went with what we assessed to be both the standard and the best practice in the industry: Websockets. For learning purposes, we decided to use raw Websockets instead of the SignalR package which is the standard for Dotnet. We unfortunately overlooked the HTTP 2 protocol. This shows how it is not as known as Websockets. We found out about this technology only at the end of the project.

For version conflict solving, we found the Yjs package based on conflict-free replicated data types. Given that this method is the best so far and that the package was fairly easy to use, we focused our efforts on learning how to use the package instead of looking at other alternatives.

For other design and implementation decision throughout the project, we went with what was the standard in the industry to avoid as much as possible 'self-imposed challenges' and to benefit from a rich community support. Also, more generally, a lot of what we worked on was new so a lot of our efforts went to learning these new fields instead of researching the best tools.

9.3 Use of artificial intelligence

Development tool

When we did not know how to implement something or when it only came down to finding the appropriate syntax, we would first try to research it through traditional sources online. When this was not enough to get our answer or we were not able to transfer the general solution to our specific use-case, we used ChatGPT.

For example, in C#, we did not know at all how to convert an object datatype to a higher datatype such as an integer or a string. The object datatype in Dotnet is the root of the type hierarchy. We created a dictionary to store the Json objects that the backend received via Websocket. But because sometimes, integers, strings or boolean values would be received, the type of the dictionary was set to string for the keys and object for the values so that any possible type could be stored in the dictionary. But further in the code, those values needed to be converted in their specific types to be then sent to a cache database. We were struggling to find resources online. ChatGPT provided a functioning solution. In this case, we took code produced by ChatGPT. The solution provided by ChatGPT was to convert the data from object to Json element, and then use the built-in methods of this type to further convert the value in a string or in an integer.

In the case of Websocket testing in the backend and testing in the frontend, we understood what needed to be done after watching tutorials on the matter and we had a good overview of the mechanic of the frameworks that we were using. But we still struggled with the implementation for our specific use-cases. We also struggled to find resources related to our specific use-case. We then used ChatGPT and used the code and syntax it provided. Most of the code for the automated tests of the Websocket controller is syntax that was obtained from ChatGPT. In the frontend, some components contained multiple variables to mock and some of them were not covered in the tutorial that we followed, and because it was specific to our use-case, finding further information online was difficult, so ChatGPT was used. So also in the frontend, a lot of the code for for testing is from ChatGPT, especially for mocking.

We could not assess with great confidence if a functioning solution was also a good quality solution. However, we knew enough to reject solutions that appeared to be overly complicated.

It was overall very useful when traditional sources could not solve our problem.

It did not manage to solve every problem. For example, there was only little information online about the Yjs package and so ChatGPT was not providing good answers on it. Thankfully, after a good amount of research, we found the answer in a community forum.

Research tool

We also used it to ask purely theoretical questions. However, we always tried to verify the given information online. For example, in one case, we weren't sure if the code to fetch an environment variable in Javascript would work in the same way to fetch an environment variable on a computer or in a GitLab CI/CD pipeline, or if we needed to take any other additional step to insert the GitLab variable in the CI/CD pipeline. We asked this specific question to ChatGPT. The provided answer was that it worked in exactly the same way and that no additional step was needed to make it work in the CI/CD pipeline. We found out it was accurate when we ran the pipeline for the first time after implementing an environment variable.

Review tool

We used ChatGPT to review our project plan and this report. The reviews gave general advice on the grammar and on the structure. We implemented the advice we found to be relevant. We did the same for this thesis report. For the structure, we received interesting advice such as supporting our explanations with flow charts for example. For grammar, we already knew most of the provided advice. For example, among other things, it advised us to ensure that we consistently use the same verb tense in a paragraph or section. This is something that we already knew. We no longer have the review of the plan, but the review for this report is in appendix.

9.4 Sustainable development assessment

Sustainable development is ‘development that meets the needs of the present without compromising the ability of future generations to meet their own needs’[94].

To assess if something is sustainable, it has to be examined against 3 factors:

1. It’s social impact
2. It’s economic impact
3. It’s environmental impact [94]

Social impact

Komtek’s job handler digitalizes public services. This is done to make them more effective. Making public services more effective is something that can only be beneficial socially because the government’s broad mission is to serve the common good. The currently aging population in Norway [32] makes it even more relevant because it creates worker shortages.

The idea of avoiding case handlers completing the same job multiple times falls into this notion of making the process more effective and ultimately, being able to do more with less.

Economic impact

There is the expression that ‘time is money’. When the Suez canal got blocked by a cargo ship in 2021, the estimated cost was 6.7 American dollars per minute [82]. This illustrates the fact that there is an economic loss if processes hold up economic transactions. The opposite to that is that there is therefore an economic benefit to having those processes go as quickly as possible.

In the case of Komtek’s job handler, if a real estate promoter needs to have electricity meters installed in his building before he can begin to sell apartments, a delay in having them installed could potentially delay his sales. So the faster this process goes, the better. a case handler that avoids completing an already completed job can work on another job instead. A gain of effectiveness is a gain of speed. And there is an economic benefit to having faster processes.

Environmental impact

Internet traffic, such as watching a video, sending an email and sharing pictures has an environmental impact. Among other things, it has a carbon footprint [45]. Komtek is a web application. It is used through the internet.

Reducing inefficiencies in the application therefore has a positive environmental impact. The process that we worked on most certainly does not have a high carbon footprint. However, scaled at the amount of Kometk users throughout Norway, the improvement that we worked on may actually have a meaningful impact on the environment.

9.5 Critical review

The aspect where the thesis falls short the most is that it used Websocket instead of the HTTP 2 protocol for the implementation of the real-time collaboration features. Using HTTP 2 would have adapted better to a Rest API, it would have been more modular and would have been less complex to implement in the frontend because it would have allowed to have multiple listeners on a single TCP connection. Because of that single TCP connection, the HTTP 2 protocol provides the same technical advantage as the Websocket protocol.

The second aspect where our thesis falls short is that our prototype did not reproduce all of the features of the Komtek's job handler. Our prototype did not have 'sub-jobs' in a job page. Because of that, the real-time collaboration features that we implemented cannot be inserted in the Komtek system without additional work and additional changes. Because of that, we also did not conduct formal user tests because it was not worth the investment of soliciting a customer support specialist at Norkart.

9.6 New projects and further work

A new project would be to reproduce exactly the same project with the use of the HTTP 2 protocol.

Further work would be to integrate in our prototype 'sub-jobs' in our job page. Such that the job page would contain distinct groups of elements. Each group, or 'sub-job', would then be completed and when every sub-job would be completed, the job itself would be considered completed.

Another step would be to have extensive user testing to identify which features works best and improve the identified features according to the user feedback, at which point the work to integrate those features in the job handler of Komtek could start.

9.7 Asessment of the group work

Introduction

Overall, our group work has been very effective. We put in place processes right at the start and, above all, we consistently followed them. Those processes made us effective. The fact that we did not over-promise and kept reasonable objectives also gave us the time to have proper group work.

Team member contributions

The way the tasks were distributed at the start of the thesis were for the most part followed throughout the project.

Arnaud took care of the administrative aspects of the thesis and worked on the backend. However, he also worked with Websockets, and this involved work in both the frontend and the backend. At the end of the project, he implemented the dynamic rendering of the elements of a job in the frontend. He also implemented the automated tests, both in the backend and the frontend. At the start of the project, he set up the automatic deployment and the CI/CD pipeline in both repositories.

Sergei was in charge of the backend. He set up the API in the backend and did multiple refactoring of the codebase to keep the architecture in accordance with best practices. He set up the Websocket infra-structure, which involved work in both the backend and the frontend. He also implemented multiple real-time collaboration features. This also involved work in both the backend and frontend. He also implemented some features in the frontend such as the authentication feature and requiring a user to be authenticated before being allowed to enter the main job list page and a job page.

Ghais was in charge of the frontend. He developed all the components of the frontend.

There has been a difference in the amount of hours worked on the thesis between each member, as can be seen in appendix. However, each team member brought unique contributions and expertise to the table in such a way that each member had a big impact on the final product. We could not have reached the final product that we handed in without the contribution of everyone.

Arnaud could give significant time to the thesis. Sergei had prior experience with Dotnet and React and Ghais has extensive experience with React. Also because of this expertise, Sergei usually reviewed merge requests of both Ghais and Arnaud. Ghais usually reviewed Sergei's merge requests in the frontend when it was not about Websockets. Arnaud usually reviewed Sergei's merge requests in the backend and in the frontend when it was about Websockets.

9.8 Project-based work

With the structure that we established right at the start and followed consistently throughout, we aimed to conduct this thesis as much as possible as a development project would be conducted in a work setting. We believe that we succeeded for the most part. We managed to bring together different expertise and truly rely on each other.

The project had to have enough work for a group of three. Reading the project description for the first time felt like we had a lot of challenging work ahead of us. However, going at it step by step, objective by objective, it proved to be definitely manageable. There is always a form of risk when moving forward in a project, especially when working with something completely new, which was the case for everything that had to do with real-time collaboration features. What if we try something and it doesn't work? However, with our approach of starting with what is easiest and building step by step, we could comfort ourselves that if we did something that did not work, we had something to fall back on.

9.9 Closing statement

Throughout this project, we believed that we acquired completely new knowledge on real-time collaboration in web applications with the use of Websockets. We also acquired new knowledge on version conflict solving in collaborative editing, including theoretical knowledge about how the main algorithms in the field work.

This was done while following a process that meets the best practice standard of the industry at the time of this thesis. Above all, we conducted this project as a real team by communicating often and daring to raise and welcome uncomfortable issues.

Finally, we believe that we managed to deliver value to Norkart that made their investment in this thesis worth it even though we do recommend exploring the use of a server-sent events architecture with the HTTP 2 protocol as a next step in the project.

Bibliography

[1] Render. URL: https://render.com/ (visited on 05/06/2024).

[2] *3-tier-architecture*. IBM. URL: https://www.ibm.com/topics/three-tier-architecture.

[3] *9.2 Server-sent events*. HTML Living Standard. Apr. 2024. URL: https://html.spec.whatwg.org/multipage/server-sent-events.html.

[4] Ani Alaverdyan. *Importing images breaks jest test*. Stack Overflow. Feb. 2019. URL: https://stackoverflow.com/questions/46898638/importing-images-breaks-jest-test (visited on 05/05/2024).

[5] Rasmus Appelqvist and Oliver Örnmyr. 'Performance comparison of XHR polling, Long polling, Server sent events and Websockets'. In: 2017. URL: https://api.semanticscholar.org/CorpusID:198960652.

[6] Teyim Asobo. *Effortless Testing Setup for React with Vite, TypeScript, Jest, and React Testing Library*. Jan. 2024. URL: https://dev.to/teyim/effortless-testing-setup-for-react-with-vite-typescript-jest-and-react-testing-library-1c48 (visited on 05/05/2024).

[7] *Automatic Revalidation*. SWR. URL: https://swr.vercel.app/docs/revalidation#revalidate-on-interval.

[8] *await operator - asynchronously await for a task to complete*. Microsoft Learn. Jan. 2024. URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await (visited on 05/04/2024).

[9] *Brené Brown*. University of Houston. Graduate College of Social Work. URL: https://www.uh.edu/socialwork/about/faculty-directory/b-brown/ (visited on 05/09/2024).

[10] Richard Brock. *Collabora Online Development Edition 2.0 released*. Collabora. Nov. 2016. URL: https://www.collaboraoffice.com/press-releases/collabora-online-development-edition-2-0-released/.

[11] Brené Brown. *Courage and Vulnerability Part I: Definitions and Myths*. 2020. URL: https://brenebrown.com/resources/courage-and-vulnerability-part-i-definitions-and-myths/ (visited on 05/09/2024).

[12] Brené Brown. *Courage Over Comfort*. Mar. 2018. URL: https://brenebrown.com/articles/2018/03/13/courage-comfort-rumbling-shame-accountability-failure-work/ (visited on 05/10/2024).

[13] Microsoft 365 help for small businesses. *Best practices for collaborating in Microsoft 365*. Youtube. 2024. URL: https://www.youtube.com/watch?v=7by2fPfaezA&list=PLnWjfDdQkUQSZVpTGdy6homDyZ_RbA5SJ.

[14] *CI/CD YAML syntax reference*. GitLab. URL: https://docs.gitlab.com/ee/ci/yaml/#onlyrefs--exceptrefs (visited on 05/07/2024).

[15] *CI/CD YAML syntax reference*. GitLab. URL: https://docs.gitlab.com/ee/ci/yaml/#onlychanges--exceptchanges (visited on 05/07/2024).

[16] *CI/CD YAML syntax reference*. GitLab. URL: https://docs.gitlab.com/ee/ci/yaml/#workflowrules (visited on 05/07/2024).

[17] *CODE. Collabora Online Development Edition*. Collabora. URL: https://www.collaboraoffice.com/code/.

[18] Codevolution. *React Testing Tutorial*. Youtube. 2023. URL: https://www.youtube.com/playlist?list=PLC3y8-rFHvwirqe1KHFCHJ0RqNuN61SJd (visited on 05/03/2024).

[19] Codevolution. *React Testing Tutorial - 13 - Filename Conventions.* Youtube. 2023. URL: https://www.youtube.com/watch?v=YZtNLuwGCMA&list=PLC3y8-rFHvwirqe1KHFCHJ0RqNuN61SJd&index=14 (visited on 05/02/2024).

[20] Codevolution. *React Testing Tutorial - 27 - Priority Order for Queries.* Youtube. 2023. URL: https://www.youtube.com/watch?v=_e0Jhf0lR2w&list=PLC3y8-rFHvwirqe1KHFCHJ0RqNuN61SJd&index=27.

[21] Szymon Cofalik and Anna Tomanek. *How collaborative editing drove CKEditor 5's architecture.* CKEditor. Nov. 2023. URL: https://ckeditor.com/blog/lessons-learned-from-creating-a-rich-text-editor-with-real-time-collaboration/.

[22] *Comparison of the usage statistics of HTTP/2 vs. HTTP/3 for websites.* W3Techs. Web Technologies Survey. Apr. 2024. URL: https://w3techs.com/technologies/comparison/ce-http2,ce-http3.

[23] *CSS.* Wikipedia. URL: https://en.wikipedia.org/wiki/CSS.

[24] Adrian Defus. *The Myers Diff Algorithm and Kotlin Observable Properties — how to connect them to make a developer's life easier.* Medium. 2019. URL: https://medium.com/skyrise/the-myers-diff-algorithm-and-kotlin-observable-properties-69dfb18541b.

[25] Tim Deschryver. *How to test your C# Web API.* Sept. 2023. URL: https://timdeschryver.dev/blog/how-to-test-your-csharp-web-api (visited on 05/04/2024).

[26] TL;DR // JavaScript codecasts for working devs. *Conflict-Free Replicated Data Types (CRDT) for Distributed JavaScript Apps.* Youtube. 2020. URL: https://www.youtube.com/watch?v=M8-WFTjZoA0.

[27] *DI container.* Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection.

[28] *Docker on Render.* Render. URL: https://docs.render.com/docker#docker-builds-on-render.

[29] dotnet. *What is .NET? [Pt 1] | .NET for Beginners.* Youtube. 2024. URL: https://www.youtube.com/watch?v=6BcPIvVfVAw#t=18s.

[30] Charles Duhigg. *What Google Learned From Its Quest to Build the Perfect Team.* New-york Times. Feb. 2016. URL: https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html (visited on 05/09/2024).

[31] Amy Edmondson. *Psychological Safety and Learning Behavior in Work Teams.* Johnson Graduate School of Management, Cornell University. June 1999. URL: https://web.mit.edu/curhan/www/docs/Articles/15341_Readings/Group_Performance/Edmondson%20Psychological%20safety.pdf (visited on 05/09/2024).

[32] *Eldrebølgen skaper stor usikkerhet for helsesektoren.* Statistisk sentralbyrå. URL: https://www.ssb.no/arbeid-og-lonn/sysselsetting/artikler/eldrebolgen-skaper-stor-usikkerhet-for-helsesektoren.

[33] *Facebook / React. Releases.* GitHub. URL: https://github.com/facebook/react/releases?page=10.

[34] *FakeItEasy. The easy mocking library for .NET.* URL: https://fakeiteasy.github.io/ (visited on 05/04/2024).

[35] Tomas Fernandez. *Vitest: Replacing Jest on Vite Projects.* Apr. 2024. URL: https://semaphoreci.com/blog/vitest#:~:text=Since%20Jest%20doesn't%20natively,layers%20of%20complexity%20and%20configuration.&text=Vitest%2C%20on%20the%20other%20hand,speeding%20up%20the%20testing%20process. (visited on 05/05/2024).

[36] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* University of California - Irvine. 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[37] Roy Thomas Fielding. *CHAPTER 5 - Representational State Transfer (REST).* University of California - Irvine. 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_2.

[38] *FIGMA DESIGN.* Figma. URL: https://www.figma.com/design/.

[39] *Firebase CLI reference.* Firebase. URL: https://firebase.google.com/docs/cli/ (visited on 05/07/2024).

[40] *Firebase Hosting.* Firebase. URL: https://firebase.google.com/docs/hosting (visited on 05/06/2024).

[41] Beyond Fireship. *10 Rendering Patterns for Web Apps*. Youtube. 2023. URL: https://www.youtube.com/watch?v=Dkx5ydvtpCA#t=1m50s.

[42] *Fluent Assertions*. URL: https://fluentassertions.com/ (visited on 05/04/2024).

[43] Neil Fraser. *Differential Synchronization*. Google. 2009. URL: https://static.googleusercontent.com/media/research.google.com/no//pubs/archive/35605.pdf.

[44] Michael Goodwin. *What is an API (application programming interface)?* IBM. Apr. 2024. URL: https://www.ibm.com/topics/api.

[45] Sarah Griffiths. *Why your internet habits are not as clean as you think*. BBC. Mar. 2020. URL: https://www.bbc.com/news/business-56559073.

[46] Thomas Hamilton. *Decision Table Testing (Example)*. Guru99. Feb. 2024. URL: https://www.guru99.com/decision-table-testing.html (visited on 05/02/2024).

[47] *History*. Libre Office. The Document Foundation. URL: https://www.documentfoundation.org/history/.

[48] *How long can you work on making a routine task more efficient before you're spending more time than you save?(across five years)*. XKCD. URL: https://imgs.xkcd.com/comics/is_it_worth_the_time.png.

[49] *How to make http2 requests with persistent connection ? (Any language)*. StackOverflow. URL: https://stackoverflow.com/questions/37007100/how-to-make-http2-requests-with-persistent-connection-any-language.

[50] *How you design , align , and build matters. Do it together with Figma*. Figma. URL: https://www.figma.com/.

[51] *HTTP/3*. Internet Engineering Task Force (IETF). May 2022. URL: https://datatracker.ietf.org/doc/html/rfc9114.

[52] *Hypertext Transfer Protocol Version 2 (HTTP/2)*. Internet Engineering Task Force (IETF). May 2015. URL: https://datatracker.ietf.org/doc/html/rfc7540.

[53] *Integration tests in ASP.NET Core*. Microsoft Learn. Feb. 2024. URL: https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-8.0 (visited on 05/04/2024).

[54] *Introduction*. Yjs Docs. URL: https://docs.yjs.dev/.

[55] *Introduction to SignalR*. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr#transports-and-fallbacks.

[56] Jamsheedsaeed. *how to setup a Ci/CD pipeline for React app using gitlab*. Medium. Feb. 2023. URL: https://blog.devops.dev/how-to-setup-a-ci-cd-pipeline-for-react-app-using-gitlab-9e7729d49732 (visited on 05/06/2024).

[57] *Jotform. About Us*. Jotform. URL: https://www.jotform.com/about/.

[58] *jsdiff*. NPM. URL: https://www.npmjs.com/package/diff.

[59] Thomas Labonne. *8 features for better collaboration on Figma*. Digidop. Mar. 2023. URL: https://www.digidop.fr/en/blog/features-better-collaboration-figma#working-simultaneously.

[60] Black Diamond Learning. *GitLab Release Management*. Youtube. 2022. URL: https://www.youtube.com/watch?v=q_QGNwQJH9g (visited on 05/06/2024).

[61] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. 1st. USA: Prentice Hall Press, 2017. ISBN: 0134494164.

[62] *MongoDB Atlas*. MongoDB. URL: https://www.mongodb.com/cloud/atlas/register (visited on 05/06/2024).

[63] *Move faster with intuitive React UI tools*. MUI. URL: https://mui.com/.

[64] Mycelial. *An introduction to Conflict-Free Replicated Data Types (CRDTs)*. Youtube. 2022. URL: https://www.youtube.com/watch?v=gZP2VUmH05A.

[65] Eugene W. Myers. *An O(ND) Difference Algorithm and Its Variations*. Department of Computer Science, University of Arizona, Tucson. URL: http://www.xmailserver.org/diff2.pdf.

[66] Kunal Nalawade. *How to Dockerize a React Application – A Step by Step Tutorial*. FreeCodeCamp. July 2023. URL: https://www.freecodecamp.org/news/how-to-dockerize-a-react-application/ (visited on 05/06/2024).

[67] Brice Nédelec et al. *LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing*. HAL. Open Science. 2013. URL: https://hal.science/hal-00921633/document.

[68] neomede. *chattp2*. GitHub. URL: https://github.com/neomede/chattp2.

[69] Norkart. *KOMTEK*. Norkart. URL: https://www.norkart.no/offentlig/komtek.

[70] Norkart. *Sammen skaper vi smartere samfunn*. Norkart. URL: https://www.norkart.no/.

[71] *Object and Collection Initializers (C# Programming Guide)*. Microsoft Learn. Apr. 2024. URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/object-and-collection-initializers (visited on 05/04/2024).

[72] *Our expertise*. Collabora. URL: https://www.collabora.com/about-us/our-expertise.html.

[73] Jaydeep Patil. *Containerization of the .NET Core 7 Web API using Docker*. Medium. Sept. 2023. URL: https://medium.com/@jaydeepvpatil225/containerization-of-the-net-core-7-web-api-using-docker-3abdd543f78a (visited on 05/06/2024).

[74] *Persistent Connections*. O'REILLY. URL: https://www.oreilly.com/library/view/http-the-definitive/1565925092/ch04s05.html.

[75] Sten Pittet. *Exhaustive testing not possible.., Why?* LinkedIn. Jan. 2024. URL: https://www.linkedin.com/pulse/exhaustive-testing-possible-why-rakib-hasan-lq2hc/ (visited on 05/02/2024).

[76] Sten Pittet. *What is code coverage?* Altassian. URL: https://www.atlassian.com/continuous-delivery/software-testing/code-coverage (visited on 05/02/2024).

[77] *Predefined CI/CD variables reference*. GitLab. URL: https://docs.gitlab.com/ee/ci/variables/predefined_variables.html (visited on 05/07/2024).

[78] Yamini Priya. *What is Testing Pyramid? How Does It Benefit Agile Teams?* Testsigma. Apr. 2024. URL: https://testsigma.com/blog/testing-pyramid/ (visited on 05/02/2024).

[79] *Push mirroring*. GitLab. URL: https://gitlab.stud.idi.ntnu.no/help/user/project/repository/mirror/push#keep-divergent-refs (visited on 05/06/2024).

[80] *Push mirroring*. GitLab. URL: https://gitlab.stud.idi.ntnu.no/help/user/project/repository/mirror/push#set-up-a-push-mirror-from-gitlab-to-github (visited on 05/06/2024).

[81] *React. The library for web and native user interfaces*. Meta Open Source. URL: https://react.dev/.

[82] Mary-Ann Russon. *The cost of the Suez Canal blockage*. BBC. Mar. 2021. URL: https://www.bbc.com/news/business-56559073.

[83] *Separation of concerns*. URL: https://nalexn.github.io/separation-of-concerns/.

[84] sfn. *Test suite failed to run import.meta.env.VITE_*. Stack Overflow. May 2022. URL: https://stackoverflow.com/questions/72128718/test-suite-failed-to-run-import-meta-env-vite (visited on 05/05/2024).

[85] Marc Shapiro et al. *Conflict-free Replicated Data Types*. Aug. 2011. URL: https://pages.lip6.fr/Marc.Shapiro/papers/RR-7687.pdf.

[86] ShowMeYourCode! *Mirroring repositories from GitLab to GitHub | pull changes automatically*. Youtube. 2023. URL: https://www.youtube.com/watch?v=E4Y6A1HplWc.

[87] Teddy Smith. *Unit Testing in C# 2022: 1. Intro + First Test*. Youtube. 2022. URL: https://www.youtube.com/watch?v=aq3lbO0RwAQ&list=PL82C6-O4XrHeyeJcl5xrywgpfbrqdkQd4 (visited on 05/02/2024).

[88] *Software Developers, Quality Assurance Analysts, and Testers. Summary*. U.S. Bureau of Labor Statistics. Apr. 2024. URL: https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-1 (visited on 05/01/2024).

[89] *Software Developers, Quality Assurance Analysts, and Testers. What they do*. U.S. Bureau of Labor Statistics. Apr. 2024. URL: https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-2 (visited on 05/01/2024).

[90] *Software Testing – Use Case Testing*. Geeks for Geeks. Jan. 2024. URL: https://www.geeksforgeeks.org/software-testing-use-case-testing/ (visited on 05/02/2024).

[91] *Software Testing Strategies*. Geeks for Geeks. Feb. 2023. URL: https://www.geeksforgeeks.org/software-testing-strategies/ (visited on 05/02/2024).

[92] Coursera Staff. *What Is a QA Tester? Skills, Requirements, and Jobs in 2024.* Coursera. Mar. 2024. URL: https://www.coursera.org/articles/qa-tester (visited on 05/01/2024).

[93] *Supported Languages.* Render. URL: https://docs.render.com/language-support#:~:text=Render%20natively%20supports%20Node.,Go%2C%20Rust%2C%20and%20Elixir..

[94] *Sustainable Development.* International Institute for Sustainable Development. URL: https://www.iisd.org/mission-and-goals/sustainable-development (visited on 05/12/2024).

[95] *SWR. React Hooks for Data Fetching.* SWR. URL: https://swr.vercel.app/.

[96] Codium AI Team. *Automated Test Suites: Best Practices for Effective Implementation.* Guru99. Feb. 2024. URL: https://www.codium.ai/blog/automated-test-suites-best-practices-for-effective-implementation/ (visited on 05/02/2024).

[97] *The WebSocket Protocol.* Internet Engineering Task Force (IETF). Dec. 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455.

[98] *Tutorial: Explore ideas using top-level statements to build code as you learn.* Microsoft Learn. Nov. 2023. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tutorials/top-level-statements (visited on 05/04/2024).

[99] *TypeScript.* URL: https://www.typescriptlang.org/.

[100] Unimicro. *Er du opptatt av kvalitet?* Finn. Apr. 2024. URL: https://www.finn.no/job/fulltime/ad.html?finnkode=345084592 (visited on 05/01/2024).

[101] *Unit Testing ASP.NET Web API 2.* Microsoft Learn. Sept. 2022. URL: https://learn.microsoft.com/en-us/aspnet/web-api/overview/testing-and-debugging/unit-testing-with-aspnet-web-api (visited on 05/02/2024).

[102] *Use HttpContext in ASP.NET Core.* Microsoft Learn. Mar. 2023. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/use-http-context?view=aspnetcore-8.0 (visited on 05/04/2024).

[103] *Using secrets in GitHub Actions.* GitHub. URL: https://docs.github.com/en/actions/security-guides/using-secrets-in-github-actions#storing-large-secrets.

[104] *Using server-sent events.* MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events.

[105] *Vite. Next Generation Frontend Tooling.* URL: https://vitejs.dev/.

[106] *Vitest. Next Generation Testing Framework.* URL: https://vitest.dev/ (visited on 05/05/2024).

[107] *WebSockets.* HTML Living Standard. Jan. 2024. URL: https://websockets.spec.whatwg.org//.

[108] *What is .NET?* Microsoft. URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet#:~:text=.NET%20is%20a%20secure%2C%20reliable,concurrency%20and%20automatic%20memory%20management..

[109] *Who we are.* Collabora. URL: https://www.collabora.com/about-us/who-we-are/.

[110] *Xunit.* Nuget. URL: https://www.nuget.org/packages/xunit (visited on 05/04/2024).

[111] Anto Zagorskii. *Operational Transformations as an algorithm for automatic conflict resolution.* Medium. July 2018. URL: https://medium.com/coinmonks/operational-transformations-as-an-algorithm-for-automatic-conflict-resolution-3bf8920ea447.

Appendix A

Terms and Acronyms

Terms & Acronyms

API Application Programming Interface. 3, 10, 13, 20–22, 34, 38, 40–42, 55, 64, 65, 69, 70, 72, 73

CI/CD Continuous Integration / Continuous Deployment. iv, viii, 56, 57, 65–68, 70, 71, 73

Collaboration feature feature that provides information in real time to users about other users.. vi, 1, 16

Collaborative editing feature that allows multiple users to edit the same document at the same time.. 1, 14

CRUD abbreviation of operations within the context an API: create, read, update, delete. 43, 46

CSS Cascading Style Sheet. Style sheet language used for specifying the presentation and styling of a document written in a markup language such as HTML [23]. 3

DOM Document Object Model. A programming interface for web documents.. 12, 13

Git distributed version control system primarily used for tracking changes in files. 1

HTML HyperText Markup Language. The standard markup language used to create and design web pages.. 3, 12, 13, 21, 22, 61

job A page in the Komtek software that contains several tasks. When all tasks in the job are completed, the job is completed.. i, vi, vii, 2, 13, 14, 20, 33, 35, 40, 52, 69, 70, 72, 73

Komtek An all in one software solution to handle tasks related to drinking water and sewage, fire prevention, property tax, renovations and invoices for city planning departments ([69]).. i, vi, 2, 3, 5, 6, 10, 12–14, 50, 52, 54, 69, 70, 72, 73

Norkart Norkart AS, a software and data provider with the objective to develop solutions that contribute to simplify the life of people, the industry, the state and municipalities ([70]).. i, ii, 2–4, 10, 12–14, 20, 34, 35, 50–54, 68–70, 73, 74

React A popular library written in Javascript for making single page applications.. 3, 13, 20, 61, 67, 70, 73

Redis Redis (Remote Dictionary Server) is an open-source, in-memory data structure store used as a database and a cache. 35

REST Representational State Transfer.An architectural style for designing networked applications, particularly web services.. 13, 34, 40, 42

TCP Transmission Control Protocol. Used for transmitting data reliably across networks.. 21, 22, 73

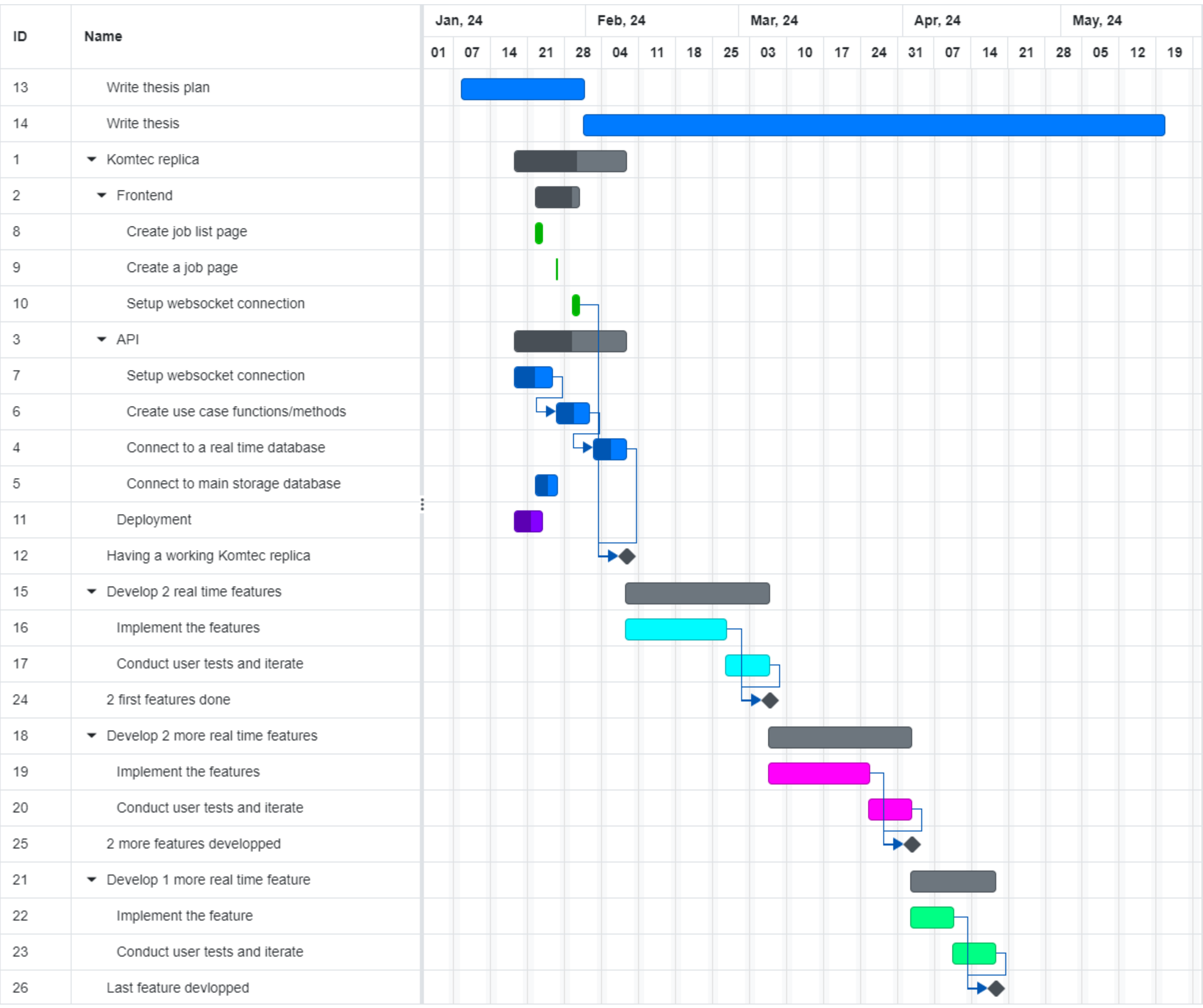
UML Unified Modeling Language. A standardized, general-purpose modeling language used in software engineering for visualizing, specifying, constructing, and documenting the artifacts of a software system. 32

URI Uniform Resource Identifier. 22

Websocket A communication protocol that provides full-duplex communication channels over a single, long-lived connection between clients and servers.. 3, 21, 22, 33, 34, 39, 40, 42, 57, 59, 60, 62, 69, 71–74

Appendix B

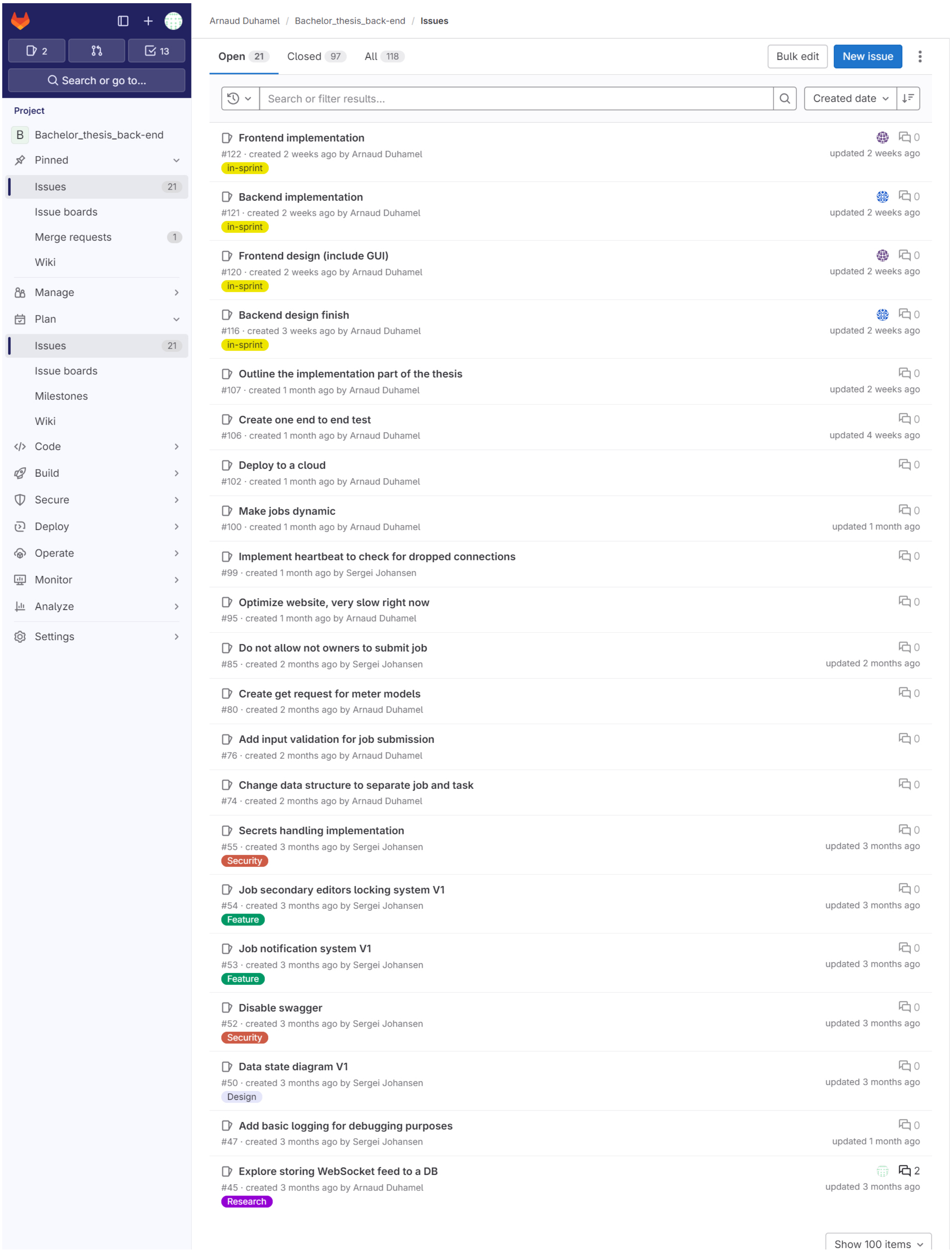
Gantt Diagram / Project Schedule



ID	Name
13	Write thesis plan
14	Write thesis
1	Komtec replica
2	Frontend
8	Create job list page
9	Create a job page
10	Setup websocket connection
3	API
7	Setup websocket connection
6	Create use case functions/methods
4	Connect to a real time database
5	Connect to main storage database
11	Deployment
12	Having a working Komtec replica
15	Develop 2 real time features
16	Implement the features
17	Conduct user tests and iterate
24	2 first features done
18	Develop 2 more real time features
19	Implement the features
20	Conduct user tests and iterate
25	2 more features developped
21	Develop 1 more real time feature
22	Implement the feature
23	Conduct user tests and iterate
26	Last feature devlopped

Appendix C

Open backend issues at the end of the project



Arnaud Duhamel / Bachelor_thesis_back-end / Issues

Open 21 Closed 97 All 118 Bulk edit New issue

Search or filter results... Created date

Project
Bachelor_thesis_back-end
Pinned
Issues 21
Issue boards
Merge requests 1
Wiki
Manage
Plan
Issues 21
Issue boards
Milestones
Wiki
Code
Build
Secure
Deploy
Operate
Monitor
Analyze
Settings

Frontend implementation 0
#122 · created 2 weeks ago by Arnaud Duhamel updated 2 weeks ago
in-sprint

Backend implementation 0
#121 · created 2 weeks ago by Arnaud Duhamel updated 2 weeks ago
in-sprint

Frontend design (include GUI) 0
#120 · created 2 weeks ago by Arnaud Duhamel updated 2 weeks ago
in-sprint

Backend design finish 0
#116 · created 3 weeks ago by Arnaud Duhamel updated 2 weeks ago
in-sprint

Outline the implementation part of the thesis 0
#107 · created 1 month ago by Arnaud Duhamel updated 2 weeks ago

Create one end to end test 0
#106 · created 1 month ago by Arnaud Duhamel updated 4 weeks ago

Deploy to a cloud 0
#102 · created 1 month ago by Arnaud Duhamel

Make jobs dynamic 0
#100 · created 1 month ago by Arnaud Duhamel updated 1 month ago

Implement heartbeat to check for dropped connections 0
#99 · created 1 month ago by Sergei Johansen

Optimize website, very slow right now 0
#95 · created 1 month ago by Arnaud Duhamel

Do not allow not owners to submit job 0
#85 · created 2 months ago by Sergei Johansen updated 2 months ago

Create get request for meter models 0
#80 · created 2 months ago by Arnaud Duhamel

Add input validation for job submission 0
#76 · created 2 months ago by Arnaud Duhamel

Change data structure to separate job and task 0
#74 · created 2 months ago by Arnaud Duhamel

Secrets handling implementation 0
#55 · created 3 months ago by Sergei Johansen updated 3 months ago
Security

Job secondary editors locking system V1 0
#54 · created 3 months ago by Sergei Johansen updated 3 months ago
Feature

Job notification system V1 0
#53 · created 3 months ago by Sergei Johansen updated 3 months ago
Feature

Disable swagger 0
#52 · created 3 months ago by Sergei Johansen updated 3 months ago
Security

Data state diagram V1 0
#50 · created 3 months ago by Sergei Johansen updated 3 months ago
Design

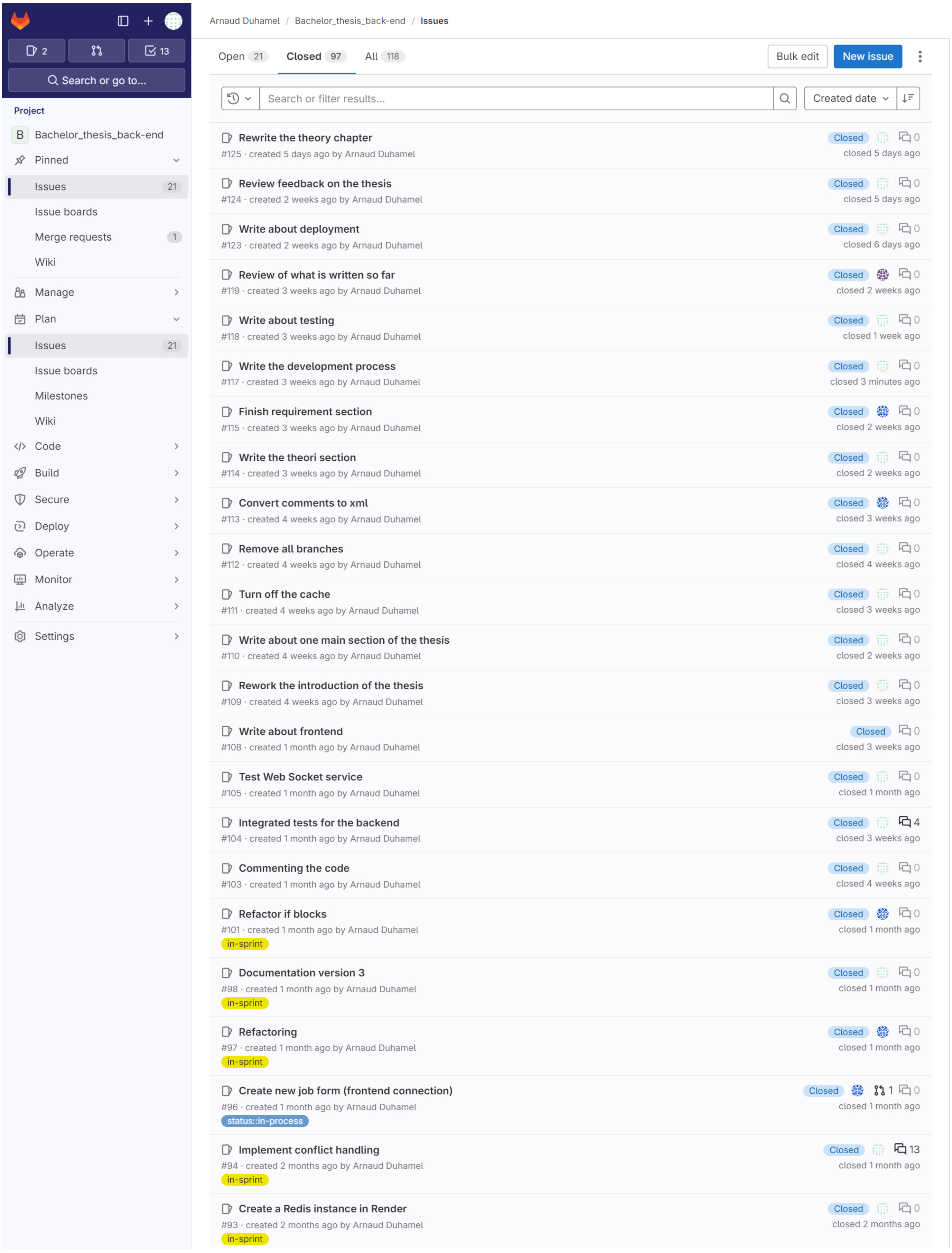
Add basic logging for debugging purposes 0
#47 · created 3 months ago by Sergei Johansen updated 1 month ago

Explore storing WebSocket feed to a DB 2
#45 · created 3 months ago by Arnaud Duhamel updated 3 months ago
Research

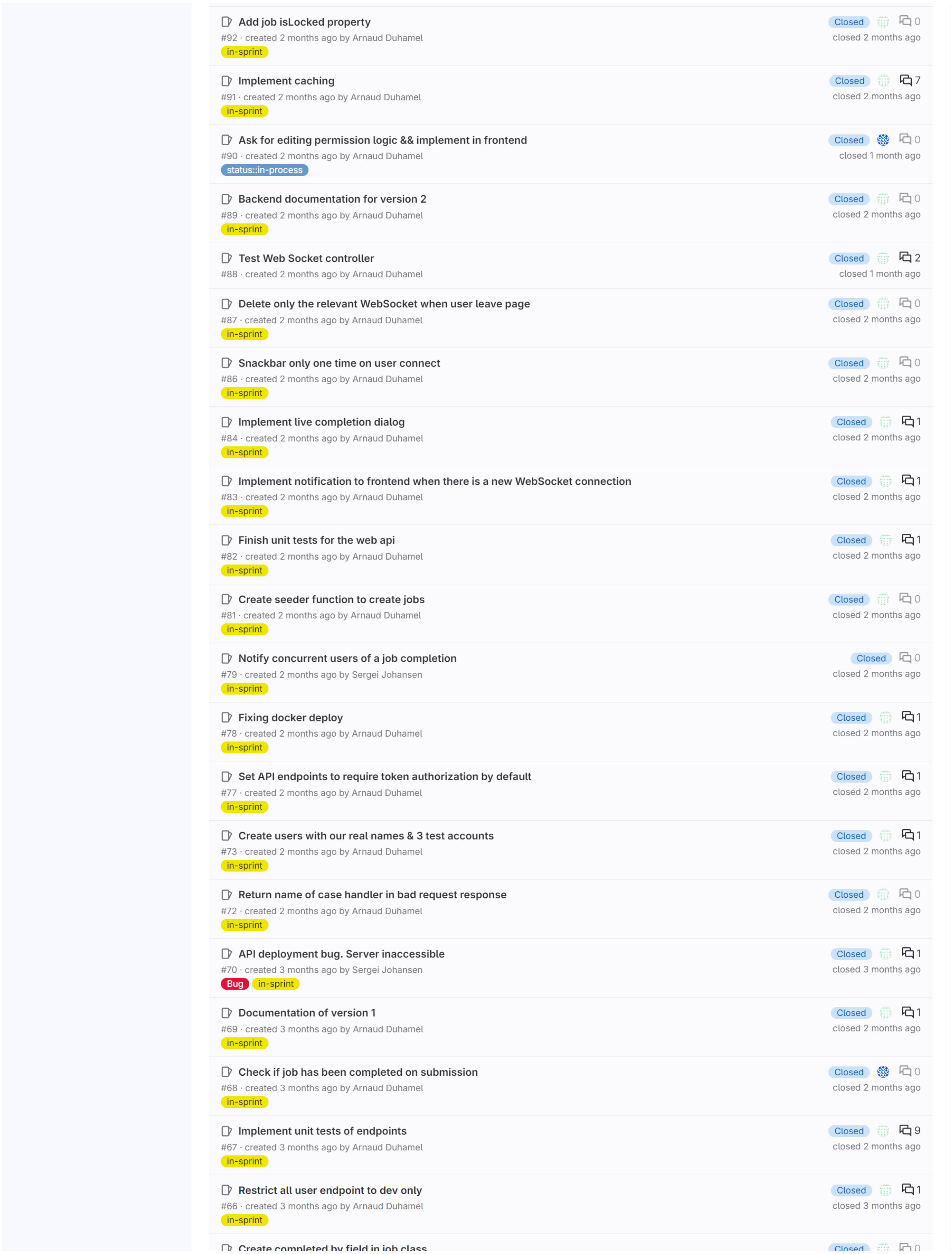
Show 100 items

Appendix D

Closed backend issues at the end of the project



Issue	Details	Status	Comments	Closed
Rewrite the theory chapter	#125 · created 5 days ago by Arnaud Duhamel	Closed	0	closed 5 days ago
Review feedback on the thesis	#124 · created 2 weeks ago by Arnaud Duhamel	Closed	0	closed 5 days ago
Write about deployment	#123 · created 2 weeks ago by Arnaud Duhamel	Closed	0	closed 6 days ago
Review of what is written so far	#119 · created 3 weeks ago by Arnaud Duhamel	Closed	0	closed 2 weeks ago
Write about testing	#118 · created 3 weeks ago by Arnaud Duhamel	Closed	0	closed 1 week ago
Write the development process	#117 · created 3 weeks ago by Arnaud Duhamel	Closed	0	closed 3 minutes ago
Finish requirement section	#115 · created 3 weeks ago by Arnaud Duhamel	Closed	0	closed 2 weeks ago
Write the theori section	#114 · created 3 weeks ago by Arnaud Duhamel	Closed	0	closed 2 weeks ago
Convert comments to xml	#113 · created 4 weeks ago by Arnaud Duhamel	Closed	0	closed 3 weeks ago
Remove all branches	#112 · created 4 weeks ago by Arnaud Duhamel	Closed	0	closed 4 weeks ago
Turn off the cache	#111 · created 4 weeks ago by Arnaud Duhamel	Closed	0	closed 3 weeks ago
Write about one main section of the thesis	#110 · created 4 weeks ago by Arnaud Duhamel	Closed	0	closed 2 weeks ago
Rework the introduction of the thesis	#109 · created 4 weeks ago by Arnaud Duhamel	Closed	0	closed 3 weeks ago
Write about frontend	#108 · created 1 month ago by Arnaud Duhamel	Closed	0	closed 3 weeks ago
Test Web Socket service	#105 · created 1 month ago by Arnaud Duhamel	Closed	0	closed 1 month ago
Integrated tests for the backend	#104 · created 1 month ago by Arnaud Duhamel	Closed	4	closed 3 weeks ago
Commenting the code	#103 · created 1 month ago by Arnaud Duhamel	Closed	0	closed 4 weeks ago
Refactor if blocks	#101 · created 1 month ago by Arnaud Duhamel in-sprint	Closed	0	closed 1 month ago
Documentation version 3	#98 · created 1 month ago by Arnaud Duhamel in-sprint	Closed	0	closed 1 month ago
Refactoring	#97 · created 1 month ago by Arnaud Duhamel in-sprint	Closed	0	closed 1 month ago
Create new job form (frontend connection)	#96 · created 1 month ago by Arnaud Duhamel status::in-process	Closed	1 0	closed 1 month ago
Implement conflict handling	#94 · created 2 months ago by Arnaud Duhamel in-sprint	Closed	13	closed 1 month ago
Create a Redis instance in Render	#93 · created 2 months ago by Arnaud Duhamel in-sprint	Closed	0	closed 2 months ago



Add job isLocked property — Closed · 0
#92 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Implement caching — Closed · 7
#91 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Ask for editing permission logic && implement in frontend — Closed · 0
#90 · created 2 months ago by Arnaud Duhamel — closed 1 month ago
status::in-process

Backend documentation for version 2 — Closed · 0
#89 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Test Web Socket controller — Closed · 2
#88 · created 2 months ago by Arnaud Duhamel — closed 1 month ago

Delete only the relevant WebSocket when user leave page — Closed · 0
#87 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Snackbar only one time on user connect — Closed · 0
#86 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Implement live completion dialog — Closed · 1
#84 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Implement notification to frontend when there is a new WebSocket connection — Closed · 1
#83 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Finish unit tests for the web api — Closed · 1
#82 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Create seeder function to create jobs — Closed · 0
#81 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Notify concurrent users of a job completion — Closed · 0
#79 · created 2 months ago by Sergei Johansen — closed 2 months ago
in-sprint

Fixing docker deploy — Closed · 1
#78 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Set API endpoints to require token authorization by default — Closed · 1
#77 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Create users with our real names & 3 test accounts — Closed · 1
#73 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Return name of case handler in bad request response — Closed · 0
#72 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

API deployment bug. Server inaccessible — Closed · 1
#70 · created 3 months ago by Sergei Johansen — closed 3 months ago
Bug in-sprint

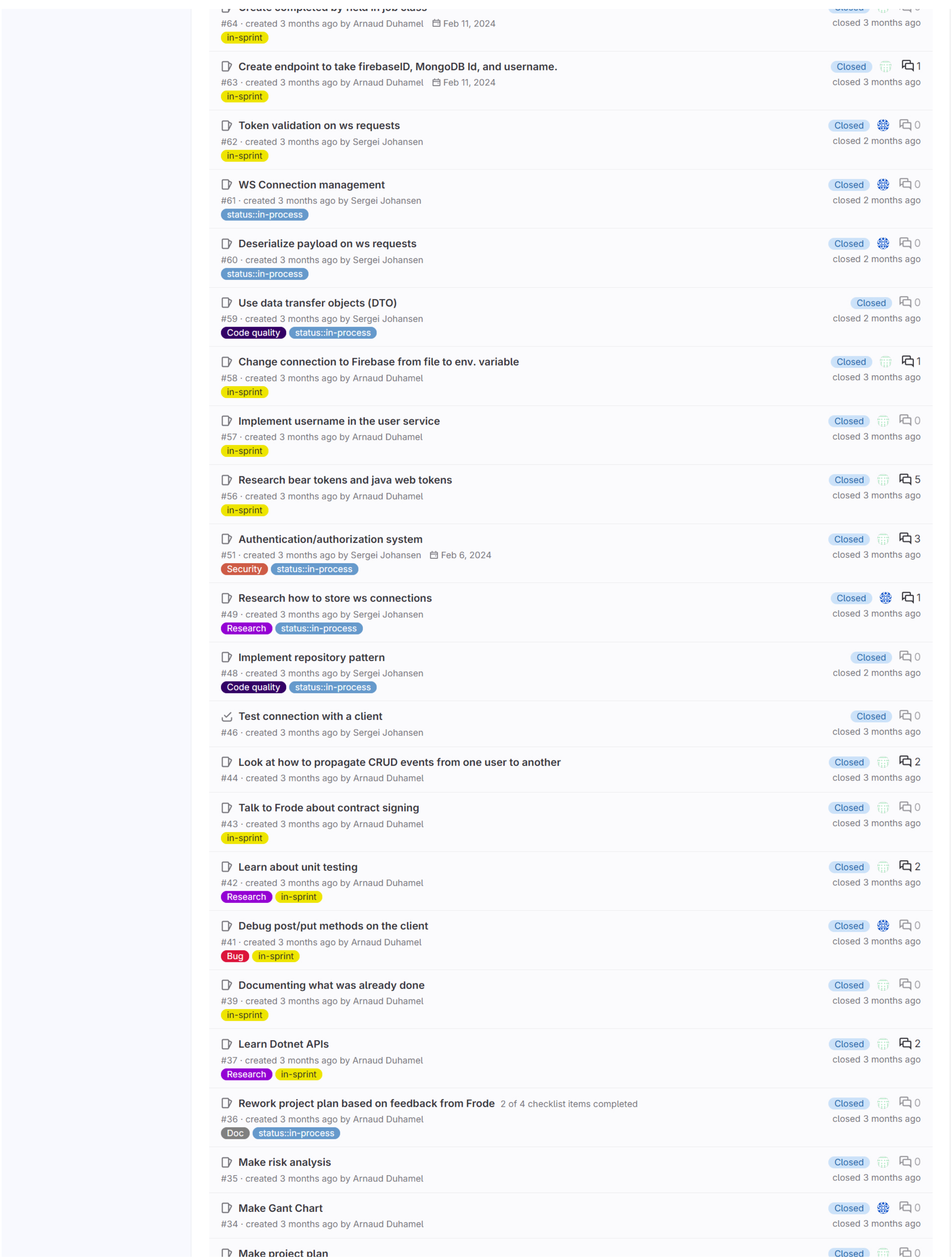
Documentation of version 1 — Closed · 1
#69 · created 3 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Check if job has been completed on submission — Closed · 0
#68 · created 3 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Implement unit tests of endpoints — Closed · 9
#67 · created 3 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Restrict all user endpoint to dev only — Closed · 1
#66 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Create completed by field in job class — Closed · 0


#64 · created 3 months ago by Arnaud Duhamel Feb 11, 2024 — closed 3 months ago
in-sprint

Create endpoint to take firebaseID, MongoDB Id, and username. — Closed 1
#63 · created 3 months ago by Arnaud Duhamel Feb 11, 2024 — closed 3 months ago
in-sprint

Token validation on ws requests — Closed 0
#62 · created 3 months ago by Sergei Johansen — closed 2 months ago
in-sprint

WS Connection management — Closed 0
#61 · created 3 months ago by Sergei Johansen — closed 2 months ago
status::in-process

Deserialize payload on ws requests — Closed 0
#60 · created 3 months ago by Sergei Johansen — closed 2 months ago
status::in-process

Use data transfer objects (DTO) — Closed 0
#59 · created 3 months ago by Sergei Johansen — closed 2 months ago
Code quality status::in-process

Change connection to Firebase from file to env. variable — Closed 1
#58 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Implement username in the user service — Closed 0
#57 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Research bear tokens and java web tokens — Closed 5
#56 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Authentication/authorization system — Closed 3
#51 · created 3 months ago by Sergei Johansen Feb 6, 2024 — closed 3 months ago
Security status::in-process

Research how to store ws connections — Closed 1
#49 · created 3 months ago by Sergei Johansen — closed 3 months ago
Research status::in-process

Implement repository pattern — Closed 0
#48 · created 3 months ago by Sergei Johansen — closed 2 months ago
Code quality status::in-process

Test connection with a client — Closed 0
#46 · created 3 months ago by Sergei Johansen — closed 3 months ago

Look at how to propagate CRUD events from one user to another — Closed 2
#44 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Talk to Frode about contract signing — Closed 0
#43 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Learn about unit testing — Closed 2
#42 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
Research in-sprint

Debug post/put methods on the client — Closed 0
#41 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
Bug in-sprint

Documenting what was already done — Closed 0
#39 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

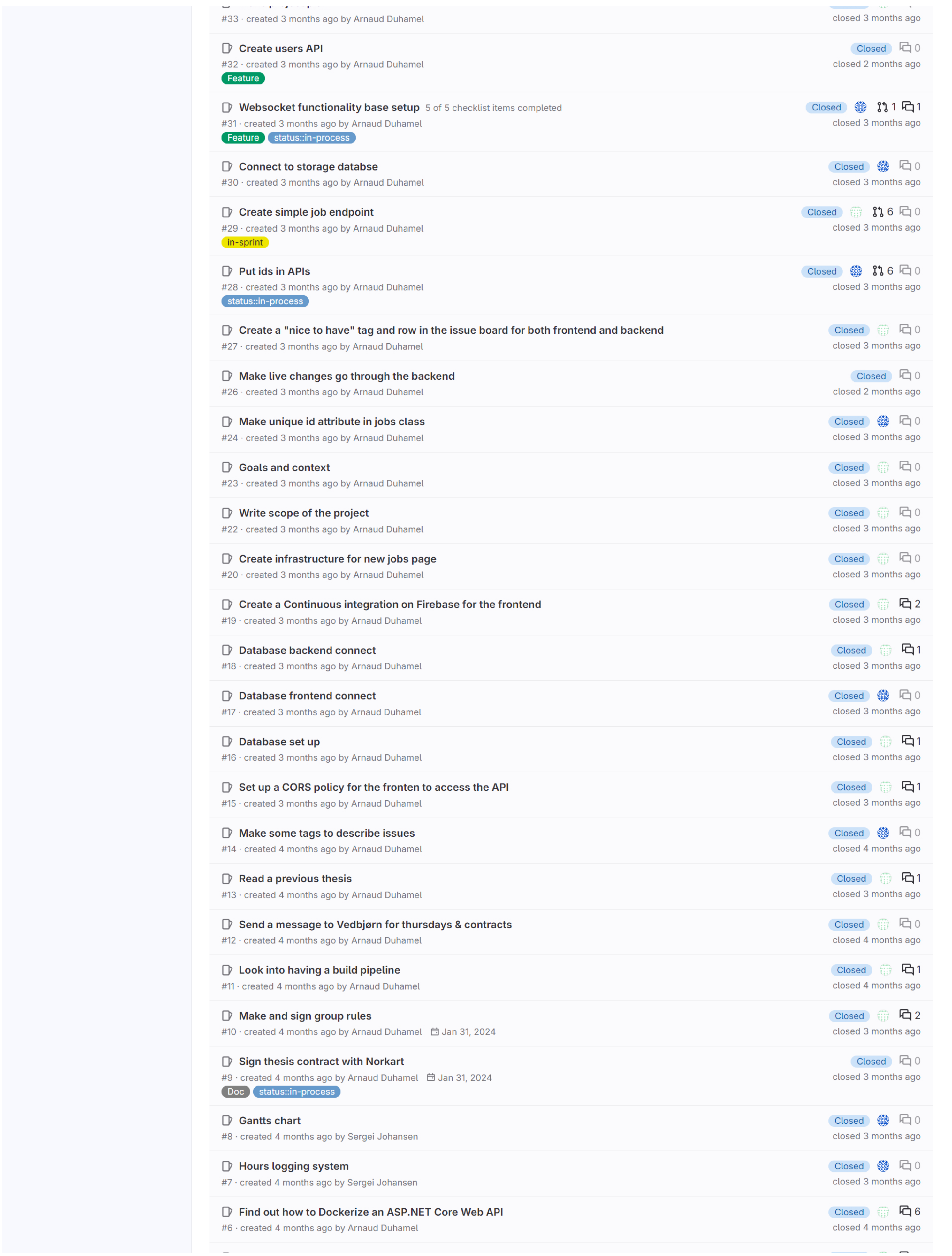
Learn Dotnet APIs — Closed 2
#37 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
Research in-sprint

Rework project plan based on feedback from Frode 2 of 4 checklist items completed — Closed 0
#36 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
Doc status::in-process

Make risk analysis — Closed 0
#35 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Make Gant Chart — Closed 0
#34 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Make project plan — Closed 0



#33 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Create users API — Closed · 0
#32 · created 3 months ago by Arnaud Duhamel — closed 2 months ago
Feature

Websocket functionality base setup 5 of 5 checklist items completed — Closed · 1 · 1
#31 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
Feature status::in-process

Connect to storage databse — Closed · 0
#30 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Create simple job endpoint — Closed · 6 · 0
#29 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Put ids in APIs — Closed · 6 · 0
#28 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
status::in-process

Create a "nice to have" tag and row in the issue board for both frontend and backend — Closed · 0
#27 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Make live changes go through the backend — Closed · 0
#26 · created 3 months ago by Arnaud Duhamel — closed 2 months ago

Make unique id attribute in jobs class — Closed · 0
#24 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Goals and context — Closed · 0
#23 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Write scope of the project — Closed · 0
#22 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Create infrastructure for new jobs page — Closed · 0
#20 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Create a Continuous integration on Firebase for the frontend — Closed · 2
#19 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Database backend connect — Closed · 1
#18 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Database frontend connect — Closed · 0
#17 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Database set up — Closed · 1
#16 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Set up a CORS policy for the fronten to access the API — Closed · 1
#15 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Make some tags to describe issues — Closed · 0
#14 · created 4 months ago by Arnaud Duhamel — closed 4 months ago

Read a previous thesis — Closed · 1
#13 · created 4 months ago by Arnaud Duhamel — closed 3 months ago

Send a message to Vedbjørn for thursdays & contracts — Closed · 0
#12 · created 4 months ago by Arnaud Duhamel — closed 4 months ago

Look into having a build pipeline — Closed · 1
#11 · created 4 months ago by Arnaud Duhamel — closed 4 months ago

Make and sign group rules — Closed · 2
#10 · created 4 months ago by Arnaud Duhamel · Jan 31, 2024 — closed 3 months ago

Sign thesis contract with Norkart — Closed · 0
#9 · created 4 months ago by Arnaud Duhamel · Jan 31, 2024 — closed 3 months ago
Doc status::in-process

Gantts chart — Closed · 0
#8 · created 4 months ago by Sergei Johansen — closed 3 months ago

Hours logging system — Closed · 0
#7 · created 4 months ago by Sergei Johansen — closed 3 months ago

Find out how to Dockerize an ASP.NET Core Web API — Closed · 6
#6 · created 4 months ago by Arnaud Duhamel — closed 4 months ago

Mirror this repository with a GitHub repository Closed 3
#5 · created 4 months ago by Arnaud Duhamel closed 4 months ago

Connect to the database Closed 0
#4 · created 4 months ago by Arnaud Duhamel closed 3 months ago

Deploy the backend Closed 2
#3 · created 4 months ago by Arnaud Duhamel closed 4 months ago

Finne en .gitignore mal for .NET prosjekter Closed 1
#2 · created 4 months ago by Arnaud Duhamel closed 4 months ago

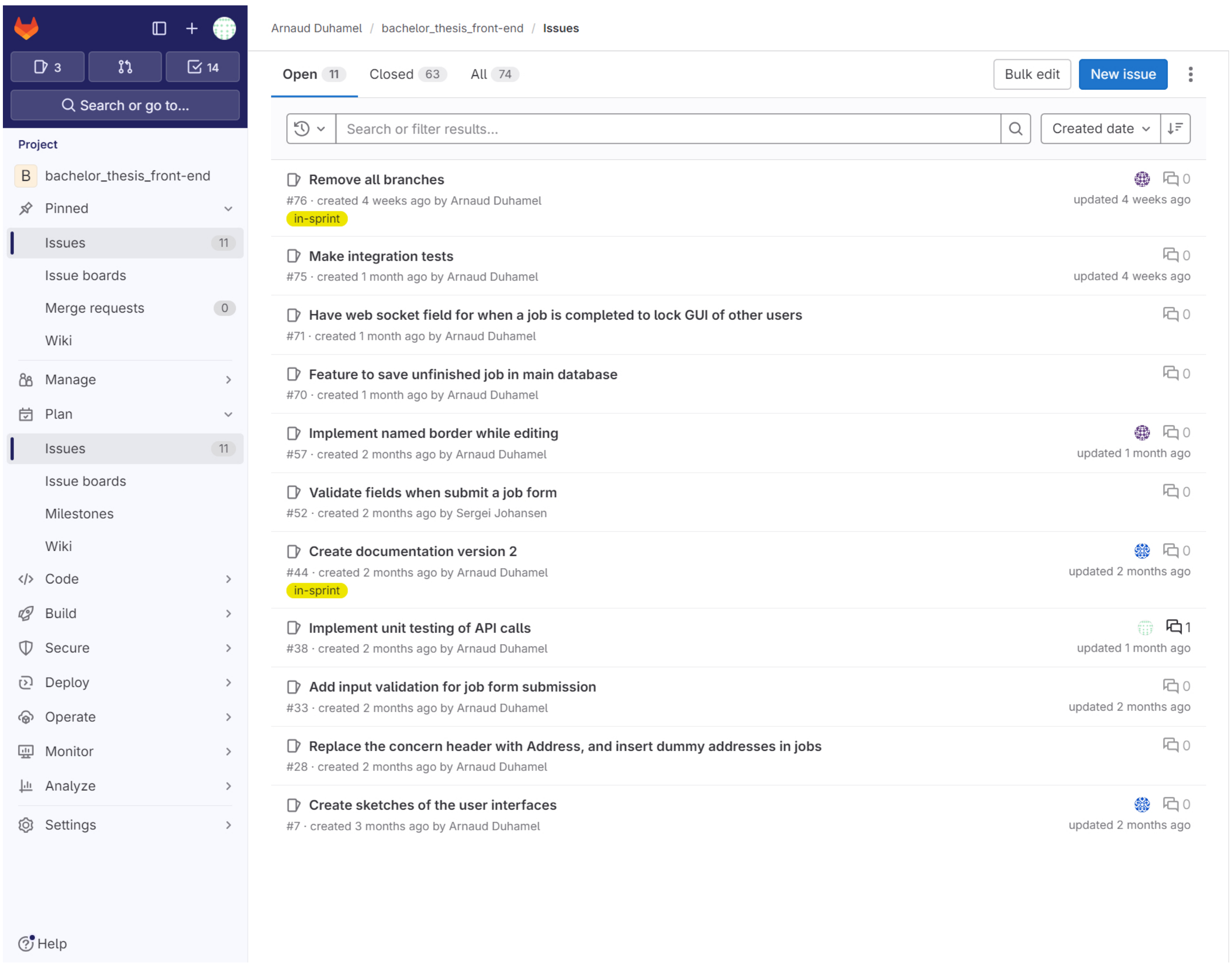
Lage en grunnleggende Web API i .NET Closed 2
#1 · created 4 months ago by Arnaud Duhamel closed 4 months ago

Show 100 items

Help

Appendix E

Open frontend issues at the end of the project



Remove all branches
#76 · created 4 weeks ago by Arnaud Duhamel
in-sprint
0
updated 4 weeks ago

Make integration tests
#75 · created 1 month ago by Arnaud Duhamel
0
updated 4 weeks ago

Have web socket field for when a job is completed to lock GUI of other users
#71 · created 1 month ago by Arnaud Duhamel
0

Feature to save unfinished job in main database
#70 · created 1 month ago by Arnaud Duhamel
0

Implement named border while editing
#57 · created 2 months ago by Arnaud Duhamel
0
updated 1 month ago

Validate fields when submit a job form
#52 · created 2 months ago by Sergei Johansen
0

Create documentation version 2
#44 · created 2 months ago by Arnaud Duhamel
in-sprint
0
updated 2 months ago

Implement unit testing of API calls
#38 · created 2 months ago by Arnaud Duhamel
1
updated 1 month ago

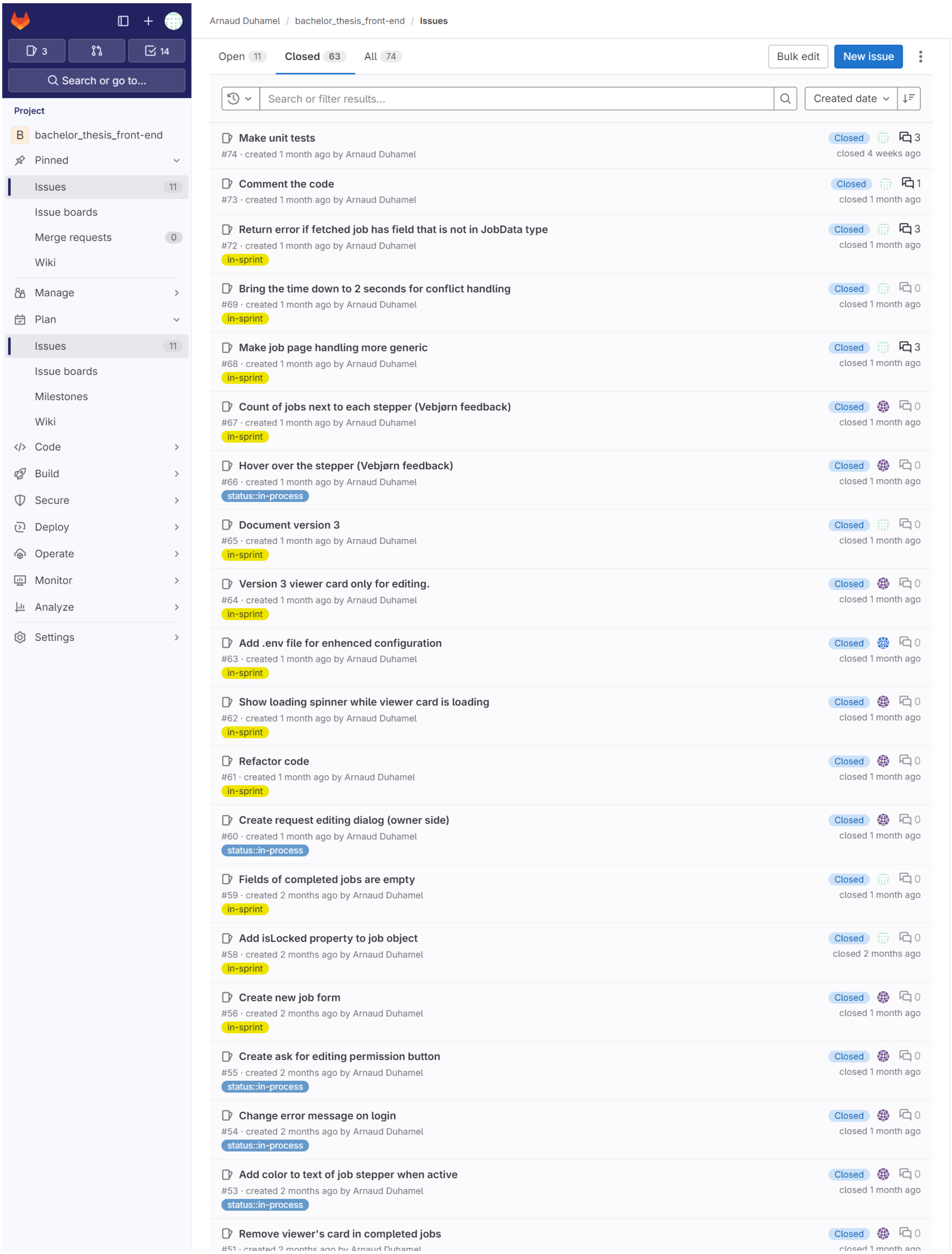
Add input validation for job form submission
#33 · created 2 months ago by Arnaud Duhamel
0
updated 2 months ago

Replace the concern header with Address, and insert dummy addresses in jobs
#28 · created 2 months ago by Arnaud Duhamel
0

Create sketches of the user interfaces
#7 · created 3 months ago by Arnaud Duhamel
0
updated 2 months ago

Appendix F

Closed frontend issues at the end of the project



Make unit tests
#74 · created 1 month ago by Arnaud Duhamel
Closed 3
closed 4 weeks ago

Comment the code
#73 · created 1 month ago by Arnaud Duhamel
Closed 1
closed 1 month ago

Return error if fetched job has field that is not in JobData type
#72 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 3
closed 1 month ago

Bring the time down to 2 seconds for conflict handling
#69 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Make job page handling more generic
#68 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 3
closed 1 month ago

Count of jobs next to each stepper (Vebjørn feedback)
#67 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Hover over the stepper (Vebjørn feedback)
#66 · created 1 month ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Document version 3
#65 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Version 3 viewer card only for editing.
#64 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Add .env file for enhenced configuration
#63 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Show loading spinner while viewer card is loading
#62 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Refactor code
#61 · created 1 month ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Create request editing dialog (owner side)
#60 · created 1 month ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Fields of completed jobs are empty
#59 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Add isLocked property to job object
#58 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

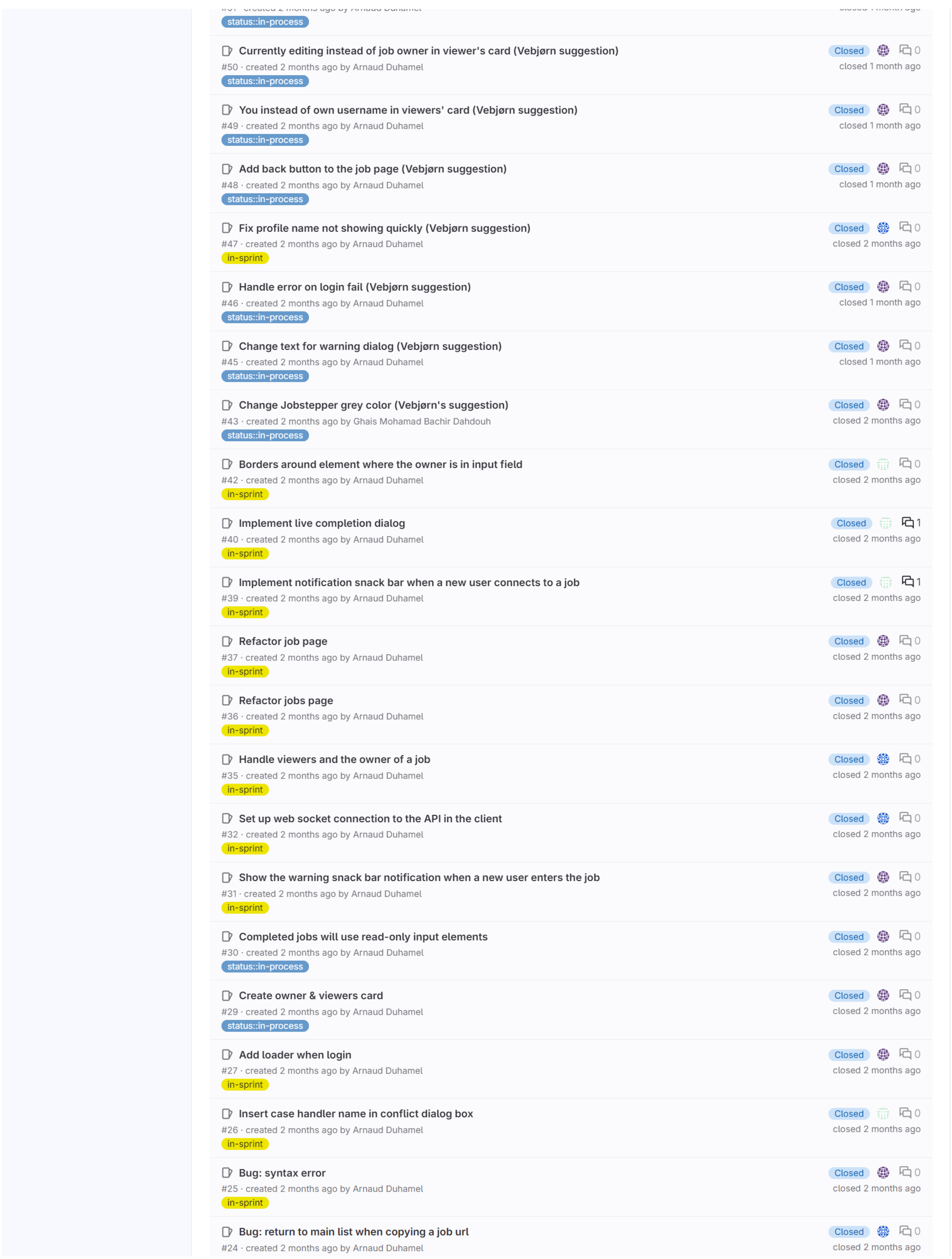
Create new job form
#56 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 1 month ago

Create ask for editing permission button
#55 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Change error message on login
#54 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Add color to text of job stepper when active
#53 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Remove viewer's card in completed jobs
#51 · created 2 months ago by Arnaud Duhamel
Closed 0
closed 1 month ago



status::in-process

Currently editing instead of job owner in viewer's card (Vebjørn suggestion)
#50 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

You instead of own username in viewers' card (Vebjørn suggestion)
#49 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Add back button to the job page (Vebjørn suggestion)
#48 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Fix profile name not showing quickly (Vebjørn suggestion)
#47 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Handle error on login fail (Vebjørn suggestion)
#46 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Change text for warning dialog (Vebjørn suggestion)
#45 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 1 month ago

Change Jobstepper grey color (Vebjørn's suggestion)
#43 · created 2 months ago by Ghais Mohamad Bachir Dahdouh
status::in-process
Closed 0
closed 2 months ago

Borders around element where the owner is in input field
#42 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Implement live completion dialog
#40 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 1
closed 2 months ago

Implement notification snack bar when a new user connects to a job
#39 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 1
closed 2 months ago

Refactor job page
#37 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Refactor jobs page
#36 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Handle viewers and the owner of a job
#35 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Set up web socket connection to the API in the client
#32 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Show the warning snack bar notification when a new user enters the job
#31 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Completed jobs will use read-only input elements
#30 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 2 months ago

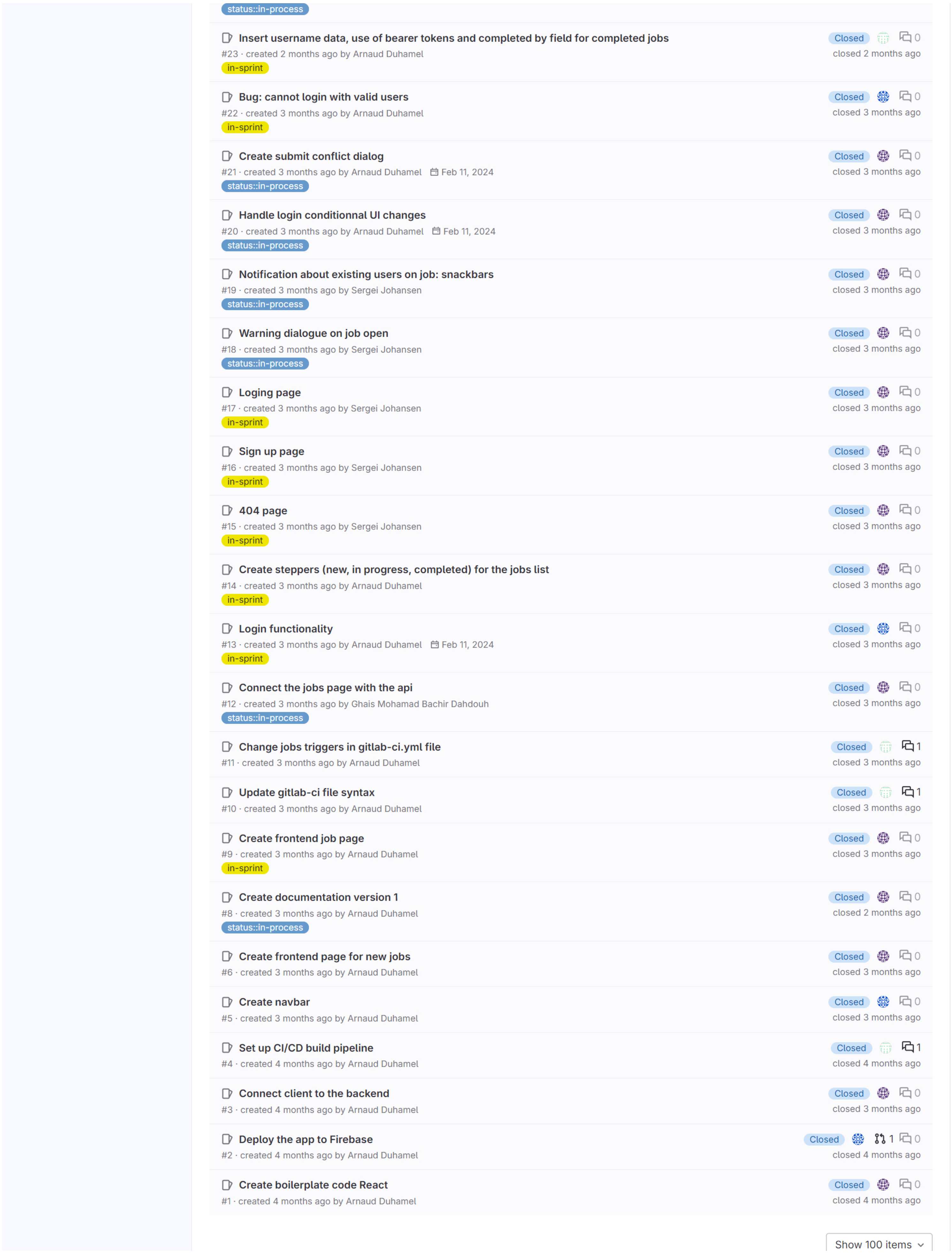
Create owner & viewers card
#29 · created 2 months ago by Arnaud Duhamel
status::in-process
Closed 0
closed 2 months ago

Add loader when login
#27 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Insert case handler name in conflict dialog box
#26 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Bug: syntax error
#25 · created 2 months ago by Arnaud Duhamel
in-sprint
Closed 0
closed 2 months ago

Bug: return to main list when copying a job url
#24 · created 2 months ago by Arnaud Duhamel
Closed 0
closed 2 months ago



status::in-process

Insert username data, use of bearer tokens and completed by field for completed jobs — Closed · 0
#23 · created 2 months ago by Arnaud Duhamel — closed 2 months ago
in-sprint

Bug: cannot login with valid users — Closed · 0
#22 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Create submit conflict dialog — Closed · 0
#21 · created 3 months ago by Arnaud Duhamel · Feb 11, 2024 — closed 3 months ago
status::in-process

Handle login conditionnal UI changes — Closed · 0
#20 · created 3 months ago by Arnaud Duhamel · Feb 11, 2024 — closed 3 months ago
status::in-process

Notification about existing users on job: snackbars — Closed · 0
#19 · created 3 months ago by Sergei Johansen — closed 3 months ago
status::in-process

Warning dialogue on job open — Closed · 0
#18 · created 3 months ago by Sergei Johansen — closed 3 months ago
status::in-process

Loging page — Closed · 0
#17 · created 3 months ago by Sergei Johansen — closed 3 months ago
in-sprint

Sign up page — Closed · 0
#16 · created 3 months ago by Sergei Johansen — closed 3 months ago
in-sprint

404 page — Closed · 0
#15 · created 3 months ago by Sergei Johansen — closed 3 months ago
in-sprint

Create steppers (new, in progress, completed) for the jobs list — Closed · 0
#14 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Login functionality — Closed · 0
#13 · created 3 months ago by Arnaud Duhamel · Feb 11, 2024 — closed 3 months ago
in-sprint

Connect the jobs page with the api — Closed · 0
#12 · created 3 months ago by Ghais Mohamad Bachir Dahdouh — closed 3 months ago
status::in-process

Change jobs triggers in gitlab-ci.yml file — Closed · 1
#11 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Update gitlab-ci file syntax — Closed · 1
#10 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Create frontend job page — Closed · 0
#9 · created 3 months ago by Arnaud Duhamel — closed 3 months ago
in-sprint

Create documentation version 1 — Closed · 0
#8 · created 3 months ago by Arnaud Duhamel — closed 2 months ago
status::in-process

Create frontend page for new jobs — Closed · 0
#6 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Create navbar — Closed · 0
#5 · created 3 months ago by Arnaud Duhamel — closed 3 months ago

Set up CI/CD build pipeline — Closed · 1
#4 · created 4 months ago by Arnaud Duhamel — closed 4 months ago

Connect client to the backend — Closed · 0
#3 · created 4 months ago by Arnaud Duhamel — closed 3 months ago

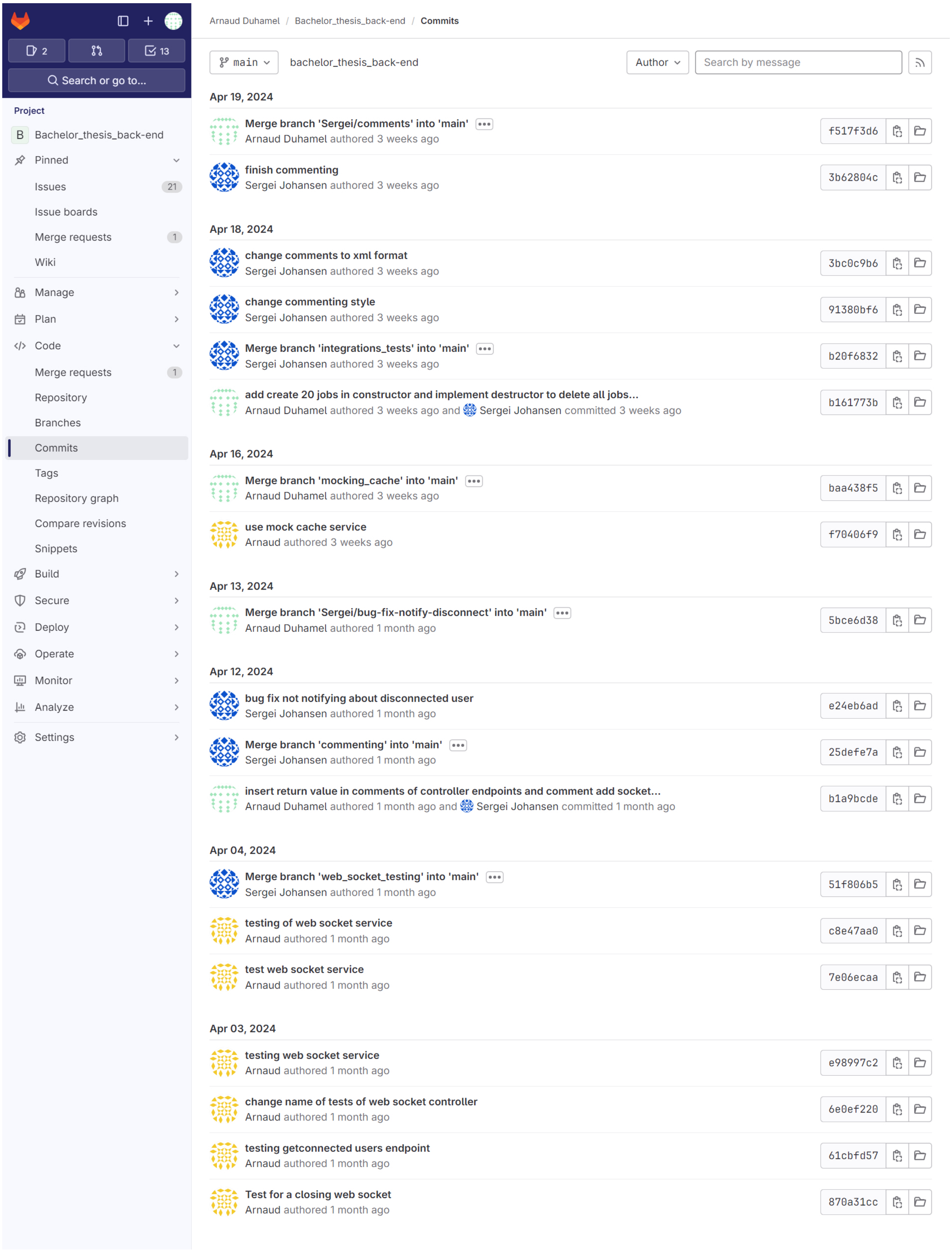
Deploy the app to Firebase — Closed · 1 · 0
#2 · created 4 months ago by Arnaud Duhamel — closed 4 months ago

Create boilerplate code React — Closed · 0
#1 · created 4 months ago by Arnaud Duhamel — closed 4 months ago

Show 100 items

Appendix G

Commit history of the backend repository



bachelor_thesis_back-end

Apr 19, 2024

Merge branch 'Sergei/comments' into 'main'
Arnaud Duhamel authored 3 weeks ago — f517f3d6

finish commenting
Sergei Johansen authored 3 weeks ago — 3b62804c

Apr 18, 2024

change comments to xml format
Sergei Johansen authored 3 weeks ago — 3bc0c9b6

change commenting style
Sergei Johansen authored 3 weeks ago — 91380bf6

Merge branch 'integrations_tests' into 'main'
Sergei Johansen authored 3 weeks ago — b20f6832

add create 20 jobs in constructor and implement destructor to delete all jobs...
Arnaud Duhamel authored 3 weeks ago and Sergei Johansen committed 3 weeks ago — b161773b

Apr 16, 2024

Merge branch 'mocking_cache' into 'main'
Arnaud Duhamel authored 3 weeks ago — baa438f5

use mock cache service
Arnaud authored 3 weeks ago — f70406f9

Apr 13, 2024

Merge branch 'Sergei/bug-fix-notify-disconnect' into 'main'
Arnaud Duhamel authored 1 month ago — 5bce6d38

Apr 12, 2024

bug fix not notifying about disconnected user
Sergei Johansen authored 1 month ago — e24eb6ad

Merge branch 'commenting' into 'main'
Sergei Johansen authored 1 month ago — 25defe7a

insert return value in comments of controller endpoints and comment add socket...
Arnaud Duhamel authored 1 month ago and Sergei Johansen committed 1 month ago — b1a9bcde

Apr 04, 2024

Merge branch 'web_socket_testing' into 'main'
Sergei Johansen authored 1 month ago — 51f806b5

testing of web socket service
Arnaud authored 1 month ago — c8e47aa0

test web socket service
Arnaud authored 1 month ago — 7e06ecaa

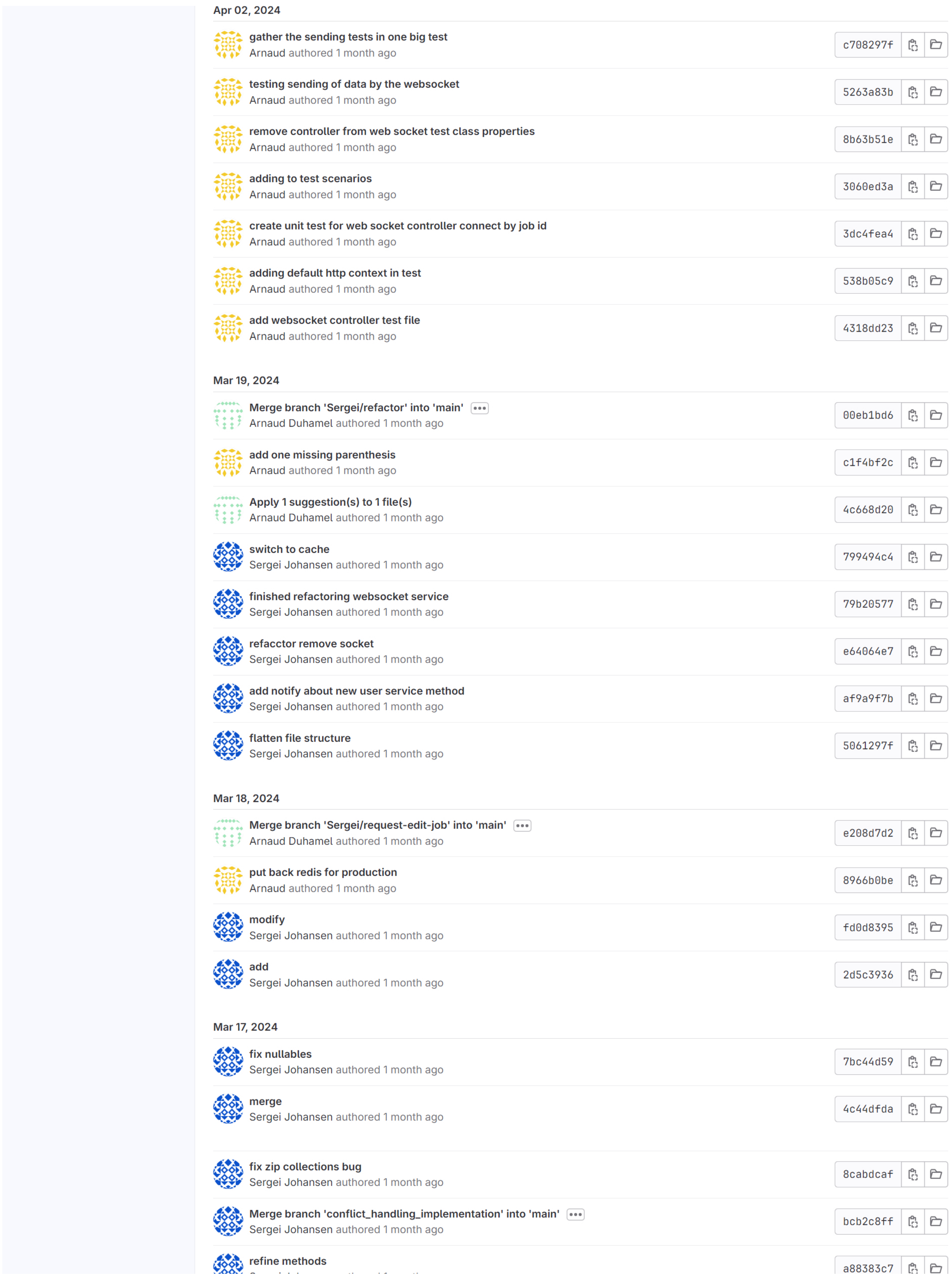
Apr 03, 2024

testing web socket service
Arnaud authored 1 month ago — e98997c2

change name of tests of web socket controller
Arnaud authored 1 month ago — 6e0ef220

testing getconnected users endpoint
Arnaud authored 1 month ago — 61cbfd57

Test for a closing web socket
Arnaud authored 1 month ago — 870a31cc



Apr 02, 2024

gather the sending tests in one big test
Arnaud authored 1 month ago
c708297f

testing sending of data by the websocket
Arnaud authored 1 month ago
5263a83b

remove controller from web socket test class properties
Arnaud authored 1 month ago
8b63b51e

adding to test scenarios
Arnaud authored 1 month ago
3060ed3a

create unit test for web socket controller connect by job id
Arnaud authored 1 month ago
3dc4fea4

adding default http context in test
Arnaud authored 1 month ago
538b05c9

add websocket controller test file
Arnaud authored 1 month ago
4318dd23

Mar 19, 2024

Merge branch 'Sergei/refactor' into 'main'
Arnaud Duhamel authored 1 month ago
00eb1bd6

add one missing parenthesis
Arnaud authored 1 month ago
c1f4bf2c

Apply 1 suggestion(s) to 1 file(s)
Arnaud Duhamel authored 1 month ago
4c668d20

switch to cache
Sergei Johansen authored 1 month ago
799494c4

finished refactoring websocket service
Sergei Johansen authored 1 month ago
79b20577

refacctor remove socket
Sergei Johansen authored 1 month ago
e64064e7

add notify about new user service method
Sergei Johansen authored 1 month ago
af9a9f7b

flatten file structure
Sergei Johansen authored 1 month ago
5061297f

Mar 18, 2024

Merge branch 'Sergei/request-edit-job' into 'main'
Arnaud Duhamel authored 1 month ago
e208d7d2

put back redis for production
Arnaud authored 1 month ago
8966b0be

modify
Sergei Johansen authored 1 month ago
fd0d8395

add
Sergei Johansen authored 1 month ago
2d5c3936

Mar 17, 2024

fix nullables
Sergei Johansen authored 1 month ago
7bc44d59

merge
Sergei Johansen authored 1 month ago
4c44dfda

fix zip collections bug
Sergei Johansen authored 1 month ago
8cabdcaf

Merge branch 'conflict_handling_implementation' into 'main'
Sergei Johansen authored 1 month ago
bcb2c8ff

refine methods
a88383c7

Sergei Johansen authored 1 month ago

update broadcast request edit perm method
Sergei Johansen authored 1 month ago `b4263bc0`

add broadcast about new job editor
Sergei Johansen authored 1 month ago `38e9f8a8`

putting the controllers get jobs endpoints to closed
Arnaud authored 1 month ago `d5bf8eb7`

merge main in the branch
Arnaud authored 1 month ago `7bd65043`

changing staring string to be read into a yjs doc
Arnaud authored 1 month ago `5ddb2084`

Mar 16, 2024

change permission grant logic on connect
Sergei Johansen authored 1 month ago `bb630a5c`

add request edit permission method
Sergei Johansen authored 1 month ago `f605040f`

Merge branch 'Sergei/enable-create-job' into 'main' •••
Sergei Johansen authored 1 month ago `fff210ac`

meter number to 0 and remove writeline in create job
Arnaud authored 1 month ago `cc76eba6`

change meter number to string and handle ydoc
Arnaud authored 1 month ago `bb78bffb`

add null check due to errors
Sergei Johansen authored 1 month ago `e96ff642`

add temporary cache mock, because of cache timeout errors
Sergei Johansen authored 1 month ago `d6a0fc67`

enable create job
Sergei Johansen authored 1 month ago `81b9d3c5`

Mar 12, 2024

Merge branch 'integrate_caching' into 'main' •••
Sergei Johansen authored 2 months ago `644b61cd`

Mar 11, 2024

finishing implementing caching
Arnaud authored 2 months ago `9fc721eb`

change tests to add cache service to jobs controller
Arnaud authored 2 months ago `4c761260`

inserting caching of water meter elements
Arnaud authored 2 months ago `5e8cdae6`

making changes to the constructor of the cache service
Arnaud authored 2 months ago `3fe51cf7`

Mar 10, 2024

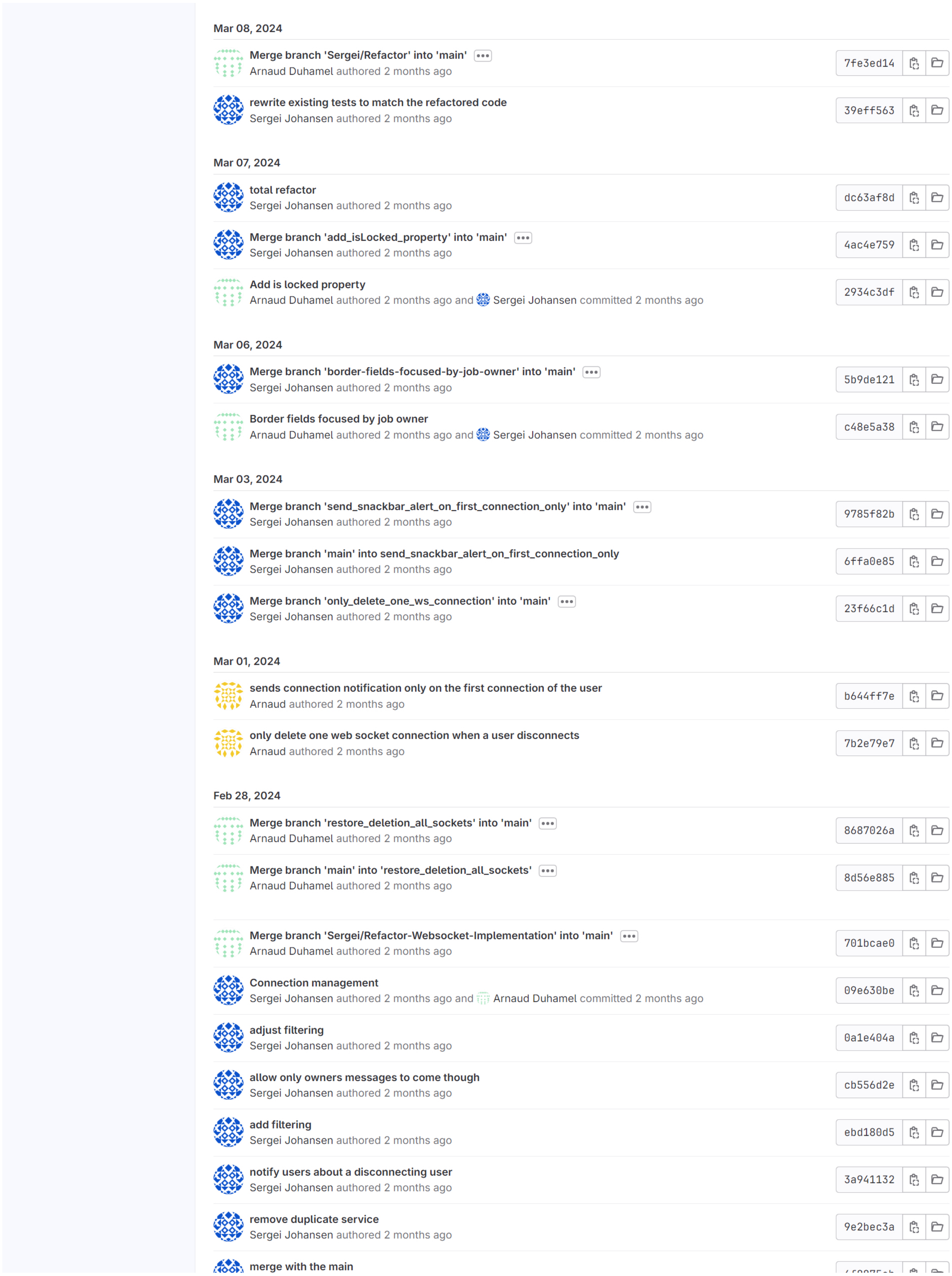
add delete all method
Arnaud authored 2 months ago `60e2d71d`

change to the post endpoint of the cachecontroller
Arnaud authored 2 months ago `72992a82`

finish cace controller
Arnaud authored 2 months ago `80a48bc6`

insert cache service
Arnaud authored 2 months ago `19c79f7b`

adding relevant packages
Arnaud authored 2 months ago `d08dc847`



Mar 08, 2024

- **Merge branch 'Sergei/Refactor' into 'main'** ••• — Arnaud Duhamel authored 2 months ago — `7fe3ed14`
- **rewrite existing tests to match the refactored code** — Sergei Johansen authored 2 months ago — `39eff563`

Mar 07, 2024

- **total refactor** — Sergei Johansen authored 2 months ago — `dc63af8d`
- **Merge branch 'add_isLocked_property' into 'main'** ••• — Sergei Johansen authored 2 months ago — `4ac4e759`
- **Add is locked property** — Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — `2934c3df`

Mar 06, 2024

- **Merge branch 'border-fields-focused-by-job-owner' into 'main'** ••• — Sergei Johansen authored 2 months ago — `5b9de121`
- **Border fields focused by job owner** — Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — `c48e5a38`

Mar 03, 2024

- **Merge branch 'send_snackbar_alert_on_first_connection_only' into 'main'** ••• — Sergei Johansen authored 2 months ago — `9785f82b`
- **Merge branch 'main' into send_snackbar_alert_on_first_connection_only** — Sergei Johansen authored 2 months ago — `6ffa0e85`
- **Merge branch 'only_delete_one_ws_connection' into 'main'** ••• — Sergei Johansen authored 2 months ago — `23f66c1d`

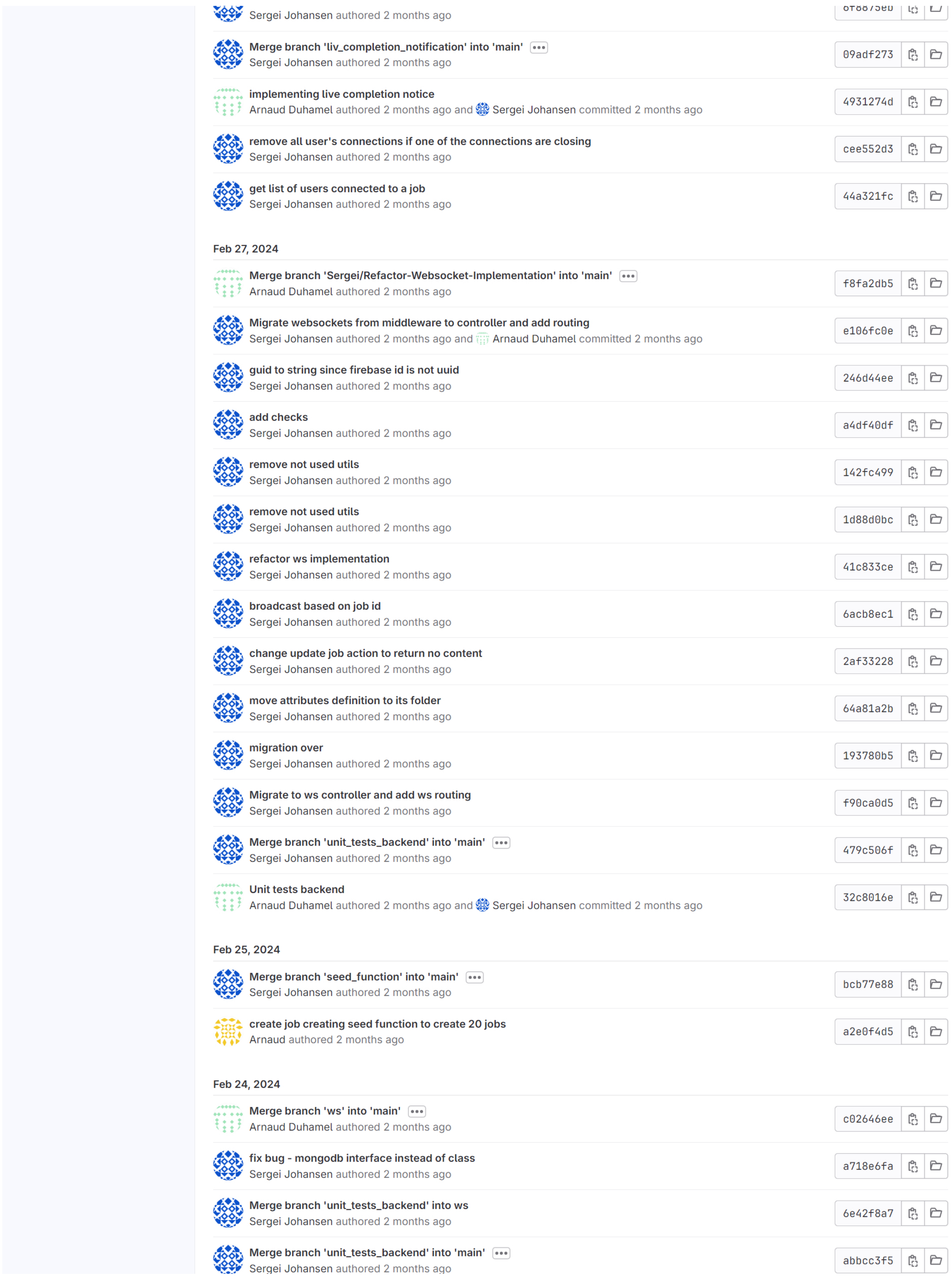
Mar 01, 2024

- **sends connection notification only on the first connection of the user** — Arnaud authored 2 months ago — `b644ff7e`
- **only delete one web socket connection when a user disconnects** — Arnaud authored 2 months ago — `7b2e79e7`

Feb 28, 2024

- **Merge branch 'restore_deletion_all_sockets' into 'main'** ••• — Arnaud Duhamel authored 2 months ago — `8687026a`
- **Merge branch 'main' into 'restore_deletion_all_sockets'** ••• — Arnaud Duhamel authored 2 months ago — `8d56e885`

- **Merge branch 'Sergei/Refactor-Websocket-Implementation' into 'main'** ••• — Arnaud Duhamel authored 2 months ago — `701bcae0`
- **Connection management** — Sergei Johansen authored 2 months ago and Arnaud Duhamel committed 2 months ago — `09e630be`
- **adjust filtering** — Sergei Johansen authored 2 months ago — `0a1e404a`
- **allow only owners messages to come though** — Sergei Johansen authored 2 months ago — `cb556d2e`
- **add filtering** — Sergei Johansen authored 2 months ago — `ebd180d5`
- **notify users about a disconnecting user** — Sergei Johansen authored 2 months ago — `3a941132`
- **remove duplicate service** — Sergei Johansen authored 2 months ago — `9e2bec3a`
- **merge with the main**



Sergei Johansen authored 2 months ago

Merge branch 'liv_completion_notification' into 'main' •••
Sergei Johansen authored 2 months ago — `09adf273`

implementing live completion notice
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — `4931274d`

remove all user's connections if one of the connections are closing
Sergei Johansen authored 2 months ago — `cee552d3`

get list of users connected to a job
Sergei Johansen authored 2 months ago — `44a321fc`

Feb 27, 2024

Merge branch 'Sergei/Refactor-Websocket-Implementation' into 'main' •••
Arnaud Duhamel authored 2 months ago — `f8fa2db5`

Migrate websockets from middleware to controller and add routing
Sergei Johansen authored 2 months ago and Arnaud Duhamel committed 2 months ago — `e106fc0e`

guid to string since firebase id is not uuid
Sergei Johansen authored 2 months ago — `246d44ee`

add checks
Sergei Johansen authored 2 months ago — `a4df40df`

remove not used utils
Sergei Johansen authored 2 months ago — `142fc499`

remove not used utils
Sergei Johansen authored 2 months ago — `1d88d0bc`

refactor ws implementation
Sergei Johansen authored 2 months ago — `41c833ce`

broadcast based on job id
Sergei Johansen authored 2 months ago — `6acb8ec1`

change update job action to return no content
Sergei Johansen authored 2 months ago — `2af33228`

move attributes definition to its folder
Sergei Johansen authored 2 months ago — `64a81a2b`

migration over
Sergei Johansen authored 2 months ago — `193780b5`

Migrate to ws controller and add ws routing
Sergei Johansen authored 2 months ago — `f90ca0d5`

Merge branch 'unit_tests_backend' into 'main' •••
Sergei Johansen authored 2 months ago — `479c506f`

Unit tests backend
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — `32c8016e`

Feb 25, 2024

Merge branch 'seed_function' into 'main' •••
Sergei Johansen authored 2 months ago — `bcb77e88`

create job creating seed function to create 20 jobs
Arnaud authored 2 months ago — `a2e0f4d5`

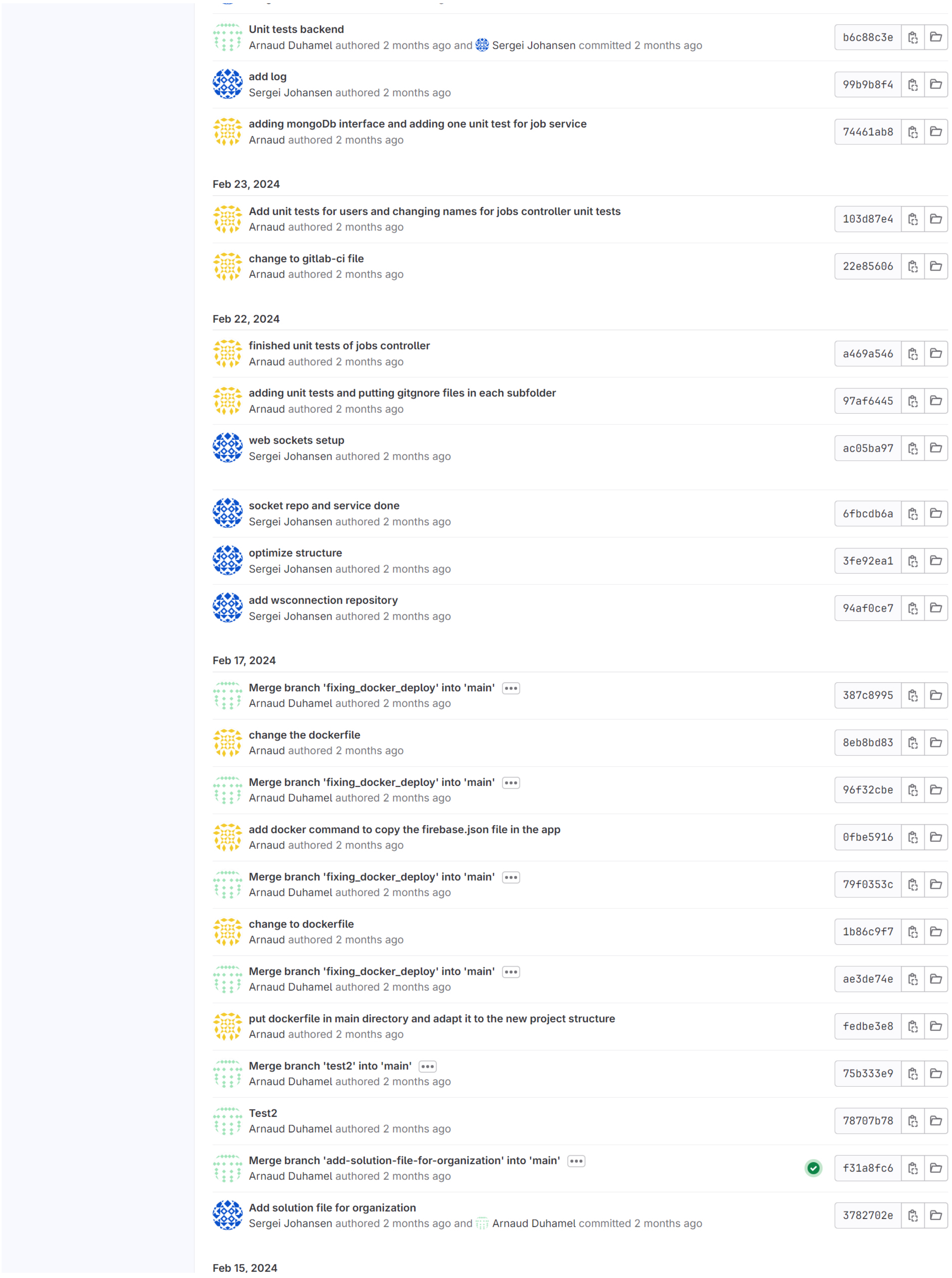
Feb 24, 2024

Merge branch 'ws' into 'main' •••
Arnaud Duhamel authored 2 months ago — `c02646ee`

fix bug - mongodb interface instead of class
Sergei Johansen authored 2 months ago — `a718e6fa`

Merge branch 'unit_tests_backend' into ws
Sergei Johansen authored 2 months ago — `6e42f8a7`

Merge branch 'unit_tests_backend' into 'main' •••
Sergei Johansen authored 2 months ago — `abbcc3f5`



Unit tests backend
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — `b6c88c3e`

add log
Sergei Johansen authored 2 months ago — `99b9b8f4`

adding mongoDb interface and adding one unit test for job service
Arnaud authored 2 months ago — `74461ab8`

Feb 23, 2024

Add unit tests for users and changing names for jobs controller unit tests
Arnaud authored 2 months ago — `103d87e4`

change to gitlab-ci file
Arnaud authored 2 months ago — `22e85606`

Feb 22, 2024

finished unit tests of jobs controller
Arnaud authored 2 months ago — `a469a546`

adding unit tests and putting gitgnore files in each subfolder
Arnaud authored 2 months ago — `97af6445`

web sockets setup
Sergei Johansen authored 2 months ago — `ac05ba97`

socket repo and service done
Sergei Johansen authored 2 months ago — `6fbcdb6a`

optimize structure
Sergei Johansen authored 2 months ago — `3fe92ea1`

add wsconnection repository
Sergei Johansen authored 2 months ago — `94af0ce7`

Feb 17, 2024

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — `387c8995`

change the dockerfile
Arnaud authored 2 months ago — `8eb8bd83`

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — `96f32cbe`

add docker command to copy the firebase.json file in the app
Arnaud authored 2 months ago — `0fbe5916`

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — `79f0353c`

change to dockerfile
Arnaud authored 2 months ago — `1b86c9f7`

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — `ae3de74e`

put dockerfile in main directory and adapt it to the new project structure
Arnaud authored 2 months ago — `fedbe3e8`

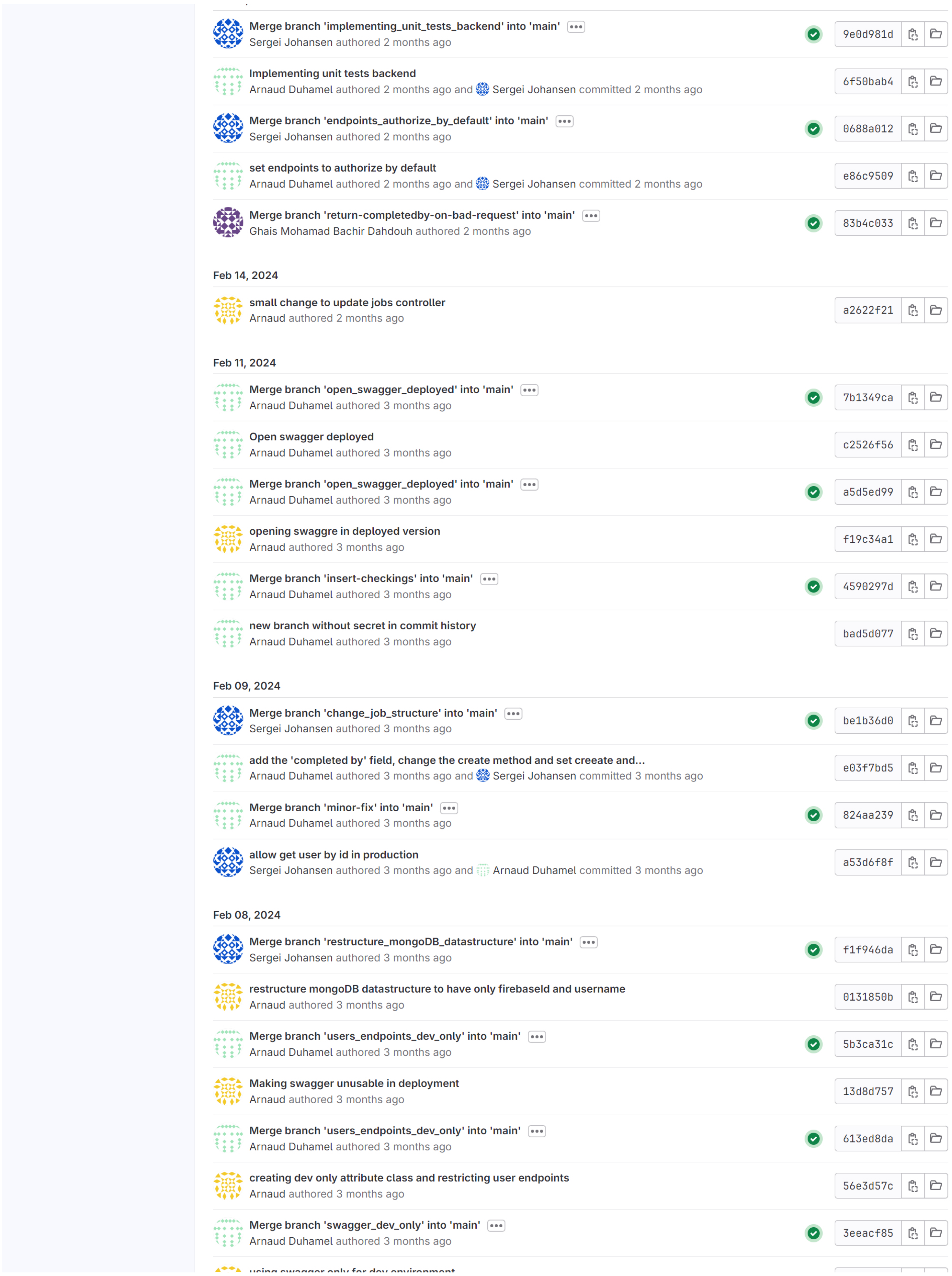
Merge branch 'test2' into 'main' •••
Arnaud Duhamel authored 2 months ago — `75b333e9`

Test2
Arnaud Duhamel authored 2 months ago — `78707b78`

Merge branch 'add-solution-file-for-organization' into 'main' •••
Arnaud Duhamel authored 2 months ago — ✓ `f31a8fc6`

Add solution file for organization
Sergei Johansen authored 2 months ago and Arnaud Duhamel committed 2 months ago — `3782702e`

Feb 15, 2024



Merge branch 'implementing_unit_tests_backend' into 'main' ••• 9e0d981d
Sergei Johansen authored 2 months ago

Implementing unit tests backend 6f50bab4
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago

Merge branch 'endpoints_authorize_by_default' into 'main' ••• 0688a012
Sergei Johansen authored 2 months ago

set endpoints to authorize by default e86c9509
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago

Merge branch 'return-completedby-on-bad-request' into 'main' ••• 83b4c033
Ghais Mohamad Bachir Dahdouh authored 2 months ago

Feb 14, 2024

small change to update jobs controller a2622f21
Arnaud authored 2 months ago

Feb 11, 2024

Merge branch 'open_swagger_deployed' into 'main' ••• 7b1349ca
Arnaud Duhamel authored 3 months ago

Open swagger deployed c2526f56
Arnaud Duhamel authored 3 months ago

Merge branch 'open_swagger_deployed' into 'main' ••• a5d5ed99
Arnaud Duhamel authored 3 months ago

opening swaggre in deployed version f19c34a1
Arnaud authored 3 months ago

Merge branch 'insert-checkings' into 'main' ••• 4590297d
Arnaud Duhamel authored 3 months ago

new branch without secret in commit history bad5d077
Arnaud Duhamel authored 3 months ago

Feb 09, 2024

Merge branch 'change_job_structure' into 'main' ••• be1b36d0
Sergei Johansen authored 3 months ago

add the 'completed by' field, change the create method and set creeate and... e03f7bd5
Arnaud Duhamel authored 3 months ago and Sergei Johansen committed 3 months ago

Merge branch 'minor-fix' into 'main' ••• 824aa239
Arnaud Duhamel authored 3 months ago

allow get user by id in production a53d6f8f
Sergei Johansen authored 3 months ago and Arnaud Duhamel committed 3 months ago

Feb 08, 2024

Merge branch 'restructure_mongoDB_datastructure' into 'main' ••• f1f946da
Sergei Johansen authored 3 months ago

restructure mongoDB datastructure to have only firebaseId and username 0131850b
Arnaud authored 3 months ago

Merge branch 'users_endpoints_dev_only' into 'main' ••• 5b3ca31c
Arnaud Duhamel authored 3 months ago

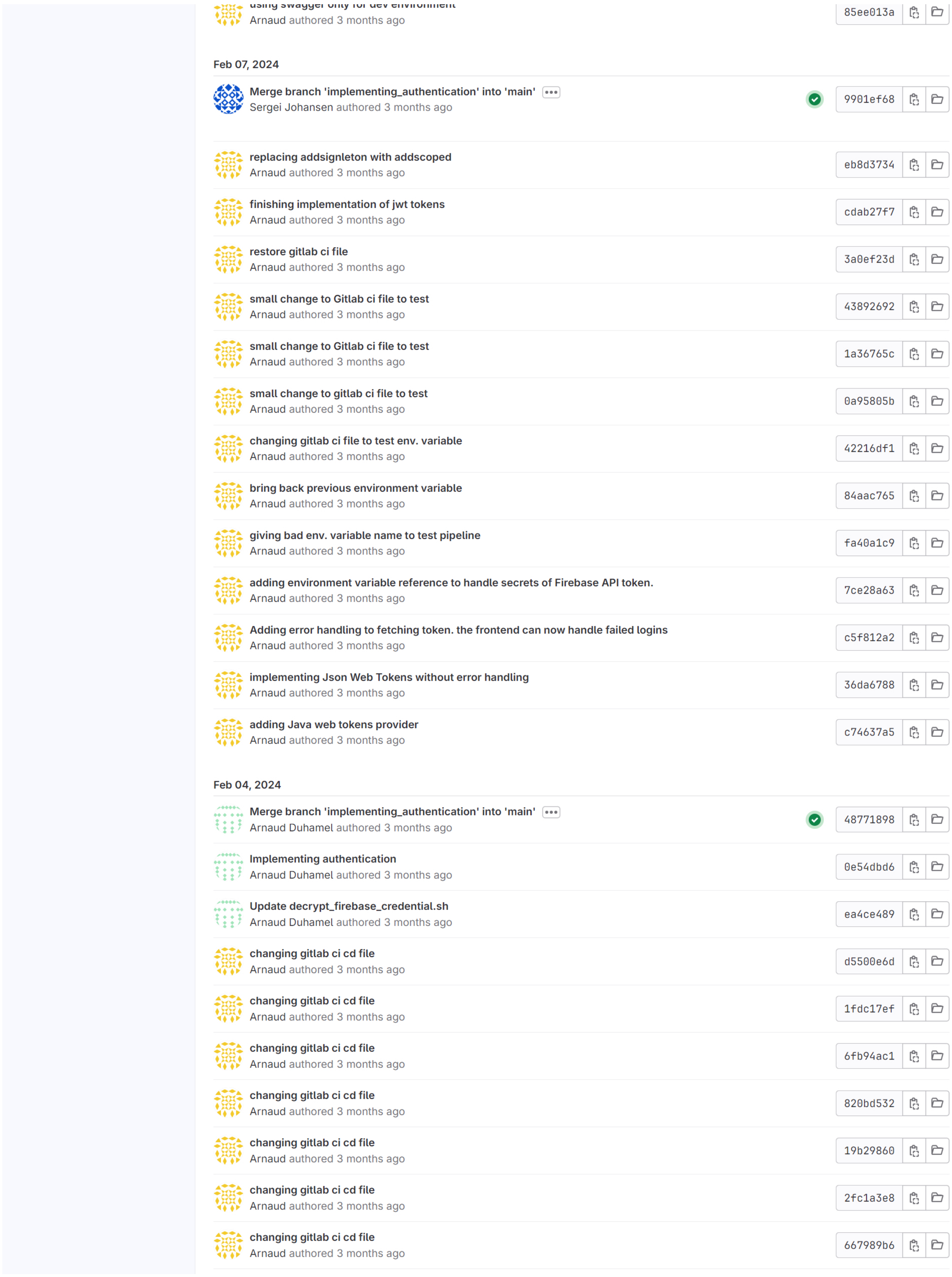
Making swagger unusable in deployment 13d8d757
Arnaud authored 3 months ago

Merge branch 'users_endpoints_dev_only' into 'main' ••• 613ed8da
Arnaud Duhamel authored 3 months ago

creating dev only attribute class and restricting user endpoints 56e3d57c
Arnaud authored 3 months ago

Merge branch 'swagger_dev_only' into 'main' ••• 3eeacf85
Arnaud Duhamel authored 3 months ago

using swagger only for dev environment



using swagger only for dev environment
Arnaud authored 3 months ago — 85ee013a

Feb 07, 2024

Merge branch 'implementing_authentication' into 'main' •••
Sergei Johansen authored 3 months ago — ✓ — 9901ef68

replacing addsignleton with addscoped
Arnaud authored 3 months ago — eb8d3734

finishing implementation of jwt tokens
Arnaud authored 3 months ago — cdab27f7

restore gitlab ci file
Arnaud authored 3 months ago — 3a0ef23d

small change to Gitlab ci file to test
Arnaud authored 3 months ago — 43892692

small change to Gitlab ci file to test
Arnaud authored 3 months ago — 1a36765c

small change to gitlab ci file to test
Arnaud authored 3 months ago — 0a95805b

changing gitlab ci file to test env. variable
Arnaud authored 3 months ago — 42216df1

bring back previous environment variable
Arnaud authored 3 months ago — 84aac765

giving bad env. variable name to test pipeline
Arnaud authored 3 months ago — fa40a1c9

adding environment variable reference to handle secrets of Firebase API token.
Arnaud authored 3 months ago — 7ce28a63

Adding error handling to fetching token. the frontend can now handle failed logins
Arnaud authored 3 months ago — c5f812a2

implementing Json Web Tokens without error handling
Arnaud authored 3 months ago — 36da6788

adding Java web tokens provider
Arnaud authored 3 months ago — c74637a5

Feb 04, 2024

Merge branch 'implementing_authentication' into 'main' •••
Arnaud Duhamel authored 3 months ago — ✓ — 48771898

Implementing authentication
Arnaud Duhamel authored 3 months ago — 0e54dbd6

Update decrypt_firebase_credential.sh
Arnaud Duhamel authored 3 months ago — ea4ce489

changing gitlab ci cd file
Arnaud authored 3 months ago — d5500e6d

changing gitlab ci cd file
Arnaud authored 3 months ago — 1fdc17ef

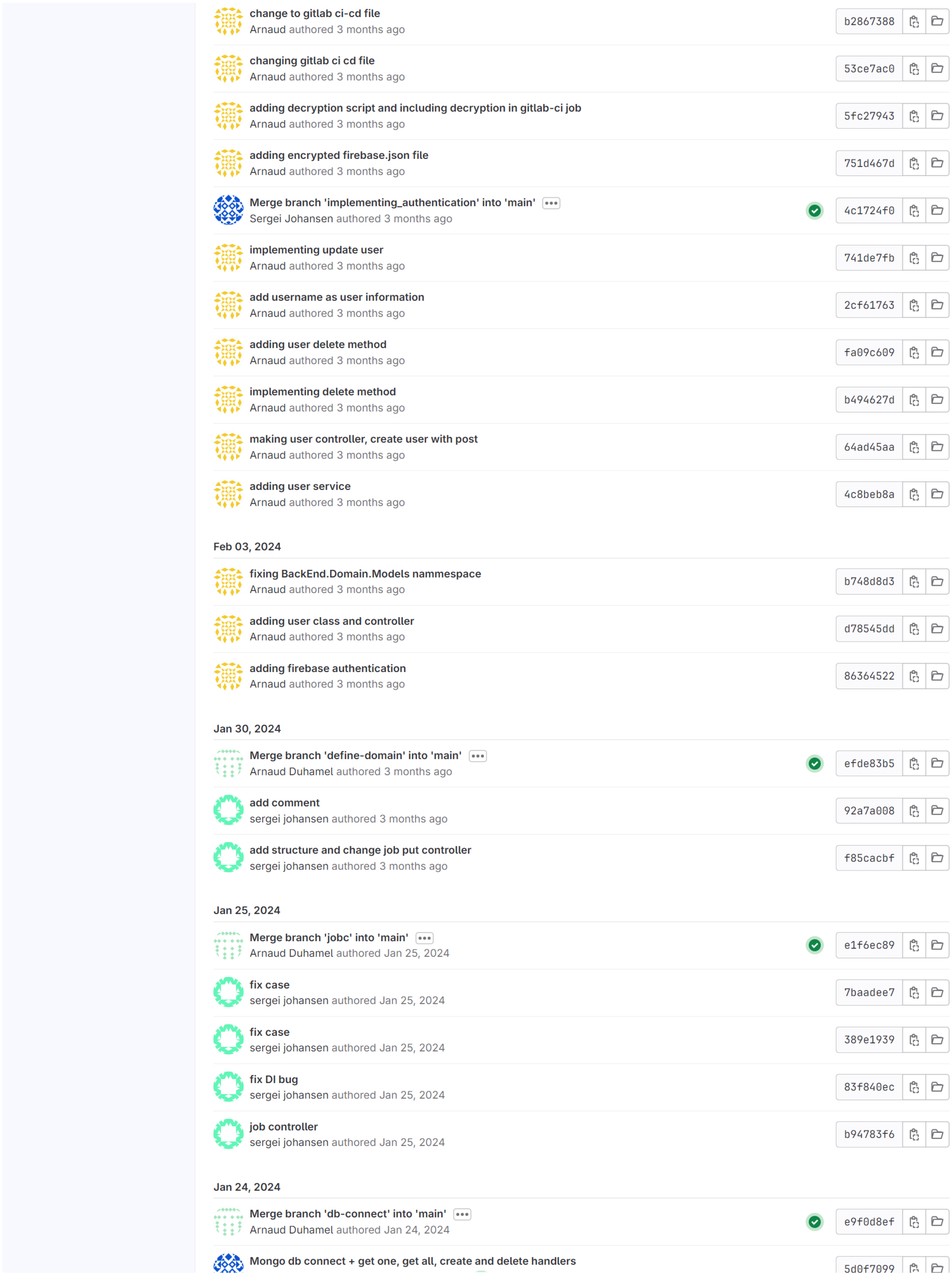
changing gitlab ci cd file
Arnaud authored 3 months ago — 6fb94ac1

changing gitlab ci cd file
Arnaud authored 3 months ago — 820bd532

changing gitlab ci cd file
Arnaud authored 3 months ago — 19b29860

changing gitlab ci cd file
Arnaud authored 3 months ago — 2fc1a3e8

changing gitlab ci cd file
Arnaud authored 3 months ago — 667989b6



change to gitlab ci-cd file
Arnaud authored 3 months ago — b2867388

changing gitlab ci cd file
Arnaud authored 3 months ago — 53ce7ac0

adding decryption script and including decryption in gitlab-ci job
Arnaud authored 3 months ago — 5fc27943

adding encrypted firebase.json file
Arnaud authored 3 months ago — 751d467d

Merge branch 'implementing_authentication' into 'main' •••
Sergei Johansen authored 3 months ago — ✓ 4c1724f0

implementing update user
Arnaud authored 3 months ago — 741de7fb

add username as user information
Arnaud authored 3 months ago — 2cf61763

adding user delete method
Arnaud authored 3 months ago — fa09c609

implementing delete method
Arnaud authored 3 months ago — b494627d

making user controller, create user with post
Arnaud authored 3 months ago — 64ad45aa

adding user service
Arnaud authored 3 months ago — 4c8beb8a

Feb 03, 2024

fixing BackEnd.Domain.Models nammespace
Arnaud authored 3 months ago — b748d8d3

adding user class and controller
Arnaud authored 3 months ago — d78545dd

adding firebase authentication
Arnaud authored 3 months ago — 86364522

Jan 30, 2024

Merge branch 'define-domain' into 'main' •••
Arnaud Duhamel authored 3 months ago — ✓ efde83b5

add comment
sergei johansen authored 3 months ago — 92a7a008

add structure and change job put controller
sergei johansen authored 3 months ago — f85cacbf

Jan 25, 2024

Merge branch 'jobc' into 'main' •••
Arnaud Duhamel authored Jan 25, 2024 — ✓ e1f6ec89

fix case
sergei johansen authored Jan 25, 2024 — 7baadee7

fix case
sergei johansen authored Jan 25, 2024 — 389e1939

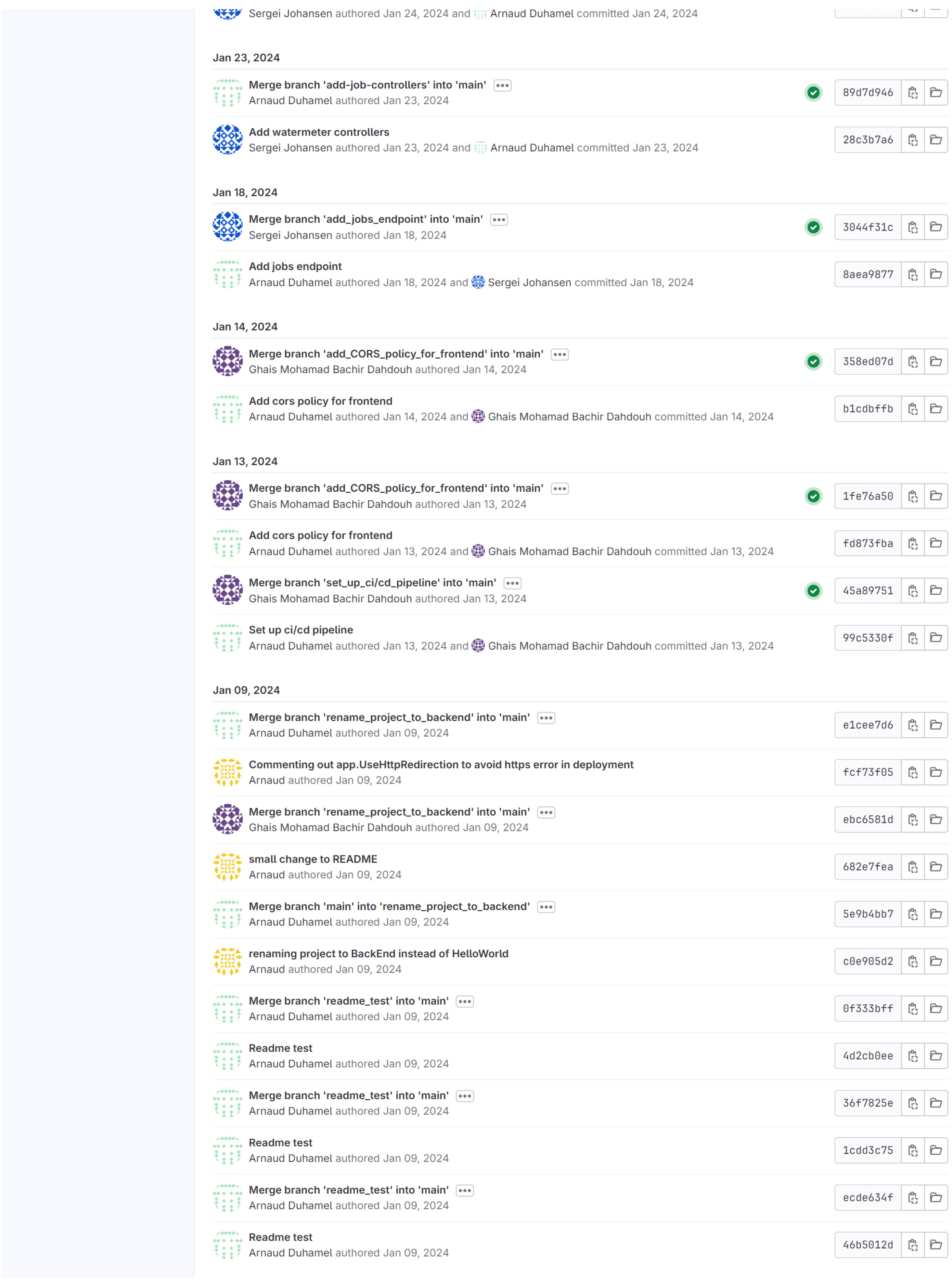
fix DI bug
sergei johansen authored Jan 25, 2024 — 83f840ec

job controller
sergei johansen authored Jan 25, 2024 — b94783f6

Jan 24, 2024

Merge branch 'db-connect' into 'main' •••
Arnaud Duhamel authored Jan 24, 2024 — ✓ e9f0d8ef

Mongo db connect + get one, get all, create and delete handlers
5d0f7099



Sergei Johansen authored Jan 24, 2024 and Arnaud Duhamel committed Jan 24, 2024

Jan 23, 2024

Merge branch 'add-job-controllers' into 'main' ••• ✓ 89d7d946
Arnaud Duhamel authored Jan 23, 2024

Add watermeter controllers 28c3b7a6
Sergei Johansen authored Jan 23, 2024 and Arnaud Duhamel committed Jan 23, 2024

Jan 18, 2024

Merge branch 'add_jobs_endpoint' into 'main' ••• ✓ 3044f31c
Sergei Johansen authored Jan 18, 2024

Add jobs endpoint 8aea9877
Arnaud Duhamel authored Jan 18, 2024 and Sergei Johansen committed Jan 18, 2024

Jan 14, 2024

Merge branch 'add_CORS_policy_for_frontend' into 'main' ••• ✓ 358ed07d
Ghais Mohamad Bachir Dahdouh authored Jan 14, 2024

Add cors policy for frontend b1cdbffb
Arnaud Duhamel authored Jan 14, 2024 and Ghais Mohamad Bachir Dahdouh committed Jan 14, 2024

Jan 13, 2024

Merge branch 'add_CORS_policy_for_frontend' into 'main' ••• ✓ 1fe76a50
Ghais Mohamad Bachir Dahdouh authored Jan 13, 2024

Add cors policy for frontend fd873fba
Arnaud Duhamel authored Jan 13, 2024 and Ghais Mohamad Bachir Dahdouh committed Jan 13, 2024

Merge branch 'set_up_ci/cd_pipeline' into 'main' ••• ✓ 45a89751
Ghais Mohamad Bachir Dahdouh authored Jan 13, 2024

Set up ci/cd pipeline 99c5330f
Arnaud Duhamel authored Jan 13, 2024 and Ghais Mohamad Bachir Dahdouh committed Jan 13, 2024

Jan 09, 2024

Merge branch 'rename_project_to_backend' into 'main' ••• e1cee7d6
Arnaud Duhamel authored Jan 09, 2024

Commenting out app.UseHttpRedirection to avoid https error in deployment fcf73f05
Arnaud authored Jan 09, 2024

Merge branch 'rename_project_to_backend' into 'main' ••• ebc6581d
Ghais Mohamad Bachir Dahdouh authored Jan 09, 2024

small change to README 682e7fea
Arnaud authored Jan 09, 2024

Merge branch 'main' into 'rename_project_to_backend' ••• 5e9b4bb7
Arnaud Duhamel authored Jan 09, 2024

renaming project to BackEnd instead of HelloWorld c0e905d2
Arnaud authored Jan 09, 2024

Merge branch 'readme_test' into 'main' ••• 0f333bff
Arnaud Duhamel authored Jan 09, 2024

Readme test 4d2cb0ee
Arnaud Duhamel authored Jan 09, 2024

Merge branch 'readme_test' into 'main' ••• 36f7825e
Arnaud Duhamel authored Jan 09, 2024

Readme test 1cdd3c75
Arnaud Duhamel authored Jan 09, 2024

Merge branch 'readme_test' into 'main' ••• ecde634f
Arnaud Duhamel authored Jan 09, 2024

Readme test 46b5012d
Arnaud Duhamel authored Jan 09, 2024

Help

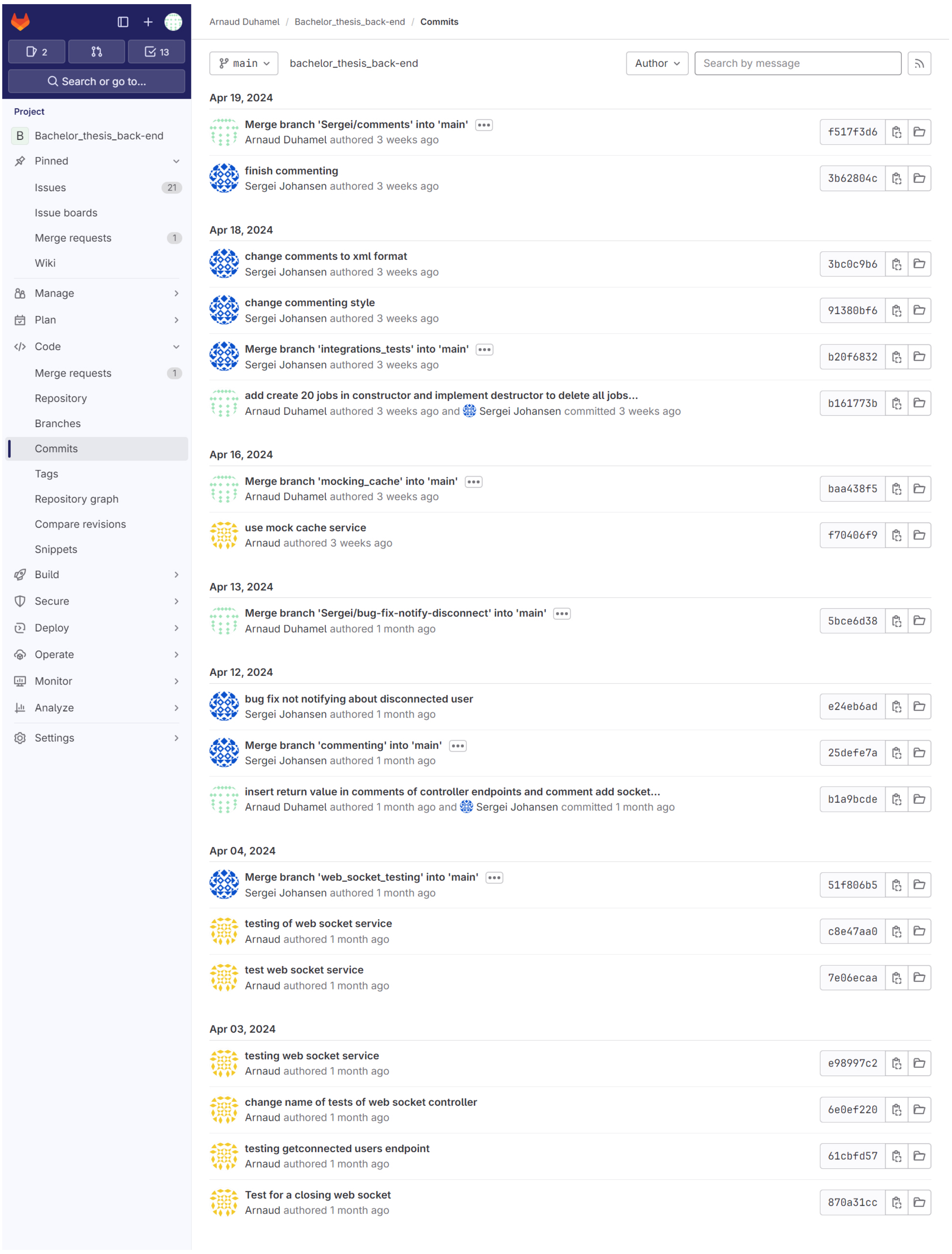
Jan 04, 2024

Initial commit
Arnaud Duhamel authored Jan 04, 2024

948fbb7b

Appendix H

Commit history of the frontend repository



Apr 19, 2024

Merge branch 'Sergei/comments' into 'main'
Arnaud Duhamel authored 3 weeks ago — f517f3d6

finish commenting
Sergei Johansen authored 3 weeks ago — 3b62804c

Apr 18, 2024

change comments to xml format
Sergei Johansen authored 3 weeks ago — 3bc0c9b6

change commenting style
Sergei Johansen authored 3 weeks ago — 91380bf6

Merge branch 'integrations_tests' into 'main'
Sergei Johansen authored 3 weeks ago — b20f6832

add create 20 jobs in constructor and implement destructor to delete all jobs...
Arnaud Duhamel authored 3 weeks ago and Sergei Johansen committed 3 weeks ago — b161773b

Apr 16, 2024

Merge branch 'mocking_cache' into 'main'
Arnaud Duhamel authored 3 weeks ago — baa438f5

use mock cache service
Arnaud authored 3 weeks ago — f70406f9

Apr 13, 2024

Merge branch 'Sergei/bug-fix-notify-disconnect' into 'main'
Arnaud Duhamel authored 1 month ago — 5bce6d38

Apr 12, 2024

bug fix not notifying about disconnected user
Sergei Johansen authored 1 month ago — e24eb6ad

Merge branch 'commenting' into 'main'
Sergei Johansen authored 1 month ago — 25defe7a

insert return value in comments of controller endpoints and comment add socket...
Arnaud Duhamel authored 1 month ago and Sergei Johansen committed 1 month ago — b1a9bcde

Apr 04, 2024

Merge branch 'web_socket_testing' into 'main'
Sergei Johansen authored 1 month ago — 51f806b5

testing of web socket service
Arnaud authored 1 month ago — c8e47aa0

test web socket service
Arnaud authored 1 month ago — 7e06ecaa

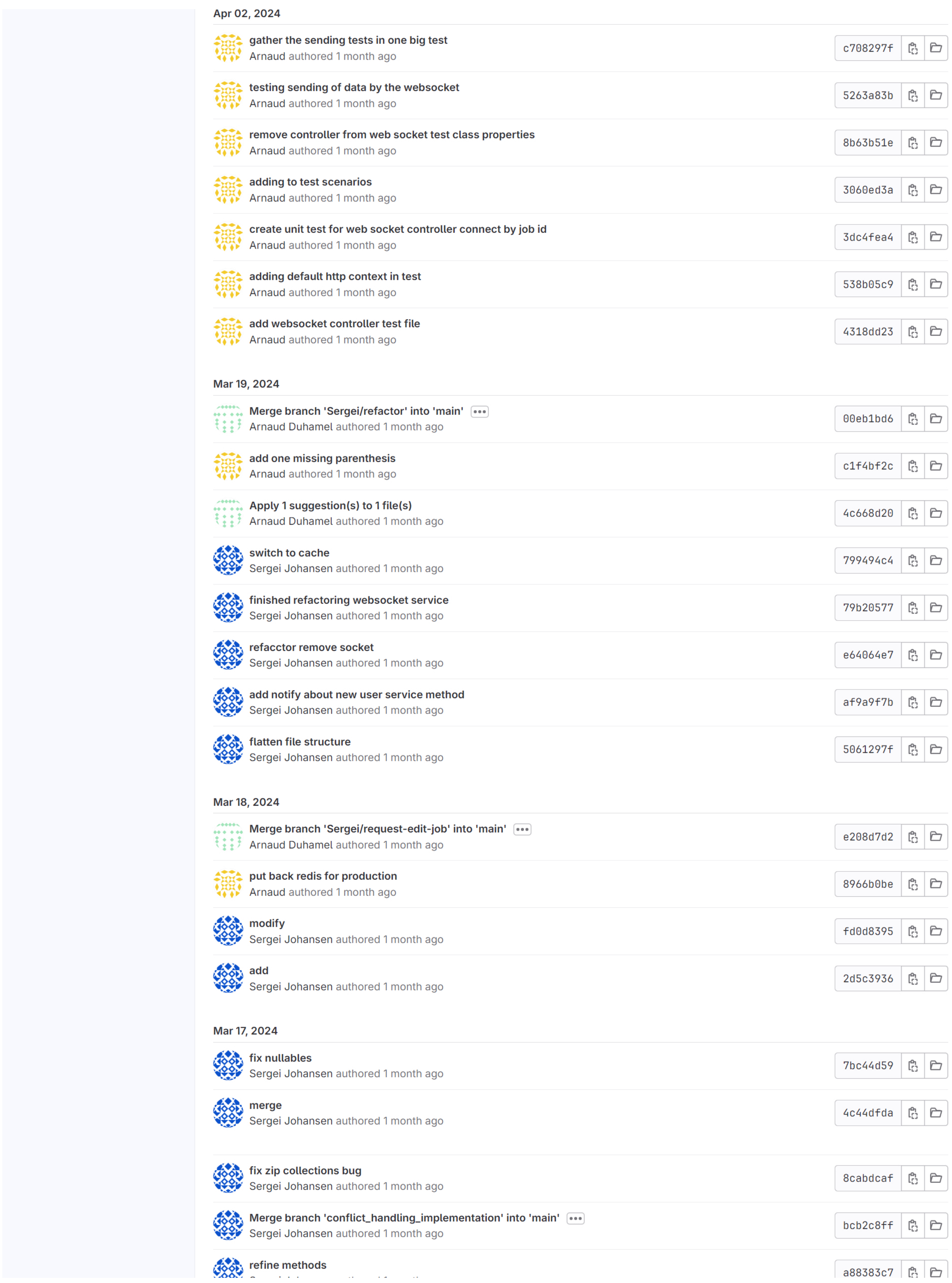
Apr 03, 2024

testing web socket service
Arnaud authored 1 month ago — e98997c2

change name of tests of web socket controller
Arnaud authored 1 month ago — 6e0ef220

testing getconnected users endpoint
Arnaud authored 1 month ago — 61cbfd57

Test for a closing web socket
Arnaud authored 1 month ago — 870a31cc



Apr 02, 2024

gather the sending tests in one big test
Arnaud authored 1 month ago — c708297f

testing sending of data by the websocket
Arnaud authored 1 month ago — 5263a83b

remove controller from web socket test class properties
Arnaud authored 1 month ago — 8b63b51e

adding to test scenarios
Arnaud authored 1 month ago — 3060ed3a

create unit test for web socket controller connect by job id
Arnaud authored 1 month ago — 3dc4fea4

adding default http context in test
Arnaud authored 1 month ago — 538b05c9

add websocket controller test file
Arnaud authored 1 month ago — 4318dd23

Mar 19, 2024

Merge branch 'Sergei/refactor' into 'main' •••
Arnaud Duhamel authored 1 month ago — 00eb1bd6

add one missing parenthesis
Arnaud authored 1 month ago — c1f4bf2c

Apply 1 suggestion(s) to 1 file(s)
Arnaud Duhamel authored 1 month ago — 4c668d20

switch to cache
Sergei Johansen authored 1 month ago — 799494c4

finished refactoring websocket service
Sergei Johansen authored 1 month ago — 79b20577

refacctor remove socket
Sergei Johansen authored 1 month ago — e64064e7

add notify about new user service method
Sergei Johansen authored 1 month ago — af9a9f7b

flatten file structure
Sergei Johansen authored 1 month ago — 5061297f

Mar 18, 2024

Merge branch 'Sergei/request-edit-job' into 'main' •••
Arnaud Duhamel authored 1 month ago — e208d7d2

put back redis for production
Arnaud authored 1 month ago — 8966b0be

modify
Sergei Johansen authored 1 month ago — fd0d8395

add
Sergei Johansen authored 1 month ago — 2d5c3936

Mar 17, 2024

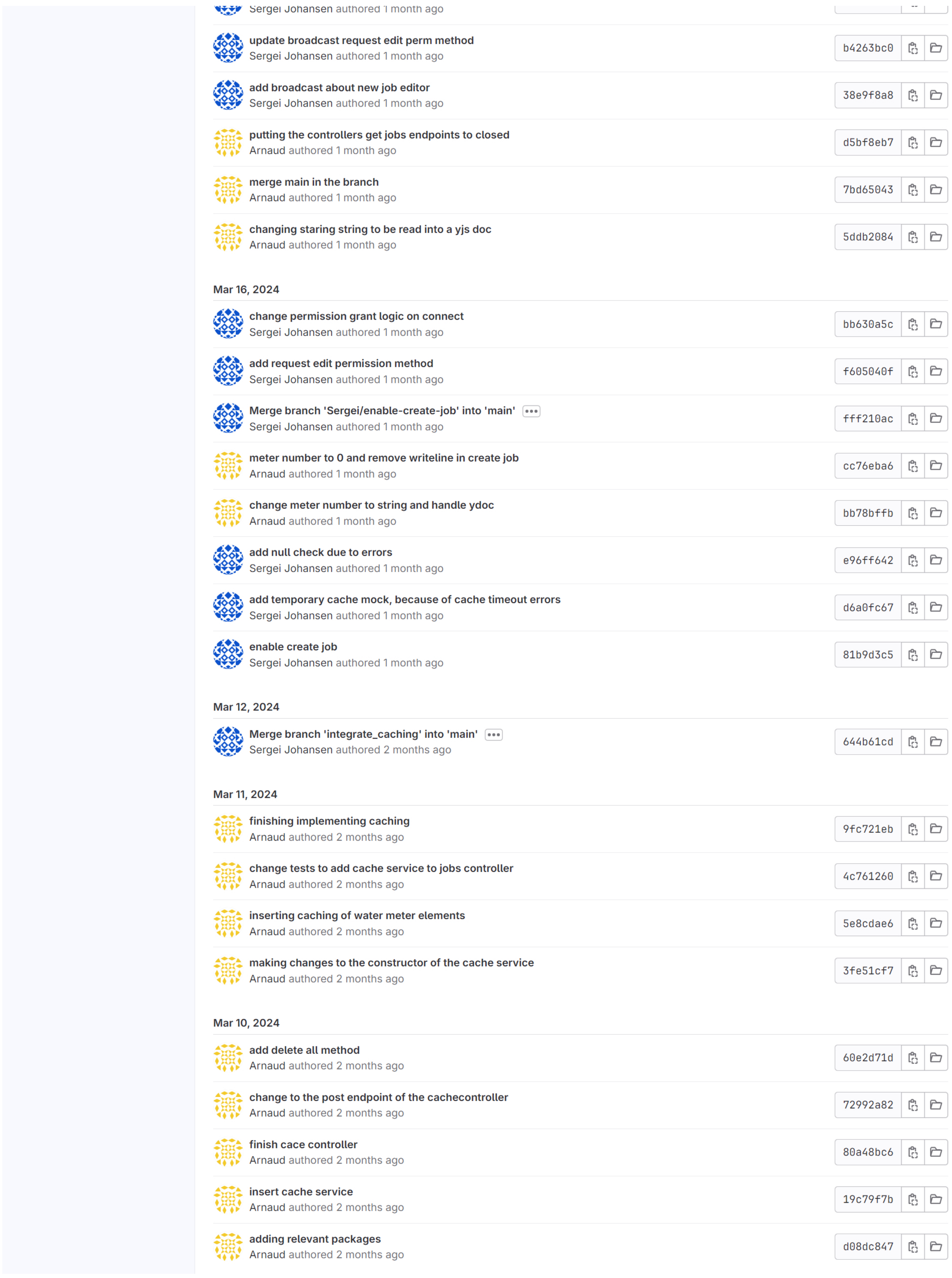
fix nullables
Sergei Johansen authored 1 month ago — 7bc44d59

merge
Sergei Johansen authored 1 month ago — 4c44dfda

fix zip collections bug
Sergei Johansen authored 1 month ago — 8cabdcaf

Merge branch 'conflict_handling_implementation' into 'main' •••
Sergei Johansen authored 1 month ago — bcb2c8ff

refine methods
— a88383c7



Sergei Johansen authored 1 month ago

update broadcast request edit perm method
Sergei Johansen authored 1 month ago — b4263bc0

add broadcast about new job editor
Sergei Johansen authored 1 month ago — 38e9f8a8

putting the controllers get jobs endpoints to closed
Arnaud authored 1 month ago — d5bf8eb7

merge main in the branch
Arnaud authored 1 month ago — 7bd65043

changing staring string to be read into a yjs doc
Arnaud authored 1 month ago — 5ddb2084

Mar 16, 2024

change permission grant logic on connect
Sergei Johansen authored 1 month ago — bb630a5c

add request edit permission method
Sergei Johansen authored 1 month ago — f605040f

Merge branch 'Sergei/enable-create-job' into 'main' •••
Sergei Johansen authored 1 month ago — fff210ac

meter number to 0 and remove writeline in create job
Arnaud authored 1 month ago — cc76eba6

change meter number to string and handle ydoc
Arnaud authored 1 month ago — bb78bffb

add null check due to errors
Sergei Johansen authored 1 month ago — e96ff642

add temporary cache mock, because of cache timeout errors
Sergei Johansen authored 1 month ago — d6a0fc67

enable create job
Sergei Johansen authored 1 month ago — 81b9d3c5

Mar 12, 2024

Merge branch 'integrate_caching' into 'main' •••
Sergei Johansen authored 2 months ago — 644b61cd

Mar 11, 2024

finishing implementing caching
Arnaud authored 2 months ago — 9fc721eb

change tests to add cache service to jobs controller
Arnaud authored 2 months ago — 4c761260

inserting caching of water meter elements
Arnaud authored 2 months ago — 5e8cdae6

making changes to the constructor of the cache service
Arnaud authored 2 months ago — 3fe51cf7

Mar 10, 2024

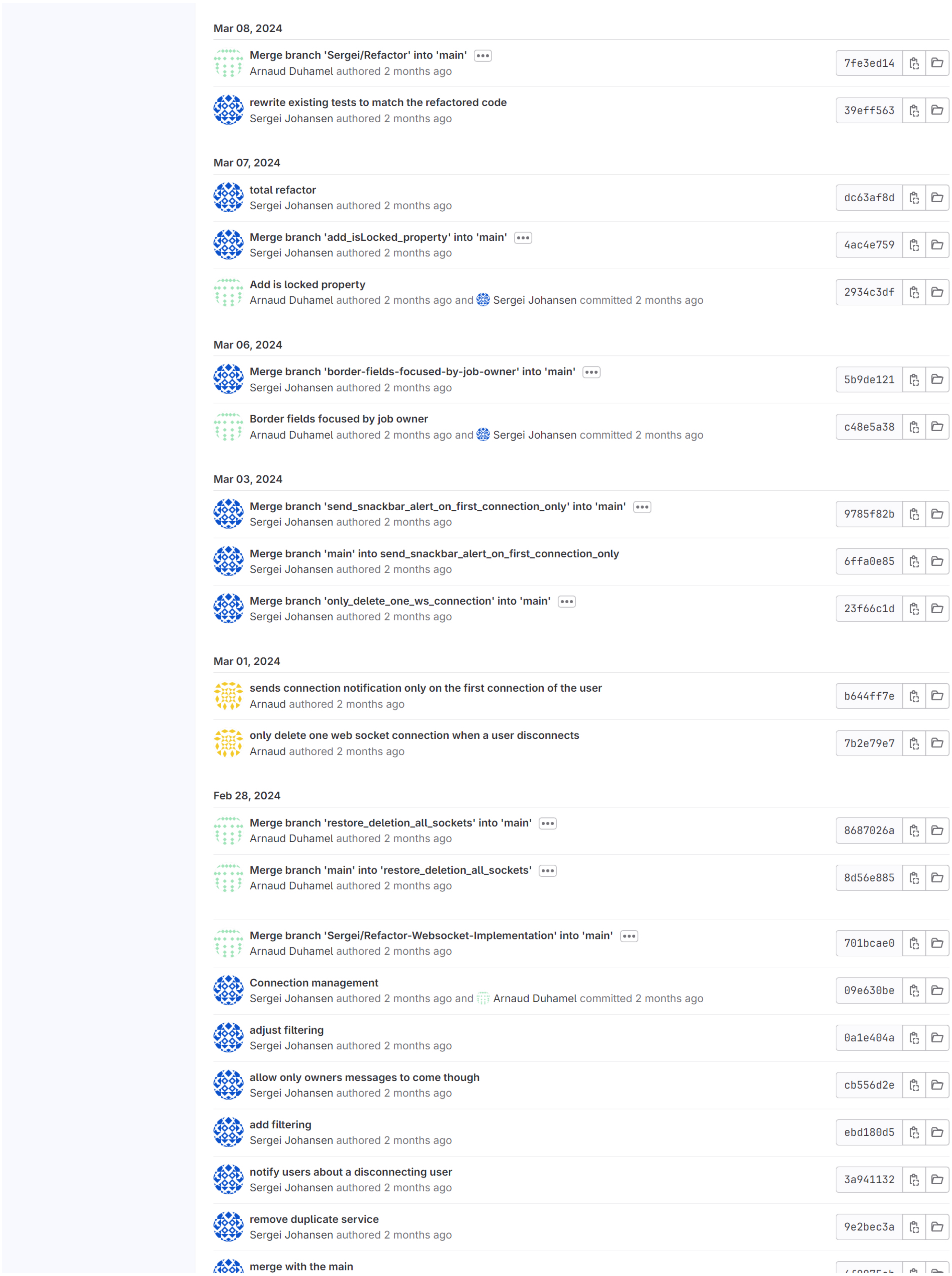
add delete all method
Arnaud authored 2 months ago — 60e2d71d

change to the post endpoint of the cachecontroller
Arnaud authored 2 months ago — 72992a82

finish cace controller
Arnaud authored 2 months ago — 80a48bc6

insert cache service
Arnaud authored 2 months ago — 19c79f7b

adding relevant packages
Arnaud authored 2 months ago — d08dc847



Mar 08, 2024

Merge branch 'Sergei/Refactor' into 'main' ••• — 7fe3ed14
Arnaud Duhamel authored 2 months ago

rewrite existing tests to match the refactored code — 39eff563
Sergei Johansen authored 2 months ago

Mar 07, 2024

total refactor — dc63af8d
Sergei Johansen authored 2 months ago

Merge branch 'add_isLocked_property' into 'main' ••• — 4ac4e759
Sergei Johansen authored 2 months ago

Add is locked property — 2934c3df
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago

Mar 06, 2024

Merge branch 'border-fields-focused-by-job-owner' into 'main' ••• — 5b9de121
Sergei Johansen authored 2 months ago

Border fields focused by job owner — c48e5a38
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago

Mar 03, 2024

Merge branch 'send_snackbar_alert_on_first_connection_only' into 'main' ••• — 9785f82b
Sergei Johansen authored 2 months ago

Merge branch 'main' into send_snackbar_alert_on_first_connection_only — 6ffa0e85
Sergei Johansen authored 2 months ago

Merge branch 'only_delete_one_ws_connection' into 'main' ••• — 23f66c1d
Sergei Johansen authored 2 months ago

Mar 01, 2024

sends connection notification only on the first connection of the user — b644ff7e
Arnaud authored 2 months ago

only delete one web socket connection when a user disconnects — 7b2e79e7
Arnaud authored 2 months ago

Feb 28, 2024

Merge branch 'restore_deletion_all_sockets' into 'main' ••• — 8687026a
Arnaud Duhamel authored 2 months ago

Merge branch 'main' into 'restore_deletion_all_sockets' ••• — 8d56e885
Arnaud Duhamel authored 2 months ago

Merge branch 'Sergei/Refactor-Websocket-Implementation' into 'main' ••• — 701bcae0
Arnaud Duhamel authored 2 months ago

Connection management — 09e630be
Sergei Johansen authored 2 months ago and Arnaud Duhamel committed 2 months ago

adjust filtering — 0a1e404a
Sergei Johansen authored 2 months ago

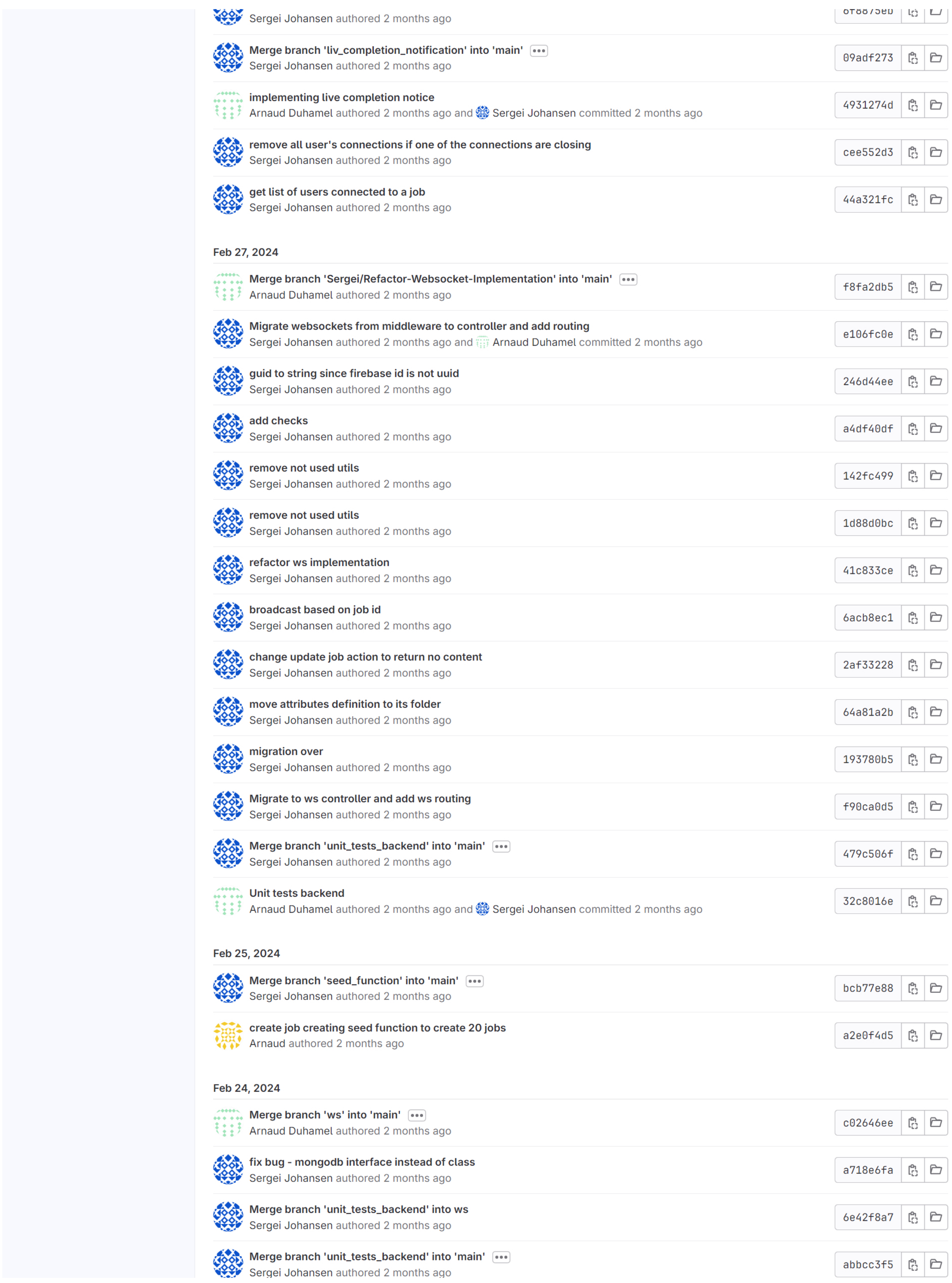
allow only owners messages to come though — cb556d2e
Sergei Johansen authored 2 months ago

add filtering — ebd180d5
Sergei Johansen authored 2 months ago

notify users about a disconnecting user — 3a941132
Sergei Johansen authored 2 months ago

remove duplicate service — 9e2bec3a
Sergei Johansen authored 2 months ago

merge with the main



Sergei Johansen authored 2 months ago

Merge branch 'liv_completion_notification' into 'main' •••
Sergei Johansen authored 2 months ago — 09adf273

implementing live completion notice
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — 4931274d

remove all user's connections if one of the connections are closing
Sergei Johansen authored 2 months ago — cee552d3

get list of users connected to a job
Sergei Johansen authored 2 months ago — 44a321fc

Feb 27, 2024

Merge branch 'Sergei/Refactor-Websocket-Implementation' into 'main' •••
Arnaud Duhamel authored 2 months ago — f8fa2db5

Migrate websockets from middleware to controller and add routing
Sergei Johansen authored 2 months ago and Arnaud Duhamel committed 2 months ago — e106fc0e

guid to string since firebase id is not uuid
Sergei Johansen authored 2 months ago — 246d44ee

add checks
Sergei Johansen authored 2 months ago — a4df40df

remove not used utils
Sergei Johansen authored 2 months ago — 142fc499

remove not used utils
Sergei Johansen authored 2 months ago — 1d88d0bc

refactor ws implementation
Sergei Johansen authored 2 months ago — 41c833ce

broadcast based on job id
Sergei Johansen authored 2 months ago — 6acb8ec1

change update job action to return no content
Sergei Johansen authored 2 months ago — 2af33228

move attributes definition to its folder
Sergei Johansen authored 2 months ago — 64a81a2b

migration over
Sergei Johansen authored 2 months ago — 193780b5

Migrate to ws controller and add ws routing
Sergei Johansen authored 2 months ago — f90ca0d5

Merge branch 'unit_tests_backend' into 'main' •••
Sergei Johansen authored 2 months ago — 479c506f

Unit tests backend
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — 32c8016e

Feb 25, 2024

Merge branch 'seed_function' into 'main' •••
Sergei Johansen authored 2 months ago — bcb77e88

create job creating seed function to create 20 jobs
Arnaud authored 2 months ago — a2e0f4d5

Feb 24, 2024

Merge branch 'ws' into 'main' •••
Arnaud Duhamel authored 2 months ago — c02646ee

fix bug - mongodb interface instead of class
Sergei Johansen authored 2 months ago — a718e6fa

Merge branch 'unit_tests_backend' into ws
Sergei Johansen authored 2 months ago — 6e42f8a7

Merge branch 'unit_tests_backend' into 'main' •••
Sergei Johansen authored 2 months ago — abbcc3f5

Unit tests backend
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago — b6c88c3e

add log
Sergei Johansen authored 2 months ago — 99b9b8f4

adding mongoDb interface and adding one unit test for job service
Arnaud authored 2 months ago — 74461ab8

Feb 23, 2024

Add unit tests for users and changing names for jobs controller unit tests
Arnaud authored 2 months ago — 103d87e4

change to gitlab-ci file
Arnaud authored 2 months ago — 22e85606

Feb 22, 2024

finished unit tests of jobs controller
Arnaud authored 2 months ago — a469a546

adding unit tests and putting gitgnore files in each subfolder
Arnaud authored 2 months ago — 97af6445

web sockets setup
Sergei Johansen authored 2 months ago — ac05ba97

socket repo and service done
Sergei Johansen authored 2 months ago — 6fbcdb6a

optimize structure
Sergei Johansen authored 2 months ago — 3fe92ea1

add wsconnection repository
Sergei Johansen authored 2 months ago — 94af0ce7

Feb 17, 2024

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — 387c8995

change the dockerfile
Arnaud authored 2 months ago — 8eb8bd83

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — 96f32cbe

add docker command to copy the firebase.json file in the app
Arnaud authored 2 months ago — 0fbe5916

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — 79f0353c

change to dockerfile
Arnaud authored 2 months ago — 1b86c9f7

Merge branch 'fixing_docker_deploy' into 'main' •••
Arnaud Duhamel authored 2 months ago — ae3de74e

put dockerfile in main directory and adapt it to the new project structure
Arnaud authored 2 months ago — fedbe3e8

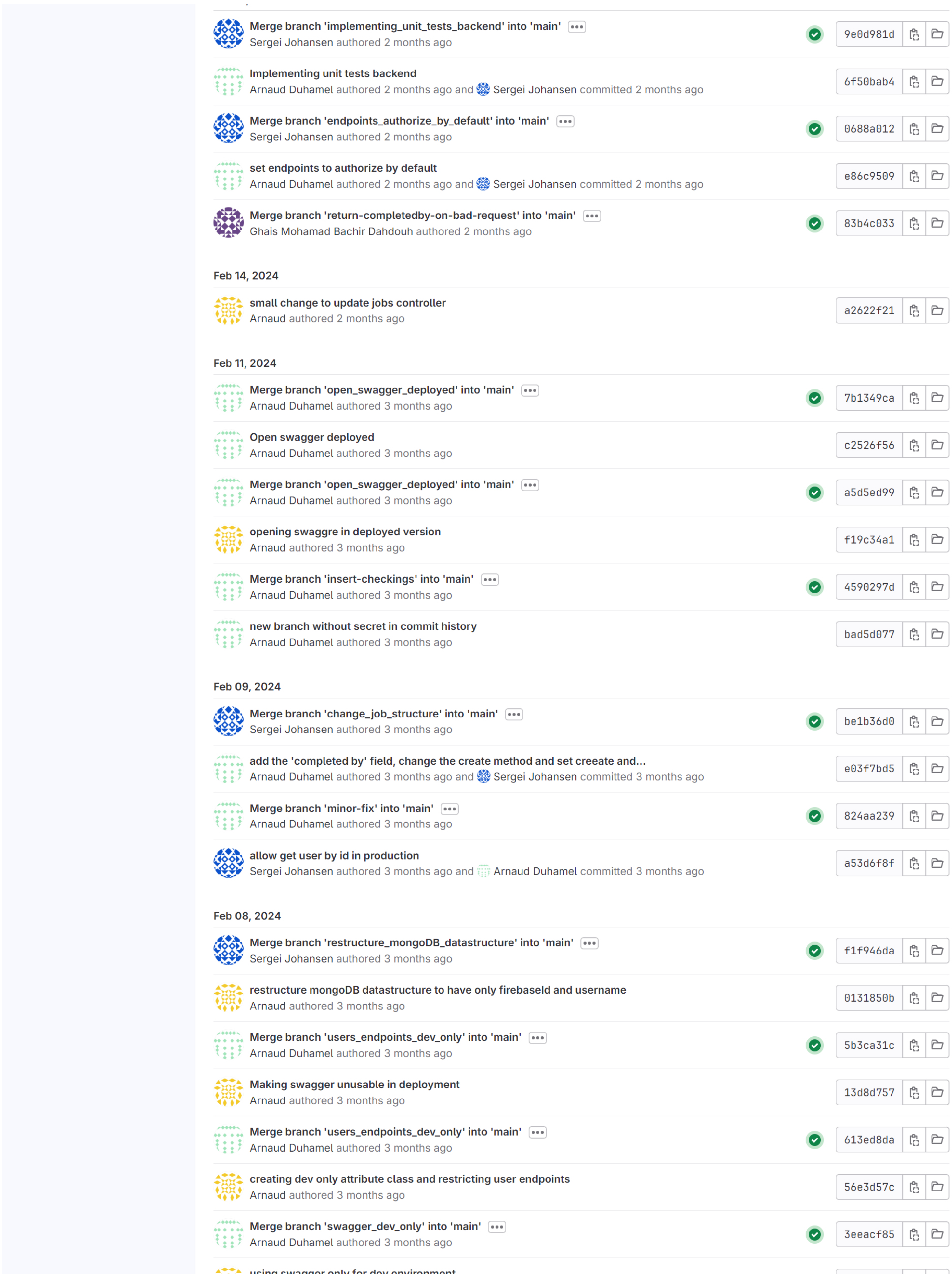
Merge branch 'test2' into 'main' •••
Arnaud Duhamel authored 2 months ago — 75b333e9

Test2
Arnaud Duhamel authored 2 months ago — 78707b78

Merge branch 'add-solution-file-for-organization' into 'main' •••
Arnaud Duhamel authored 2 months ago — ✓ f31a8fc6

Add solution file for organization
Sergei Johansen authored 2 months ago and Arnaud Duhamel committed 2 months ago — 3782702e

Feb 15, 2024



Merge branch 'implementing_unit_tests_backend' into 'main' •••
Sergei Johansen authored 2 months ago
9e0d981d

Implementing unit tests backend
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago
6f50bab4

Merge branch 'endpoints_authorize_by_default' into 'main' •••
Sergei Johansen authored 2 months ago
0688a012

set endpoints to authorize by default
Arnaud Duhamel authored 2 months ago and Sergei Johansen committed 2 months ago
e86c9509

Merge branch 'return-completedby-on-bad-request' into 'main' •••
Ghais Mohamad Bachir Dahdouh authored 2 months ago
83b4c033

Feb 14, 2024

small change to update jobs controller
Arnaud authored 2 months ago
a2622f21

Feb 11, 2024

Merge branch 'open_swagger_deployed' into 'main' •••
Arnaud Duhamel authored 3 months ago
7b1349ca

Open swagger deployed
Arnaud Duhamel authored 3 months ago
c2526f56

Merge branch 'open_swagger_deployed' into 'main' •••
Arnaud Duhamel authored 3 months ago
a5d5ed99

opening swaggre in deployed version
Arnaud authored 3 months ago
f19c34a1

Merge branch 'insert-checkings' into 'main' •••
Arnaud Duhamel authored 3 months ago
4590297d

new branch without secret in commit history
Arnaud Duhamel authored 3 months ago
bad5d077

Feb 09, 2024

Merge branch 'change_job_structure' into 'main' •••
Sergei Johansen authored 3 months ago
be1b36d0

add the 'completed by' field, change the create method and set creeate and...
Arnaud Duhamel authored 3 months ago and Sergei Johansen committed 3 months ago
e03f7bd5

Merge branch 'minor-fix' into 'main' •••
Arnaud Duhamel authored 3 months ago
824aa239

allow get user by id in production
Sergei Johansen authored 3 months ago and Arnaud Duhamel committed 3 months ago
a53d6f8f

Feb 08, 2024

Merge branch 'restructure_mongoDB_datastructure' into 'main' •••
Sergei Johansen authored 3 months ago
f1f946da

restructure mongoDB datastructure to have only firebaseId and username
Arnaud authored 3 months ago
0131850b

Merge branch 'users_endpoints_dev_only' into 'main' •••
Arnaud Duhamel authored 3 months ago
5b3ca31c

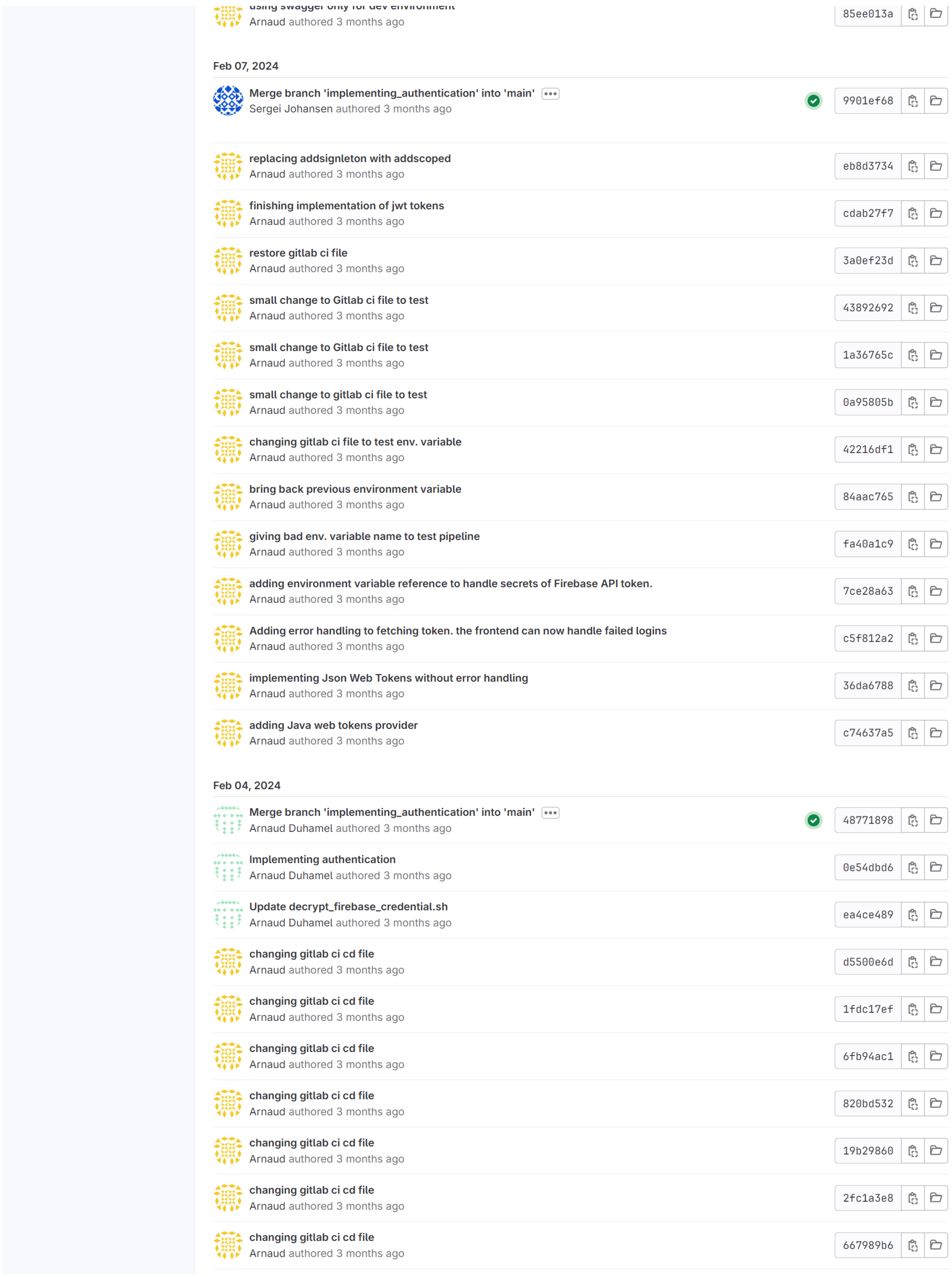
Making swagger unusable in deployment
Arnaud authored 3 months ago
13d8d757

Merge branch 'users_endpoints_dev_only' into 'main' •••
Arnaud Duhamel authored 3 months ago
613ed8da

creating dev only attribute class and restricting user endpoints
Arnaud authored 3 months ago
56e3d57c

Merge branch 'swagger_dev_only' into 'main' •••
Arnaud Duhamel authored 3 months ago
3eeacf85

using swagger only for dev environment



using swagger only for dev environment
Arnaud authored 3 months ago — 85ee013a

Feb 07, 2024

Merge branch 'implementing_authentication' into 'main' ✓
Sergei Johansen authored 3 months ago — 9901ef68

replacing addsignleton with addscoped
Arnaud authored 3 months ago — eb8d3734

finishing implementation of jwt tokens
Arnaud authored 3 months ago — cdab27f7

restore gitlab ci file
Arnaud authored 3 months ago — 3a0ef23d

small change to Gitlab ci file to test
Arnaud authored 3 months ago — 43892692

small change to Gitlab ci file to test
Arnaud authored 3 months ago — 1a36765c

small change to gitlab ci file to test
Arnaud authored 3 months ago — 0a95805b

changing gitlab ci file to test env. variable
Arnaud authored 3 months ago — 42216df1

bring back previous environment variable
Arnaud authored 3 months ago — 84aac765

giving bad env. variable name to test pipeline
Arnaud authored 3 months ago — fa40a1c9

adding environment variable reference to handle secrets of Firebase API token.
Arnaud authored 3 months ago — 7ce28a63

Adding error handling to fetching token. the frontend can now handle failed logins
Arnaud authored 3 months ago — c5f812a2

implementing Json Web Tokens without error handling
Arnaud authored 3 months ago — 36da6788

adding Java web tokens provider
Arnaud authored 3 months ago — c74637a5

Feb 04, 2024

Merge branch 'implementing_authentication' into 'main' ✓
Arnaud Duhamel authored 3 months ago — 48771898

Implementing authentication
Arnaud Duhamel authored 3 months ago — 0e54dbd6

Update decrypt_firebase_credential.sh
Arnaud Duhamel authored 3 months ago — ea4ce489

changing gitlab ci cd file
Arnaud authored 3 months ago — d5500e6d

changing gitlab ci cd file
Arnaud authored 3 months ago — 1fdc17ef

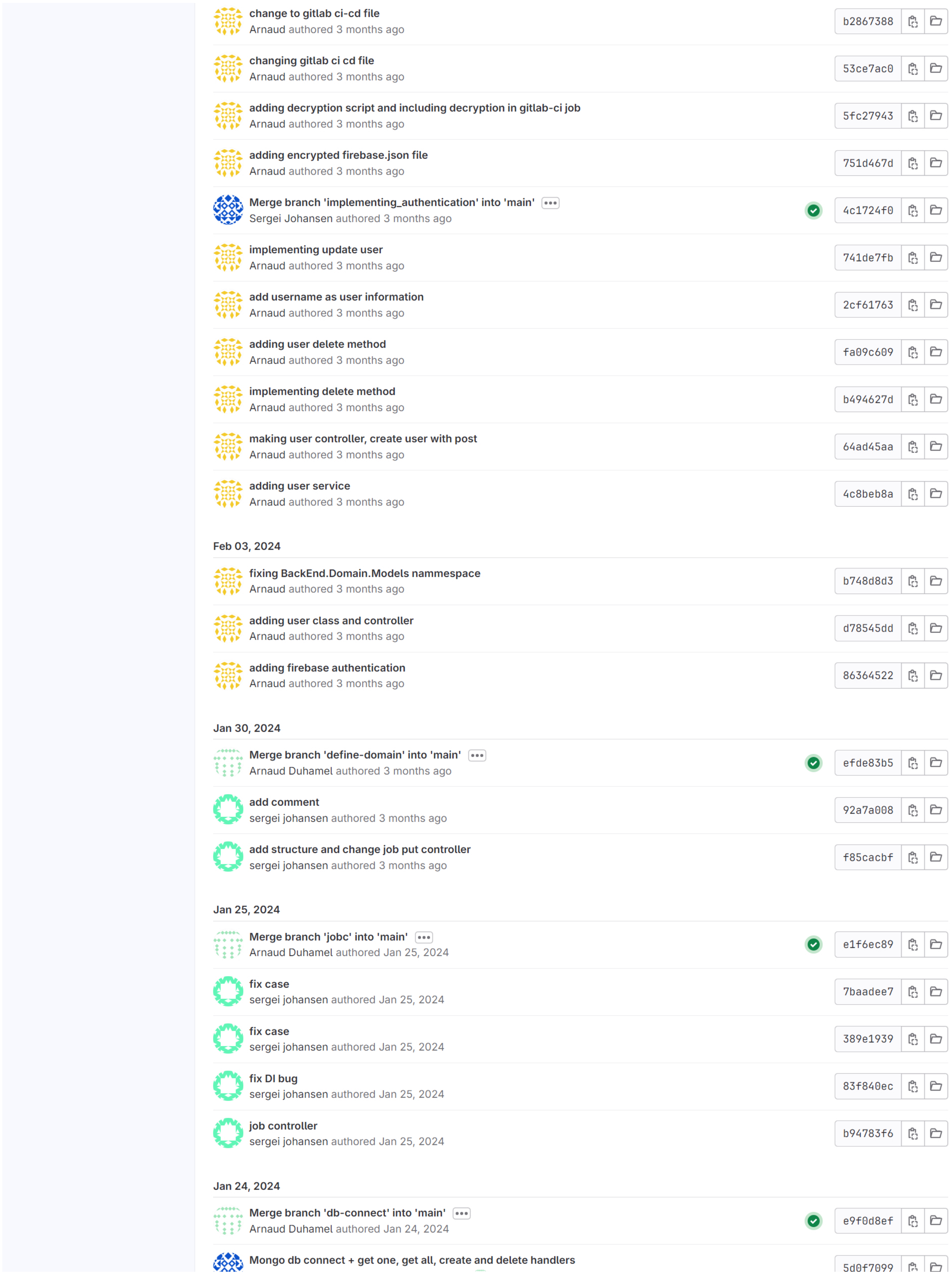
changing gitlab ci cd file
Arnaud authored 3 months ago — 6fb94ac1

changing gitlab ci cd file
Arnaud authored 3 months ago — 820bd532

changing gitlab ci cd file
Arnaud authored 3 months ago — 19b29860

changing gitlab ci cd file
Arnaud authored 3 months ago — 2fc1a3e8

changing gitlab ci cd file
Arnaud authored 3 months ago — 667989b6



change to gitlab ci-cd file
Arnaud authored 3 months ago — b2867388

changing gitlab ci cd file
Arnaud authored 3 months ago — 53ce7ac0

adding decryption script and including decryption in gitlab-ci job
Arnaud authored 3 months ago — 5fc27943

adding encrypted firebase.json file
Arnaud authored 3 months ago — 751d467d

Merge branch 'implementing_authentication' into 'main' •••
Sergei Johansen authored 3 months ago — ✓ 4c1724f0

implementing update user
Arnaud authored 3 months ago — 741de7fb

add username as user information
Arnaud authored 3 months ago — 2cf61763

adding user delete method
Arnaud authored 3 months ago — fa09c609

implementing delete method
Arnaud authored 3 months ago — b494627d

making user controller, create user with post
Arnaud authored 3 months ago — 64ad45aa

adding user service
Arnaud authored 3 months ago — 4c8beb8a

Feb 03, 2024

fixing BackEnd.Domain.Models nammespace
Arnaud authored 3 months ago — b748d8d3

adding user class and controller
Arnaud authored 3 months ago — d78545dd

adding firebase authentication
Arnaud authored 3 months ago — 86364522

Jan 30, 2024

Merge branch 'define-domain' into 'main' •••
Arnaud Duhamel authored 3 months ago — ✓ efde83b5

add comment
sergei johansen authored 3 months ago — 92a7a008

add structure and change job put controller
sergei johansen authored 3 months ago — f85cacbf

Jan 25, 2024

Merge branch 'jobc' into 'main' •••
Arnaud Duhamel authored Jan 25, 2024 — ✓ e1f6ec89

fix case
sergei johansen authored Jan 25, 2024 — 7baadee7

fix case
sergei johansen authored Jan 25, 2024 — 389e1939

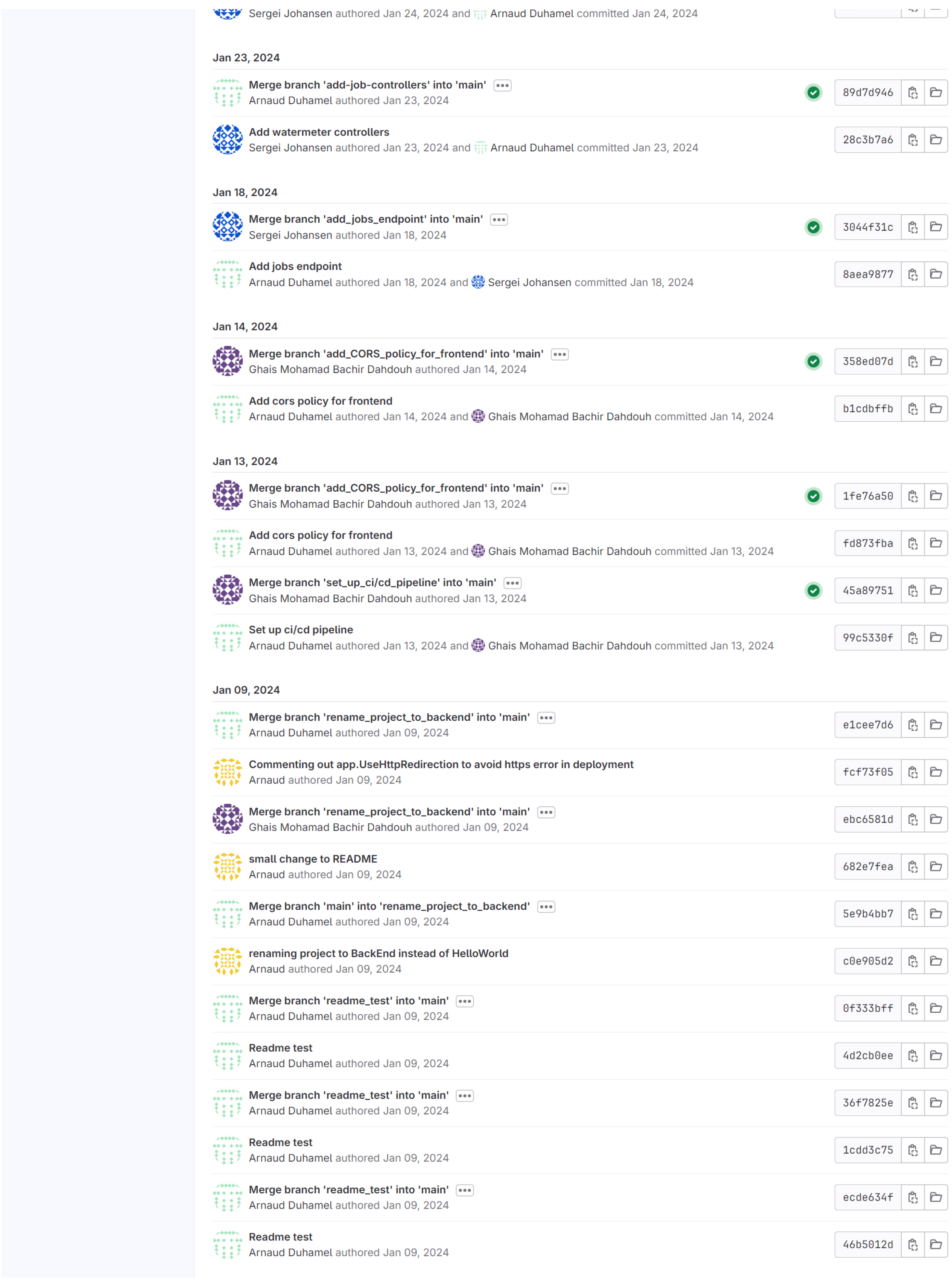
fix DI bug
sergei johansen authored Jan 25, 2024 — 83f840ec

job controller
sergei johansen authored Jan 25, 2024 — b94783f6

Jan 24, 2024

Merge branch 'db-connect' into 'main' •••
Arnaud Duhamel authored Jan 24, 2024 — ✓ e9f0d8ef

Mongo db connect + get one, get all, create and delete handlers — 5d0f7099



Sergei Johansen authored Jan 24, 2024 and Arnaud Duhamel committed Jan 24, 2024

Jan 23, 2024

Merge branch 'add-job-controllers' into 'main' ••• ✓ `89d7d946`
Arnaud Duhamel authored Jan 23, 2024

Add watermeter controllers `28c3b7a6`
Sergei Johansen authored Jan 23, 2024 and Arnaud Duhamel committed Jan 23, 2024

Jan 18, 2024

Merge branch 'add_jobs_endpoint' into 'main' ••• ✓ `3044f31c`
Sergei Johansen authored Jan 18, 2024

Add jobs endpoint `8aea9877`
Arnaud Duhamel authored Jan 18, 2024 and Sergei Johansen committed Jan 18, 2024

Jan 14, 2024

Merge branch 'add_CORS_policy_for_frontend' into 'main' ••• ✓ `358ed07d`
Ghais Mohamad Bachir Dahdouh authored Jan 14, 2024

Add cors policy for frontend `b1cdbffb`
Arnaud Duhamel authored Jan 14, 2024 and Ghais Mohamad Bachir Dahdouh committed Jan 14, 2024

Jan 13, 2024

Merge branch 'add_CORS_policy_for_frontend' into 'main' ••• ✓ `1fe76a50`
Ghais Mohamad Bachir Dahdouh authored Jan 13, 2024

Add cors policy for frontend `fd873fba`
Arnaud Duhamel authored Jan 13, 2024 and Ghais Mohamad Bachir Dahdouh committed Jan 13, 2024

Merge branch 'set_up_ci/cd_pipeline' into 'main' ••• ✓ `45a89751`
Ghais Mohamad Bachir Dahdouh authored Jan 13, 2024

Set up ci/cd pipeline `99c5330f`
Arnaud Duhamel authored Jan 13, 2024 and Ghais Mohamad Bachir Dahdouh committed Jan 13, 2024

Jan 09, 2024

Merge branch 'rename_project_to_backend' into 'main' ••• `e1cee7d6`
Arnaud Duhamel authored Jan 09, 2024

Commenting out app.UseHttpRedirection to avoid https error in deployment `fcf73f05`
Arnaud authored Jan 09, 2024

Merge branch 'rename_project_to_backend' into 'main' ••• `ebc6581d`
Ghais Mohamad Bachir Dahdouh authored Jan 09, 2024

small change to README `682e7fea`
Arnaud authored Jan 09, 2024

Merge branch 'main' into 'rename_project_to_backend' ••• `5e9b4bb7`
Arnaud Duhamel authored Jan 09, 2024

renaming project to BackEnd instead of HelloWorld `c0e905d2`
Arnaud authored Jan 09, 2024

Merge branch 'readme_test' into 'main' ••• `0f333bff`
Arnaud Duhamel authored Jan 09, 2024

Readme test `4d2cb0ee`
Arnaud Duhamel authored Jan 09, 2024

Merge branch 'readme_test' into 'main' ••• `36f7825e`
Arnaud Duhamel authored Jan 09, 2024

Readme test `1cdd3c75`
Arnaud Duhamel authored Jan 09, 2024

Merge branch 'readme_test' into 'main' ••• `ecde634f`
Arnaud Duhamel authored Jan 09, 2024

Readme test `46b5012d`
Arnaud Duhamel authored Jan 09, 2024

Help

Jan 04, 2024

Initial commit
Arnaud Duhamel authored Jan 04, 2024

`948fbb7b`

Appendix I

Project plan

NTNU
Kunnskap for en bedre verden

DEPARTMENT OF COMPUTER SCIENCE

PROG2900 - BACHELOR THESIS

Project Plan

Author:
Sergei Johansen
Arnaud Duhamel
Ghais Dahdouh

1st February 2024

1 Project scope

Software like Teams, Google docs and GitHub have made collaboration among teams much easier then it used to be.

With those tools, users can modify in real time the same document. This actually does not remove the need to handle potential version conflicts between users, but it is now handled immediately, in real time. With such tools, users are pretty well equipped to avoid unwillingly overriding each other's work.

These tools draw, among others, from those two fields:

1.1 Fields

Real time features

Real-time features provide information in real time to users about what other users are doing. This allows users to collaborate with each other effectively and above all, to avoid overriding each other's work.

This is wider then collaborative editing. Collaborative editing is one way among many to address the issue of having many users working together and potentially overriding each other's work. For example, notifications could be used, warnings, having document statuses being updated in real time and knowing when other users are inside a document are also features that can be implemented.

Conflict handling

In the case where multiple users work on the same document and end up with two different versions, there needs a way to decide what will be the final version.

Git is the best example of that. With git, the user can decide what will be the final version for each part of a file where there are two different versions. The current version, the incoming version, or a whole new version can be chosen.

This is something that can be implemented with or without real-time features.

1.2 Project limitation

Our project will focus on implementing features related to those fields in one of Norkart's products called Komtek.

Our project will be restricted to the part of the Komtek systems that handles tasks.

1.3 Project description

To achieve this, our project includes the following steps, taken from the project description submitted by Norkart:

- Map out modern user interface solutions that handles various processes in both a synchronus and asynchronus way.
- Research and document good design practices for collaborative editing in complex systems.
- Develop a design system for collaborative editing in the relevant processes of the Komtek systems such as processing active tasks, version conflict handling and task handling.
- Research if it's possible to develop a generic system architecture that makes it possible to use real time features in many different products.
- Implement a proof-of-concept with a user interface imitating the relevant processes of the Komtek systems.

- Conduct qualitative user tests in partnership with Norkart.

The full project description is in appendix 1.

2 Background and framework

2.1 Background

Here is the typical issue in the task handling part of the Komtek systems faced by Norkart that lead to this project:

Two case workers will see the same unassigned task. Both of them will click on the task. They do not know that someone has been assigned to the task unless they refresh their page after the task has been assigned. Because there is no real-time feedback, they do not know that there is two users working on the same task. Both will complete the task. The first case worker submits the changes he made to the task. After, the second case worker also submits his changes to the task. Because he is the last one to submit the task, his version replaces the version of the first worker. As a result, two case workers spent time working on the same task, and the work done by one of the case worker is completely lost.

2.2 Objectives

Deliverables to Norkart

What Norkart wishes to obtain at the end of this project is a prototype that will replicate the task handling processes of the Komtek systems and that will include various real time and conflict handling features aimed at avoiding users overriding each other's work.

They would also like that we develop our prototype with a generic architecture that could easily be implemented in many other products.

Benefits to Norkart

Based on this prototype, Norkart will be able to integrate many features that will either reduce or prevent case workers from overriding each other's work in the task handling processes of the Komtek systems. And potentially many other systems.

Learning objectives

The learning objectives are the following:

- Learn about web technologies used in the Norwegian industry(React, .NET)
- Learn about real-time technologies(WebSocket, Firebse Realtime database)
- Learn about API and frontend testing
- Learn about continuous deployment and integration
- Learn about developing generic architecture design
- Learn to conduct literature reviews, and especially to draw actionable insights from the review
- Further develop team work skills
- Learn about writing documentation
- Learn to program and to develop with a sustainable development perspective

2.3 Framework

The main framework of our project will be the technologies used by Norkart. This is not something that was imposed on us, but we made the decision to use the same technologies that Norkart uses: React and .NET. Both in order to maximize the value of what we will produce for Norkart and because those technologies are widespread in Norway. For the rela-time technology, WebSockets will be implemented for the learning opportunity. But the real-time technology used by Norkart is SignalR.

3 Project organizing

3.1 Responsibilities and roles of the group members

- Arnaud is the group leader and will also work on the backend part of the project.
- Sergei is responsible for the backend part of the project
- Ghais is responsible for the frontend part of the project
- Vebjørn Fonstad Leiros is our contact point at Norkart. He is the one directing our project and providing us feedback.
- Frode Haug is our thesis supervisor at NTNU.

3.2 Processes and group rules

Below are the signed group rules:

Group rules

Meetings

- There will be group meetings on sundays and thursdays.
- The meeting on sundays is to assess the work done in the previous week, how it went, what should be done to improve the working methods, and planning tasks for the coming week, as well as any other issues.
- The meeting on thursdays is to follow up on the tasks of the week and take up any other issues. We will be meeting at Norkart on those days.
- There will be a meeting with the supervisor on tuesdays.
- There will be a meeting with the client on thursdays.
- Group members commit to attend meetings prepared and ready. If a member cannot attend a meeting for a serious reason, he commits to let the other members know as soon as possible and provide the reason.

Working methods

- Each group members commit to work at least 30 hours per week. The group acknowledges that Ghais is working full time and that he will not be able to commit those hours, but nonetheless commits to get his tasks done within the set deadlines.
- The group acknowledges that Ghais works and lives in Lillehammer, reducing the possibilities to work together physically.
- Arnaud and Sergei commit to work on campus physically together as much as possible. Meeting times will be agreed on during the Sunday meetings.
- Group members commit to work one day per week at Norkart's offices during the planning phase of the project. After that, it is not mandatory, but highly encouraged.
- During the meetings, group members commit to focus on working effectively and to not be distracted on other topics.
- In any case, group members commit to get their weekly tasks done.
- If a group member struggles with a task to the point that he may not complete it in time, he commits to let the others know as soon as possible; he commits to not leave it undone and let the others know only at the Sunday meeting.
- The main communication channel between the group members will be Discord.
- Group members commit to be available to answer messages during the day from 10 am to 7 pm. The group acknowledges that Ghais is working full time and is therefore not available to answer messages during his working hours.
- Group members commit to not intentionally ignore other members' messages
- In case of disagreement, a simple majority will have the final say
- Group members commit to be loyal to the decisions taken
- Hours will be logged on a weekly basis in a common Google Sheet
- Issues related to the frontend part of the project will be managed on the issue board of the frontend repository. Every other issue will be managed on the issue board of the backend repository.

- Group members commit to track issues
- Group members commit to document their use of artificial intelligence so that it can be described in the final report how artificial intelligence was used, especially if code produced by artificial intelligence is inserted in the project.
- Arnaud is the group leader and will also work on the backend part of the project.
- Sergei is in charge of the backend part of the project
- Ghais is in charge of the frontend part of the project
- If Arnaud is sick, Ghais will be group leader
- If Sergei is sick, Arnaud will take charge of the backend part
- If Ghais is sick, Sergei will take charge of the frontend part and Arnaud will take over the backend part
- Group members commit to sharing dissatisfactions as soon as possible in a respectful way, preferably within 24 hours.

In case of a rule violation

1. A group conversation about the issue
2. If the situation has not improved within 3 days, a written warning from all of the other group members where:
   - the rule violation is mentioned
   - what is asked to correct the situation
3. If the situation has not improved within 3 days, a group conversation with the supervisor
4. If the situation has not improved within one week, after discussing it with the supervisor, a written exclusion of the group from the other group members

Signature and date:

Sergei Johansen, 23.01.24

Sergei Johansen

[illegible], 18 january 2024

Ghais Mohamad Bashir Dahdouh

[illegible], 18 january 2024

Arnaud Duhamel

4 Planning, following up and reporting

4.1 Project management model

Komtek replica

For this part of the project, the requirements are clear and well known, because it boils down to duplicating what already exists.

For this, a stricter model will be followed and there will be little discussion and changes around the development of this part of the project.

For this part of the project, a waterfall model will be followed.

A list of tasks will be created in advance and that list will be followed, purely and simply.

Choosing real-time features to implement

For this part of the project, a more scientific method will be used. There will be an existing product review and a literature review. Based on that, we will make a decision on which feature to implement, in discussion with Norkart as well. Our reviews may also not exactly give us an answer to our problem, and in such case we will have to make an hypothesis that this feature would be helpful, and then it will be implemented. User tests will be able to tell us if it was a good choice or not.

Implementing the real-time features

This part of the development process will follow a more fluid approach. Because most of the features will be new to the team, it will be more uncertain what we will manage to implement and at what pace. So for this part of the project, a scrum approach will be followed where we will follow up more tightly and adjust underway as needed.

And so, in this spirit, we implemented an issue board and we update it weekly, every Sunday, for the coming week. We have a column for open issues, in-sprint issues, in-process issues and closed issues. We do not have a scrum master but we have a group leader.

On our Sunday meetings, we have both a retrospective on how the week went and planning the upcoming week. And on Thursdays, we have follow up meetings to see how things are going.

4.2 Plan for status meetings and decision making moments

The meetings for the project are set up as follows: One meeting Sunday to have a retrospective on the past week and to plan the upcoming week, one meeting on Tuesdays with our supervisor, one meeting on Thursday with Norkart. We also spend the day working at Norkart's offices on Thursdays.

We decided to have meetings with our supervisor and Norkart every week so as to maximize the feedback that we receive during our thesis.

5 Organizing and quality assurance

5.1 Documentation, standards, configuration, tools, etc

For coding with React, Visual Studio Code is used. The latest version of React is used.

For coding with Dotnet, Visual Studio is used. The latest version of Dotnet is used.

For version control, we use the GitLab instance of the department of computer science of NTNU in Trondheim.

We have two repositories, one for the frontend and one for the backend. Both the frontend and backend repositories have an issue board. Issues unrelated to the frontend are inserted in the backend board.

The backend API is deployed on Render. The frontend is deployed on Firebase.

MongoDB will be used as a permanent database and Firebase real-time Database will be used for the live features.

5.2 Plan for inspection and testing

Before a merge is made on the main branch, another group member reviews it.

There is currently a pipeline at every commits that tests to see if the application builds successfully.

We will introduce automated tests for both the frontend and the backend. this is something that will need to be researched.

5.3 Risk analysis at the project level

The following risk standard comes from the bachelor thesis written in 2022 by Sebastian Lindtvedt, Salvador Bascunan and Dennis Kristiansen at the Norwegian University of Science and Technology (Lindtvedt et al., 2022).

Likelihood/ severity	**Minimal**	**Minor**	**Moderate**	**Significant**	**Critical**
Highly likely	🟨	🟥	🟥	🟥	🟥
Likely	🟨	🟨	🟥	🟥	🟥
Probable	🟩	🟨	🟨	🟥	🟥
Unlikely	🟩	🟩	🟨	🟨	🟥
Highly unlikely	🟩	🟩	🟩	🟨	🟨

Table 1: Risk standard

The following risk table is inspired from the bachelor thesis written in 2022 by Sebastian Lindtvedt, Salvador Bascunan and Dennis Kristiansen at the Norwegian University of Science and Technology (Lindtvedt et al., 2022). The table format is the same.

Risk	**Description**	**Likelihood/Severity**
1	One of the group members getting sick	Unlikely/Significant
2	Inadequate communication	probable/significant
3	Unequal contribution	probable/significant
4	Technical challenges	highly likely/moderate
5	Lack of documentation	probable/moderate
6	Not delivering what the company asked for	likely/critical
7	Spending too much time and energy on irrelevant aspects	highly likely/critical
8	Losing source code	unlikely/critical
9	Losing documentation	unlikely/significant
10	Losing equipment	highly unlikely/minor

Table 2: Risks

Priority	Risk	Mitigation
Low	1	Members will keep a healthy lifestyle so as to avoid getting sick. In the case of someone getting sick, tasks will be re-attributed and if needed, talks will be held with Norkart and our supervisor to reduce the scope of the project.
High	2	Everyone committed to be available during the day and to not intentionally ignore each other. We have a designated communication channel on Discord and we meet two times per week.
High	3	Everyone committed to work equally on the thesis. There is a defined process in case that this commitment is broken. We also meet twice per week where members detail what they have worked on.
High	4	This is almost certain to happen. We started working early and the group produces well so far. This gives us a margin of maneuver and we are also giving ourselves a good margin of maneuver in our planning.
Medium	5	We will be creating our documentation underway, along with writing the thesis itself. There is also a recognition that documentation is as important and necessary as the code itself.
High	6	This is one of the major risks and the most likely to materialize. It definitely not as easy as it looks to understand and deliver what the company ask for. It boils down to having frequent meetings with Norkart, preparing meetings, noting down questions for Norkart throughout the week, showing them our work, obtaining their approval before starting work on features and asking for their feedback.
High	7	This is also one of the major risks and the most likely to materialize. Planning will be key to avoid getting off track. We will discuss during our team meetings if what we intend to work on is necessary and worth it. It was outlined early in the project that one of the priority is to follow the 20/80 principle. The idea is that 20% of effort is necessary to achieve an 80% percent quality, but that it takes 80% of effort to achieve the remaining 20% quality. The group agreed early to not try to over achieve and to hold itself in the area of this 20% of effort.
Low	8	GitLab is used.
Low	9	Overleaf is used for all of our documents. This stores them on the Cloud, a Google sheet is used for logging hours and our Software documentation will be stored on the wiki pages of GitLab. Everything will be on the Cloud.
Low	10	Both our source code and our documentation will be on the Cloud. This is our best protection.

Table 3: Risk mitigation

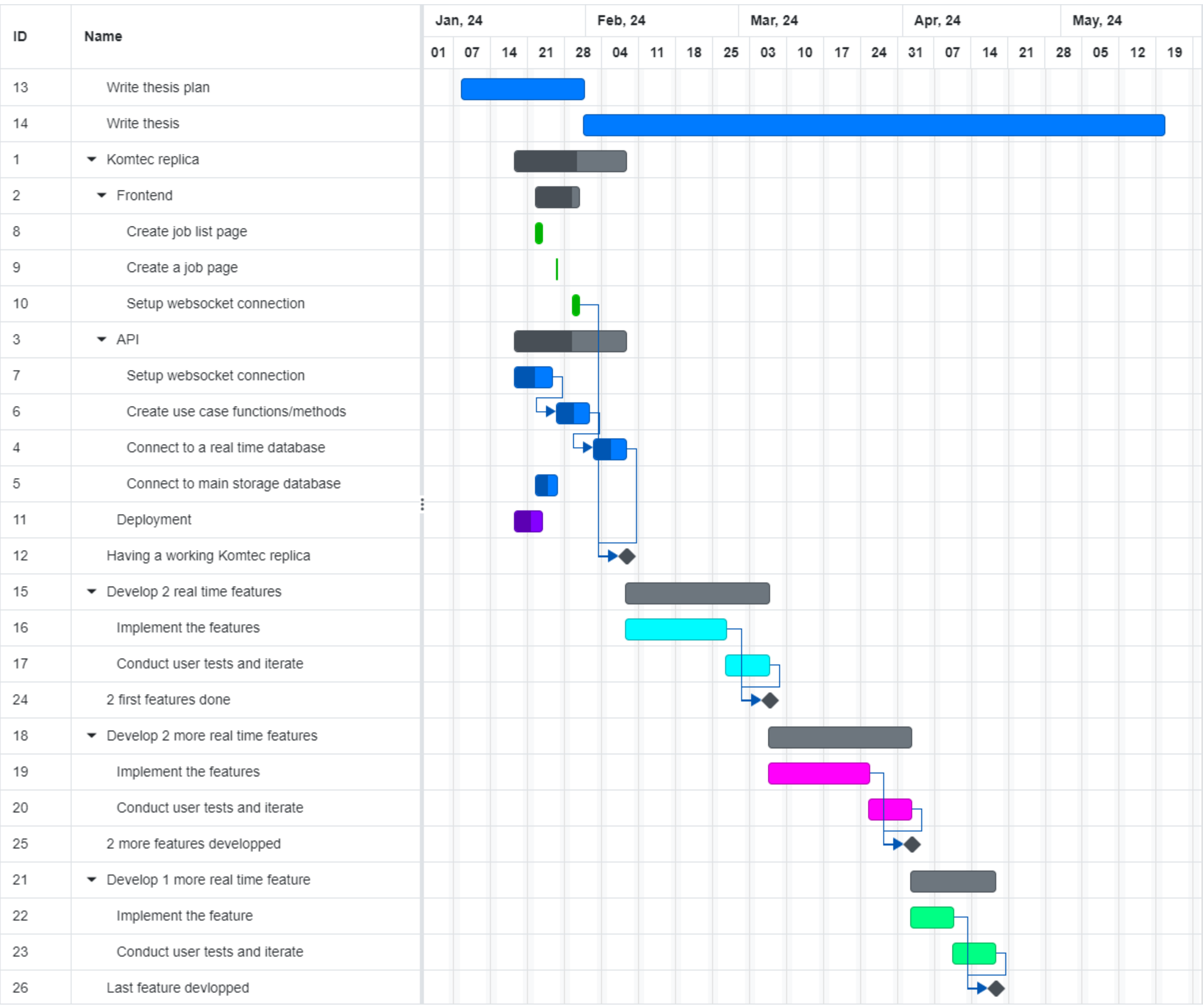
The following risk mitigation table is inspired from the bachelor thesis written in 2022 by Sebastian Lindtvedt, Salvador Bascunan and Dennis Kristiansen at the Norwegian University of Science and Technology (Lindtvedt et al., 2022). The table format is the same.

6 Implementation plan

7 Decision making moments

- By January 7: chose the technologies for the project
- By February 4: chose which features to work on for the first development sprint
- By February 18: chose a user test methodology for the features developed in the first sprint
- By February 25: chose which features to work on for the second development sprint
- By March 10: chose a user test methodology for the features developed in the second sprint
- By March 17: chose which feature to work on for the third development sprint
- By March 24: chose a user test methodology for the feature developed in the third sprint

7.1 Gantt chart with milestones



ID	Name
13	Write thesis plan
14	Write thesis
1	Komtec replica
2	Frontend
8	Create job list page
9	Create a job page
10	Setup websocket connection
3	API
7	Setup websocket connection
6	Create use case functions/methods
4	Connect to a real time database
5	Connect to main storage database
11	Deployment
12	Having a working Komtec replica
15	Develop 2 real time features
16	Implement the features
17	Conduct user tests and iterate
24	2 first features done
18	Develop 2 more real time features
19	Implement the features
20	Conduct user tests and iterate
25	2 more features developped
21	Develop 1 more real time feature
22	Implement the feature
23	Conduct user tests and iterate
26	Last feature devlopped

References

Lindtvedt, Sebastian, Salvador Bascunan and Dennis Kristiansen (2022). *Cloud-native solution for building digital twins*. NTNU Open. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3002601.

Appendix 1
Project proposal

Oppgave 41 - Samtidig redigering og konflikthåndtering i komplekse informasjonssystemer (BPROG) - RESERVERT

Bedrift:	Norkart
Kontaktperson:	Vebjørn Fonstad Leiros
E-post:	vebjorn.fonstad.leiros@norkart.no
Telefon:	46944101
Lokasjon:	Lillehammer
RESERVERT	

Beskrivelse av oppgaven:

BAKGRUNN:
Norkart er et Norsk teknologiselskap som tilbyr markedsledende løsninger innen kommunalteknikk, kart- og eiendomsinformasjon til offentlig og privat sektor. KOMTEK er et totalkonsept for kommunalteknisk forvaltning som dekker fagområdene vann og avløp, brannforebygging, eiendomsskatt, gebyr og fakturering og renovasjon. KOMTEK benyttes i dag i over 300 kommer og i flere enn 50 interkommunale selskaper.

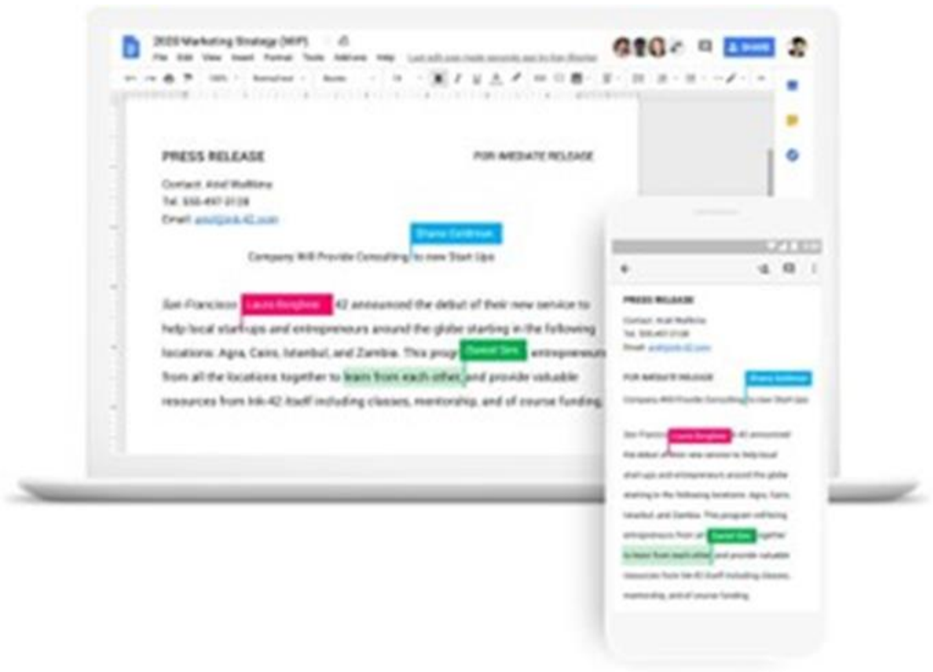
Norkarts visjon er «sammen skaper vi smartere samfunn» – og med dette mener vi å skape et smartere samfunn sammen med våre kunder, våre partnere og alle de menneskene som bruker løsningene i KOMTEK hver eneste dag.

KOMTEK er en skyløsning basert på .NET-teknologi i Azure, med React som hovedverktøy for frontend-utvikling.

OPPGAVER OG DELMÅL
KOMTEK-systemene gir for øyeblikket ikke støtte for samtidig redigering av data, og har begrenset konflikthåndtering hvor "siste endring gjelder" prinsippet anvendes. Med økende behov for hyppige oppdateringer fra flere brukere distribuert over ulike lokasjoner, ser vi mot moderne forbrukergrensesnitt som Office 365, Sharepoint og Google Drive for inspirasjon. Målet med dette prosjektet er å utforske og utvikle løsninger for samtidig redigering tilpasset KOMTEKs typiske arbeidsprosesser.

- Kartlegge moderne løsninger innenfor brukergrensesnitt som håndterer ulike arbeidsprosesser (asynkrone vs synkrone)
- Utforske og dokumentere gode designmønster (UX) for samtidig redigering i komplekse fagsystemer
- Utvikle et designmønster for samtidig redigering på relevante arbeidsprosesser i KOMTEK-systemene (eksempler: «aktive oppgaver», «konflikthåndtering», «oppgavebehandling»)

- Utforske om det er mulig å lage en generisk arkitektur / software-design / maskingrensesnitt som gjør det mulig å bruke i flere fagmoduler/klienter.
- Implementere et proof-of-concept system med webgrensesnitt på en/flere utvalgte reelle arbeidsprosesser i KOMTEK
- Gjennomføre kvalitative brukertester i samarbeid med Norkart



Appendix 2
Project contract

NTNU
Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for datateknologi og informatikk
Veileder ved NTNU: Frode Haug e-post og tlf. frode.haug@ntnu.no 611 35 191
Ekstern virksomhet: Norkart AS Ekstern virksomhet sin kontaktperson, e-post og tlf.: Vebjørn Leiros, veblei@norkart.no 469 44 101
Student: Arnaud Duhamel Fødselsdato: 25. juni 1993
Student: Ghais Mohamad Bachir Dahdouh Fødselsdato: 1. april 1997
Student: Sergei Johansen Fødselsdato: 17. april 1997

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	x
Prosjektoppgave	
Annen oppgave	

Startdato: 8. januar 2024
Sluttdato: 21. mai 2024

Oppgavens arbeidstittel er: Samtidig redigering og konflikthåndtering i komplekse informasjonssystemer

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:
- Lunsj minst 1 dag i uken ved fysiske møter

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

X	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: Dato:
Veileder ved NTNU: Frode Haug Frode Haug Dato:
Ekstern virksomhet: Norkart AS, Innovasjonsdirektør Arild Nomeland Dato: 31.01.2024 Arild Nomeland
Student: Arnaud Duhamel Dato: 31. jan 2024 Arnaud Duhamel
Student: Ghais Mohamad Bachir Dahdouh Dato: 31. Jan 2024 Ghais Dahdouh
Student: Sergei Johansen Dato: 31.01.2024 Sergei Johansen

Prosjektbakgrunn

Eid av den eksterne

KOMTEK: et totalkonsept for kommunalteknisk forvaltning som dekker fagområdene vann og avløp, brannforebygging, eiendomsskatt, gebyr og fakturering og renovasjon.

Appendix J

Project contract

NTNU
Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for datateknologi og informatikk
Veileder ved NTNU: Frode Haug e-post og tlf. frode.haug@ntnu.no 611 35 191
Ekstern virksomhet: Norkart AS Ekstern virksomhet sin kontaktperson, e-post og tlf.: Vebjørn Leiros, veblei@norkart.no 469 44 101
Student: Arnaud Duhamel Fødselsdato: 25. juni 1993
Student: Ghais Mohamad Bachir Dahdouh Fødselsdato: 1. april 1997
Student: Sergei Johansen Fødselsdato: 17. april 1997

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	x
Prosjektoppgave	
Annen oppgave	

Startdato: 8. januar 2024
Sluttdato: 21. mai 2024

Oppgavens arbeidstittel er: Samtidig redigering og konflikthåndtering i komplekse informasjonssystemer

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven: - Lunsj minst 1 dag i uken ved fysiske møter

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

X	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder:	
Dato:	
Veileder ved NTNU: Frode Haug	Frode Haug
Dato:	
Ekstern virksomhet:	Norkart AS, Innovasjonsdirektør Arild Nomeland
Dato:	31.01.2024 Arild Nomeland
Student: Arnaud Duhamel	
Dato:	31. jan 2024 Arnaud Duhamel
Student: Ghais Mohamad Bachir Dahdouh	
Dato:	31. Jan 2024 Ghais Dahdouh
Student: Sergei Johansen	
Dato:	31.01.2024 Sergei Johansen

Prosjektbakgrunn

Eid av den eksterne

KOMTEK: et totalkonsept for kommunalteknisk forvaltning som dekker fagområdene vann og avløp, brannforebygging, eiendomsskatt, gebyr og fakturering og renovasjon.

Appendix K

Meeting notes

Gruppemøte tirsdag 19. Desember 2023

Tilstede: Arnaud, Ghais, Sergei

start tid: 17:00

Vi ble kjent med hverandre i første omgang.

Ghaisd forklarte prosjektet vi skal jobbe på ved hjelp av en liten app. Han skal utarbeide den litt mer å pushe kode på versjonskontrollplatformen vi skal bruke.

Som prosjektstyringsmetode har vi ikke avtalt en formell prosess. Vi vil spørre Vedbjørn om han foretrekker en metode over en annen eller hvis han har noe ønsker der.

Vi vil også spørre Frode hvis det er greit for oss å bruke vanlig Gitlab for versjonskontroll fordi det var opplevd at NTNU Gjøvik sin Gitlab er vanskelig å bruke når det er mye traffik.

Gruppen avtalte å bruke engelsk som arbeidspråk.

Alle er enige i å ha sakssporing og at det er viktig å ha for å gjennomføre oppgaven.

Arnaud sin gruppe i integrasjonsprosjekt hadde ingen prosjektstyringsmetode og ingen sakssporing. Og dette gjorde prosjektet veldig vanskelig. Det var en dårlig opplevelse.

Som sakssporings verktøy er det avtalt å bruke Figma. Det skal brukes Figma som prototypings verktøy. Prototypene skal brukes i sakssporingen.

Sergei foreslo å enten bruke Jira, Azure Devops eller Gitlab for sakssporing. Han foreslo å se over alternativene og å velge det som passer best. Ghaisd foreslo for sin del å ha det enklest mulig og at dette er Gitlab. Arnaud brukte for sin del Gitlab og opplevde at det gikk bra. Og at det er nyttig å kunne koble saker til sammenslåingsforespørsler.

For møtter i løpet av oppgaven, det var avtalt å ha et møtte på mandagene for å planlegge uken. Et møtte på onsdagene for å følge opp, for å se hvordan det går med alle og for å oppdatere hverandre. Og et møtte på lørdagene for å se over uken som har gått og å prate om ting. Dette er noe fleksibelt, møttene vil kunne flyttes ved behov.

Det var avtalt følgende roller:

-Arnaud blir gruppeleder og vil ellers fokusere på backenden. -Ghaisd vil ta vare på frontenden i React. -Sergei vil jobbe på backenden og frontenden ved behov.

Backenden vil være i .NET.

Som forventninger, Arnaud har ikke noe presist mål enn å gjennomføre hele prosjektet. Å gjore alt som står i oppgavebeskrivelsen. Han har også ikke noe presist mål når det gjelder karakteren. Arnaud også regner det å jobbe på oppgaven som en fulltidsjobb. Bortsett fra at gruppen skal ha et fag til og at han skal være student assistant i faget Advansert Programmering. Han vil måtte gi tid til stillingen men skal ellers fokusere på oppgaven som fulltidsjobb.

Sergei ønsker særlig å lære noe nytt. Ved å legge innsatsen til å lære skal karakteren følge.

Ghaisd, for sin del, ønsker å holde seg til løsningene som er enklest. Å holde alt enklest mulig og å bygge derfra. Han også ønsker å ta master så han sikter helst for en B eller A for å få opptak.

Bruppen ble enig i ordentlig prosjektstyring og sakssporing, det er en veldig god start på oppgaven.

Som oppgave i feriene ble det foreslått å lage en backen struktur i .NET som kan kobles med applikasjonen Ghaisd lagde i React.

Sergei blir opptatt i løpet av feriene og derfor kan ikke garantere at han skal jobbe på det men Arnaud vil kunne jobbe på det.

slutt: 17:50

Veiledningsmøtte onsdag 20. Desember 2023

Tilstede: Frode, Arnaud, Ghais, Sergei

start tid: 15:15

Starten av bacheloroppgaves prosess er i januar. I 31. år ha Frode aldri hatt møter med studenter før juleferien.

Frode er ledig hver tirsdag og torsdag før lunsj. Før klokka 12.

Han kan gi oss 1 time per uke.

Frode skal ikke være rådgiver om programvaret eller teknologien. Han kan egentlig ikke noe om Norkart sine programvarer.

Han skal hjelpe oss med rapporten og planleggingen. Men ikke om teknologien selv.

Vi skal få 2 timerslynkurs 10. januar fra Tom om programvareutvikling.

Vi fortalte at Ghaisd jobber i Lillehammer og at han skal delta digitalt. Frode svarte at det går fint men at vi bør prøve å møte fysisk mest mulig og at det er ikke corona lenger.

Vi har planlagt å ha møter digitalt men vi skal se hvordan det går og om noe bør endres underveis.

Vi spurte om det er greit å bruke vanlig Gitlab for prosjektet. Han fortalte oss å bruke idi sin Gitlab. Det skal vi gjøre.

Vi spurte om hva vi bør fokusere på: dokumentasjon, utvikling, etc. Frode svarte at Tom skal fortelle om alt dette her.

I Januar vil vi måtte lage en prosjektplan. Vi vil se på koden til selskapet og det de ønsker og så videre og lage en prosjektplan. Fordi oppgavebeskrivelsen er ikke ferdige spesifikasjoner. Vi vil måtte utvide det.

Når faglærerne ser på en oppgaveforslag, de anslår om det er en oppgave som passer for 3 eller 4 stykker. De bare vurderer ut av forslaget.

Så hvis det er litt lite, studentene må finne en måte å utvide den selv her og der. Ellers blir det ikke A eller B for å si det sånn.

Nest møte med Frode er 11. januar klokka 11:00.

slutt: 15: 50

Gruppmøtte torsdag 4. Januar 2024

Tilstede: Arnaud og Ghais

start tid: 13:00

For kunstig intelligens, den beste måten skulle være at, hvis det kopieres kode, og klart legge som referanse i oppgaven og å legge samtalen ved rapporten. Ellers skal det være nok med å beskrive hvordan vi har brukt det, som utviklingsverktøy og sånn.

Planen nå er å begynne fra bunen av og å utvikle derfra. Derfor skal vi lage grunnleggende kode i både React og .NET i hver sin repo og å bygge opp derfra.

For oss vil det være best for å holde orden i oppgaven vår at vi har to Git repo: en for frontenden og en for backenden.

Da er planen til neste lørdag å ha grunnleggende, hello world kode i REACT i front-end repo og å ha en grunnleggende, hello world, API i .NET i backend repoet.

Det er også planlagt å undersøke for mulige samtidig redigering og konflikt håndtering trekk på nett og andre mulige produkter som gjør det å se hva de gjør og se hva som er nyttig.

Det var også gjentatt at Ghaisd skal være ansvarlig for frontenden, og Arnaud og Sergei backenden.

Neste revidering og planleggingsmøtte blir neste lørdag klokka 7 pm.

slutt: 13: 30

Møtte med Norkkart torsdag 4. Januar 2024

Tilstede: Vebjørn, Arnaud, Ghais, Sergei(på nett)

start tid: 10:00

Meningen med oppgaven for Norkart er at vi skal finne en løsning til samtidig redigering og konflikthåndtering som har verdi og som gir en god brukeropplevelse.

Det er mange løsninger som er kjent og som kan brukes. Men meningen med oppgaven vår er å finne på noe som er brukervennlig og verdiful.

Google docs er til inspirasjon fordi den viser i samtid hver som redigerer. Den viser hvor andre brukerne redigere med navnet ved siden av markøren. I tillegg til å vise endringene i samtid. Dette var noe kulle punkter som kunne være til inspirasjon for oss.

Komtek er en løsning som brukes for å løse kommunal oppgaver. Det er oppgaver hvor en saksbehandler går gjennom. En typisk oppgave er bytting av en vannmåler.

En rørlegger vil bytte vannmåleren og etter det sendes en sak til kommunen for behandling. Saksbehandleren åpner saken og går over for å sikre at byttingen er ordentlig dokumentert. Det er bare verifisering altså.

Det kan også være andre oppgaver.

En oppgave i KOMTEK er tildelt automatisk til den første brukere som åpner oppgaven. Men det er mulig for en bruker å frigi oppgaven. Da er den første neste brukeren som åpner oppgaven som får den tildelt.

Rørleggere også bruker Komtek. Det er de to brukergruppere: rørleggere og saksbehandlere.

Saksbehandlere er brukere som er typisk sånn cirka 50 år gammel og har ikke mye teknologiske erfaring eller evner. Og de følger fremgangsmåten ganske slavisk.

For rørleggere er det mer variert med flere unge rørleggere. De også bruker mye mmobil i feltet. Det derfor også kunne være en mulighet for oss å utvikle noe som er egnet for mobil. Da må vi se om omfanget til oppgaven og veivalgene vi skal ta.

Det er også noen sikkerhetskrav knyttet til produktene. Det er også GDPR regler man må følge hvis man bruker personlig informasjon.

Derfor er det beste for oss som bachelor gruppe å bygge opp noe som er mest eksternt mulig for å unngå mulige krav om konfidentialitet og å måtte skjulle noen informasjon i rapporten vår.

Det er også viktig for oss å vite at opplysningene som beskriver en oppgave faller ikke inn i vår bacheloroppgave. Det som faller inn er opplysningene som brukes for å løse oppgaven.

Når det gjelder samtidig redigering, det som er viktig at vi får med er å se at oppgaven er åpen av mange saksbehandlere, og se i sanntid redigeringen til andre brukere og å håndtere konfliktene som oppstår.

Vi har frihet til å lage den oppgaveløser selv som vi ønsker det. Vi har frihet til å bygge opp brukerinterfacen vi ønsker eller mener er best.

Vi bør også se på både backend og frontend løsninger.

Altså, for oppgaven, for å ha noen som er mer fullstendig, bør vi ha en backend infrastruktur, et trekk om samtidig redigering og et trekk om konflikthåndtering, og en bruker interface.

Første fasen i prosjektetet vårt er å kartlegge mulige løsninger. Det er uansett det man gjør i første måneden av oppgaven: forskning. Så vi bør bruke mye tid på å vurdere og å gjøre veivalg tidlig. Vi bør designe litt og lage noen små prototyper for å prøve ut ting og å vise frem, men vi bør først og fremst forske og rapportere resultatene våre. Hoveddelen er forskning.

Vi ble anbefalt å bruke MaterialUI for å designe og å lage prototyper. Eller å bruke det for frontenden.

Det som skal gi mest verdi til Norkart er å programmere i .NET for backenden og REACT.js for frontenden.

Hvis vi fokuserer bare på samtidig redigering og konflikthåndtering så er oppgaven tynn for oss. Så vi kan godt ta for oss å lage en hel oppgavesystem. Med en backend, samtidig redigering og konflikthåndtering trekk og en frontend del. Det beste er å utvikle noe som kommer til å ligne KOMTEK.

Rørleggerne også bruker en annen platform som heter Entreprenørdialog. I denne platformen er det også aktuelt med samtidig redigering og konflikt håndtering. Så det er egentlig best at vi utvikler noe som er uavhengig av en platform. Eller som kan lett brukes av mange forskjellige platformer. Det er derfor det er tenkt om en API til backenden.

Men i tillegg til samtidig redigering og konfliktehåndtering kunne også Entreprenørdialog ha mulighet til å planlegge oppgaver. Å ha en scheduler.

Entreprenørdialog har også en fleksibel format som gjør at man kan utvikler maler til mange forskjellige oppgaver. Det brukes en JSON format til det.

Altså, det er tenkt om en API til backenden for oppgavebehandling. Et system for samtidig redigering og konflikthåndtering. Dette også innebærer et bruker system. Som business to business bruker Norkart Azure Active Directory. Eller Microsoft bruker systemet. Man kan bare bruke Microsoft kontoer for å bruke teknologiene til Norkart som er business to business. For business to consumer så er det helt åpent. Det kan være Discord, GitHub, Google og så videre. Creativity app engang. Det er noen løsninger hvor det kan legges til flere bruker systemer enn det som er gitt allerede. Vi bør i hvertfall bruke et eksternt brukersystem og ikke lage vår egen. Fordi da er sikkerhet og alt tatt vare på.

Bør vi ha en endringshistorie brukerne kan se på? Dette kan være litt for mye fordi det er ikke samme frihet i KOMTEK enn i Google Docs. Fordig Google Docs har egentlig bare en input felt. Det er hele dokumentet. I komtek er det flere input felter. Fordi hver skjema felt er sin egen felt med sin egen historie. Derfor kan det bli mye å holde oversikt over alle endringene.

Kanskje er det ikke viktig å ha så nøye kontroll over historien. Det vi ønsker og vår jobb å finne ut av er å finne hva i dette gir verdi, kanskje først begynne med det som er enklest, og å utvikle trinnvis på det. Det er egentlig veldig lurt.

Fordi det kan være kort vei til verdi og det kan også være en lang vei med mye arbeid for lite verdi. Vi må på en måte finne en balansegang i det. Og helst først sikte for det som gir mest verdi ut av minst arbeid mulig. Og bygge derfra.

Er det meningen at, i utgangspunktet skal det ikke være samtidig redigering? Er det meningen at det skal være bare en saksbehandler for hver oppgave? Svaret til det er ja. Så det kunne være en ide å låse oppgaver, eller å gi varsler, og så videre. Det er noe å utforske der.

Bør vi ha to repoer i Gitlab? En for backenden og en for frontenden? Vi ble anbefalt å bare ha ett repo fordi det er det som er lettest for kontinuerlig integrering. Men det er ganske opp til oss å styre det som vi ønsker det.

Sist ba vi om generelle råd for oppgaven. Vi ble anbefalt å dokumentere alt vi gjør. Bare med det skal vi går et langt stykke fram. Dokumenter, dokumenter og dokumenter alt. Vi bør også ikke nøle om å skrive om det som var feil eller det vi sleit med fordi det er også noe verdiful å skrive om i rapporten. Vi bør også forkalre hvorfor det var feil.

Vi bør også jobbe jevnt og trutt. Vi bør begynne å jobbe fra dag 1 og vi bør særlig skrive underveis. Vi bør egentlig jobbe på rapporten samtidig som vi jobber på programvaret.

Vedbjørn spurte om det vi skal gjøre i oppgaven nest. Det blir sikkert en gruppekontrakt og etter det skal vi jobbe på en prosjektplan. Vi skal også ha lynskurs om prosjektstyring 10. januar. Vi hadde et litte møtte før jul men der ble vi ikke fortalt mye om prosessen. Vi skal møtte vår veileder 11. Januar etter lynkurset. Altså, 10. og 11. januar skal vi sikkert få bedre oversikt over oppgavene fremover. Det vet vi ikke helt enda.

Som regn praktisk skal det sikkert gå greit å kunne møtte hos Norkart hver onsdag. Vi skal få et grupperom, men med ingen utstyr. Og det går helt fint for oss. Det skal også ordnes kantina for oss.

Vi skal ha et møtte med Vedbjørn om uken. Den skal vare opp til 1 time. Møttet skal også være på onsdagene.

Det beste for Vebjørn er at vi samler spørsmålene våre og stille dem på møttene. Hvis vi har et spørsmål går det greit å spørre med en gang men hvis det er mange så blir det for tungvint og da bør vi samle dem å stille dem på møttene.

Altså, oppgavene fremover for oss er å begynne å se etter mulige løsninger og mulige trekk angående samtidig redigering og konflikt håndtering.

slutt: 11: 00

Gruppmøtte lørdag 6. Januar 2024

Tilstede: Arnaud og Ghais

start tid: 19:05

Dette møttet var dedikert til tilbakemelding angående siste uken.

Tilkbakemeldingen var positivt overalt. Vi er egentlig kanskje kommet lengre enn vi bør så tildig i prosessen. Så det er bra.

Møtet vi hadde med Norkart siste onsdag var veldig bra. Det var en god kjemi mellom gruppemedlemene og selskapet.

God arbeid var allerede gjort. Arnaud har pusha en veldig grunnleggende .NET, hello world API og Ghais også har pusha en veldig grunnleggende, hello world React applikasjon.

I morgen så skal vi slå det sammen med main grennen i Gitlab og planlegge uka videre.

ChatGPT var brukt for å gi ideer om planlegging og løsninger. Kanskje var det for mye her.

Det som er viktig å si er at Ghaisd er en erfaren React utvikler. Han kan rammeverket godt og derfor bruker han ChatGPT mye i hverdagen.

Arnaud må lære seg .NET fra bunnen av. Også React. Så han skal sikkert lære mer ut av strukturerte tutorialer.

Sergei var ikke med på møttet og hadde ikke anledgning til å jobbe med gruppen onsdag og i løpet av uken. Vi satser på at det hadde bare med den ene uken å gjøre.

Ideen å satse på per nå er å ha en database som håndterer samtidig redigering og konflikt løsing. Det er det som er enklest og det er det vi satser på nå. Det er noe som skal diskuteres med Vedbjørn fra Norkart. Fordi nå bruker Norkart en Azure database som er egnet til samtidig redigering. Vi bare får se om det er kompatibel med det de gjør nå og at det er ok for dem å betale. Firebase er gratis. I hvert fall for oss. Kanskje er det ikke for bedrifter eller etter et viss bruksnivå.

Vi møttes i morgen for å legge planen for uken. Vi møttes i morgen klokka 19:00.

slutt: 19:20

Gruppmøtte søndag 7. Januar 2024

Tilstede: Arnaud og Ghais

start: 19:00

I dette møttet så planla vi uken som kommer.

For frontend prosjektet var det planlagt 3 oppgaver: lage grunnleggende React kode, utrulle applikasjonen til Firebase og tilkoble appen til backend delen.

For backend serveren var det planlagt 2 oppgaver: utrulle applikasjonen til render og tilkoble backenden til databasen.

Til neste uken skal det sendes melding til Vebjørn for å se om det er mulig å komme tirsdag i stedet for onsdag fordi det er lynkurs på campus vhor vi bør delta. Hvis ikke holder vi det til onsdag og Arnaud vil bli med digitalt. Det er fortsatt planen å møttes på Norkart hver onsdag.

Ellers møttes vi på torsdag med Frode. Klokka 11 på den 11. Januar 2024.

Sergei er litt usikker om han vil være med på onsdagene på Norkart på grunn av mangel på utstyr og ekstra skjermer der.

Nå skal det som var gjort bli verifisert og merget.

Det var også avtalt at den som verifiser også merger.

slutt: 19:39

Lynkurs i prosjektstyring 10. Januar 2024

start tid: 12:00

For bacheloroppgaven er det i utgangspunktet ansvaret vårt i det hele.

Neste innlevering er en prosjektplan.

Rapportant vil handle om å dele det vi har lært over hele oppgaven. Det blir ikke en rpport hvor vi beskriver det vi har gjort hver dag.

For prosjektplanen så skal vi bruke en mal som var brukt i mange år. Det er ikke tvang til det men det er et godt utgangspunkt.

Ikke klipp og limm.

NTNU open har mange forskjellige bacheloroppgave.

Vi skal lever avtaler på papir til Line.

Det er to nye ting for oppgaven: kunstig intelligens og bærekraft. NTNU skal prøve å ruste oss til disse to i verden.

Det er mange kilder vi må forholde oss til. Vi bør lese emnebeskrivelsen. Sånn er det. Det er bare til oss å ta det beste av alt.

Vi bør begynne tidlig på rapporten. Vi bør dokumentere våre veilvag **underveis**.

Dette er ikke konsulentoppdrag. Vi må prioritere læringsutbyttene. Dette innebærer rapporten.

Det er vurderingskriterier som følges men ikke slavisk. Det er helheten som ses på til slutt.

Vi kan være ambisiøse men vi bør være realistiske. C er en god karakter ved NTNU. Gjennomsittlig karakter for bacheloroppgaven er C. 43% får C.

Vi har nesten ingenting i timeplanen.

Vi bør derfor lage en timeplan.

Veileder får ikke sensurere vår arbeid.

Vi må skape får eget eierskap til bacheloroppgaven. Det må drøftes og avklares med oppdragsgiver.

Møter med oppdragsgiver må avtales på forhånd. De er opptatte folk. Vi bør bruke muligeter fra oppdragsgivere om opplæring og alt.

Vi må plannlegge. Vi har ikke noe gode unnskyldninger for å ikke planlegge. Planleggingsprosessene er uunngåelige. Selv om det kan fravikes fra planen etterhvert.

Bakgrunnen for et prosjekt plan er for å legge ned en kontekst.

Vi trenger ikke å ha noen teori inføring som helst i det vi produserer.

Effektmål er hva oppdragsgiver ønsker ut av vår arbeid. Resultatmål er det vi skal levere.

Våre objektiver må være konkrete. Det må være målbar.

Rammer kan være lovgivning. Det er restriksjoner vi må forholde oss til.

Problemområde er beskrivelse av problemstillingen. Problemstillingen beskrives der i sin helhet. Avgrensning er der vi skriver om det vi ikke skal jobbe på.

Det er ikke noe fasit på det men mellom 10 til 12 sider kan være noe godt.

Vi bør sette opp rutiner og grupperegler. Vi må sette opp kontrakt og legge den som vedlegg.

Hvordan skal vi føre timer.

Vi må ha beslutningsmøter og statusmøter.

Vi bør legge mye innsats på plannlegging. Arbeidet avhenger mye på grunnlaget som legges med planen. Og å skrive rapporten om oppgaven. Dette er viktig ting.

Det er også eksempler av prosjekplaner i undervisningsmateriells.

Det er mange prosjektplaner på NTNU open. Vi kan lese mange av dem.

Vi må legge en plan for testing og kvalitetsikring. Vi må ha det i prosjektet.

Vi må kartlegge og vi må begrunne det vi gjør. Vi bør gjøre det underveis.

Vi må ha risiko analyse. Ikke fordypt, med hovedtrekk.

Som neste oppgave er det for oss å undertegne grupperegler, avatale mellom oppdragsgivere, studenter og NTNU. Kanskje en avtale om konfidensialitet.

Vi kan ikke bruke kunstig intelligens for å skrive rapporten. Vi kan bruke det til språkvask.

Vi må fortelle om det vi har brukt det til.

Vi må henvise hvor vi har brukt kunstig intelligens i arbeidet.

Vi må også ha bærekraft.

slutt: 14: 00

Gruppmøtte onsdag 10. Januar 2024

Tilstede: Arnaud, Ghais og Sergei

start tid: 19:00

Arnaud viste det han har gjort siste dagene med utrulling.

Så

Spørsmål å tille :

- Vise det vi har gjort så langt.

Arnaud viste det han gjorde så langt med utrullingen. I Render, GitHub og Gitlab.

- Er det greit å ha møter på torsdag hos Norkart? Ikke nødvendigvis alltid med Vedbjørn men i hvert fall for å jobbe på oppgaven? Det er det som passer best egentlig med andre fag og seminarer.

Ja dette går fint. Arnaud skal spørre Vedbjørn i en epost i dag.

- Er det normalt at jeg må kjøre kommandoen "npm install –save-dev vite" hver gang jeg laster ned React prosjektet? Hvis ja, så kan hello world prosjektet merges.

Ja det er noe normalt. Sergei skal se over det og skal merge det uansett.

- Skal vi også ha en kontinuerlig deployment av React prosjektet?

Det er noe Sergei vil jobbe på. I utgangspunktet så skal vi utrulle frontenden på Firebase. Etter det skal databasen tilkobles til backenden. Da er vår utviklingsmiljø i gang.

- Jeg tenkte å undersøke for å ha pipeline tester. Vi må ha noen testinger og vi må legge en plan for det uansett. Går det greit?

Dette er et helt felt i seg selv og vi bør ikke gå inn i det. Kanskje kan vi ha for nå bare build: å sikre at prosjektet vårt bygges og bare det.

- På hvilket språk vi vil skrive? Vi kan egentlig velge å ha det på engelsk eller norsk. Det er opp til oss.

Da bytter vi til engelsk for alt. Vi hadde det på Norsk siden 4. januar 2024. Nå skal vi bytte til engelsk.

- Tom snakket om å ha SkyHigh ressurs og at vi kan søke om det. Jeg tenkte vi kunne kanskje bruke det hvis det er nødvendig for å kjøre pipeline tester? Det er en per gruppe som søker for det.

Det trenger vi ikke. Hvis det blir nødvendig så kan vi se.

- Vi må lage gruppe kontrakt med innleveringsfrist 31. januar. Jeg kan jobbe på det ut av det som forrige grupper har allerede gjort før.

Dette var snakket om og gruppen vet nå om 31. januar fristen. Arnaud skal jobbe på grupperegler.

Gruppereglene kan også kobles til risikoanalysen. For å forutse vha som skje hvis en uhell skjer.

- Vi må begynne på å lage en prosjektplan som skal også innleveres senest 31. januar. Fordeling av oppgaver? Sergei tenkte å lage gant chart (vi må ha det) og timeføring.

Vi skal dele arbeid rundt prosjektplanen neste søndag.

- Vi må velge en utviklingsmetode: Scrum, fossefall eller noe annet.

Det skal være en blanding av scrum og kanbal for utviklingen og for brukertesting så skal vi ha noe som ligner mer forskningsmetoden.

- Vi må også signere en avtale med Norkart om oppgaven og prosjektet.

Det skal Arnaud sende til Norkart, også når det gjelder møttene.

- Vi må lage dokumentasjon. Hva med å bruke wiki sidene til GitLab?

Det kan vi bruke. Dette er for dokumentasjonen til programvaret.

Vi vil også måtte ha timeføring.

- Vi ble anbefalt å lese en bacheloroppgave hver, har vi lyst til det?

Ja det kan vi gjøre.

Saker som er knyttet frontenden skal være i frontenden repository. All resten skal være i backenden repository.

Arnaud kan lage som tags.

slutt: 19:50

Meeting with supervisor 11 January 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 11:00

The thesis is not so well described by Norkart, we should detail it with Norrkart.

Make good group rules now, when everything is going fine.

We will also sign a project contract.

The first 2 points in the project plan should be we written so that it can be directly put in the final report.

We should also have best specs description as possible because it will also be in the final report. It should be done in partnership with Norkart.

We should have hours writing. It will be attached to the final report.

The supervisor wants status reports. It is 1 to 2 pages. We don't need meeting minutes. It is attached to the final report though.

Development method. We will most likely use scrum. If we do scrum, we should also have good specs description nonetheless. Scrum should not be used to loose on the work.

We definitely should have a project leader. Which is different then a scrum leader.

If we don't need to meet, send a message the day before. It will be thursdays at 11:00 for now. If he should look at documents, we should send it at least 24 hours before.

We can do the defense in english.

The project plan is something that we should follow. We should switch point 1 with point 2 in the template. If we do a good job in the plan, the report will be easy to write.

2 scope: this is what should attract the attention of the reader. This is what should see the thesis to the reader.

Fagområde is what the thesis will be about. We have to explain that. We have to sell and describe what we will do. Effektmål is what Norkart gets out of it. Resultatmål is what is done at the end. Læringsmål is what we will have learned out of the project.

Bakgrunn is why Norkart came up with this thesis for us.

We should work on the project plan now, but not just work on that.

Begrensning is what we couldn't do. Avgrensning is what we will not do.

end time: 15: 50

Group meeting 14 January 2024

Present: Arnaud, Ghais, Sergei

starting time: 19:00

- Retrospective. How did the week go? Arnaud: - There was a bit of issues during the weekend so Arnaud Will be more available during the weekend - Would like to be working more with Sergei on campus. Is available most days of the week. - Would like to all meet on thursdays at Norkart.

Sergei: - Seminars will be on mondays. - The week started yesterday for the bachelor. Now he can fully dedicate to the bachelor thesis. Happy to announce that now can focus full time on thesis. - Looked into C. - Text channel for links would be nice to have. - Channel for ressources and for links. He can do that.

Ghais: - It was a bit bad that we couldn't meet in Lillehammer. But weekly meetings from now on is good news. - Meeting with Frode was good, we will not be asking him technologie related questions. Just academic questions - There was this bug related to the use of the frontend with backend with two different urls and making it work together, but it got fixed. - Went through some previous thesis and we could definitely use them. It was a good week. - Even if we didn't meet in Lillehammer we met on wednesday so that was good.

Arnaud and Sergei will meetup tomorrow. Sergei will let Arnaud know for the time.

For the group rules, it was went through and modified. The group rules file contains all of the rules. The file rules_summary contains a summary of those rules from ChatGPT. It is mentionned in the document that it is a ChatGPT summary. And Frode will be consulted on it because Arnaud is a bit nervous about using the summary.

It is ok with Ghais if Frode disapproves the ChatGPT summary.

Should we have a scrum master? No. Let's not do that. That would create unnecessary work. Let's chose a development method and get on with it instead. We will have retrospective meetings and planning meetings anyway.

Fo the Signing of the project contract and the declaration of confidentiality, Arnaud is waiting for feedback from Vebjørn. And we will move from there.

Sergei plans to come to Norkart on Thursday so we will most likely be able to sign the contract and declaration of confidentiality.

The database was not set up. Arnaud will do this task. After that, the team proceeded to plan the tasks for the week. Issues were created live.

Arnaud will also make a continuous deployment system for the frontend.

Arnaud Will try to look at some pipeline tests, if it's feasible without too much issues. So that we can test the end points of the API.

This is not high on the priority list and not much energy should be put into that, but it sure is good to have. It would be about looking at tests for endpoints and for websocket connection.

For the big picture strategy, we will now create a backend with a real time database. This will be presented to Norkart. After that, other backend solutions can be implemented. To expand because we have to map the possible solutions here. This will also depend on what Norkart wants. But for now, it is a real time database.

At the same time, we will work on the frontend as well. For the front end, we can implement many possible solutions and also what Norkart wants as well.

We can also have user tests most definitely. We should have user tests for that matter.

Other tests should also be technical and specific. This could also be looked up. How should APIs be tested and how should frontend solutions be tested.

Should we also have some functionality on the job itself? Like locking the job, giving an alert, having a new setup on hover, a different color if it is in use, etc.

For the meeting with Norkart, we will ask if we should test some real time solution at the job list level. Before entering the job. About showing a warning, and perhaps the name of the person inside the job. And then ask about Firebase real time database and CosmosDB.

Lastly, Arnaud found in the thesis "Cloud-native solution for building digital twins" that there is some information about realtime update of information. So this is a potential source of information.

slutt: 20:44

Meeting with supervisor 16 January 2024

Present: Arnaud, Ghais, Sergei & Frode

start time: 11:00

Use AI to validate, and not to rewrite.

Be more concrete. Instead of reasonable time, say the actual time.

Instead of the intentional ignoring rule, use online activity rule: for example everyone must be active on discord atleast 3 times a week.

Rewrite the rule about written and oral warnings

The next meeting with Frode will be next tuesday at 10 am.

stop time: 11:15

meeting with Norkart 18 January 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:00

We first reported a bit to Vebjørn what we have done so far. We showed him the job list that was created.

In reaction to that, Vebjørn told us that now it's planning time, and we should not code anything before we have a plan in place. We can code now, but we should not lock ourselves in what we coded and we should be cleear over the fact that what we have coded so far could be thrown out of the window.

So right now, we should focus on making the plan.

So right now, we should think about what we want to solve and what we want to work on. We have a job solver right now in which we need to handle collaborative editing. We can either make a whole solution or just work on the collaborative editing part. This is up to us. But in any case it will be about collaborative editing.

Komtek can be a template and a starting point for specs: We need a user login system, a job solver with a job list page, a page inside the job, a button to submit the job and a collaborative editing features. We need to have that. For the collaborative editing, it is up to us to solve the issue and find out about it.

The idea would be that we will have one main solution, or one main development environment, which would be this replica of Komtek, and then we will be free to develop features related to collaborative editing. And there, we will have the opportunity to build small or bigger features and have them user tested.

Arnaud asked and mentioned that, between creating a Komtek duplicate, it would be more valuable to develop a solution that could potentially be integrated directly in Komtek. That would be more valuable. Vebjørn told us that doing that would involve confidentiality and so it would not be benefecial to us. So it is actually better to create a working duplicate.

It would also be positive to have solutions for which we conclude that are not feasable or do not provide a good user experience. This would be as valuable to Norkart as a solution that works. It would still be instructive.

Norkart will like that we give them demonstrations of the features that we develop, and then we can receive feedback on the spot from Vebjørn for improvements and/or proceed with some user feedbacks from their customer support team.

Hopefully, in the best scenario, the solution that we would develop would be generic and easily transferrable between applications. We will look into that for sure and hopefully, by nature, it should be easily transferable because it will boil down to establishing a connection to the API that will handle the real-time data that will be of value.

Right now, the technology used by Norkart for collaborative editing is SignalR because it's the most convenient to use with DOTNET.

For the database, it is up to us really to use the database that we would like the use. The type of data that is used by Komtek is very relational. It is basically strings. And that would make it pretty well suited for a relational SQL database.

Norkart would also like if the solution could be decoupled from the database choice, so that it would be easy to change database for a customer.

But at the end of the day, it is up to us to find out. So a real time database or a SQL database would be fine for Norkart.

Vebjørn could most likely get us an Azure ressource if we need to because right now there are mainly two choices for a realtime database: Firebase and Azure Cosmos DB. But we should not lock ourselves on one specific database to be as flexible as possible for customers.

For the collaborative editing feature that we will create, we will essentially be free to explore each and every processes of Komtek. We will stand pretty free to work on whatever we would like.

Vebjørn also told us that many interfaces actually access the Komtek solutions, there are users and other APIs. If we could figure out a way to track all of these different interfaces, when they enter a job, that would be good.

We should look at design but we should not spend too much time on it. It should be given as much weight as the weight it had in our studies, which is not a lot. But yes it would be good to get some good design practices and implement those.

Otherwise the main task will be to develop features handling collaborative editing.

So what should we be working on right now? Right now, we should work on the project plan. For now. Describe what we would like to work on and we can also send it to Norkart and get feedback on it. And if we can make a solution that works on many software, it is a good thing.

When it comes to rules regarding collaborative editing, we should start small, and expand from there. So our first feature should be as simple as possible. And so for now, the simplest feature is to give a warning to user entering a job if there is already someone in the job. That's the simplest thing we can work on for now.

And then it goes down to what is useful information to give a user in the collaborative editing. Who can decide to save the task. This could also be possible features to develop.

Our features should not be chatty solutions. It should be one warning, nothing continuous with back and forth messaging.

We should also use version numbers for the features that we develop.

It was asked if preventing access to a job except to the person to whom it is assigned would be something worth exploring. We were told that everyone should still have access to all of the jobs, even if they are assigned. But inform and/or restrict.

In the task solver, there is a possibility to upload files. But this is not something in which we should get involved. This should be left as is, unmodified.

When it comes to feedback from Vebjørn, if we would like to have him give us feedback, we should provide him some sketches and screenshots. Vebjørn is not so interested in looking at our source code. He doesn't really want to do code review so much.

But he definitely wants to see our demos. And we can show him along the way.

We should make some objectives, with many demos. And show it underway. This would make our development a bit more as a waterafall model. And in a bachelor thesis with milestones and plan, it is a bit unavoidable but we do have the possibility to change the plan underway, especially in the context of an agile development method.

We will have to do some research on the solutions. But actually, it matters more to have user acceptance then to have a research based solution.

We will mainly ask academic questions to Frode, but We should not be scared to ask Frode about solutions and more technical questions as well.

We asked if it would be a possible solution to not have a backend or an API at all for that matter. And have everything move between the real time database and the user interface. They use SignalR to deliver status messages. The task solver should have an API. This is something that we are asked to have.

Norkart has been working on a collaborative editing in Komtek and it will be deployed soon. Someone inside a job will receive a notification when another user submits. And so we should not work on that feature ourselves.

Vebjørn reiterated that We should only develop one software, but if what we create could work on other products would be best. This is something to have it at the back of our head. Creating a little library or something like that.

Norkart uses SignalR. It is the simplest solution.

Arnaud asked if it would just be ok to just go for the technology that Norkart uses right now: SignalR and so on. The answer is that, when it comes to the technology choices, we can pick our battles and not every choice has to be researched and justified thoroughly.

When we will make a choice choose, we will justify the choice, and see how deep we want to reasearch and defend the choice.

-In the thesis description, when it comes to the products that already exist out there, do you want us to deliver anything to you? Same thing for best design practices?

The answer to that is no. But this is something that we will include in our thesis.

-When it comes to the more technical part, would you like to have anything at the job level? Or would you like us to do some research before and then submit you anything? How would you like to go at it?

Yes, this is something that we are free to explore.

-Anything new with the contract?

Vebjørn did not receive anything about the contracts. He asked us for the deadline and I told him the deadline is January 31. It should be fixed and arranged by then.

This was a very productive meeting that greatly clarified the meeting for us.

Is the lack of collaborative editing actually an issue? Yes. The main issue is that two case workers will see a job that is not assigned to anyone. Both will click on it. Both will work on it. And then one will submit before the other. The other user will not know something was submitted, and he will submit himself, and his submission will override the other case worker's submission. And then someone worked litteraly for nothing. This is a real issue.

end time: 11:08

Group meeting 18 January 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00

For this meeting we kind of debriefed the meeting that we had with Norkart.

We are asked to work on solutions tailored to Komtek. And so we should have a Komtek replica. Not an exact replica, but something that mocks Kommtek's processes. This is not a choice.

We have to have an API. This is a request from Norkart.

And so now, the plan is the following:

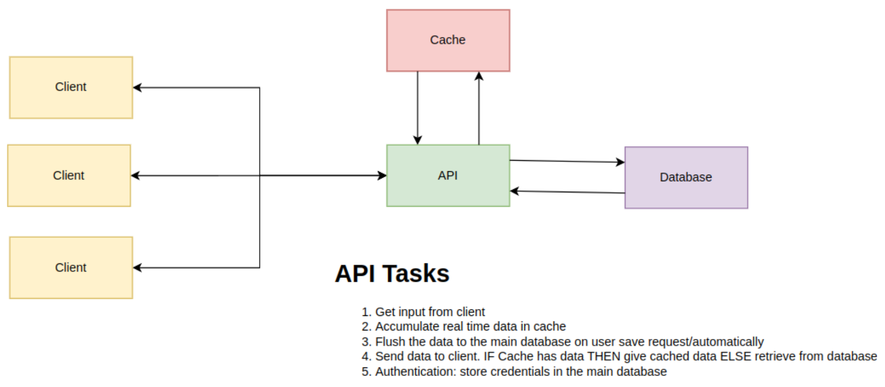
Once the Komtek replica will be done, the main project will be to implement and test collaborative editing features. And user test them. We should keep track of all of our version. And the final product that we would submit would include all of the versions that we worked on.

When it comes to the set up of the Komtek replica, our choices of technology is not so important. This does not matter too much. What matters most is the collaborative editing feature that we will work on.

However, it would be good to have something that could be turned into something generic. Some sort of library that could be used out of the box. Same thing for the databases. But we are quite free there.

And so, no problem for Sergei to work on Websockets to create the connection between the backend and the frontend. Whatever works best and is the most convenient.

And so now the architecture would look as the following:



API Tasks

1. Get input from client
2. Accumulate real time data in cache
3. Flush the data to the main database on user save request/automatically
4. Send data to client. IF Cache has data THEN give cached data ELSE retrieve from database
5. Authentication: store credentials in the main database

Every piece of data that will be modified in real time will go through the api. It will be placed in a real time database and changes will be handled in that database. On submission, the job will be sent to persistent storage, or sent wherever else it needs to go in the software.

There will be a technology that ensures that the backend speaks to the short term database in real time and a technology that ensures there will be real time communication between the frontend and the API.

The choice of an API was the right one, because the only thing that will be in the backend will be data. The backend will not send back any React content.

end time: 15:10

Group meeting 21 January 2024

Present: Arnaud, Ghais, Sergei

starting time: 15:05

Sergei worked on Web Sockets for the week. He will be finishing that soon. He will push it tonight.

Ghaisd had some hickups with the API but it was fixed today.

Arnaud has been working on the project plan. The process was good for the week. He asked if we should have a development deployment. This would be nice to have and it would be useful, it would also give us extra points. But it is not necessary. Which is not a problem, as long as we don't merge something into main without being sure that it works.

Gant Chart and risk analysis. Sergei will provide it tomorrow.

For the next steps of this week. We will work on creating the backend api. And so we will create the endpoints for the job pages themselves.

We will also connect this API to a SQL database as well. Not a real time database, just a normal database. It will be Firestore.

The database itself doesn't matter. It's not important.

Ghais will try to recreate Komtek, but not completely. It will not be the full database.

So, the gameplan for the databases are one that will be used for real time editing and the other for more permanent storage. The first one will be firebase real time database and the other will be Firestore.

When we will connect, we will first see if there is data in the real time database. If there is, it will be taken there. If there is not, it will be taken from the main database.

And then, when there will be no more users in a job, what is in the real time database will be moved to the permanent database.

In Komtek, to submit a job, you click on a button. When that button is clicked, it is moved to anothe column in Komtek and then it stays there for 14 days as still visible. After that, it disappears.

When a job is in that column, or that list, it can no longer be modified.

On the frontend, we will start working on a navigation bar and on the jobs page.

The "submitted" column will also come but later.

end time: 15:55

Supervisor meeting 23 January 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 10:00

We should have a Gant chart for the whole semester and at the end of the thesis we should have a new one explaining what we did. And if it is different from the original plan we should write and explain why it is different.

It will be our job to have enough features to impress Norkart and make them satisfied with our work. That's on us.

We should have a good volume of features to present.

Our thesis is somewhat restrained and so it will be on use to kind of expand and bring in a good volume of features to present Norkart.

We should be working on the implementation of new features up to Mars/April. Because after that we will be working on writing the thesis and we will no longer be developing.

Ok so, as a general commentary, we should reduce the length of the whole plan and we should write it with line spacing of 1.

This should not be more then 5 to 6 pages long.

And we should switch the section 1 and 2 of the plan. Our section 2 will become section 1. We should do this right now otherwise we will have to change the writing again for the main thesis.

We should not have any pictures or diagrams inside our project plan. This is not common to have it here.

Our content in the project scope is also too technical. This is not something that we should talk about.

When it comes to the fields, it should be real time features and conflict handling.

So this is whaat it will be summarized to.

And we can have a short introduction that should not be more then 25% of the page.

So our section 1, project scope, that should be 2.5 pages. The section 2, background and framework.

At the end of the day, our thesis is not meant to be read by people without IT knowledge. This will be read by our supervisor, the department and an evaluator. It is isn't meant to be read by someone without IT knowledge.

For the risk analysis, we should also use a color code for the various risks and especially mention how we plan to address thoses risks.

end time: 10: 30

meeting with Norkart 25 January 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:00

We thought of coming and present what we were thinking of working on before even starting to work on it. So as to get approval and then start working on it. That's ok.

-What did you implement again regarding the issue? A warning to users inside a job when another user saves his changes to the job?

Right now. What is happening in real time is that the list get's refreshed. And it gets refreshed automatically. So that when a new job is added or deleted, it is upated automatically. However, this is done by having http calls to the api every 5 seconds, which is not the most effective. This would be best done with WebSockets or SignalR. That's all.

-Are you currently implementing any other features regarding this issue? Do you want us to work on it or not?

We can work on what we want it's no problem. It should be basic. It should not be anything complicated. But it should be in real time. That's the idea. And we should try to make it simple.

How deep should we go?

It is not the idea that we will create something that will be integrated in Komtek right now. The idea is that what we will do will pave the way for Norkart.

-Do you plan to work on any other such feature?

Right now, Norkart is working on the possibility of deleting a job. But it works on the same principle, which is that there is a call made every 5 seconds or so to update the list. WebSocket is a better way of handling that.

-Would you be interested in presentations on existing products and on good design practices?

We can share what we found during the meetings, but we should mainly keep our findings for the bachelor thesis itself. This is where we should put our findings. And share a summary in the meetings.

-Do you want us to develop a design system? So that the the design of our features is coherent?

We should have little of a design system. In fact, we can use Norkart's own design system. It is public and it is accessible. They have a library for that. The library is not accessible to the public, but the design system itself is. And we should use that.

The url to the design system is https://toitsu.norkart.no/.

There is also the url https://brandpad.io/norkart. This one describes the Norkart logo, along with other design information. This is something we can use.

In any case, if we use Material UI, Norkart can translate it to their own design system.

-What is meant with synchronous and asynchronous?

For asynchronous, the synchronizing between the different versions is delayed. Kind of like in Git. For synchronous, the different versions are synchronized in real time.

For that, we can try the Incremental Static Regeneration (ISR) system. This is something that we could look into.

-Is it ok if we use WebSockets?

Yes no problem. For Norkart, it's ok. But we will have to justify our choice anyway in the thesis.

-For the design, are we allowed to use color? Should we use color?

We don't really need to start thinking about adding colors and what not. This is not something that Norkart requires us to go into.

-Would you like us to present you sketches of our implementations? Higher fidelity prototypes?

Yes we should present them sketches. We don't need to show the lower fidelity sketches. But we should show the higher fidelity sketches that we make.

If we make domain diagrams we should also show it to Norkart.

-For user tests, would you take charge of that completely and provide us feedback? Or would you provide us raw results that we would have to analyse ourselves?

For the user tests, we will all do it together. So we would be involved in them ourselves. This is something that we would participate in. But then we would have to be careful, as developers, in our presentation, to not give the answers to what we want. To not be leading. And we should not provide too much directions.

For us, when we save a job, we should not think more about it. This is where our process stops. We don't need to bother more with what happens to the job after it is saved.

-Is there specific features that you would like us to develop and/or work on?

V1

For version 1 , we could have the following:

When two users are inside a job, if one of them saves and/or submits a job, when the other user will try to save and/or submit his changes, he will not be able to save and he will receive a warning message that will interrupt his workflow and tell him that someone else saved the job. And that he can either stay on the job and have it refreshed, or he can just go back to the list page.

To implement this, we should use e tags. We, the team, decided to use numbers representing certain statuses. 1 for new job, 2 for in progress/assigned jobs, 3 for completed jobs.

Another option is that if someone is inside the job already, another user that enters the job after cannot update it at all.

Another possibility in the version one would be that, if someone in inside the job, and someone else enters the job, the first user receives a notification that someone entered the job. The notification will not break his flow. It will not interrupt whatever he is doing.

The user entering the job, however, will receive a warning that will interrupt his flow and offer him to not enter the job or enter anyway.

V2

For version 2, we could highlight in real time which element is modified by another user inside the job. So, if I am in a job and someone else is modifying a textbox, the textbox will be highlighted. And this would be done for each user that is inside a textbox. This could also be done on other elements. It would basically highlight the element that a user edits actively.

We are proposed to have automatic updates. But another solution will be to use WebSockets. That will create a two-way, real-time communication.

-Would you rather have few thoroughly implemented features, or many different features that are more on the prototype side?

The most developed it is, the easier it is to follow, to replicate and then to implement for Norkart. But it is up to us ultimately to see how far we want to go. And how much we want to develop our solution.

For testing, we should see what is worth testing. The best tests are integration tests. We should also have a unit test logic.

We should also ask Frode about how much testing we should have really. He will be able to guide us. But it is a balancing act for sure. To have the right amount of testing. It's not good to overdo it either.

For the contract, we will receive an email about it containing what they wish us to sign. They will most likely ask us to sign their own contract as well. On top of the NTNU contract. Vebjørn already received a contract, and he sent it back because of some of its provisions. He is working for us here.

Sergei asked if it would be ok for us to use caching. So, caching can actually be counterproductive to have if the data is stored in there for a long time. If we use the entity framework, it already has cache built in. The caching is under the hood and automatic.

For now, we will stick with only using MongoDB directly, without using the entity framework. The entity framework is kind of the standard framework for interacting with SQL databases in Dotnet.

We don't need to overcomplicate anything either. We could also use an interface that uses MongoDB. In any case SQL handles the caching automatically.

Norkart would like to go over our project plan. So Arnaud sent it to Vebjørn and he will go over it and give us feedback on it.

From now on, for the next meetings, we should send an agenda to Vebjørn, along with our written questions, along with any sketches and or other documents that we would like him to go through.

For logistics. There is one bus at 8:15 and another at 9:15. But then we would arrive at 10:00 am at the bus station. It is ok for Vebjørn that we arrive at 10:15. That is ok. He books the room from 10:00 to 11:30 even if we finish the meeting at 11:00 in case we overflow.

And we could also take some local busses to go to Norkart with our own ticket.

We then asked Vebjørn about testing APIs.

We were talked about X unit to test APIs. There is also the library fluent insertions for testing. We should look at this one. Because it makes the tests more readable. And it gives better output as well. Docker is also good to use for integration tests but this is something that we should look into ourselves.

But we should prioritize integration tests.

To do load testing on APIs, there is a library to do that kind of tests. We should look at videos from Nick Chapsas. There is good information from him. But we should not use everything from him because he is no longer a career developper and what he presents is always the newest. And this may not be the best to use in an industry context because it does not have a lot of support and it may have compatibility issues as well. So we should be critical.

We also have to look at sustainable development.

Rate limiting is a good start for sustainable development. SQL requests are actually not so costly. What is costly is to go from JSON to Dotnet. We shouldn't focus on performance too much in the start as well. This is a trap to avoid that could give us problems in our development. We should develop first. Performance should not stop us from developing the whole application. We should focus on making something that works first. And we should also not use too much time and energy to optimize, because in the end it does not have so much value.

But optimizing is part of sustainable development.

end time: 11:08

Group meeting 25 January 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00

At this point, we have a water meter endpoint. And the water meter controller has an enpoint that takes one job with one id. It was proposed and agreed to move it to the job endpoint. Because for now, we will have only one job type. That's the only thing we need.

And at this point, our job page only has the minimal field type that are used in the different jobs in the actual Komtek software.

So for now, there is a text field, a radio type button, a select menu and a checkbox. Because this is what is used. So for now, our job page is actually very simple.

It was discussed if we need job ids at all, or not, becaue there is the meter number that is unique. But in the end it is best practice to have an id field no matter what. Even if there are orther uther attributes to the class.

So right now, the feature that we can work on as version 1 is that if a user saves his job, and another user tries to save a job after that, he will receive a notification that the job has been saved. And he will have the choice to either stay in the job, in which case the page will be refreshed or he will have the opportunity to go back to the jobs page. In which case his jobs list will be updated.

To have that, we will implement the 3 following job statuses: new which will be the number 1, in progress that will have the number 2 and completed which will be number 3.

We will have a post method to add dumy data to the database. This is not something that will be used by the frontend.

When the data will be updated, a put method will be used. This is what will update the data. And this is what will change the data.

Arnaud needs to learn about Dotnet. He cannot do anything right now and that's not good. And he will learn about testing. Arnaud will handle everything about testing. We need to have that one way or another in the thesis. So Arnaud can take care of that.

So we will have a put request. If the job is opened for the first time, the status is changed to 2. That is a job in progress. And then, when the job is saved, it is changed to 3. And at that time, If the user tries to save, he will get a notification telling him that someone already saved.

end time: 15:55

Group meeting 28 January 2024

Present: Arnaud, Ghais, Sergei

starting time: 21:30

Retrospective:

- Ghais: it went well, we had our plan and we followed it. It's great tha Vebjørn accepted to delay the meeting from about 15 minutes and all and that we don't have to wait any longer to meet. And that we can wake up later. It was good that we asked. The worst thing was that he could have said no. It was also the first time that we met all 3 at Norkart.

- Sergei: we managed to set up the server. That's good. Not sure tht we did everything that we planned to. He watched a lot of videos about backend and dotnet.

- Arnaud: was not able to follow along last week because of a lack of knowledge about Dotnet. Will work to fix that. It would be good to have the sunday meetings at a different time to make sure it always happens at this time.

The group decided on sunday at 11:00 am. From 11:00 am to 11:45.

It is good to have space between meetings. To avoid having too many meetings close to each other.

For this week, the goal is to finish the baseline application on which we will develop things. This should be done this thursday. Because there is a bug to fix in the backend. The post and put methods do not work right now. Otherwise Ghais is done creating the jobs page and this is ready for review and to merge. Sergei will look at this.

We will also start on the documentation. Because it is better to document underway.

It will be good to document this baseline application.

And then, we will start working on a feature that prevents a user from submitting a job if the job is already submitted.

The project plan is pretty much done. The deadline to submit it is January 31. Arnaud will put in there the Gantt chart, send it to Frode for the last review and then submit it to the faculty.

We are still waiting on Norkart for the thesis contract.

The deadline for both the plan and the contract is february 1.

end time: 10:25

Supervisor meeting 30 January 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 10:00

- What do we do if we don't have an answer from Norkart within the deadline?

This is something that we will have to take up to Tom because he is the one in charge of that.

We mentioned that Norkart wanted to have an extra agreement with us on top of the agreement with NTNU and he mentioned that this is up to us to handle that as we want to. And that NTNU will not involve itself in it.

- Project plan:

We are too long. We have too much content, and we repeat ourselves. And he did outline some changes that we should make in specific sections.

We should have decision points as well.

end time: 10: 30

meeting with Norkart 01 February 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:15

We received in our inbox an email inviting us to register in Huma. This is the Human Ressources solution of Norkart. This is something in which we should register and all of our contracts with Norkart will be there. This was not a scam. And all of our data will deleted in June.

We presented the solution that we built so far, the latest version. And it was good. The feedback was good. But we were told to implement inlogging both in the frontend and in the backend. In the frontend we should have an external service that arranges inlogging. And we need to implement bear token in the backend as well.

We were told to get started on innlogging early, before we lose track of it. This is important for security.

We presented Redis durign the presentation. That this is a possible architecture that we could use with WebSocket. It reminded Vebjørn about what is called event grid. This is a queue service for Azure. This is somemthing that we could look at later.

We will also need inlogging because we will need to use user information. Because one possible solution is to allow multiple connection from one user, but not allow many connected users.

And one possible feature is to synchronize the different connections of the same user. Just for the multiple connections of one user, not between multiple users.

We were also told that we should look in the performance costs and the bandwidth use, especially in a cloud context because there are high costs associated to that.

Some case handlers, more on the younger type, can have up to 10 windows open at the same time. If all of that is WebSocket connections that are consuming a lot of energy, that could be a problem. So this is something that we should look into. So we could try both techniques here: one WebSocket cconnection that sends a little bit of data and triggers a get request, or have the WebSocket connection send all of the data. To have the WebSocket provide all of the necessary data. We should look at what is the most efficient. We should look at what is more performant and use less traffic.

We will need to look at innlogging in relations to websockets as well. Because we want only authenticated and authorized users to be able to open WebSockets connection. So this is something to fix as well.

We will need to use JavaWebtokens, and Bear token. Or let's try both.

There is also an issue of organizing jobs here. In the current solution, we have a job. And our job pretty much only have one task. But, in the current solution, one job actually has more then one task inside. And, some other solutions of Norkart interact with only the tasks themselves, whereas some other solutions interact with the whole job. This is something that we could look at. At this interaction between jobs and tasks. And especially, if we could build a solution that could be generic to both jobs and tasks.

For now, our case manager solution can only work on the jobs. It is up to us for now. And perhaps later we could work on making it work on both jobs and tasks in a generic way.

Our focus should remain on the conflict handling and collaborative editing. The structure of jobs and tasks is not the main focus. But it is something good to keep in mind and potentially develop in a generic perspective.

We will use warnings in what is version 1. If a user enters a job in which someone already is, he will get a dialog box telling him that someone is inside the job, does he really want to proceed? Yes or no? This will be in itself a very effective feature that will solve a lot of problems. Because at the outset it is not the goal to have many case handlers on one job. The goal is actually to avoid that.

And we will need innlogging for that because Vebjørn wants to have the names of the users that are inside the job to appear in the dialog box, something like "John Doe is in the job right now, do you really want to go inside that job"?

We will also have locking. There will be one user at a time that will be allowed to modify a job. And we will need to handle that in real time. We will also need to handle warnings in real time. The agenda for this meeting details this pretty well with diagrams.

We will not have just a check. We already need real time implementation.

To summarize, when it comes to cybersecurity, we need to use inlogging, bear tokens and/or JavaWeb-tokens. We should also have up to date dependencies and handle our secrets. For inlogging, we should use FireAuth. We should also use a white list system for connections. Everyone is blocked out except chosen addresses. We also need to use those tokens in the backend because we need verification on boths ends. One in the backend and one in the frontend. Because this is a cybersecurity best practice. So we need to implement that.

We were also recommended by Vebjørn to read one bachelor thesis that is graded A, one graded B and one graded C. So that we get an idea of the what is required and what is a good thesis. The thesis we were given by Tom were ones that he considered good, most likely A thesis.

Version 1 Plan

Verification before submitting a task

This was agreed to by Vebjørn. It changes the priority of the versions but at the end of the day it did not prevent a case handler from working for nothing because he would only know if the job is saved when he submits his own job.

- Simplest simultaneity architecture The simplest architecture was shown and it was explained that the only part of the structure that would need to be with WebSocket was the part that sends the data in real time to the clients. And that can be inserted in the controller of a post request. So the incoming data can still remain with https. This is the easiest because this is what would require the least amount of changes.

- With Redis buffer This is something that we do not need to go into, at least for now. For now, we can keep it as simple as possible.

What does Norkart want? For now, the simplest solution will be prefered. We will not use any form of buffer for now. Perhaps later.

- Locking workflow proposal We will implement it.

- Alerts workflow proposal This is also something we will implement.

Which implementation does Norkart want first? Locking or alerts? It will be both.

Vebjørn will also look at having our product hosted on their Azure ressources. And see if we could use their Azure product for the grid, if we ever decide to implement that.

end time: 11:30

team meeting 01 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 10:15

We discussed following the meeting that we had with Vebjørn.

We especially looked at the issues to open in light of the meeting. We did it the same day, while it was fresh in our memories, instead of doing it on Sunday.

So, we decided to add innloging, both on the frontend and the backend. So we opened new issues relating to that in the frontend, such as creating a sign up page and a 404 not found page. In the backend, pretty much everything relate to software security was included as upcoming issues.

end time: 11:30

team meeting 04 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 11:00 am

Retrospective:

Ghais: Last week was good but confusing. We did not follow 80/20 because we were asked to implement a good bunch of features. But we can still manage to implement it we have good time.

Sergei: Excited for the future of the project. Sergei worked on WebSockets. WebSocket is implemented. Was reading a lot about dotnet. Ready to merge and we could look into using SignalR because using WebSocket is quite verbose.

Arnaud: How about we look at merge requests right away and then return to other tasks. This is Ok with others. Sergei mentioned to use notifications. This will make it god faster. Ghais also suggested to dedicate time everyday to take a look and see if there is a merge request waiting.

Issues were added last wednesday after the meeting with Vebjørn. Should we add anything more? No. let's stick with what we have now.

For stand up:

Ghais: there is a bug right now in the development branch of Ghais, he doesn't manage to connect to the backend. Sergei will take a look at it. But it does not impact the logic of what he is doing. So Ghais did all of the login and registering page. He also did the error page. What is missing now is the avatar and managing the logged in state. This is what Ghais will work on this week. We will also need to connect the frontend to the firebase authentication database. Sergei will take a look at it and if Arnaud is done before he can also take a look at it. Ghais would like to have this for thursday. Arnaud as well to be able to show it to Vebjørn.

Ghais also suggested having a stand up meetig on wednesday to prepare for the meeting with Vebjørn on thursday. We will have a meeting about it at 8 pm on wednesday.

Ghais implemented username. It was discused was to wether or not we should have it at all. And we should have it because this will be needed for error messages and warnings to return the name of other users. The username field was changed to full name in the frontend.

Sergei worked on WebSockets and managed to implement the full setup. He will need our collaboration more this week for building on it so be available on Discord a bit more often. The next step for Sergei will be to integrate WebSocket in the backend. Right now it is just the initial set up. He also made documentation for it. It is in the readme page but he will move it to the wiki pages. And he will be writing more documentation about it this week.

Arnaud worked on implementing user authentication with Firebase. And now 4 methods are implemented. Create a user, get all users, get a user by id and delete a user. Only the put method needs to be added and then this part of the infrastructure can be merged. The use of tokens is not implemented yet. Arnaud also implement that this week.

We will keep track of the different versions of what we will impelement. On the frontend, when login, registration and avatars will be implemented, we will be done with version 0.

end time: 11:45

Supervisor meeting 6 February 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 11:01

The meeting started with Frode telling me that we did not hand in the project plan. And that he was not seeing the project plan on Black Board. And that Arnaud was not in the group. It was just Ghais and Sergei and the project plan was not handed in.

Arnaud told Frode that he handed it in. And he showed the hand in to Frode.

The issue was that he was not in the bachelor group. So he sent a message to Tom, the course coordinator and the issue was resolved. Arnaud was put in the bachelor group and the documents were handed in appropriately. Both the project contract and the project plan.

The main purpose of the meeting was to get a bit of a briefing on writing the main bachelor thesis.

It is good that Frode is our supervisor because he is also the one giving the crash course on writing the thesis.

And so he gave us a plan on how the thesis should be written and what each part should contain.

He gave us his plan from 2023.

And he told us that it was not updated for 2024. But the changes for 2024 is the use of artificial intelligence. It has to be mentioned how it was used. And we need to have sustainable development as well. We will need to have it in our conclusion.

end time: 11:10

team meeting Wednesday 07 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 08:00 pm

stand up meeting to prepare meeting with Vebjørn.

So, we updated each other from the work that we have done so far.

Ghais started by showing what he did. All of the login page is done. He also showed the warning message and the snack bar. What Ghais had in mind was to have warning given to users that are entering jobs for which they are not assigned. And snack bar warnings would be received by the assignee when someone else is entering his job.

Arnaud proposed that the interrupting warning would come to anyone who enters a job in which someone already is. And that the snack bar warning would come up to users in a job when another user enters the job.

So what would happen to assignees? Because Ghais was thinking that if someone clicks "continue" on the warning, he would become the assignee. Arnaud was thinking that the assignee will be chosen at the first click, and that's it, it will never change and anyone can finish the job anyway.

Arnaud showed what he did which was the management of users, and the management of tokens. To be able to get the token, and then to use it in the requests where it is requuired. So for now, every path relating to the jobs require a token, and everything related to users require a token except for creating a user and getting the token, which is basically what is used for loging in.

Ghais mentioned using endpoints in the singular, because right now they are all in the plural. For example, "/Users". But it was looked up and the best practice is to have it in the plural.

Sergei showed websockets. What is implemented right now is that after a connection is initiated, a message sent by a user is sent right back to him, and not other users. Sergei wanted to implement websockets in the first place because it doesn't require much code on the frontend, wehreas for SignalR, you have to download a package on both the frontend and backend. But it turns out it is a lot of code in the backend, and you have to download packages anyways. So we will try SignalR and decide from there.

For testing, at least testing the backend, we just need a framework that can make http requests and then we test the endpoints, and assert the result obtained with what was expected. And then those scripts can be triggered either manually or automatically. So it will actually be easy to integrate in the pipeline. There are most likely cshapr frameworks for that.

For tomorrow, we will show Vebjørn what we did. Ghais can show the frontend, arnaud the users and Sergei the websockets.

end time: 11:45

meeting with Norkart 08 February 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:15

We did not send Vebjørn an agenda today. Because we were just gonna show him stuff. He asked us to send him one anyway, and describe what the meeting will be about. We told him that was received and we will send him an agenda, and the day before. On Wednesdays.

So for the users part, we should not do anything through the backend. We should do everything between the frontend and Firebase. This is a change that will need to be made.

It turns out we also do not need to create a sign up functionality. Just a login functionality is enough.

But if we use usernames, we will need to match the username in MongoDB with the username in Firebase.

As such, the only thing remaining data that MongoDB will need is the username and the Firebase ID. This will need to be stored.

Usernames will need to be used in the app, because Vebjørn wants to have the names of users in the warning.

We will take down Swagger for the deployed version. We will only have Swagger for the development version.

We should also take down the frontend for now as long as we do not have the login interface implemented.

Vebjørn asked why we are not using SignalR. He was just asking. Sergei mentioned that it is to avoid using dependencies. Vebjørn answered at the outset SignalR is a maintained package and so it was determined safe. But that the idea of not uselessly incorporating packages is good.

We were also asked about handling errors and so on. And for WebSockets, this is something that we have to implement ourselves.

Arnaud mentioned that it was more work to implement websockets. Sergei mentioned that it is not, the initial set up is more work but after that it is fairly easy.

For loging in, what will be important here is that the frontend directly communicates with Firebase to login users, that is to fetch the token. And then this token is to be used to access protected web pages and to make the calls to the backend.

We can be redirected for login to the different services such as Google and Microsoft.

We can also use our own pages. What matters is the the call to Firebase be made directly from the frontend.

There was discussion about the different versions to implement.

It came out the following:

For version 1, what we implement is a check on the submit button. Jobs will now have a status. When a job is submitted, we will check to see if the status is submitted. If it is not submitted, the request is accepted, and the job is submitted and saved to the database. If the job is already submitted, the new submission is refused, and the user is proposed to be sent to the submitted job, or to go back to the job list. In the dialog, we will need to have the username. So that the message tells the user "The job was already submitted by John Doe". We will also need the usernames when we will use websockets.

This solves the issue of versioning. There is only on submitted version. And so now you cannot have multiple users submitting many versions. The job will be submitted one time, and not more.

Vebjørn acknowledged that this is not a good user experience because we can still have many users working on one job at the same time and have his work be wasted in the end when he tries to submit. But this is our version 1.

So there will be no WebSockets for version 1. We will only use it for version 2.

For version 2, we will have what we will call the owner of a job. The owner of a job will be the user that have the right to edit a job. The owner will be the first user to enter a job. It will not matter who is the assigned user to the job. The owner of the job will always be the first user to enter the job at any time.

And then we have a queue of connections. The oldest in the queue will always be the ones to have the right to edit the job.

end time: 11:22

team meeting Thursday 08 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:30 pm

We talked and clarified that version 1 will not include WebSockets. It will only be the verification at the submit button.

In the jobs data structure, we will add a field for submitted by. This will be a string.

And in the submitted list, instead of the column "assigned to", we will have "submitted by".

And the error dialog when a submitted job is refused because it was already submitted, we will also have the username in there.

So now, for version 0, we need login, protected pages on the front end, menu and avatars for when a user is logged in.

In the backend, everything is there but some changes need to be made. We will now only have login, create users and delete users. And we will use only the Firebase userID.

WebSockets will come later.

end time: 13:00

team meeting Thursday 11 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 11:00 pm

Ghais: There was some confusion as to what needed to be done. Login should have been prioritized and not WebSockets. But it was a good week because the work was done. So it went well in the end.

Sergei: We had an idea of what was version 1, and then suddenly login became important. We should have documented things and had a better of what needed to be done. But he is satisfied with himself and satisified with us.

Arnaud: There was some confusion as to what was version 1. And I was also confused about it. But now there is a good common understanding of what needs to be done. And that's good. But it was good because in the end the job got done. And so that's great. It was productive and the job got done.

For the upcoming version 1, we will implement a check in the put method. The method will check if the job is already submitted, no matter what it receives. If it is not already submitted, the changes are accepted, if not an error message is returned.

We will also work on documentation of the versions on both the frontend and the backend. And creating sketches.

We will meet again on wednesday to see how things are and if we are done we will present that to Vebjørn as version 1. If not we will tell him we are not done yet and we will still show him what we have done so far.

end time: 11:45

team meeting Wednesday 14 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 8:00 pm

Arnaud mentioned that he implemented the necessary code to dynamically insert username data fetched from the database inside the web interface. He also inserted the token header in the put requests. And he created a "completed by" for the completed jobs.

What was missing was the submit conflict dialog box. Ghais implemented it on the spot.

It also turned out that when you enter the url of a specific job, it sends you to the login page and then redirects you to the main list page. What happens is that the protector redirects to the login page if he does not see a user being connected right away, and then when he sees it, redirects from the login page to the main list page. The solution for this is to have the frontend wait a bit before rendering, so that it has the chance to fetch the user credentials before.

An issue was created about this.

There was a syntax bug in the frontend that Ghais solved on the spot as well.

We also mentioned what we did during the week. Arnaud mentioned having worked the unit testing, nothing on tuesday and worked on the frontend on wednesday.

Sergei only worked on the documentation today.

Ghais worked on the documentation of the frontend as well.

end time: 9:00 pm

meeting with Norkart 08 February 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:15 am

We showed Vebjørn the prototype of version 1. The version 1 has the check to make sure only one assignee every gets assigned to a job. And it also has a check to only allow one submission of a job.

We were also told that it might be possible to store usernames directly in FireAuth, instead of using MongDB for it.

Because we mentionned to Vebjørn that we still use some endpoints of the API to make operations on users and on jobs. This is something that we should perhaps avoid, and so we got the information that FireAuth might allow to store usernames. This is something we will look into.

In the Backend, we then showed him that some endpoints are only open and avilable in development environments, and that it returns 404 not found when the urls are used in the deployed version.

And we also showed him that other endpoints that are to be used have the attribute [Authorize]. Vebjørn then mentioned that it is possible to implement a white list system. Requiring authorization by default and having to explicitly authorize in the controllers. He showed it to us on the spot, and this system was also implemented the same day. The attribute to allow the use of the endpoint without authorization is [AllowAnonymous].

It was also pointed out to us that the endpoint to fetch a user token was completely open. We were asked why we even have it. It is because the API is used to fetch a token when we need it. And so it should be set to [DevOnly].

We were advised that it is possible to fetch a token directly in the console of the browser, and so we could avoid using the endpoint altogether.

In the end it was a compromise, the best is to not have those endpoints at all but it is an acceptable solution to put them as [DevOnly] and still allowing us to manipulate the databases from the backend.

One last thing about version 1 was that it would be nice for the user that was completing a job, but got interrupted by the completion of the job by another user to have his work saved somewhere or at least still accessible.

This is something that will be completely addressed in version 2 because there users that are not the owner of a job will not have the right to edit it and they will see the changes of the owner live.

This was the feedback on version 1.

For version 2, there will be the owner of the job, which the user that has the right to edit a job. This is something that will be flexible and may be called to change, but for now it will be the first user that enters a job.

Other users that subsequently enter the job will be called viewers. They will not have the right to edit the job. For them, it is thought to show them input fields that are "read only". Read only input fields can still be selected and they are not obscured. Users see them as a normal element and can interact with them as a normal element with the only exception that it cannot be edited.

It is also asked that the element edited by the owner of the job will also be highlighted in a different color then the main color. So that viewers can follow along. Or another form of visual cue. It could also be a form of highlighting the changes made by the owner in real time.

Vebjørn also wants for viewers to see the changes made by the owner in real time.

And then there will be the warnings, everyone in the job will receive a notification when someone enters the job. There will be an interruption notification when the owner submits the job. There will also be a list of people inside the job updated in real time.

And lastly, this is something that was not talked about in the meeting, but was asked before, but a user that enters a job in which there are already other users will receive an interrupting notification as well telling him there are other users in the job and asking him if he wants to proceed or not.

The idea of having multiple tasks in the same job also came up. This is something that we can implement if we want to challenge ourselves, but it is not something that Vebjørn requires for the thesis. If we want, we can not implement it at all.

It was proposed a 3 week plan: 50% of the implementation in the first week, presentation and feedback on that 50% in the second week, the other 50% the second week, and lastly, some user tests in the 3rd week. Vebjørn agreed to that.

Specifications for version 2:

For version 2, we will have what we will call the owner of a job. The owner of a job will be the user that have the right to edit a job. The owner will be the first user to enter a job (this can change in the future). This information does not need to be saved for a long time. But this is something that we will need to track: who is the owner of a job. A job assignee has no correlation with the owner of that job. The owner of the job will always be the first user to enter the job at any time. Only the owner can complete the job.

And then we have a queue of connections. The oldest in the queue will always be the ones to have the right to edit the job and then submit it.

These are the relevant terms: The owner: the one that has the right to edit a job and submit it. There is only one owner for each job at every one time. At least for version 2. This may change in another version. Every other users in the job are viewers. Viewers: the other users inside the job that are not the owner. They can see the editing of the owner in real time, but they cannot edit the job.

Viewers get an interrupting notification when the owner submits the job.

A WebSocket manager can be inserted inside a controller, so a WebSocket can react to HTTP requests inside the API.

This is good because there are other Norkart services that make requests to Komtek, and they do it in HTTP, and so the WebSockets need to be able to react to that.

How do we handle those external requests? Do we let them make the changes, and then impose them on the users inside the job? Or is it refused? On the ground that there is another user inside the job? We could try both avenues. But the priority is to accept those requests.

end time: 11:15 am

team meeting Wednesday 15 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00 pm

For the documentation, we will start from version 1, because the versions should start from what we created. The baseline does not really matter. It is more noise. It is not so relevent. So we will start from version 1 and describe what was added to it to handle multiple users in one job.

We talked about what was to be done in the rest of the week.

For the WebSockets, it is possible to incorporate a WebSocket manager inside a controller. And such, it is possible to have a WebSocket manager react to HTTP requests.

The plan for the WebSocket connection in the case of job completion, will be that the requests will still proceed via HTTP, but their effect will be broadcasted to every user connected to the web socket. This is the most straightforward and easy way to implement web sockets in our current case.

And this will protect us against connection failure, because when entering a job, a user will still fetch the job page.

It was required of us to create a list of who is connected inside a job. We talked about this together and it was decided to have a list with all of the names of the different users. In a rectangle shape and with the owner of the job on the top of the list. To implement this in the frontend, what is required is a stateful list, the owner of the job is kept as the first one and everyone else follows after. This list will be updated according to who enter and leave a job.

Everytime a new user will connect itself to the job, every other user will receive a snack bar notification telling them who connected to the job, in addition to getting their name on the list. We will not have notifications for when a user leaves a job. But the list will be changed accordingly.

When the job is completed, it will also display the flow interrupting notification. This will take place.

We talked extensively about the fact that a job can include multiple tasks, and in the current Komtek system, every task can be submitted separately, and each task has a case handler, on top of having a case handler for the whole job.

This is something that we can implement if we want to. But we don't need to, for the whole bachelor. This is not something that we are asked to do. An issue for it was created, and it is something that we could work on if we are ahead and have time. But this will involve more work for sure. The issue is for Sergei to move the data properties inside a job right now into a separate Object called Tasks, and a job would essentially be a list of tasks.

But for now the priority is collaborative editing and WebSockets.

It was also mentioned that work on the thesis itself will need to start, and that Arnaud can do it. But then he will be working less on the development itself.

It was also discussed that at some point users will be able to ask to become the owner of a job, or that with collaborative editing, the owner of the job will be allowed to decide who can edit a job and who cannot.

It was also talked about having a focus color around the element in which the owner of the job is editing. This will be something for a later version.

When locked, the job's elements will be set to read only. They will remain selectables and not darkened, but they will not be editable.

It was talked about having input checks on both the frontend and backend. This is something necessary for safety. This is something that we have to have.

Many other issues were added to the issue boards at this point as well for next Sunday.

The plan would be to do at least 50% of the implementation before next thursday, the other 50% the next week. And then on the third week arrange user tests with Vebjørn and other Norkart workers. And we could already start the work on version 3 while seting up user tests. So the work on version 2 would be spread over 3 weeks.

We could also have a designer of Norkart look at what we developped and get feedback. But for that we need to have something worth showing.

If the plan is to put a WebSocket manager inside a controller, this may be implemented fairly quickly. And then we could jump in the frontend and help Ghais with it. Then things may be done already next week, we show Vebjørn, get feedback, make changes, and then show again to Vebjørn and have user tests.

Because in the Backend, it will most likely be only seting up WebSockets that is needed.

end time: 13:00

team meeting 18 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 11:00 am

Retrospective:

Arnaud: It's great that we got version 1 done before the meeting with Vebjørn. This was productive. He also looked at the merge requests regarding the tests and he merged it. It was a bit overwhelming so it may be better to talk about it beforehand. The team agreed.

Ghais: It's great that we have a plan for the next 3 weeks and hopefully we can get more components done in this week as well.

Sergei: It's great that we have a plan as well, and that we have clarity about what we needs to be done.

And then we did stand up and Arnaud showed the changes in the backend. Now it's two projects: one being the backend itself and the other one being the test project. All the tests of the application will be moved to the test project. Tests will be placed in classes. And it will be faster to run as well. So that tests can now be run by the pipeline everytime we push to any branch.

Arnaud will work on the thesis this week. Because in the backend we only need to setup the WebSockets. Arnaud will start writing the main thesis.

Ghais showed that he made the viewers card. Arnaud suggested of having its size change according to the amount of people inside a job. Ghais will git it a minimum height and then it will expand dynamically.

There was also the idea of giving a sign for the owner of a job. Like a different icon or something. Arnaud proposed having another header above the viewers header called "Owner" and have the username of the owner written under.

For the meeting with Frode, Arnaud thought of asking him about sustainable development. What do we put in there? Or how can we incorporate that to the project?

end time: 11:30

Supervisor meeting 20 February 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 11:00

Today we wanted to have a bit of a briefing on sustainability. How is it expected to incorporate it in the project and what are the requirements around that otherwise.

Frode told us that sustainability is about living in a way that future generations will also be able to live their life.

That it involves economic, environmental and a sosial aspect.

There is nothing that we specifically have to implement with a sustainability perspective. It will be something in the last paragraph where we will describe the economical, social and environmental benefits of our project.

But we should not stretch the thinking too far out. Like, how software makes people save time and therefore they feel better because they feel more free. But we could say that it digitalizes processes, and therefore makes it more effective. And quicker service is a sosial improvement.

For testing, we will have to implement tests and describe in the report how we tested things. This is something that we will have to implement itself.

How many pages should be the thesis. There is always many attachements and appendices to a thesis. But for the most part it is usually between 50 and 100 pages. But it is not the quantity that makes the quality. A math PhD thesis can be only 20 pages. But 20 pages of math, that's intense.

Ghais also asked what counts more in the final grade: the project or the thesis itself? At the outset, it is the report. Both are linked, but it is basically the thesis report that is the basis for the final grade.

end time: 11: 15

Group meeting 21 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 19:30

Arnaud worked on the thesis and is done with the introduction. He sent it to Frode right away for feeback to it next week. We will also ask Frode to show him what we have implemented so far. But we will do so next week. Because for now it is enough with the review of the introduction of our thesis.

Ghais worked a bit on the focus color when the owner of the job is inside a text field. This is a bit difficult to work with because when the user clicks out of the field, it leaves focus. It may be a better solution to simply put in place a normal border around the field when the owner is inside the field.

Ghais is also working on the loading page for the frontend client.

Otherwise, there is not so much to show Vebjørn tomorrow. It may be best to talk to him about our implementation plan.

Sergei worked on securing links of the frontend client. Before, when the link of a job was entered in the url bar, it was immediately going to the login page because it took a couple of seconds for the client to fetch the user information. And when it had fetched the information, it redirected to the main list page because it was now in the login page. Now, there is a little delay before rendering the page, which allows the client to properly render the job page.

end time: 19:50

meeting with Norkart 22 February 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:08

We explained to Vebjørn that, for the submission button, there will be a Web Socket manager in the controller that will react to the post requests and broadcast the changes. So, the submission button will always work with an http request. It will not use Web Sockets. So the web sockets work in only one direction.

For the rest of the data that we will look at in real time, it will be using the Web Socket both to send and to receive. It will be used in both directions.

Vebjørn told us that this seems to be in the right direction.

Vebjørn asked us if we planned to show our work to Frode, and we will, especially to know if what we do is complete enough for a bachelor's degree. Frode has many years of experience so it will be useful.

Ghais also created the card to be able to show the owner of a job and the viewers.

We got a copy of the fully signed project contract and we gave it to Vebjørn. Vebjørn will upload it in Huma.

end time: 10:47

Group meeting 22 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00

For the number of users inside a job, Web Sockets are needed.

Ghais made changes to the frontend and he will be able to open a merge request, it will not conflict with what Sergei is currently doing with the Web Sockets because he is working in the backend.

For now, Sergei is working on the set up for Web Sockets, he will create a merge request and then we will share tasks related to Web Sockets. So for now, Arnaud can do some research on Redis or he can further work on unit tests.

end time: 12:15

Group meeting 25 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 11:00

Retrospective:

Ghais: It was not so much action this week like previous weeks, but sometimes you need to relax. We decided to show Frode our client. So we can get experts opinion about it. Only 2 weeks left for version 2.

Sergei: Was a good week for Sergei, managed to set up web socket. People on the same job see the same changes. Version 2 can be done in less then that, one week.

Arnaud: Merged requests.

Stand up:

Ghais: Ghais will start refactoring the frontend. Especially the job page. It is getting bigger and bigger. Ghais looked at the merge request of Sergei. Ghais added a bunch of small features. Ghais uses features. Never approve a merge request without testing it first.

Ghais will also continue refactoring.

Sergei: Yeah it is definitely a good thing because one time, Arnaud saw an error and spotted. Sergei finished initial job set-up for the websocket. And he connected the frontend to the backend. We will also debug the login. This week, he will make the wireframe and finish the websockets by thursday.

Arnaud: Arnaud worked on the thesis the first half of the week and the second half of the week worked on testing. Arnaud finished the testing of the controllers and will now do the testing of the services. After that, Arnaud will do the testing of the requests to the API in the frontend. It will be mock requests. This may be better because this will be consts. It will not change, but the components will change so it may not be best to test them right now.

There was also an issue with the WebSocket connection. The websocket connection was trying to connect to a local host, instead of the deployed version. Sergei did some live debugging with the rest of the group.

Two bugs were solved and the changes were pushed.

Issues for the coming week were also created. For Web Sockets, it will be about creating and tracker who is the owner of a job. Sergei will work on that.

end time: 11:30

Supervisor meeting 27 February 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 11:00

For the thesis, we have a good plan. This is good.

We need a summary in both languages, Norwegian and English.

We also went through the introduction that we wrote and Frode gave us good feedback.

We should be developing meaningfully until the beginning of April. And then we will be focusing on wrapping up the development and writing the thesis itself.

We showed Frode what we worked on so far. And explained to him the features that we are working on. Everything was good. But he did suggest to also have the list be updated in real time.

end time: 11:20

Group meeting 28 February 2024

Present: Arnaud, Ghais, Sergei

starting time: 20:00

Retrospective:

Ghais: Has been refactoring the job page so that each component are in their own files and the components are called with only one line. Ghais also made small tweaks and improvements to the design.

Sergei: did coworking with Arnaud for the whole day. Sergei changed the setup from a middleware to a controller. Connection handling has also been implemented. That being which connection receives which data. The current connections are stored in-memory with variables. A later improvement would be to use Redis.

Ghais what happened with the viewers logic. All can type for now and make changes to a job, but only the owner has its changes broadcasted to other users. The list of users connected is updated everytime there is a new connection and every time there is a disconnection. It is updated with an HTTP request.

Arnaud raised the issue of asking Vebjørn should be connection based. Or if it should remain user based. That is that, on the view cards, users appear once, even if they have many windows open. The owner has the right to edit in all of its windows, not just one. This is something that Arnaud will ask to Vebjørn.

Arnaud will also ask Vebjørn if its ok if the list is updated with http requests. Maybe it is.

Ghais pointed that the snackbar openeing when one user open another window as a bug. This is a bug based on a business logic that is user based.

Ghais also saw a bug that when a third user is in the list, the view header also appears above the second user.

It was also talked about during the meeting to meet Vebjørn tomorrow in the afternoon so that we have time to tweak and test version 2 before meeting him. Arnaud sent him an email asking him to meet at 13:30.

Arnaud will work on locking the page for viewers and the alert when there are already users in one page.

Arnaud did mention to Sergei to let Vebjørn know if he does not come to Norkart. Sergei may also let Ghais know to make sure Vebjørn gets the message.

end time: 20:20

meeting with Norkart 29 February 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 13:30

We showed Vebjørn all the features that were ready to be shown: the live submission, the snackbar alert, the locking, the dynamic user list, the warning if a user is already in the job, the live changes to the job, the change of ownership, etc.

We gave the link to Vebjørn of our deployed version. He will look at it, test it himself and give us proper feedback in a document. Then we will implement that.

What came to his mind was what do we work on next. We could have conflict handling of differing versions, asking ownership of a job, or asking the right to edit. Expand the complexity of the job handler with tasks. And Vebjørn would like for a user in the task to know which users are in the job as well as the users in the same task, and users in the job to see which users are in the different tasks.

It is kind of up to us to decide how we will increase the complexity. We asked what would give the most value to Norkart. It would be tasks, because it would make our prototype resemble more Norkart's software. We could also increase complexity by having different types of tasks and more input fields. And have something that looks more like what Norkart uses.

We also discussed with Norkart about using some form of cache to be able to store the changes made to a job underway, without it being submitted. Or to have a dialog box saying that there are changes from before that have not been submitted, do you want those changes? If yes, you get them, if not, you get a blank page.

We also forgot to implement the use of highlighted borders for input fields in which the owner is.

We explained to vebjørn that the updates of the users list is done through http calls, triggered by a Web Socket message, was that ok? We could also do everything through the web socket. For him, it is ok, it made no difference. But it may be something worth discussing for us in our thesis.

Vebjørn tried the website himself in front of us and his first impression of the stepper was that the 2 other buttons were disabled because of the grey color. So their color should change.

Vebjørn was also asked if it was ok that we had a business logic based on users with websockets instead of connections. Instead of having each connection in the user list, the user appears one time, no matter how many windows he is using. And the owner has the right to edit a job in all of its windows. For Vebjørn, this is ok, but this then raises the question about about conflict handling. Because Vebjørn also tested that if the same user connects from multiple computers, both sessions have the right to edit a job.

We also asked Vebjørn if he was interested in have user tests. So, he will first test our product. He will first do that. To test it on Norkart workers, we should have an interface that looks more like Komtek.

But we thought of first finishing up version 2 to next week, fixing bugs and integrating Vebjørn's feedback, then move on to version 3.

Vebjørn asked us if it was too little of a thesis. We raised the question with Frode and we were told that having collaborative editing with conflict handling would be complex enough. So then it could be a mix of both tasks and conflict handling.

And even in his feedback, Vebjørn will also raise up issues that will most likely increase complexity of the thesis.

end time: 14:30

meeting with Norkart 3 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 11:00

Retrospective:

Ghais: this week way over exceeded expectations. So good job.

Sergei: it turns out that if we try, we do more then we think we can and if we continue like this, we can do a lot. There is something called quick implementation or something like that. Which is to implement features quickly and then to refactor.

Arnaud: was nervous at the beginning of the week because it was the 2nd week of version 2 but it worked out well and it was great.

We went through the feedback that Vebjørn gave us. A lot of the feedback was minor UI changes that Ghais will implement.

The last feedback was a bit bigger, it was proposing to add multiple user types. One that creates job, and another one that completes jobs. To make it look more like Komtek.

This is a discussion that we will have next week with Vebjørn. What do we do next.

Ghais will implement the feedback on the frontend and will also add the borders around the elements changed by the owner of the job.

In the meantime, for this week, Arnaud will work on the testing of the API calls on the frontend. And documentation of the use of web sockets for version 2 in the backend.

Sergei will work on testing the websocket controller, documenting WebSockets for version 2 in the frontend and on implementing data transfer objects and the repository structure.

Sergei mentioned the idea of disconnecting a WebSocket that is inactive for certain amount of time. This will be discussed with Vebjørn but it will not be worked on this week.

end time: 11:36

meeting with Norkart 6 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 20:15

Sergei did not work on the thesis at all.

Ghais implemented a bunch of the feedback of Vebjørn, but not all. He is still working on it. Ghais will open a merge request tonight to have his changes so far merged for tomorrow and be able to show them to Vebjørn.

Ghais also wondered if Sergei could look at the username not appearing sometimes at login. Sergei will do that and see if he can get it done tonight.

Arnaud did the documentation of version 2 in the backend. He worked on the borders around the elements that the owner is focusing on.

And Arnaud also almost finished a video series on testing in the frontend and asked what he should be testing. Ghais thinks it is not worth testing the frontend at all because things will change. Arnaud did not do integration and end to end testing in the backend, should he do taht instead of testing the frontend? Yes. So Arnaud will move to that.

The plan for tomorrow would be to discuss what to work on with Vebjørn for version 3, then focus on only that for the next 3 weeks, then move to the thesis. And wrapping up the coding, but no more new features. And if we are done in 2 weeks, work on something else for the last week after having talked with Vebjørn.

Sergei will not come to Norkart tomorrow. And Ghais will have a workshop that he will attend between 12:00 and 14:00. Arnaud will go there anyway.

Same thing for the Ramadan, Ghais will do more home office then, but we will be able to go to Norkart anyway.

end time: 20:30

meeting with Norkart 07 March 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:15

We implemented all of the feedback that Vebjørn gave us in his previous test. And we showed him that.

When showing him, he also mentioned that we could change the error message at failed login to "verify that username and password is correct" and for the stepper, to have a hover effect on it. Changes were actually implemented during the meeting. The hover effect was not implemented but rather some color was added to the words.

And then Vebjørn asked us if we implemented some of his idea at the bottom of the document.

And that's when the meeting when to its core: what should we be working on?

Because now we have 3 weeks. After that, we should be moving to writing the thesis, and be done with the development.

First we talked about the possibility of creating jobs.

In Komtek, there is different Graphic User Interface for different user types. To make it simpler for us, we could stick to one user interface, and insert in it the possibility to create jobs.

The idea here is to make our interface look more like Komtek, so that it would make sense to proceed with some proper user tests.

And then this led to the task feature. Another possible idea to make it look more like Komtek would be to add some tasks in the jobs. That a job would now be a container in which tasks are. And that a job could contain multiple different tasks. And that each task has a set of fixed fields that would be validated. And taks would be displayed as an accordion menu in the job. The idea here is that a job could now contain an unknown amount of tasks. And in such case, we would need to make the frontend dynamic. And the Web Socket handling as well. We would need to make it dynamic as well. That would be a challenge. And right now, in Komtek, the input fields are not validated in the Komtek backend, so for every task, the frontend could create a new field and send it to the backend and it would be accepted. So this is also something that we could implement. Lastly, we could have type config. This is something that we could also implement. So that the fields for every tasks and their parameters are properly controlled.

And then Ghais about the asking for ownership feature. This is something that is up to us, but if we implement conflict handling, this may not be needed.

Part of our bachelor thesis is to handle conflict and to have proper collaborative editing. And so this is also something that we should include in our development.

But then it was mentioned that we could have jobs that are open for editing by everyone and some jobs would still follow the owner logic.

We could also have a version logg. This is also a feature that we could implement.

The idea would be for us to increase the complexity of our project to have enough to write about in the report. So this is something that we should be looking at and see what has the most value for us in this aspect. Vebjørn recommended us to make a priority list as well. Where we weight what matters most to each of us and decided with that.

Vebrjøn would like to have everything: tasks and dynamic handling in the frontend and the web sockets, collaborative editing and conflict handling and version logg and version logg.

And so finally, this is all the possible features that we could implement for version 3:

- Collaborative editing (ask for ownership) - Tasks vs jobs - Create jobs - Version logg - Type config

end time: 11:30

meeting with Norkart 07 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:10

We discussed that we needed to chose what we will work on, because we have little time, and we will not manage to implement everything. We need to take it one step at a time.

So, we have three weeks on us to get as much done as possible. And in that timeframe, take feedback from Vebjørn and implement his proposed changes.

And so we discussed and decided to prioritize the following features, in order of priority.

1: ask for ownership 2: implement caching (last version applies) 3: create a job 4: implement tasks

We also created issues in the issue boards about that.

end time: 12:40

team meeting Sunday 10 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 11:00 pm

Retrospective:

Ghais: Last week went alright. We did way more then needed, so Ghais feels that we are more relaxed. He likes where we are and what we are implementing. We have a few weeks left for implementing stuff.

Sergei: Was focusing on other subjects more. Sergei was happy with the progress we made and he feels like we have plenty of timme. No stress.

Arnaud: Happy where we are. Everything that we set to do was done and this is great. Not sure we have so much time if we want to implement both collaborative editing and tasks.

Stand up:

Ghais: Request editing was put in the viewer's list so that it is easy to see. There is a merge request coming soon. Started with the job form. Not sure everything can be done in time.

Arnaud: implemented the isLocked variable. So it can now be linked to the frontend. This variable is what indicates if a job uses collaborative editing or not. If the job is locked, the owner logic applies, if it is not locked, it uses collaborative editing for everyone. For the rest of the week, worked on learning about Redis and caching.

Sergei: this week he only did refactoring. To have a good shape in the code. Because it felt to Sergei that it was more chaotic.

Plan for the week:

Ghais: will work on the form for the job, this will need to be linked to the post method in the backend. And the button to ask for ownership, he will also need someone to connect it to the backend.

Sergei: will take care of linking the new frontend features to the backend, will review merge requests and do some testing.

Arnaud: will work on caching and conflict handling.

We will switch to remote meetings with Vebjørn for the rest of the thesis. Except the last meeting where we could go there and show our final product.

Let's cancel the meetings on wednesday night. And from now own let's meet 15 minutes earlier before the meeting with Vebjørn.

And lastly we will not work on tasks because it will not add much value to our thesis and it will be a lot of work. So we will focus on collaborative editing.

end time: 11:22

Supervisor meeting 12 March 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 11:00

We essentially asked Frode at what time we should start writing the thesis report. And we were told that most start in the last month and keep developing until then.

But it is good to start early. And write underway.

Frode also asked us if we had enough to be developing because we were quite focused on the report.

Yes we do have enough but we that we planned to be done at the start of April.

And so we told Frode that we told Norkart that we had to be done with the development at the start of april and that, after that, there would be no more development of new functionalities.

end time: 11: 05

meeting with Norkart 14 March 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:00

Cache was showed to Vebjørn.

We told him that we would be working on conflict handling and instead of tasks because this is what would give most value to the bachelor and would be more in line with the bachelor thesis description.

Vebjørn mentionned that it is good that we focus on what relates the most to the bachelor thesis description. And that the idea behind implementing tasks is to have something that ressembles more reality. So that the solution created can be more realistic, usable and testable.

And so it would be valuable and pretty exciting for Norkart to be able to have the real-time features and the conflict handling in an environment that is more like Komtek.

We also showed Vebjørn the work that was done on caching.

And we explained that we found a framework, Yjs, to solve conflict handling in React. And that we will now implement it in the frontend.

That we thought we needed caching to be able to handle conflict but that in the end it can all be handled in the frontend. And that in fact it affects caching because what needs to be cached at this point is the final result after the changes, and not just the change, which is what it is right now.

We also showed him a priority list of what we have planned for the last weeks of development:

1: ask for ownership 2: implement caching (last version applies) 3: create a job 4: implement tasks

This is something that Vebjørn supported.

Vebjørn asked us what we thought of the process that we had for the thesis. And Arnaud mentioned that it was the best group work process that he had in all of his bachelor.

We will meet next week, and we will not meet for the easter vacation and then we can meet the week after.

Vebjørn will also be available to meet during the writing of the report. And will also go over it to make sure that we do not write any confidential information.

Vebjørn will also test what we have developped so far and he will ask for the input of a colleague. He will also send us a form to ask us how what we developed is expected to behave. And test it against that. And then he will send us feedback in a document like he did last time.

We also showed him the changes that are being worked on in the front end. That is to create a job and to be able to ask for ownership of a job.

end time: 10:36

meeting with Norkart 14 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 10:00

Stand up:

Ghais: worked on creating a new job. He also proposed incorporating different a new type of jobs if we find time to do it. It would be to install a new water meter. Ghais also mentionned that pages are loading too slow in the frontend, especially when loading a job page because the page is locked for many seconds before it unlocks. This is how Arnaud implemented it: a job is locked until the frontend receives from the backend the information that the user is the owner of the job and can therefore edit the job.

Arnaud: implemented caching and looked up conflcit handling. Found a framework called Yjs that can be used with React to resolve version conflicts in the front end only. It turns out that caching is not even needed to be able to handle conflicts and that the conflict handling is all done in the front end. Now it is about implementing the solution in what we have developped and adapt the caching to store only the resolved value.

Sergei: Sergei deleted the web socket data that was being sent on close because it was creating error messages in the console because when the data was sent, the web socket was already closed. It turns out that there is also a delay locally for pages to load and so this would be something to look into. Sergei will work more on the thesis from today to sunday and have some implementations done. He usually works on the thesis from thursdays and on. Arnaud mentioned it would be better to do the opposit and work on the thesis at the start of the week so that more could be shown to Vebjørn.

Some more issues were made related to asking for ownership of a job and linking the job creation on the frontend to the backend. For now, the button to ask for ownership has been removed until it will be ready in the backend as well. An Issue related to the performance of the client has also been created but will be tackled later if there is enough time.

We will also start writing notes for the report and the documentation. We will be able to meet Vebjørn around April 10, and then we can meet up and ask him about documentation.

end time: 10:36

team meeting Wednesday 17 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 11:00 am

Retrospective:

Sergei: it was a good week. Nothing extraordinary. He had been working since thursday. Got over-confident at some point that implementing requesting ownership feature would be easy, but stumbled on a bug that is now difficult to solve. Then it becomes frustrating when it does not go according to expectations.

Ghais: We are almost finished with the thesis basically. It was good to have many meetings and dead-lines. And then there will be documentation to work on and to rework. We will use ChatGPT to give us feedback on the report, not to write anything for us.

Sergei was wondering if Vebjørn could look at our code to give us feedback. He will because he wants to. And he will most likely give us what are the most obvious recommendations.

Ghais is also almost done with the edit request dialog.

Sergei showed us the tenuous bug he is working on. It happens that when a user first enters a job, and another after that enters the same job, ownership is allocated normally. But then when the other user reenters the job first and the other comes back in the job, he takes the ownership right away. It may have to do with the socket sorting in the backend.

We will also need to change the viewers card for version 3 to no longer have viewers. Because everyone will be an editor. But we will also need to keep the old card because some jobs will be open for all to edit and some others will not. We still need locked jobs to be able to request ownership.

For Arnaud, the week involved a lot of work and was difficult because although he found a framework for conflict handling. The documentation and the resources around the framework was a bit thin and so he kind of had to figure it out himself. And that was painfull. But it worked out in the end and that feels pretty good.

Ghais showed the request ownership dialog.

Arnaud showed the conflict handling branch. He forgot to point the server to the local server. And Sergei came with idea of having a .env file so that when we run the code locally, it points to a local server or not. But that for production, it always point to the backend server and that we never have to worry about forgeting to point to the right server.

Arnaud also explained the setup and how it works. And explained the bugs that he had to overcome. Ghais mentioned that it will require some refactoring for the job page. Some of the conflict handling could be placed in hooks. The file is getting a big too big here.

The feature was not tested with 3 users in one job. Arnaud tested it live during the meeting and it worked.

Ghais mentioned some other small changes to the frontend. Sergei will finish up on requesting owner-ship and link it to the frontend and then our thesis is pretty much done. Sergei will also refactor. Arnaud proposed working on testing the websocket controller but Sergei will want to refactor first.

After that we will be able to work on some automated tests and move to writing the thesis report.

Ghais mentioned that it was key to start with small objectives.

We will meet Frode tuesday to show him what we did so far.

end time: 12:00

Supervisor meeting 19 March 2024

Present: Frode, Arnaud, Sergei

start time: 10:00

We met Frode and showed him the latest that we have done. And now we can create a new job and ask for editing rights. And the conflict handling that we implemented.

He then told us that it was too little. That we should develop more. It is not about spending a lot of time on developing something that should tak a short time. It's about working regularly on it.

We told him that we could implement dynamic fields to be able to handle an amount of fields that we do not know about.

We also asked if we should comment the code. And we should comment it.

end time: 10: 05

meeting with Norkart 21 Mars 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:00

Arnaud apologized to Vebjørn for not giving him any agenda for the meeting and not sending him any messages the day before. He completely forgot.

Arnaud showed the recently added features to the program. It is the job adding and asking for ownership and conflict handling.

Vebjørn noticed how slow it was and mentioned that. He refered to the fact that anything that takes more then 100 miliseconds is experienced as slow by users.

The caching made our solution slower in the end because it is a free solution and it creates another http request. Locally it is faster. And for the conflict handling, it is slowed down on purpose so that conflict handling can be shown. But it can now be reduced. Now that it has been shown. On a pure user standpoint, the best is to have changes and conflicts managed as fast as possible to avoid having to wait for anything.

We mentionned that Frode told us to develop more in the coming week. And so we decided to try to handle the fields of a job dynamically. And Vebjørn asked us if we could also enforce a certain object structure as well, that would be great for him.

Other then that, Vebjørn conducted some testing of our current solution with a colleague.

This is the feedback that he gave us.

If we could implement a feature that asks the user if he wants to save the job in the main database when he leaves a job, that would be good. Right now, it is saved in cache. But by definition, cache is temporary and so it will disappear at one point or another. And especially now there is nothing that says what happens with the data when a job is left. So being offered the option to store the job data in the main database when leaving a job would clarify a lot of things.

It would be good to have a hover effect over the different elemenets of the stepper, to be sure the user knows that it is clickeable.

It was also recommended to have next to each stepper elements the number of jobs that are in each of the stepper element. In paranthesis.

It was also recommended to have a color code for each user so that we could know who is editing.

It would be good to have a search bar to be able to search for jobs.

It was also recommended to make the overall system faster.

We will not meet next week because it is the easter vacations. We will meet the week after.

end time: 10:50

Group meeting 21 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 9:45

Stand up:

Ghais waited for refactoring because there had many merging. But did do some refactoring. There is a merge request pending for some small changes

Sergei finished refactoring, finished the transfering of ownership feature and tested the feature when a user asks for editing when there are 2 other users that already ahve editing rights. In this case, it is the first one that clicks to either accept or reject the request have the last word.

Arnaud merged the different merge requests made by Sergei but nothing else. Worked on entrepreneurship instead.

It was decided to implement dynamic fields in the frontend because we were told by Frode we should develop more.

There will also be implemented dynamic jobs in the backend.

Vebjørn also suggested some feedback and some of it will be implemented. Some of it won't. Like a job searching feature or being able to save a job in the main database instead of the cache. Those will not be implemented. But they will nontheless be part of the final report.

Issues were made as well for the rest of the week.

end time: 10:00

Group meeting 24 Mars 2024

Present: Arnaud, Sergei

starting time: 11:00

Retrospective:

Sergei: It went well but not much was done.

Arnaud: Was uncomfortable in the beginning on being able to implement dynamic fields for the job. Because had no clue at all where to start. But now feels better because a roadmap has been found.

Stand-up:

Sergei: was done with the request editing. Nothing new was added. Was worried about the long if chains. Got educated on that. Found a resource that maybe there should be a programming pattern used. It's basically making a class for each of the functions and just have one block. It would be better for performance. Shrinks the code. The idea is to use hashmaps. If chains are sequential. A Hashmap is instant. Thinking about implementing that this week.

Arnaud: Added documentation on version 3 in the front end and reduced the time for the transmission of web socket data for the water meter from 5 seconds to 2 seconds. Also started the work on making the job page fields dynamic.

Plan for the week:

Arnaud will continue with making the job fields dynamic in the frontend.

Sergei will work on making jobs dynamic in the backend and will remove some if statements and replace it with a hashmap in the websocket controller.

end time: 11:30

Group meeting 28 March 2024

Present: Arnaud, Ghais

starting time: 10:00 am

Stand up:

Ghais: Changed the hover cursor over the stepper steps. Now, it becomes a pointed arrow. So now people know they can click on it. Also implemented the number of jobs next to each step.

Also refactored the useSWR package with Arnaud during the meeting.

Arnaud: Refactored the front end for dynamic rendering. And worked on some changes recommended by Ghais. And fixed a bug that was not saving the meter number type when the job was being saved. But my fix is not great, I should look at something else.

But it will need to be fixed further.

There are still some small tasks to do in the frontend. After that the development will pretty much be done.

end time: 10:45

Group meeting 1 April 2024

Present: Arnaud, Sergei, Ghais

starting time: 11:30 am

Retrospective:

Sergei: Not much work. Nothing practical. Nice celebration.

Ghais: Did the last tasks and Arnaud reviewed the merge requests. We are done now with the development now.

Arnaud: Went well. Did the last tasks and reviewed Ghais' merge requests. That went well and was productive. It could have been something Arnaud did earlier in the thesis as well.

Stand up:

Sergei: will have a merge request for refactoring

Arnaud: refactored the frontend, merged Ghais' merge requests and started commenting the code in the frontend.

Ghais: made the final changes to the frontend requested by Vebjørn

ChatGPT will be used to give feedback to what we write. Will not use ChatGPT to write any text.

Development is now done. It will be commenting the code, testing and documentation

fixing some bugs if we find them.

We will meet Frode at 1pm on thursday.

I will send a message to Vebjørn to meet him online at 10am on thursday. We will show him the last features that we implemented and that will be it. And then telling him that we will now write the thesis.

end time: 12:00

meeting with Norkart 04 April 2024

Present: Arnaud, Ghais & Vebjørn

starting time: 10:00

We implemented typesafety and also implemented optionnal job fields.

What is important is that every potential job field has its own component that is rendered if the jobfield is ever present.

We will meet at Norkart next thursday and after that meetings will take place on a request basis. The assumption is that the meeting takes place. So we need to tell Vebjørn in advance, preferably on mondays or tuesdays if we do not plan to meet at Norkart.

A zip file of both the frontend and the backend has been sent to Norkart.

end time: 10:15

Supervisor meeting 4 April 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 13:00

We can now start writing the thesis.

Frode is a bit worried we didn't do enough but yes, it is good that we start working on the report now.

Frode will read our thesis 2 times before handing in.

The last time is usually about 1 week before handing in.

The dates are around April 25 and around May 10.

Tips for us? The first and last part are quite important. The middle part is more customized to our project.

Can we write about what we could have worked on? Yes there is a whole section for that.

Can we use ChatGPT to give us feedback on our text? Yes. It's also good to use our own brain for that as well.

Do we log our hours? Yes.

end time: 13:05

team meeting 07 April 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:30

Retrospective:

Ghais: last week was good, we met Frode and he gave us two deadlines. One at the end of April and the other one week before the deadline.

Sergei: Good week, in terms of transition from practical to writing. Hard because he wants to code more. Wanted to do more. Never truly done with coding. Writing is a whole new job. Completely different. Transition is kind of hard.

Arnaud: Got a bit nervous when asked questions that were not answered. April 2, asked if can reassign websocket testing to himself and integration test to Sergei. No answer there. April 2, asked if we are done with refactoring? No answer there. But it was addressed today.

One day delay is ok.

Stand up: did refactoring. Tried refactoring in the backend. But it did not work. It introduced bugs so it was rolled back. It was not worth it. It would have looked better.

Now we are done with refactoring. Because before refactoring code, you should have tests in place. Because refactoring can introduce bugs without knowing it. And so we are done with refactoring.

Ghais: did final coding with Arnaud. And looked at the merge requests. Did some GPT research. Could review what we write. Not make any changes but get feedback from ChatGPT.

Arnaud: worked on commenting and testing.

Plan for the week:

Arnaud: work on commenting the code in the backend. Make unit tests in the frontend. Make integration tests in both the front and backend, andn try to do one end-to-end test.

Sergei: work on the implementation part of the thesis.

Ghais: work on the frontend design part of the frontend.

For the meetings, we will not need to meet Frode this week, but just to know from him the dats at which we should submit him the thesis for review.

For the meeting with Vebjørn, we will ask to postpone it to thursday April 18. We will ask him for feedback on the code, but won't make any changes unless its about solving a bug.

end time: 12:30

meeting with Norkart 14 April 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00

Retrospective:

Ghais: Retrospectives may not be so relevant anymore. Because it is mainly about the report at this point.

Arnaud: Had to prepare a test case for a work interview this week and so its why he was working late this week. It was not great. Is back to normal schedule. Sergei merged comments first and then worked on other changes. That was good.

Sergei: Not much to say. Happy to start writing the thesis. Has been reading a lot lately. Not too much stressed. Everything is getting done.

It will be good to have meetings on Sundays. To preserve the group atmosphere, and to meet after meeting Vebjørn.

Stand-up:

Ghais: Started with the report on the frontend.

Arnaud: Finished the unit tests on the frontend and commented the backend. Started working on integration tests.

Sergei: Review Arnaud's merge requests. Made user testing on his own and found a bug. The list was not updated when somebody left the job. Because message was not sent in proper format. Writing the report. Read about how to write. Made the report going.

Plan for the week:

Arnaud: will work a bit on integrated tests and then will move to the report because this is what is most important now. Integrated tests in the front-end and end to end testing have been dropped.

Comments for functions in the backend will be modified by Sergei.

The cache in the backend will be turned off.

Arnaud will send a message to Vebjørn that we are going to Norkart on thursday.

end time: 10:36

meeting with Norkart 18 April 2024

Present: Arnaud, Ghais, Sergei & Vebjørn

starting time: 10:05

Vebjørn did not have time to look at the code.

Arnaud asked him about some of what he wrote about Komtek.

Komtek really is a series of multiple software. There are apps for people working in the field. And there are purely web applications as well in the environnment.

The software that we work on is a software that is on the cloud and is used by case handlers in offices in a town hall. This is a purely web application.

And so Arnaud wrote that the part of Komtek that is worked on is a web application, it uses React on the frontend, it has an API for the backend, in Dotnet and it is a Rest API. Yes that's right.

Sergei asked about specifications. And if there are more specifications that we should include in there. Those specifications are kind of mentioned throughout all the meetings that we had with Vebjørn before. We have them in our notes. But safety, not having the API out in the open, using bearer tokens, etc.

Bearer tokens? Why not sessions? Because of distributed systems. It is more effective to use bearer tokens in distributed systems then to fetch session information all the time.

There was also a thing, we were working on the theory part and we found out about HTTP2. A protocol that essentially allows to achieve the same result as Web Sockets. And that this is presented as more effective then web sockets. This was mentioned to Vebjørn as a possible alternative that may be better then what we actually developed.

We sent the codebase to Vebjørn a couple of weeks ago. Since then, we added tests in the backend and comments. So this is something that we could send as well.

end time: 11:00

Group meeting 21 March 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00

Stand up:

Ghais completed the front end design part.

Sergei has been writing about the requirements section. It is not completely done. Still in progress.

Arnaud has been writing about the theori part.

Sergei will continue writing about the requirements part and start writing about the design of the backend part.

Arnaud will keep writing about the theori part this week, the testing part and the working process part.

Ghais will review what we wrote and provide feedback, also feed from ChatGPT. But we will only take the feedback. We will not have ChatGPT write anything.

We will meet with Frode this week and tell him, not have him read but tell him what we are writing about and see if it is in the right direction or not.

end time: 12:15

Supervisor meeting 23 April 2024

Present: Frode, Arnaud, Sergei

start time: 11:00

Asked Frode if it was right to put an inventory of real time features and of real time technologies in the theory part. Yes it is.

Asked if it was ok to have screenshots of other products in the theory part. Yes it's ok, but screenshots of what we made should go in the design part.

Frode asked if we had use cases for our requirements. Yes we do and we also have specific requirements that were provided by Norkart such as having login and password.

Asked what is the difference between implementation and design. Implementation is more about the tools used to implement the project.

Design is more about architecture

Asked if it is ok if we submit the thesis for review later then the end of April. Yes no problem but then maybe he won't be able to look at it two times. Asked if it we should submit for review a thin version rather then not at all. He said it's better to have a complete draft, so that the review can be useful.

There was issues with the wifi connection today.

end time: 11:10

Group meeting 29 April 2024

Present: Arnaud, Sergei, Ghais

starting time: 12:45 am

Stand up:

Arnaud finished the theori part which was the most difficult and consuming because of the theori and the concepts. research paper also had to be read and so on. So this was difficult. The other parts will be easier because they are more about reporting what we did.

Sergei finished the requirements part and is on his way to finish the backend design part. After that he will work on the implementation of the bakcend.

Ghais removed the branches in his repository and had the thesis reviewed by a gpt specialized in reviewing thesis. And provided the feedback that Arnaud will incorporate.

Arnaud can write about the testing, sergei will finish backend design and start on backend implementation. Ghais will write about the GUI in frontend design and about frontend implementation.

We will not meet Frode tomorrow but we will send him tomorrow the thesis for review and before that the feedback from gpt will be incorporated in the text.

end time: 10:45

Group meeting 5 May 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00

Arnaud mainly worked on the testing part of the thesis. Hopes to get it done today and then work on the deployment part.

Also went through the feedback from Frode. Will move what was written in the theory part in the design and implementation part. And will rewrite the theori part by focusing on the problem, and explaining the problem.

So the plan for the week will be to write about deployment and hopefully about the working process.

Sergei made a plan for the implementation part and will write more this week. There was many exams and deadlines last week so could not do much.

Ghais worked on the implementation part of the frontend and the GUI design. Ghais knows someone that could go through our thesis before submission. Which is good because Frode recommended us to do that. Will also look with Vebjørn when to look at the thesis. Vebjørn will go through the thesis one time. So it may be best to wait until the thesis is a bit more complete.

Tomorrow meeting with Frode. Will ask about what I should write in theory part before making any changes.

end time: 12:15

Supervisor meeting 6 May 2024

Present: Frode, Arnaud

start time: 13:00

So I asked about the theori part of the thesis. The main issue with this part is that it is too technical. It goes too deep in technicalities. And it is too long for what it should be. Normally, this part is between 2 and 6 pages.

What it should talk about is what the reader should know to understand the problem addressed by the thesis.

What are the typical problems and the typical solutions. But it should not be technically heavy.

The rest should be moved to the design part. Both the GUI and the technical design part.

For the requirements, it should be based on a use-case. We should see at previous bachelor thesis to get an idea of what it does.

In the design part, more detail should be given about the solution, the various alternatives and the choices that we made, and why we made the choices that we made.

The implementation part should explain how our codebase is built. It could show code examples but not too much. And more details can be given on specific part. Some people put their whole codebase in appendix. We can do that if we don't have too much code. And we put only our code in there.

We have to send him what we have for the 2nd review on May 13. Because he is gone on May 15. And our meeting will be online then.

He also mentioned that we should get a Norwegian to check our Norwegian summary.

end time: 13:12

team meeting Sunday 12 May 2024

Present: Arnaud, Ghais, Sergei

starting time: 12:00 pm

Ghais: Went through the thesis and provided some feedback;

1. Explain what Discord is
2. Reduce the meeting section in the process chapter. If Frode or Ghais' contact person says it is too long and not necessary, Arnaud will reduce it.
3. In the process chapter, change the name of the theory section
4. Remove the Golang example.
5. Mention Norkart in the acknowledgement section

Sergei: Has been working on the implementation part, will write more in the coming days. Moved the requirements part of the testing chapter to requirements. Arnaud would prefer to keep it in the testing chapter, so that everything about testing is in the testing chapter.

Also either did or planned to do changes in the theory section. Will let Arnaud know.

Arnaud: Has been writing the thesis and will finish the conclusion today.

We will have a meeting after receiving Frode's second review to decide who works on what.

We will provide Ghais' contact person that will review our thesis a Word document of the pdf and a clone of our Overleaf project.

Arnaud will contact Frode and ask at what time max can we send him the draft monday.

end time: 12:30

Supervisor meeting 15 May 2024

Present: Frode, Arnaud, Ghais, Sergei

start time: 12:45

In the design, we have a lot of technical material. We should have more specifically about user interface design and using screeshots. We should show our solution.

One thing that we could do would be to have GUI design as a full chapter.

In the development process, we don't need to write too much about milestones. We don't need to write too much about meeting discussion. For testing, we should shorten as well.

Same thing for artificial intelligence. We should be shorter on that as well.

Our chapters should have an introduction each.

How many pages should we have? There are no hard ranges. But something between 50 and 90 is good.

For the presentation, we were thinking of having a video presentation. Is that ok? Or should we have a live presentation? It is better with a live presentation.

Our chapters are imbalanced. There should be more in Design and Implementation then in testing for example.

We will not meet Frode again before handing in.

end time: 13: 15

Team meeting 14 May 2024

Present: Arnaud, Ghais, Sergei

starting time: 13:15

Arnaud will shorten the testing and development process section.

Sergei will finish the implementation and design of the backend.

Ghais will create a GUI design chapter with screenshots.

Ideally we will keep the thesis under 90 pages.

We will also add an introduction in each of the chapters we worked on.

end time: 13:45

Team meeting 18 May 2024

Present: Arnaud, Ghais, Sergei

starting time: 19:00

We went through the thesis together and made some small modifications and additions.

We added introductions at each chapter, among other things.

Sergei will make an entity diagram. Arnaud will add the hours logg at the appendices.

Arnaud will add to the implementation the message when the job is completed through http and then a message is sent through websocket to other users.

Arnaud will also add websocket flow diagram to the implementation section. It will show data being sent to multiple users.

Arnaud will also review the conclusion for grammar. After that, it can be handed in.

Arnaud will take care of handing in.

Sergei will let Arnaud know when he finishes the entity diagram.

We will meet on June 1 to talk about the presentation.

end time: 20:30

Appendix L

Feedback and advice from ChatGPT on our thesis report

Specific Feedback on Grammar and Writing Style with Examples

1. **Terminology Consistency:** Define and use terms such as "collaborative editing" consistently throughout the document.
   Example: Define 'collaborative editing' the first time it appears and differentiate it consistently from 'real-time editing' throughout the document.

2. **Active Voice Usage:** Convert passive constructions to active voice to enhance clarity and directness.
   Example: Change 'The project was directed by us' to 'We directed the project'.

3. **Verb Tense Consistency:** Ensure verb tenses are consistent within each section to improve readability.
   Example: If you begin describing your methodology in the past tense, continue using past tense throughout that section without switching to present tense.

4. **Sentence Structure:** Simplify complex sentences to make the text easier to read and understand.
   Example: Instead of 'The server, which processes data, sends it to users after filtering through complex algorithms', use 'The server processes data and, after filtering, sends it to users.'

5. **Precision in Descriptions:** Be specific in your descriptions, especially when discussing technical details.
   Example: Instead of 'We used .NET for backend development', specify 'We utilized .NET Core 3.1 to leverage its support for real-time web functions through SignalR.'

6. **Article and Preposition Usage:** Correct the use of articles and prepositions, which are often minor but impact the professional quality of writing.
   Example: Change 'We implemented features in software' to 'We implemented features in the software system.'

7. **Avoid Redundancy:** Reduce unnecessary repetition of words and phrases.
   Example: Instead of 'We asked users to give their feedback and users responded to our queries', use 'We requested feedback from users, who responded promptly.'

8. **Punctuation Accuracy:** Ensure punctuation marks are used correctly to avoid confusion and enhance readability.
   Example: Replace 'We developed a system, that integrates seamlessly' with 'We developed a system that integrates seamlessly.'

9. **Transitions Between Sections:** Use transitions to ensure a logical flow of ideas from one section to the next.
   Example: After concluding the methodology section, transition to the results section with 'Following our detailed methodology, we now present the system's performance results.'

10. **Refining the Abstract and Conclusion:** Ensure that the abstract and conclusion sections succinctly summarize the thesis and reflect the significance of your findings.
    Example: In the abstract, succinctly summarize the problem, your approach, and the main findings, such as 'Our research addresses the implementation of real-time collaborative editing in municipal job management systems, enhancing operational efficiency by 30

Specific Feedback on Bachelor Thesis Content with Examples

1. **Scope and Clarity:** The thesis ambitiously covers a broad range of features. Focus more on critical components like conflict resolution mechanisms.
   Example: Instead of covering multiple collaboration features broadly, deeply analyze the conflict resolution mechanisms in Google Docs and their applicability to the Komtek system.

2. **Literature Review:** Expand the comparison of existing technologies.
   Example: Detail the response times and user experiences of real-time collaboration features in Google Docs vs. Microsoft Word, discussing their applicability to municipal job management systems.

3. **Technical Choices:** Justify the selection of .NET and C# more robustly.
   Example: Explain the choice of .NET by discussing specific libraries that support real-time data processing unavailable in Node.js or Python.

4. **System Architecture and Data Flow:** Elaborate on these aspects with more detailed diagrams or examples.
   Example: Provide a detailed diagram illustrating the data flow when two users edit the same document simultaneously, explaining how data conflicts are managed.

5. **Results and Analysis:** Clarify the presentation of results with more quantifiable data.
   Example: Show a graph comparing the time it takes for changes to become visible to all users before and after implementing the new real-time features.

6. **User Testing and Feedback:** Describe the user testing process in more detail.
   Example: Describe how user feedback was collected through surveys after testing sessions and how this feedback led to specific improvements in the GUI layout.

7. **Discussion of Limitations:** Address scalability and potential system limitations.
   Example: Analyze how the system would perform under the load of 100 simultaneous users and suggest potential infrastructure improvements.

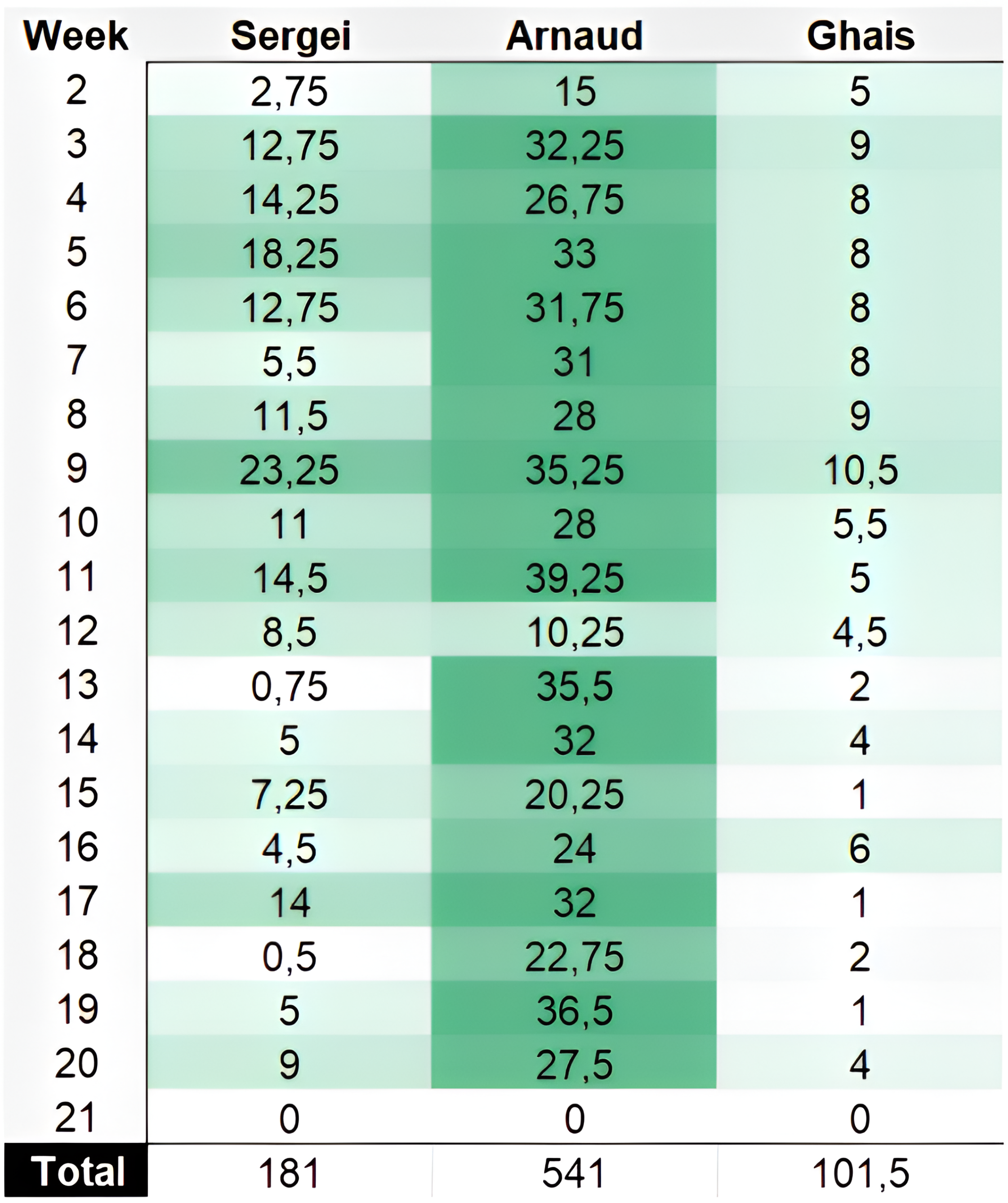
8. **Integration of Diagrams and Visuals:** Ensure all visuals are clearly integrated and labeled.
   Example: Refer to Figure 2.4 directly in your text when discussing the influence of user interface design on real-time collaboration and ensure all figures are clearly labeled.

9. **Citations and References:** Enhance citation practices.
Example: Provide citations for each technical claim, such as 'WebSocket provides lower latency compared to HTTP polling', from reliable sources.

10. **Conclusions and Recommendations for Future Work:** Strengthen your conclusions.
Example: Summarize the effectiveness of different conflict resolution strategies and propose areas for further research, like adapting these strategies for mobile platforms.

Appendix M

Hours logg



Week	Sergei	Arnaud	Ghais
2	2,75	15	5
3	12,75	32,25	9
4	14,25	26,75	8
5	18,25	33	8
6	12,75	31,75	8
7	5,5	31	8
8	11,5	28	9
9	23,25	35,25	10,5
10	11	28	5,5
11	14,5	39,25	5
12	8,5	10,25	4,5
13	0,75	35,5	2
14	5	32	4
15	7,25	20,25	1
16	4,5	24	6
17	14	32	1
18	0,5	22,75	2
19	5	36,5	1
20	9	27,5	4
21	0	0	0
Total	181	541	101,5