Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Hans Kristian Olsen Granli
Torje Dahll-Larssøn Thorkildsen

Developing the next iteration of Qs

Can we complete the new Qs so that it will be an improvement compared to the current system?

Bachelor's project in Computer Engineering
Supervisor: Tomas Holt
May 2021

NTNU
Norwegian University of
Science and Technology

Hans Kristian Olsen Granli
Torje Dahll-Larssøn Thorkildsen

Developing the next iteration of Qs

Can we complete the new Qs so that it will be an improvement compared to the current system?

Bachelor's project in Computer Engineering
Supervisor: Tomas Holt
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Preface

This report is written by two students at NTNU as a part of a bachelor thesis in the course TDAT 3001 - Bachelor Thesis in Computer Engineering. The teams consists of Hans Kristian Olsen Granli and Torje Dahll-Larssøn Thorkildsen. The assignment was proposed by 3D Motion Technologies, and we chose this assignment because both members of the team have been using the old version of the Qs queue system for more than two years. We thought the old system lacked a lot of essential features, and we wanted to partake in the development of the new version.

First and foremost we would like to thank Tomas Holt of 3D Motion Technologies as both the project manager and our supervisor. We would also like to thank the student assistants, students and teachers in the course IDATT2105 - Full-stack application development - for testing our product and provided valuable feedback throughout the development process.

Hans Kristian Olsen Granli

Torje Dahll-Larssøn Thorkildsen

Assignment

This assignment is a further development of two bachelor assignments made in the spring of 2020, as well as one group (group 3) that worked on the project as a part of TDAT 3022 - Software Engineering Project in the autumn of 2020. The two bachelor groups worked separately on a front-end and a back-end. The last group tried to put it all together, and fix security flaws.

Our job was then to stitch everything together, make it a commercially viable product in a production ready state, and ultimately deliver a superior product to the old version of Qs.

The full list of functional and non-functional specifications can be found in the vision document in the appendix.

Scope

When examining the project and reading the report from group 3 we quickly found that the back-end had severe design flaws which would make any security improvements a mere band-aid fix. We therefore built the whole back-end including the database and API up from scratch.

Additionally previously conducted user tests showed a significant preference towards the old system. So the front-end would need work on both in terms functionality and appearance.

Summary

Qs is a Queue System made to ease the process of approving and getting help for exercises. It has roots back so SKS (Smart Køsystem - Smart Queue System). And this will be the second iteration of the system known as Qs. Qs is a two-part application with a front-end web client and back-end server. The old iteration of Qs uses NodeJS back-end and AngularJS front-end. We inherited a project which had been worked on by three previous groups, which uses Java back-end and React front-end.

Our job was to rectify any bugs, and make the system ready for production. The end goal for the product owner, 3D Motion Technologies, is to make Qs commercially viable.

During the exploration phase we quickly realized that the new back-end had inherited some severe design flaws from the old system. Additionally the implementation was lackluster and needed significant work. We therefore designed a new back-end database, a new RESTful HTTP-API, implemented a connection pooling framework, extended WebSocket service and drastically improved runtime performance. In order to choose the correct connection pooling framework we conducted extensive research as well as running benchmarks to verify performance claims. The result of our work on the back-end resulted in average response times being reduced to 1/10 of the inherited project.

The previous group who worked on this project conducted user tests which stated clear points of improvement. The client also had to be updated to meet the functionality offered by the new back-end. The style sheets on the client has been re-written using modern CSS grid to improve portability and to make the system easier to maintain. New features such as chat, room editor, user photos and the possibility of calling a group for video chat have been added.

Covid-19 made testing the system a challenge. However, we managed to conduct two large scale user test with around 80 participants. This was done to verify both the performance of the back-end during a moderate and realistic load, and to get feedback on the changes made to the user interface.

The system we are delivering is still somewhat rough around the edges. Specifically the WebSocket service should be re-factored to host multiple endpoints to reduce memory strain. Additionally, editing exercises has potential issues that needs to be sorted out.

However, we believe that this system is in a state where it is ready to replace the old version of Qs. It has more features, better performance and we believe it offers a superior mobile experience.

Table of Contents

List of Figures viii

List of Tables viii

1 Introduction 1

1.1 Clarifications . . . 1

1.2 Abbreviations and glossary . . . 1

2 Theory 2

2.1 Client-server Architecture . . . 2

2.2 Relational database systems . . . 2

2.3 REST . . . 2

2.4 HTTP . . . 3

2.4.1 HTTP REST-API . . . 3

2.5 Websocket . . . 3

2.6 Authorization and Authentication cookies . . . 4

2.6.1 Authorization . . . 4

2.6.2 Authentication token . . . 4

2.6.3 Checksum . . . 5

2.7 Teamwork Method - Agile Kanban and Lean . . . 5

3 Method and chosen technologies 7

3.1 Front-end . . . 7

3.1.1 TypeScript . . . 7

3.1.2 Babel . . . 7

3.1.3 React . . . 8

3.2 Back-end . . . 8

3.2.1 Hikari Connection Pooling . . . 8

3.2.2 Java Beans . . . 8

3.2.3 Jersey . . . 8

3.2.4 Spring and Spring Beans . . . 9

3.2.5 Security . . . 9

3.2.6 Maven & Tomcat . . . 9

3.3 Teamwork Methodology - Agile Kanban . . . 9

3.4 Testing . . . 9

3.4.1 Virtualization - QEMU & KVM . . . 9
3.4.2 Connection Pool Framework . . . 10
3.4.3 Performance test . . . 10
3.5 P5.js . . . 11
3.6 User testing . . . 11

4 Results **12**
4.1 Scientific results . . . 12
4.1.1 Back-end . . . 12
4.1.1.1 Hikari CP Perfomance benchmark . . . 12
4.1.1.2 Performance tests . . . 14
4.1.1.3 Database . . . 14
4.1.1.4 REST API . . . 15
4.1.1.5 Security . . . 15
4.1.2 Front-end and User interface . . . 15
4.1.2.1 Design changes . . . 15
4.1.2.2 User photos . . . 20
4.1.2.3 Room editor . . . 20
4.1.2.4 Chatting system . . . 21
4.1.2.5 Dark mode theme . . . 22
4.1.2.6 Student queue view . . . 23
4.1.2.7 Notifications . . . 24
4.1.3 User testing . . . 25
4.1.3.1 Production Test 1 . . . 25
4.1.3.2 Production Test 2 . . . 25
4.1.4 Code documentation . . . 26
4.2 Engineering results . . . 26
4.2.1 Functional properties . . . 26
4.2.2 Non-functional properties . . . 27
4.2.2.1 Good security . . . 27
4.2.2.2 Reliable Website . . . 27
4.2.2.3 Stability . . . 27
4.3 Administrative results . . . 27
4.3.1 Timesheet and activity . . . 27
4.3.2 Planning and organizing . . . 28

5 Discussion **30**

5.1 Scientific results . . . 30

5.1.1 Back-end . . . 30

5.1.1.1 Choice of Connection Pool Framework and Benchmarks . . . 30

5.1.1.2 Performance test . . . 30

5.1.1.3 Database Structure . . . 31

5.1.1.4 API Design . . . 32

5.1.1.5 WebSocket . . . 32

5.1.1.6 Code documentation . . . 32

5.1.2 Front-end . . . 33

5.1.2.1 Reactive design . . . 33

5.1.2.2 Dark mode (theme option) . . . 33

5.1.3 Code documentation . . . 34

5.2 Engineering results . . . 34

5.2.1 User tests . . . 34

5.2.1.1 Production Test 1 . . . 34

5.2.1.2 Production Test 2 . . . 35

5.2.2 Qs in a system perspective . . . 35

5.2.3 Professional ethics . . . 35

5.3 Administrative results . . . 36

5.3.1 Group Reflection . . . 36

6 Conclusion and Further Work **37**

Bibliography **38**

Appendix **39**

A Old REST-API endpoints **39**

B New REST-API endpoints **46**

C Gantt diagram **48**

D Vision Document **49**

E System Documentation **58**

F Requirements Documentation **74**

G Project Handbook **82**

List of Figures

1 Checksum function 5
2 External CP framework test 12
3 Cycle Connection graph 13
4 Cycle statement graph 13
5 Login page comparison 16
6 Subjects page comparison 16
7 Location page comparison 17
8 Edit room comparison 17
9 Queue comparison 18
10 Queue student comparison 18
11 New approval list 19
12 Subjects page reactivity 19
13 Queue element reactivity 20
14 Settings page 20
15 Room editor 21
16 Chatting system 22
17 Dark mode toggle 22
18 Dark mode in subjects page 23
19 Queue view for students 23
20 Session expiring warning 24
21 Video call prompt 24
22 Work activity: Hans Kristian Granli 28
23 Work activity: Torje Thorkildsen 28
24 Issue board 29
25 Gantt diagram 29
26 Bad documentation naming 32

List of Tables

1 Performance benchmark of the servers on a powerful virtual machine without accessing the database 14
2 Performance benchmark of the servers on a powerful virtual machine with accessing the database and fetching one cell of data. 14

1 Introduction

Qs is a product that solves the issue of organizing help and approval of exercises on University Courses. At its core its a queue system with added features. This means that a good implementation can be used for broader problems that involve queues. This project is a continuation of [AW20], which in turn was a project with the aim of connecting the projects made as a part of the bachelor theses [AR20] and [MM20].

In this report we aim to answer the following thesis:

Can we complete the new Qs so that it will be an improvement compared to the current system?

The report is divided into 6 main chapters. After the introduction it goes on to explain theory that is relevant to the project. Following the theory, the methods and chosen technologies of the project are described. These put the theories from the preceding chapter into use. Then, we display our scientific, engineering and administrative results, followed by discussion of these findings. The report ends with a conclusion and suggestions for further work. The bottom half of the document lists our references and attachments.

1.1 Clarifications

Since we will mention multiple iterations of the same Qs system, we need clarification on which system we mean.

The system we are delivering and have been working on is referred to as *our system*.

The system we were given at the start of the assignment will be referred to as *the new system*.

The currently running system will be referred to as *the current system*.

1.2 Abbreviations and glossary

- **Qs**: Abbreviation of Queue System and is the name of the application.
- **API**: Application programming interface. Defines interactions between two or more software-components.
- **URL**: Uniform Resource Locator is often referred to as a web address, but it specifically specifies the address of a machine on a network.
- **HTML**: Hypertext Markup Language. The standard language used to create web-pages.
- **CSS** : Cascading Style Sheets. The standard language used to style an HTML page.
- **SQL**: Structured Query Language. A language to create and interact with relational databases.
- **DAO**: Data Access Object. An interface between a database and the rest of the code. These objects create an abstraction layer and simplifies database access from other parts of the code.
- **Viewport**: The area in which the software is rendered. For a website, this will vary based on the size of the browser window. Smaller devices such as phones will have smaller viewports.
- **Cookie**: A cookie is a token stored on a client machine. Cookies are often sent along with requests to servers and provide additional information..
- **Distributed Hypermedia Systems**: Hypermedia is a medium of information including graphics, audio, video, plain text and hyperlinks.

2 Theory

2.1 Client-server Architecture

Client-server architecture is a system with one or more clients connected to one or more centralized servers across a network. In this architecture, the clients sends requests to the server, and the server either responds right away or communicates with another component, like a database.

The theory behind the client-server architecture is described in more detail in the report [MM20].

2.2 Relational database systems

A relational database is a form of data storage. It organizes data in tables and creates relational connections between the data. Every entry in a relational database must have a unique key or key pair to separate it from its peers, this key is known as a primary key.

The act of separating related data into different tables and reducing data redundancy is called *normalization*. To connect two tables together we store the primary key of a foreign table known as a foreign key. When fully utilized - normalization leads to less double saving of data, and since the data is stored as a relation updating a single row will implicitly update all connected rows, which leads to less complexity.

Most modern relational databases allow us to use SQL to interact with the databases. By using JOIN statements we can combine the related data between the tables. When designing a relational database it is important to consider the number of relations and type of relationships. There are three types of relations; one-to-one, many-to-one and many-to-many. We don't have to make any special considerations in the case of one-to-one relations, and we can even store the data in the same table. The case of many-to-one means that one entry in database A can be mapped to multiple entries in table B, but an entry in table B can only be mapped to one entry in table A. Many-to-many relations often force us to create a connection table, connecting multiple entries from table A to multiple entries from table B.

2.3 REST

Representational state transfer is a design architecture for distributed hypermedia systems [Fie00]. REST has the following characteristics:

- **Separation of Client and Server**
  Separation of client and server improved portability of the user interface, reduces complexity and improves the scalability of the system.
- **Statelessness**
  Each request must contain necessary information by itself for the server to fulfil the request.
- **Cacheable**
  Cache constraints require that the data within a response to a request be implicitly or explicitly labelled as cacheable or non-cacheable. This property leads to less strain on the network and faster response time for the end-user.
- **Uniform interface**
  Uniform interface implies a generalization in the architecture. This means that
- **Layered system**
  A layered style means that the architecture can be built using hierarchical layers and constraining each layer to the point where they cannot see the resources they aren't interacting with. As an example in the current system: Getting a user should only give access to the properties of the user, not what subjects the user is a part of.

- **Code on demand**
  This is an optional property in REST and extends the functionality of a client by downloading and executing code in the form of applets or scripts.

2.4 HTTP

HTTP - Hypertext Transfer Protocol is an application layer protocol. This protocol is a set of standards implemented across the Web (World Wide Web) and is specified by the group IETF (Internet Engineering Task Force). Any URL starting with HTTP implements the HTTP protocol. HTTP, like REST, is a stateless protocol.

The HTTP protocol gives us access to these verbs ([Doca]):

- **GET** - Allows us to fetch a resource. Get requests should only retrieve data.
- **HEAD** - Should give the same response as a GET request, but without the body. This can give us useful metadata about a resource.
- **POST** - Allows us to create an entity on the endpoint
- **PATCH** - Update a single property on the resource
- **PUT** - Update one or more properties on the resource
- **DELETE** - Deletes the resource.
- **CONNECT** - Establishes a tunnel to the server identified by the target resource. This is implemented by all modern web-browsers and is not something a developer needs to think about.
- **OPTIONS** - Asks the server for allowed methods on the target resource.
- **TRACE** - Performs a message loop-back test. This can only be used for debugging.

Using the HTTP-verbs allows us to implement multiple actions on the same endpoint, which improves readability.

2.4.1 HTTP REST-API

Combining HTTP and REST to create an API allows us to use the HTTP verbs and implement resources in a hierarchical structure, where each property in the resource must have a unique identifiers. This does also mean that we can do multiple actions on the same endpoint.

2.5 Websocket

A WebSocket is a two-way connection (full-duplex) between two networked machines. The connection uses the Transmission Control Protocol (TCP) to ensure that both parts send and receive messages successfully. The WebSocket standard was defined by IETF in RFC 6455 [For]. A resource mounted on ws:// will implement the WebSocket protocol (Docs [Docb]).

WebSockets can be used for a lot of different functionality. A website server might use WebSockets to notify their users' browsers when there has been an update that requires them to reload certain parts of the site. Another common use case is chatting systems. Without two-way communication, users would have to continually poll and check for updates, which can be resource-intensive for both the client and the server. This would also affect the responsiveness of the page.

2.6 Authorization and Authentication cookies

2.6.1 Authorization

Authentication is the process of confirming a given identity, as well as checking that information about authenticated identities remains unchanged. There are multiple ways of implementing an authentication system, but for users on a website, it usually starts with logging in with some of their personal information like email, phone number, and a password.

Authorization is the act of enabling permissions and privileges for a user. This works in conjunction with authentication by first confirming a given user's identity, and then giving them the appropriate permissions.

2.6.2 Authentication token

After a user has logged in with their credentials we can confirmed their identity. However, given the stateless nature of HTTP and REST, we must have a way to authenticate each request. We could send the user's email and password with every single request, however, this means that we have to store the password locally on the client's machine. Even though modern browsers are relatively memory safe, this is something that should be avoided. What is regarded as a more secure way is giving the user an authentication token, which stores the identity of the user. This has the added benefit of being shareable between multiple external resources. This is known as Cross-Origin Resource Sharing (CORS).

In computer software, there are several different types of tokens, which represent permissions to perform a certain operation within the system. Session tokens often keep track of an authenticated user's session in a system. If the user doesn't refresh this token within a certain time, it will expire and require authentication again. This is useful as we can create protected stateless HTTP and RESTful resources.

2.6.3 Checksum

In order to verify the authenticity of a token, the creator of the token should append a checksum to the token. A checksum can be used to verify that some information has not been modified, and is generated using a checksum function/algorithm. This function should work in such a way that even a small change in input will generate a dramatically different checksum. The checksum will be of a set length. Examples of such checksum algorithms are MD5 and SHA1.

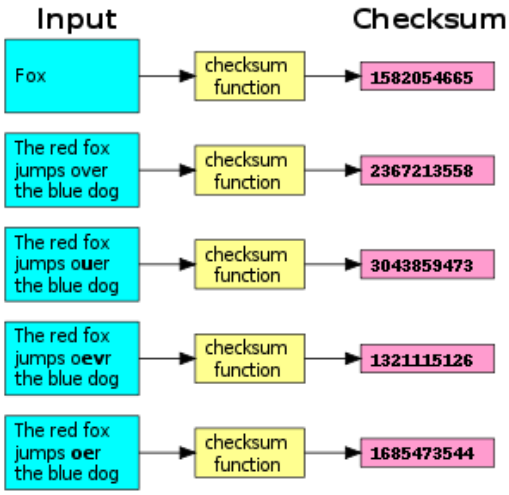


Figure 1: Visualization of how a checksum function modifies the input into a set length of seemingly random numbers. Small changes in the input produce dramatically different outputs.

Source: https://en.wikipedia.org/wiki/Checksum

2.7 Teamwork Method - Agile Kanban and Lean

Agile Kanban is a system for lean manufacturing. Lean System Development is a part of the software development paradigm Agile and was created by Mary Poppendieck and Tom Poppendieck [Mar03]. Lean tries to minimize overhead, and follows the seven Lean Principles:

- Eliminate Waste - Avoid creating something the customer doesn't want or need.
- Build quality in - Ensure quality in all produced components.
- Create Knowledge - This principle encourages teams to provide infrastructure for documentation and retain the knowledge gained from the development process.
- Defer commitment - Options should be kept open, and decisions should be made based on collected data.
- Deliver fast - Avoid over-engineering solutions that don't solve the problem at hand.
- Respect people - Communicate proactively and effectively, encourage healthy conflict, surface issues as a team and empower each other.
- Optimize the whole - A big challenge in software development is making the team greater than the sum of its parts.

Kanban was originally created by Toyota to improve manufacturing efficiency. It highlights problem areas by giving each task a lead time and cycle time. This makes it easier to identify bottleneck areas. In software development, kanban breaks user stories into tasks and makes use of kanban

cards on a kanban board. Unlike regular kanban, agile kanban also focuses on iterations. The kanban cards make the process of improving an already existing system easier, as it exposes areas of weakness.

When developing software using kanban, the kanban board may be broken down into three categories:

1. TODO
2. DOING
3. DONE

This makes it easy for all members of the team to know what is being worked on, and when a member finishes his/her task, they can simply pick a card from the TODO-board. This eliminates the need to speak with a project manager to be assigned to a new issue.

3 Method and chosen technologies

As we are the fourth team to work on this project we did not have a say in the initial decisions made for the structure of the system. We will therefore only give a brief summary of the relevant technologies, and refer to the other reports when appropriate. Many of the needs and properties described in our vision document had already been fully or partially implemented in the version of Qs we inherited. To meet 3D Motion Technologies' goal of a production ready product, and our own thesis, we had to improve all parts of the system.

3.1 Front-end

You can read about the reasoning for choosing the different front-end technologies in [AR20].

3.1.1 TypeScript

Typescript is a framework for writing type-safe JavaScript code:

```
// javascript

function double(example){
    return example+example
}

// typescript

function double(example : number):number{
    return example+example
}
```

It type checks data types, object methods and leads to fewer bugs when writing code, which in turn means that the developers need to spend less time testing and lurking out bugs and runtime errors. Additionally TypeScript serves as method documentation and makes it easy to generate JavaScriptDoc.

3.1.2 Babel

Babel is a framework made to compile code from modern JavaScript syntax to be compatible with older JavaScript versions. This is to make sure that we don't eliminate a potential user base with lower spec computers or outdated software. This means that we can utilize array prototype methods such as map, without worrying about compatibility. [Con]:

```
// Babel Input: ES2015 arrow function
[1, 2, 3].map(n => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function(n) {
  return n + 1;
});

// ref https://babeljs.io/docs/en
```

3.1.3 React

The user interface is created using React, which is a JavaScript framework. It specializes in creating reactive user interfaces in web-based applications. The report [AR20] goes into great depth as to why React was chosen as the front-end framework for this project.

3.2 Back-end

The back-end is written in Java and uses a MySQL database. The reasoning for using Java as the programming language can be found in [MM20].

3.2.1 Hikari Connection Pooling

In order to manage a big load of users at once, and make use of Java's multi-threading capabilities, we needed to implement connection pooling (CP). Connection pooling is the act of creating multiple connections to the database at once and storing them in some data structure. Different parts of the system can then request a connection, use it for its purposes, and then return it to the connection pool. This enables multiple threads to read from the same database at once, and avoids various issues with creating many new connections to a database as it recycles the ones in the pool.

The project we received did not utilize connection pooling. We therefore looked up the different connection pooling Java frameworks. There are multiple alternatives such as Apache Commons DBCP, Hikari CP and C3PO. After doing some research and running some open source benchmarks, we decided to use HikariCP.

3.2.2 Java Beans

In order to reference initiated objects between our classes, we had to implement the connections as concurrent beans. During testing, we quickly realized that the system we received didn't implement resource sharing properly.

Java beans is a standard for Java classes, a bean has the following properties:

1. All properties are private (use getters/setters)
2. A public no-argument constructor
3. Implements Serializable.

Any Java class with these properties can be called a bean. This is important because a serializable class can be written to an IO stream. In other words, a Java bean is a reusable component accessible from multiple sources. Therefore different Java frameworks can access, modify and use the shared objects.

3.2.3 Jersey

Jersey Restful Web Services (JAX-RS) is an open-source framework for developing Restful web services for Java. It requires API classes in form of Java beans. This framework was chosen by the previous group, and their reasoning for choosing Jersey can be found here [MM20].

3.2.4 Spring and Spring Beans

Jersey is a Java framework with a feature called Spring Beans. The difference between a spring bean and a Java bean is that a spring bean doesn't have to meet the same requirements as a Java bean, but still be shared between different Jersey frameworks. For this to be achieved, the Java objects have to be created by the Spring Framework Container. In our project, the Java Database Connectivity Objects (JDBC) are implemented as spring singleton beans and shared between the Jersey API classes.

A **Spring bean** is an object managed by spring. It is not held to the same constraints as a regular Java Bean, but it has to be instantiated, configured and managed by a Spring Framework container.

3.2.5 Security

The system uses a cookie-based authentication system. We have not altered the cookie-related security features we have inherited. You can read more about the security features in [AW20].

3.2.6 Maven & Tomcat

Maven is a dependency management tool for Java and Kotlin applications. It builds and stores all framework and dependencies. Maven can also be configured to run unit tests before compiling and deploying the application. The reasons for choosing Maven instead of Gradle can found in [MM20]. Maven can be used to compile a Java project to a WAR file which can be used by Tomcat.

Tomcat is a framework that helps deploy an application to the web. Since Tomcat 10 was released late during the development period, this project is only stable in Tomcat 9.

3.3 Teamwork Methodology - Agile Kanban

We decided to go for Kanban as our development methodology because looking at user stories made it easy to figure out what the biggest issues with the system were. Additionally, we had little experience in writing API back-end in Java and did not have a frame of reference for how long it would take to re-write the parts of the application which we saw as problematic.

We used git and GitLab hosted by the university for version control. Due to the Covid restrictions, we had to work remotely. Therefore most of our communication went through Discord, which is a communication platform where you can share text and talk. Additionally, we made use of Microsoft Teams for documentation sharing.

3.4 Testing

This section is related to our setup and performance tests.

3.4.1 Virtualization - QEMU & KVM

In order to eliminate as many outside factors as possible, we conducted our performance tests using a virtualized environment.

QEMU is an open-source machine emulator and virtualizer used for creating virtual machines.

KVM - Kernel Virtual Machine is a Linux kernel module that allows a program to utilize hardware virtualization. This means that a virtual machine can make use of features such as Intel IOMMU

(Input-output memory management unit). KVM allows a virtual machine to run at near-native performance.

Our performance tests were done on a virtualized Ubuntu 21.04 running on Manjaro host machine with Linux kernel version 5.12 and KVM enabled. The virtual machine was given 16GB RAM and 14 virtual CPU cores from an Intel i9 9900K processor.

3.4.2 Connection Pool Framework

In order to choose the best framework for the project we had to benchmark the different frameworks available. We ran the benchmark [Woo] on our virtualized system.

3.4.3 Performance test

The inherited project had been performance tested [MM20]. Since we have made multiple changes to the structure and functionality of the server we wanted to replicate the test to see how our changes had affected the server. Following our changes to the system, we could not perform the exact same test to measure the server's performance.

We decided to create a new test where both versions of the server would send exactly the same information back to the test client. We made two versions of the test for the current system; one with a temporary redirect and one with direct access to the endpoint. A temporary redirect was tested because the current system uses them for all new endpoints to redirect them to the ones from the old system (this maintains backward compatibility).

The test used a custom endpoint on both versions. The server would always respond with the same data, which was sent back directly without accessing the database. This way, we could avoid any potential performance differences in our databases, and directly test the servers. The following JSON object was sent back from the server to the test clients:

```
{
    "testObject": "server",
    "performanceTest": "true",
    "test1": "test",
    "test2": "test",
    "test3": "test",
    "test4": "test",
    "test5": "test",
    "test6": "test",
    "test7": "test",
    "test8": "test"
}
```

Listing 1: The test data sent from the server for performance benchmarking without database access.

The request were sent from a Python script rapidly and asynchronously, meaning they never had to wait for the preceding request to get a response. With this approach the server was tested for a load of more than a thousand requests per second, where it always would respond with a JSON object of typical size for the system.

We also made another test that accessed the database in a fair and minimal way so that it can show the differences in the database of the current system and the new system. This test sends requests in a similar Python script and is tested both with temporary redirects, and without.

3.5 P5.js

P5.js is a JavaScript library targeted at artist who want a quick and easy way to create sketches and art within a website. It has a wide range of different drawing functions and website features. P5.js has a main loop called ”draw” that runs a set amount of timer per second. This is where you can use the drawing functions and update the website, and optionally animate various features. This library is used to fulfil our need of a ”Way to create room images” described in our vision document. It is used for the room editor, as well as the room viewer.

3.6 User testing

Due to the ongoing pandemic of Covid-19 our ability to perform user tests was severely hindered. We had no physical lectures and NTNU was periodically closed. Our product manager organized testing with a real class from school to really see if the system was getting production ready.

4 Results

4.1 Scientific results

4.1.1 Back-end

4.1.1.1 Hikari CP Perfomance benchmark

JMH Microbenchmarks is a framework created to isolate and measure the overhead of pools.

- One *Connection Cycle* is defined as opening / close a connection
- One *Statement Cycle* is defined as a single prepared statement, execution and closing of a statement.

The benchmark project may be found at [Woo].

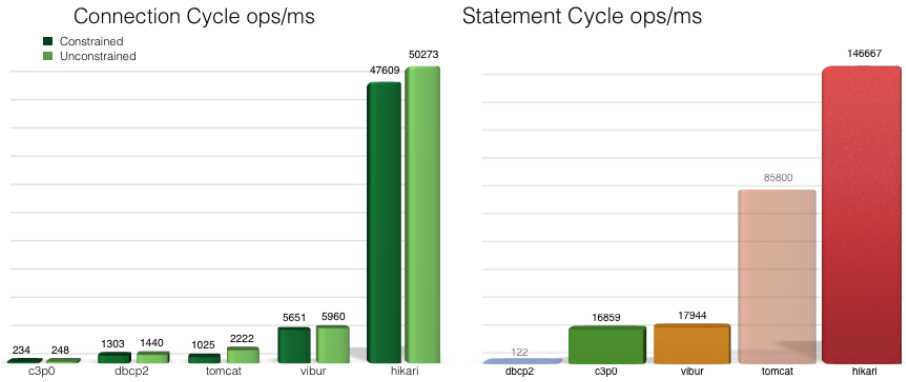


Figure 2: Performance of popular CP frameworks

Source: https://github.com/brettwooldridge/HikariCP

We reran the same open source benchmarking framework to see if Hikari is as good as the developers claim.

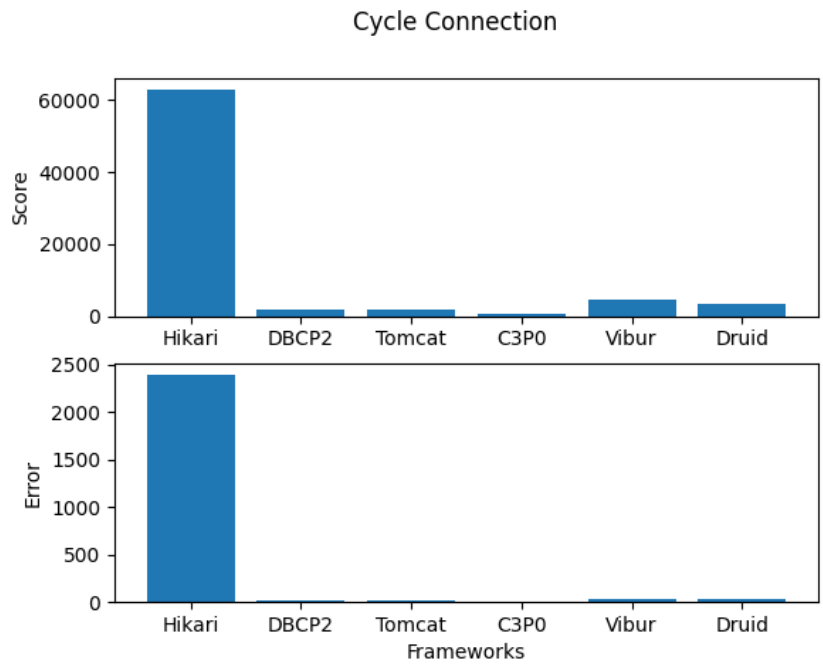


Figure 3: Cycle connection score and error rate

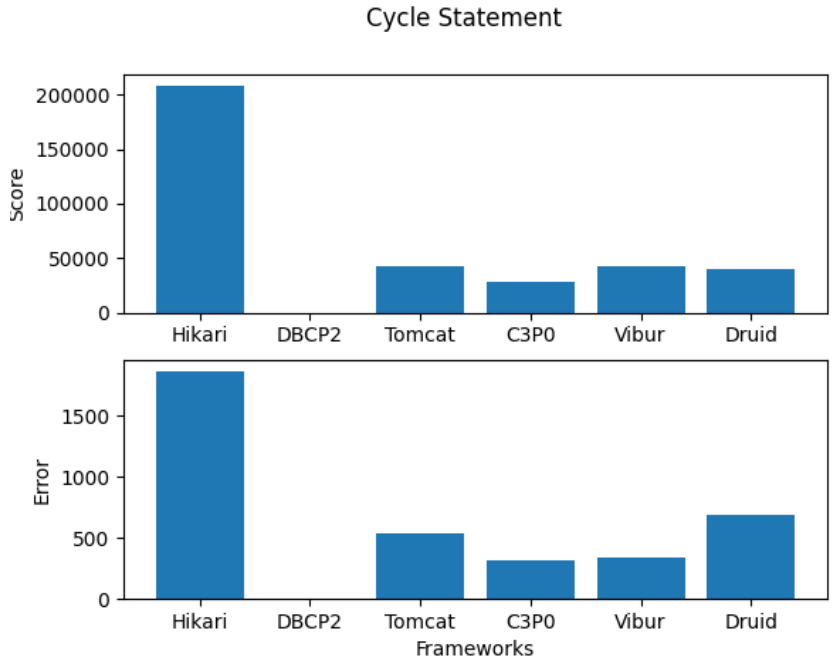


Figure 4: Cycle statement score and error rate

4.1.1.2 Performance tests

For the tables below, we have used the following abbreviations:

NTR: The new system with temporary redirect
NNR: The new system without temporary redirect
OUR: Our system

The following table displays the result of the first tests without accessing the database.

	NTR	NNR	OUR
Samples	50000	50000	50000
Time (min:sec:ms)	01:35:75	01:26:48	00:14:75
Average (ms)	85.60	77.41	9.98
Min (ms)	10.79	9.87	2.35
Max (ms)	1017.20	597.81	334.98
Error %	0.00	0.00	0.00
Throughput (requests/s)	522.21	578.14	3390.84
Sum of response time (hours:min:sec)	01:11:20	01:04:30	00:08:19

Table 1: Performance benchmark of the servers on a powerful virtual machine without accessing the database

The following table displays the result of the first tests with accessing the database and fetching one cell of data.

	NTR	NNR	OUR
Samples	50000	50000	50000
Time (min:sec:ms)	02:20:95	02:13:28	00:16:92
Average (ms)	126.21	119.45	12.09
Min (ms)	20.68	17.20	1.97
Max (ms)	760.38	685.24	430.95
Error %	0.00	0.00	0.00
Throughput (requests/s)	354.73	375.15	2955.20
Sum of response time (hours:min:sec)	01:45:11	01:39:32	00:10:05

Table 2: Performance benchmark of the servers on a powerful virtual machine with accessing the database and fetching one cell of data.

4.1.1.3 Database

We created 7 additional tables for core functionality:

- exercise : table to hold information about an exercise, such as exercise number, due-date, subject and description
- user_exercise : table to store information of which exercises a user has approved
- queue_exercise : table to connect exercises to a queue_element. This replaces the string stored in the old database
- queue_user : table to connect user and queue_element. This does not store the queue_element's owner
- requirement_exercise : table to connect one or multiple exercises to a requirement. This is opposed to the string relation on the old database

- subject_role : table made to replace subject-person. This table stores a user_id, subject_id and the role_id the user has in the subject. This controls access and makes it so that a user can be both student-assistant and student at the same time.
- queue_element_message : table where an entry is connected to a queue_element and stores a simple message from the group to student-assistant. This functionality is present on the old system, but was not a part of the system we inherited.

These were added to improve performance.

Additionally we created the following tables for chat functionality:

- chat_message
- chat_session

The whole database diagram may be found in system documentation in the appendix.

4.1.1.4 REST API

As a consequence of making the API RESTful compliant, the number of endpoints was reduced from 104 to 48. A list of endpoints of the new system and our system can be viewed in the appendix.

4.1.1.5 Security

The server has a cookie based security system. It sends a cookie with relevant information about the user and their role in the system, which is used for authorization. A user cannot change the values of their cookie without altering the checksum of the cookie. When a request is sent to the server, the user's cookie is sent along with it. The server then checks their privileges and either performs the requested actions or refuses if they are not authorized. Our system is designed in such a way that users are only able to receive the minimum amount of information about other users for the system to function. An example of what this means: a student can get the names of other students in his subject, because he might create groups with them for approval. However, he can not retrieve any information about students outside of his subjects.

4.1.2 Front-end and User interface

Throughout the project we have continuously made changes to the user interface, both based on our own knowledge and responding to input from our testers.

4.1.2.1 Design changes

Our system ended up looking quite different from the new system. The figures below show some of their differences.

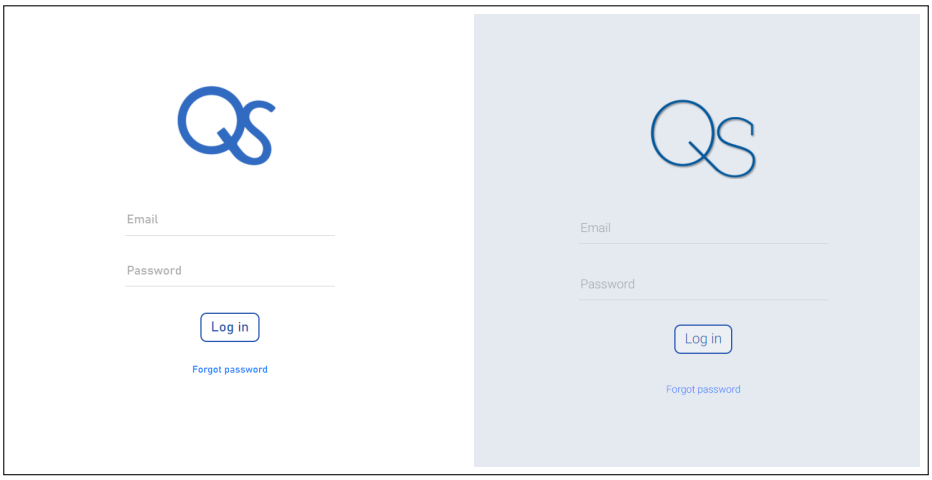


Figure 5: Comparison of the new login screen (left) and our (right).

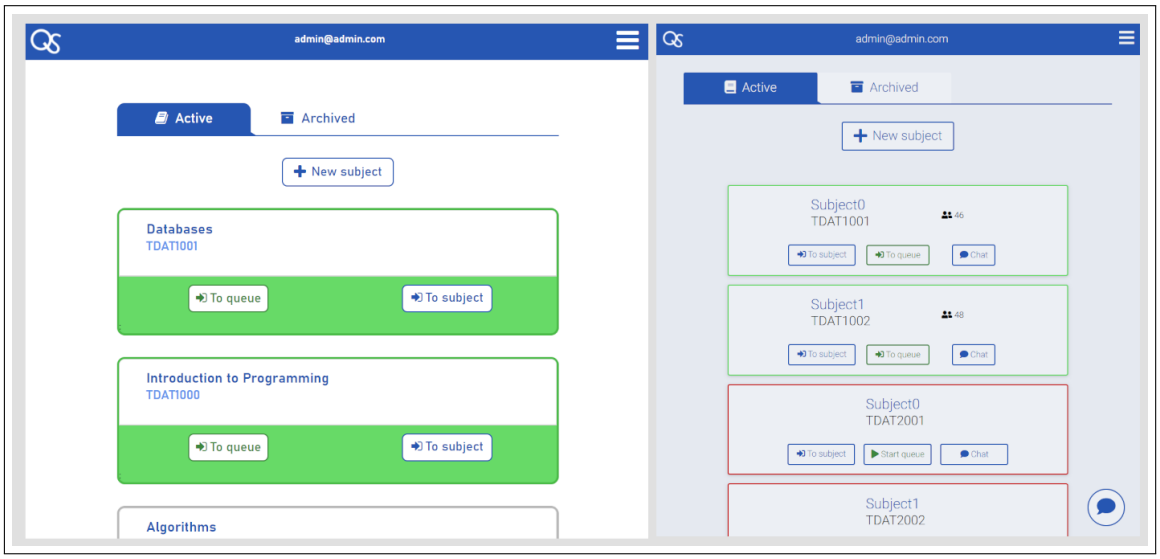


Figure 6: Comparison of the new subjects screen (left) and the our (right).

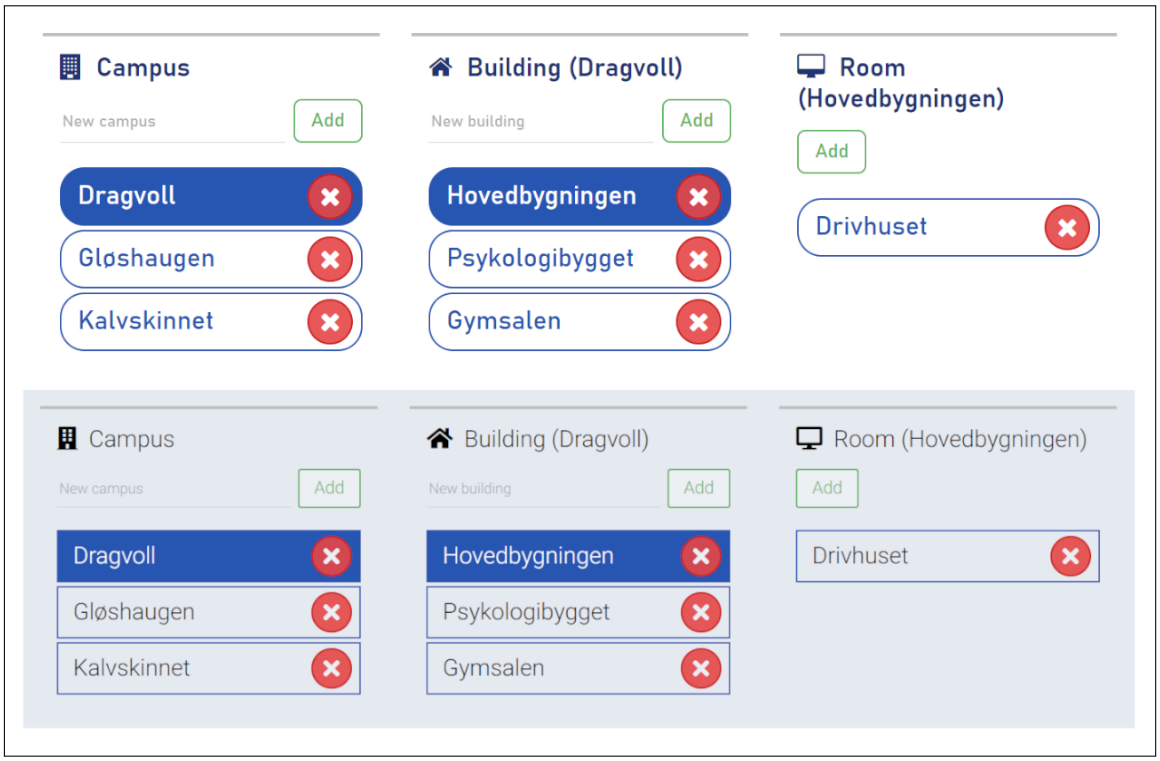


Figure 7: Comparison of the new location page (above) and our (below).

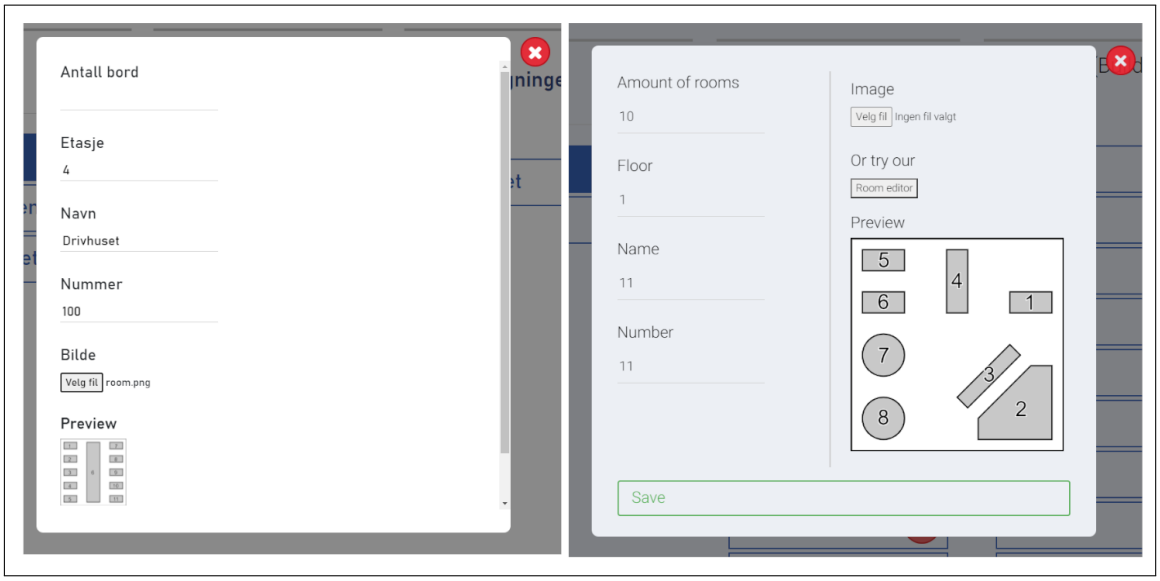


Figure 8: Comparison of the new room-edit screen (left) and our (right).

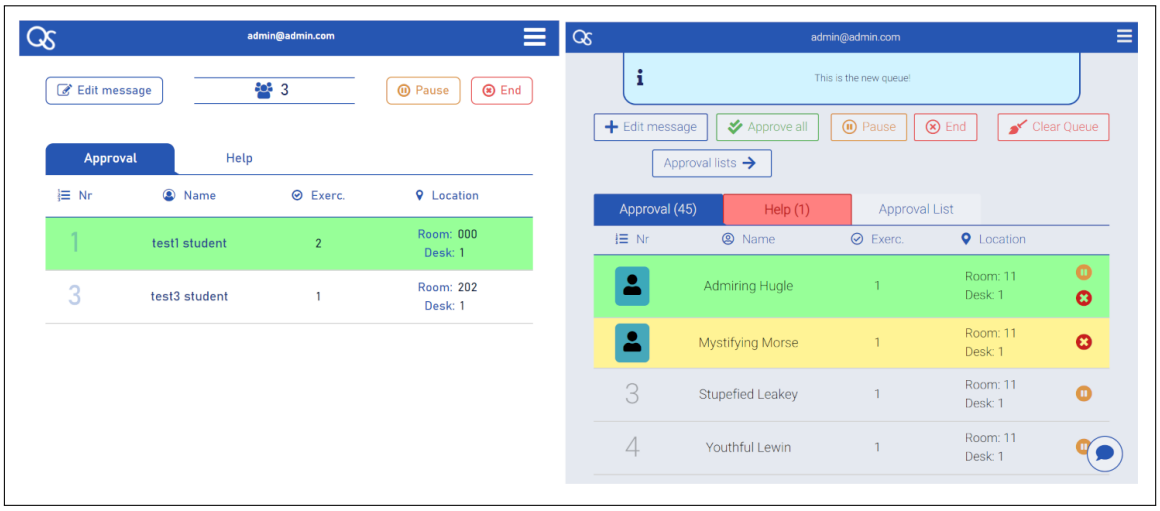


Figure 9: Comparison of the new queue page (left) and our (right).

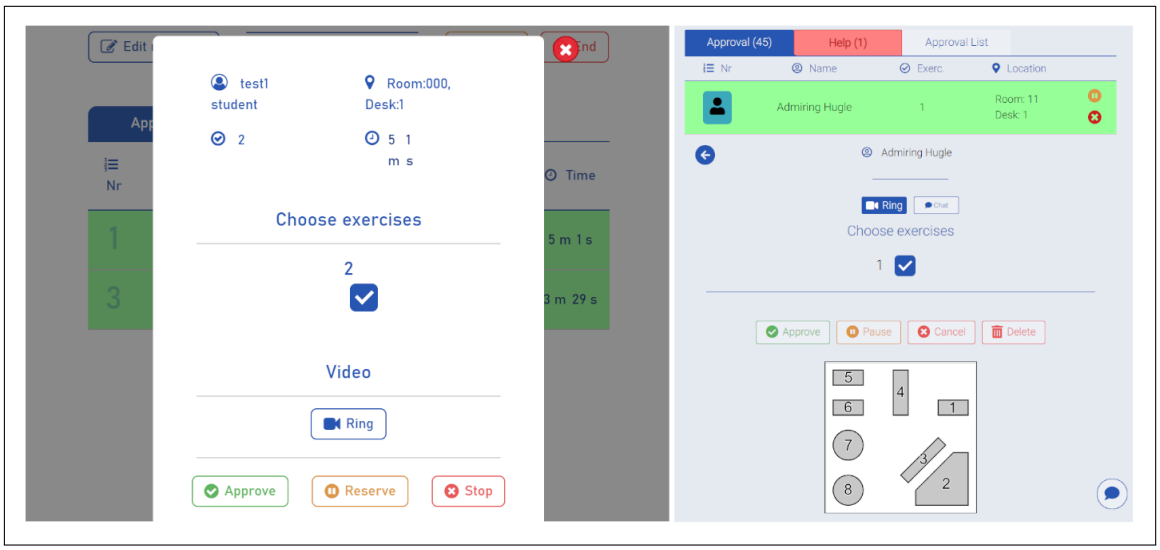


Figure 10: Comparison of the new options when selecting a student from the queue screen (left) and our (right).

Figure 11: Our system's approval list page. Functionality for exercise due date has been added, and exercises approved after due are colored orange.

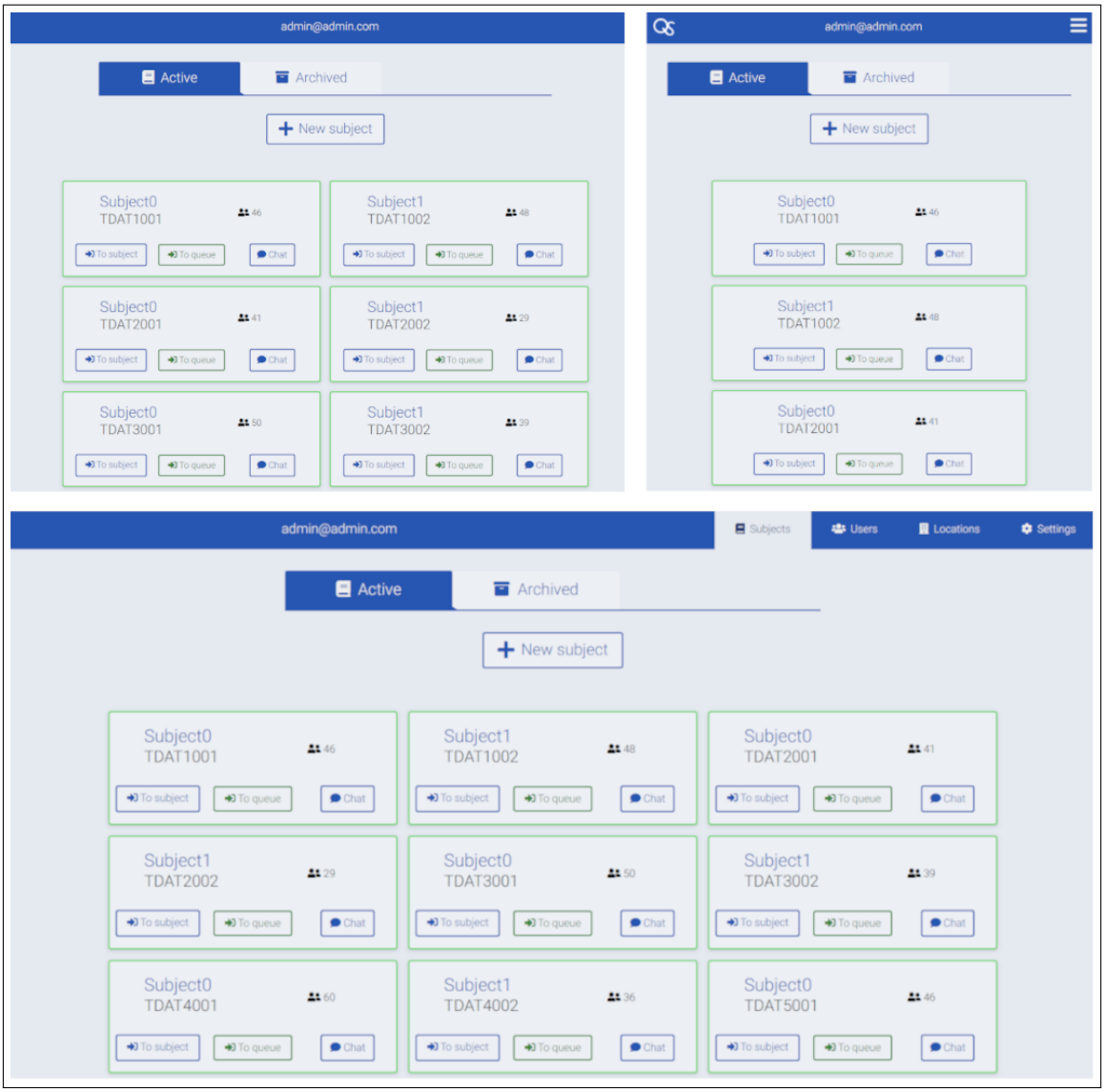


Figure 12: A figure showing how the subjects page reacts to different viewport sizes

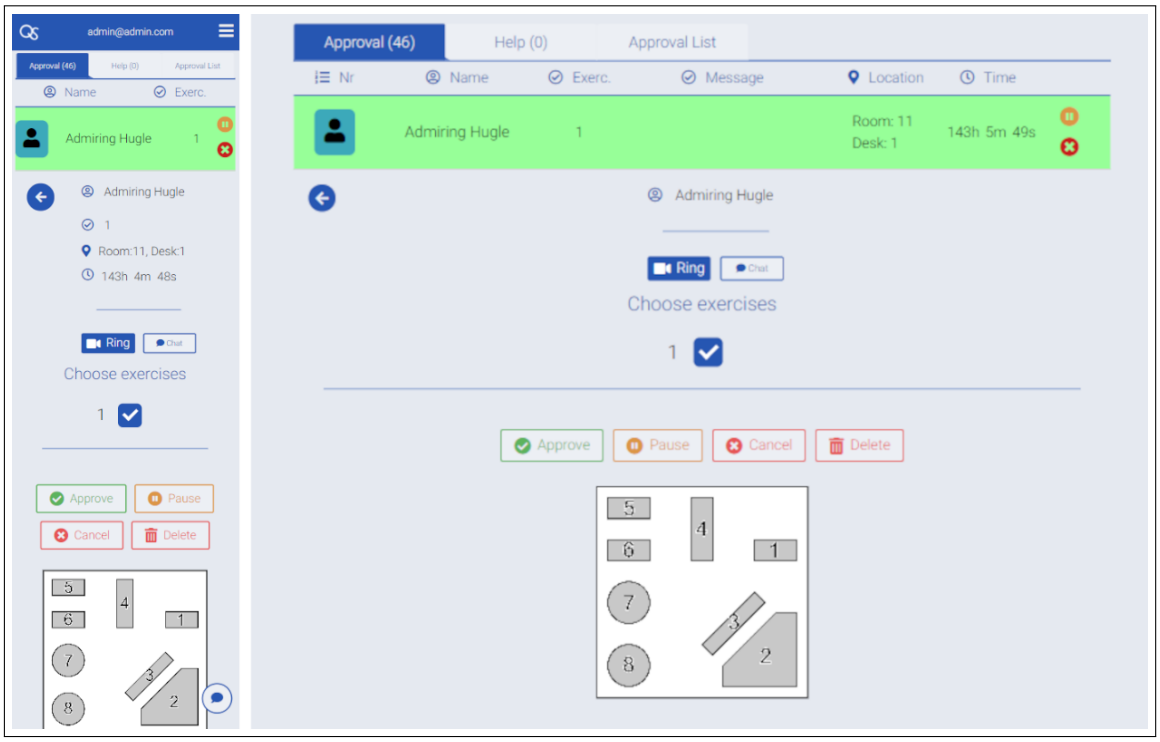


Figure 13: A figure showing how the queue element page reacts to different viewport sizes

4.1.2.2 User photos

Our system lets users upload a photo of themselves. This can only be seen by teachers and assistants, and is meant to simplify the process of finding students withing a room.

Figure 14: The new system's settings page, with an option to upload a photo of the user.

4.1.2.3 Room editor

As requested by the product manager our system includes a room editing application within the website. This can save and load rooms so that you can match the real room setup whenever it changes. It was created using p5.js. The room editor can draw rectangles, circles, text and all

kinds of polygons. All of these can be selected and edited. The editor also includes a grid option for lining up shapes, and a couple of other functions.

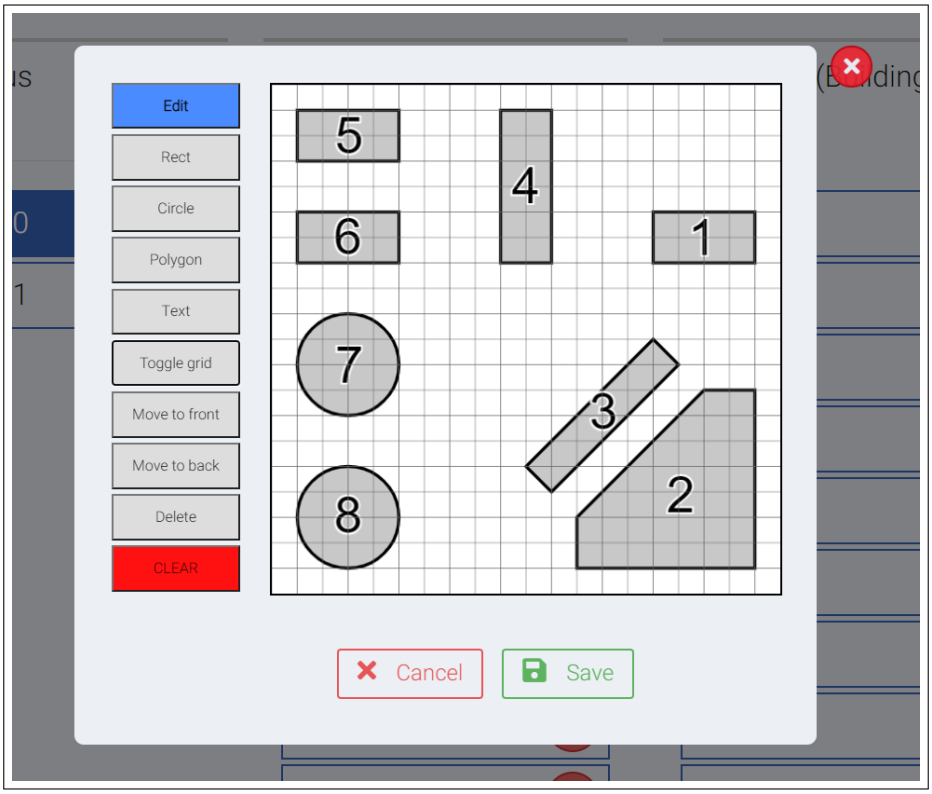


Figure 15: Our system's room editor for creating custom room images. These images can be edited at any point.

4.1.2.4 Chatting system

Our system includes chatting functionality. There are three types of chats;

- **Subject chat** is used when chatting will all users in a subject. Students can discuss tasks and ask teachers for assistants. There is also an option to send messages anonymously.
- **Queue group chat** is used for chatting within a group waiting for approval/help in a queue. The teachers can join this chat and read/write to the students. The students are notified about teachers being able to read their messages in this type of chat, and they have to accept this to start chatting.
- **User-to-user chat** is used when chatting directly to user in the system. This functionality is as of now reserved for teachers and assistants in a subject. Teachers can start conversations with whoever they want within one of their subjects.

A snippet of the chatting system is shown in 16 below.

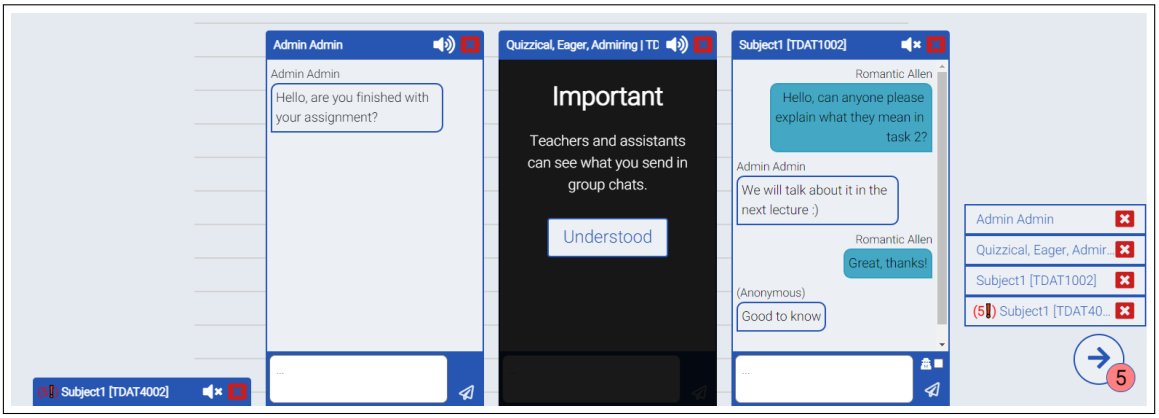


Figure 16: Our system's chatting functionality.

Figure 16 shows the state of four chats, from left to right:

1. A collapsed subject chat.
2. A user-to-user chat.
3. A queue group chat, showing the startup warning for group chats.
4. A subject chat.

4.1.2.5 Dark mode theme

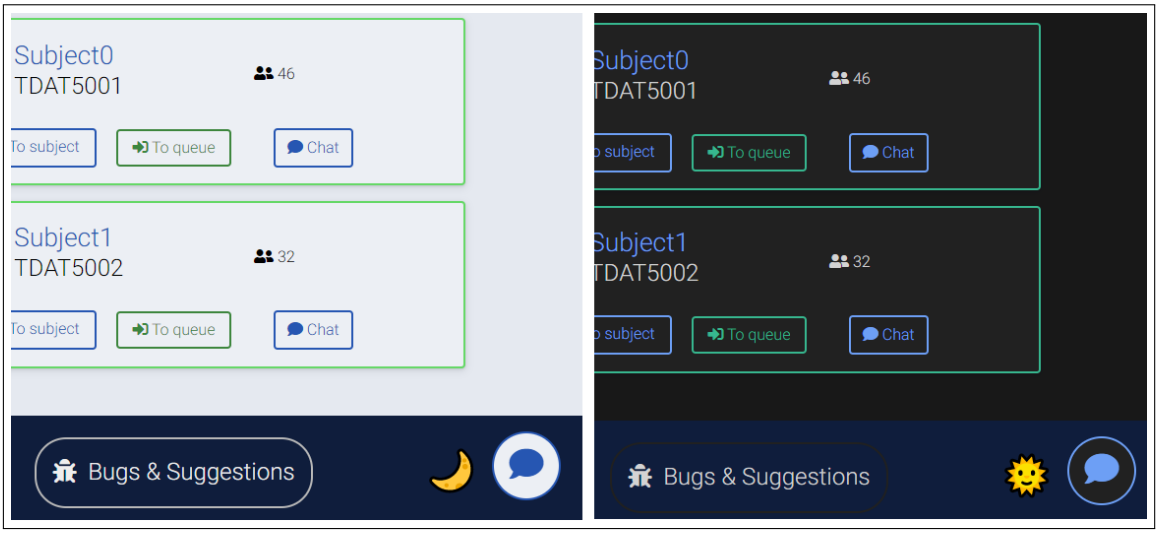


Figure 17: While in standard Qs light mode, the moon in the footer can be clicked to enter dark mode. While in dark mode, this icon turns into a sun that can be clicked to re-enable light mode.

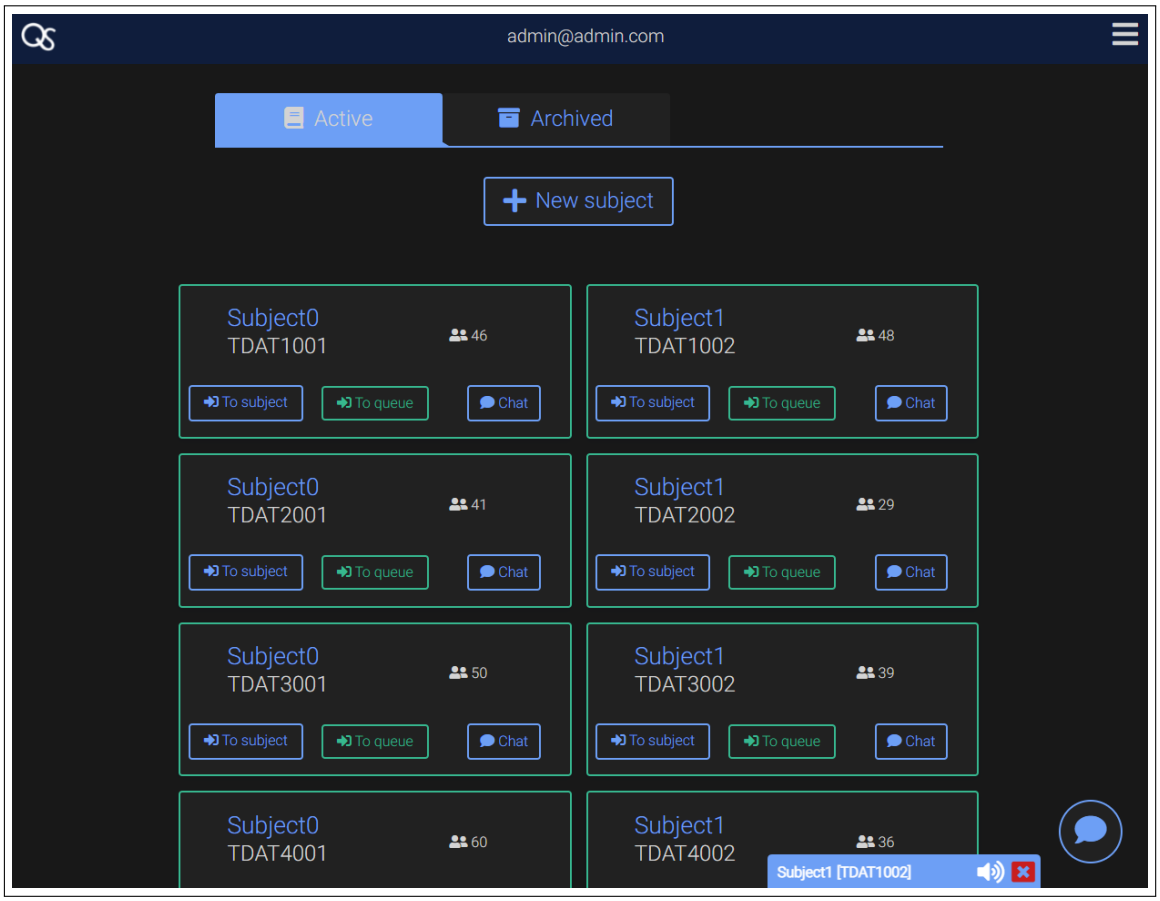


Figure 18: The subjects page in our system's dark mode theme.

4.1.2.6 Student queue view

Figure 19: Students can watch the queue in real time without getting access to other students information.

4.1.2.7 Notifications

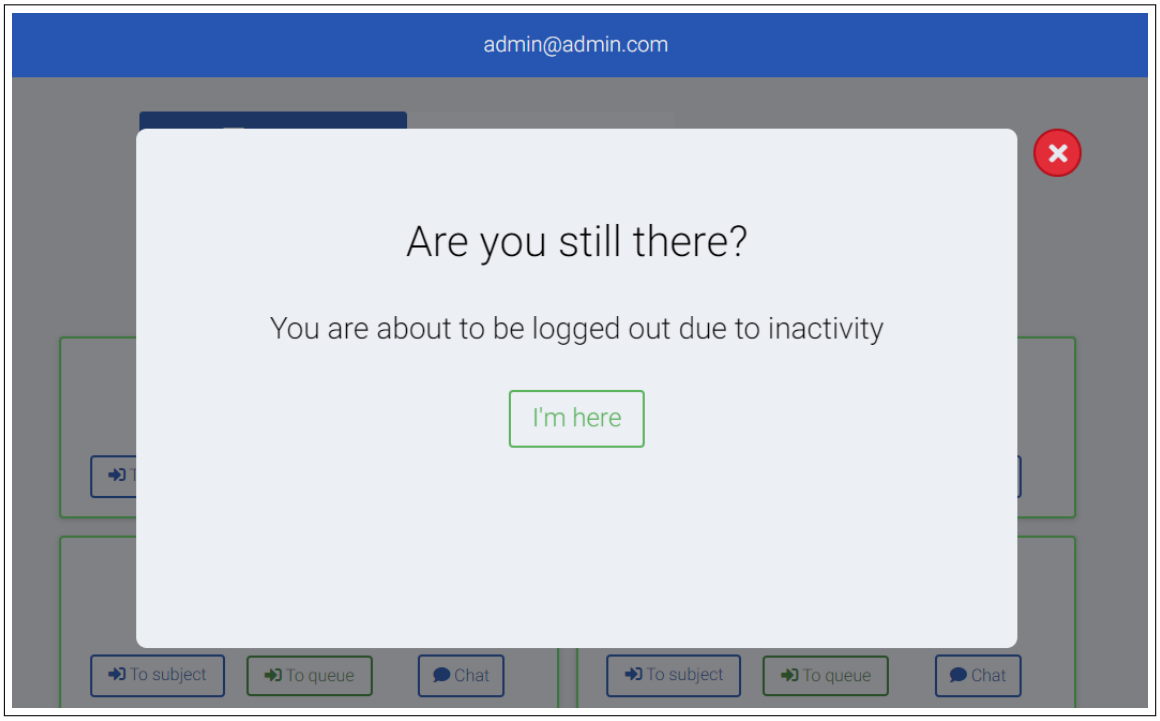


Figure 20: Our system will warn the user if he about to be logged out due to inactivity.

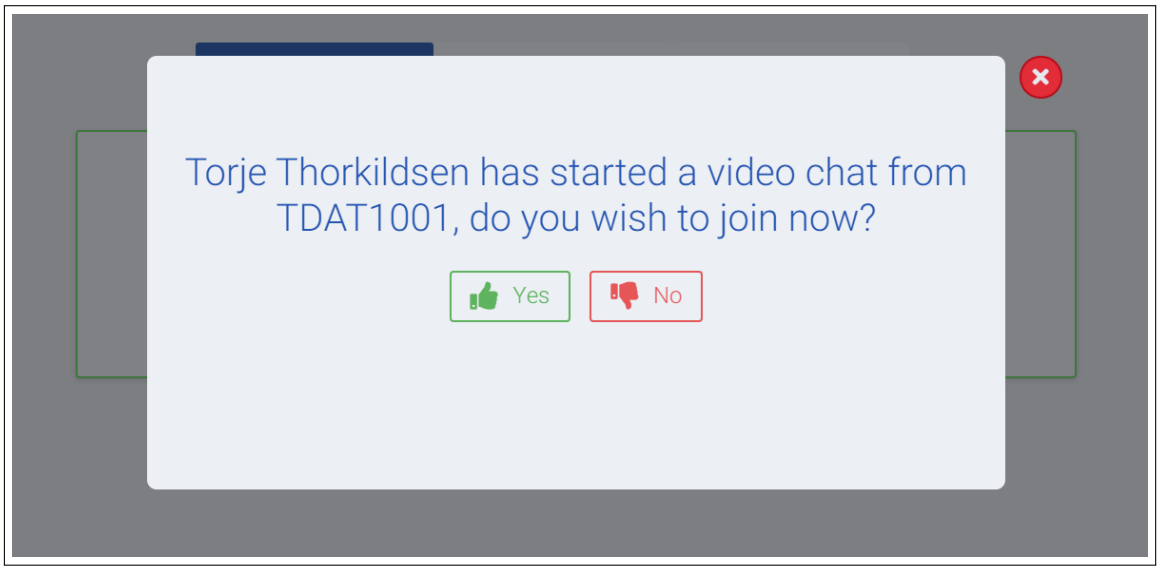


Figure 21: Teachers and assistants can call groups, and they will be notified by a sound and this prompt. Students can also call their groups to start preparing their work before getting help.

4.1.3 User testing

Our product manager gave us the opportunity to test the system during a real class at NTNU. We conducted two production tests. These sessions were digital only, which meant they got to test the online video chat feature. Many users sent valuable suggestions and bug reports through the feedback function built into the website.

4.1.3.1 Production Test 1

Feedback from students:

- Message for the queue does not show.
- Can't figure out how to enter the queue for the respective exercises.
- Queue feels much slower when we can't see the whole queue and how long the student assistant has been helping someone.
- Can we see the whole queue?
- I would like to see the whole queue, not just the position.

Feedback from student assistants:

- Me as a student assistant can delete other student assistants, on purpose?
- I think it be nice if the help tab was more distinct when something happens in it. Perhaps red text or something - so that we don't overlook them, since we want to prioritize them.

Feedback from teachers:

- Approving exercises is bugged on approval list page. Comments are removed after being saved, same goes for date. Date doesn't seem to be set automatically if you click on the exercise.
- Subject page: It may be confusing that some students may render, and that you have responsible further down. Especially on mobile, but may be applicable for PC as well. It may be better if students and responsible could have been collapsed and extended by the user.
- There are some bugs on the subject edit page. I got an error when selecting exercise 1 to be approved. There was no error on exercise 2, but this was forgotten when reloading the page. After this the option to select exercises came 4 times after each other.
- Approval List: The sorting seems to be random. Users should be sorted by surname. The option to download exercises doesn't work.

4.1.3.2 Production Test 2

Feedback from students:

- Chat is annoying
- Chat is scrolling if you get a message when scrolling and reading the backlog
- Chat is opened automatically when messages are received even if it's closed, that's kinda annoying.

- Would be nice to be able to mute the chat without muting the entire tab.
- Bug: Had to refresh the page to get updated position in queue
- There isn't any profanity filter on the chat

Feedback from student assistants:

- Jitsi in qs doesn't work neither in Safari nor Chrome
- Page-title when a student assistant wants to start video chat is not very intuitive
- Chat name for queue_element_chat with subjectcode is not very intuitive. It should be the name of the students.
- Default font size is too big
- No way to go back to the person you are approving, and it's impossible to know who you are approving. There are no buttons to delay an approval.
- Queue page has scroll even without content on the page

4.1.4 Code documentation

Since the front-end is written in TypeScript, JSDoc can be easily generated.

The back-end also has annotations for JavaDoc.

All of the documentation can be found in the system documentation document in the appendix.

4.2 Engineering results

In this chapter we will look into some the goals we set in the vision document. As many of our initial goals were partially developed when we inherited this project, we will discuss them briefly.

4.2.1 Functional properties

All features in the current system are also present on our system. The functional properties mentioned in the vision document have all been fulfilled to a degree in which they are usable and mostly bug free. Present functionality was improved, and several new features were added:

Feature	Status
Virtual Meeting	This has been implemented using JITSI Meet
See the queue as student	This feature has been implemented with limited details
Room image creation within the website	This feature has been implemented with support for both image, and a dedicated editor
Chatting system	This has been implemented with 3 different chat options
Token timeout warning	This has been implemented with warning after 30 minutes
Expiration date for exercises	This feature has been implemented
Option to upload user photos	This feature has been implemented
Notification about being called	This feature has been implemented globally

4.2.2 Non-functional properties

4.2.2.1 Good security

This is a somewhat ambiguous goal, but the security on our system is vastly superior compared to the current system. Access control to the different resources is as strict as possible, and users only have access to the users they have a relation to.

There is still some work to be done to ensure the security. Even though none of the students managed to break the security features of the system during the testing, there should be conducted some more penetration testing to ensure the security.

4.2.2.2 Reliable Website

The website is built using React, which will make it work in all modern browsers. We worked on the website with the goal of it being usable on mobile, as well as tablets.

4.2.2.3 Stability

The user test we conducted had over 80 students. We saw no issues in terms of the system being unstable or crashing. The system itself is also fairly simple to setup, and should not require much resources to maintain.

4.3 Administrative results

4.3.1 Timesheet and activity

At the end of the project we had the following total work hours:

Hans Kristian Granli: 604 hours.

Torje Thorkildsen: 597 hours.

Our main focus varied over the course of this semester. We have visualized our work distribution in the pie charts below.

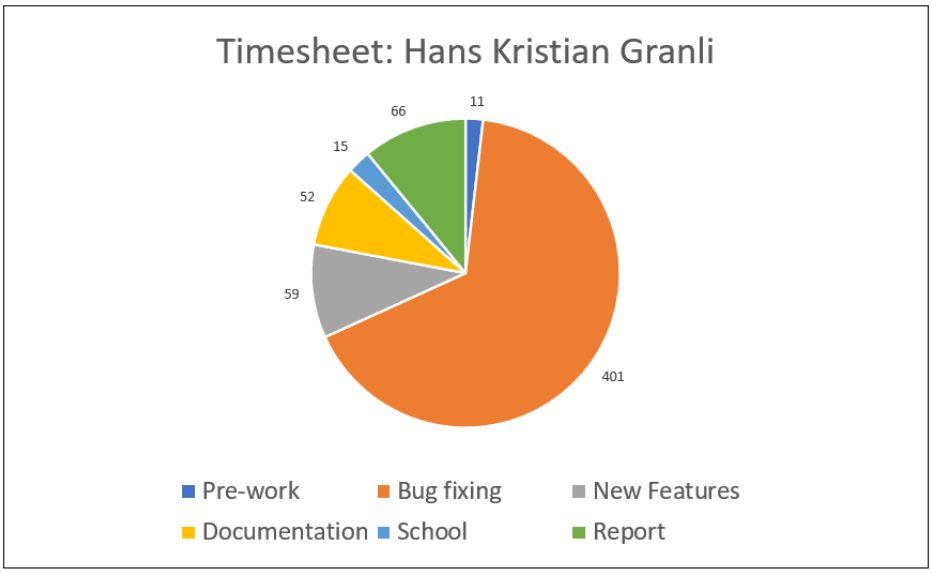


Figure 22: Hans Kristian's distribution of work hours.

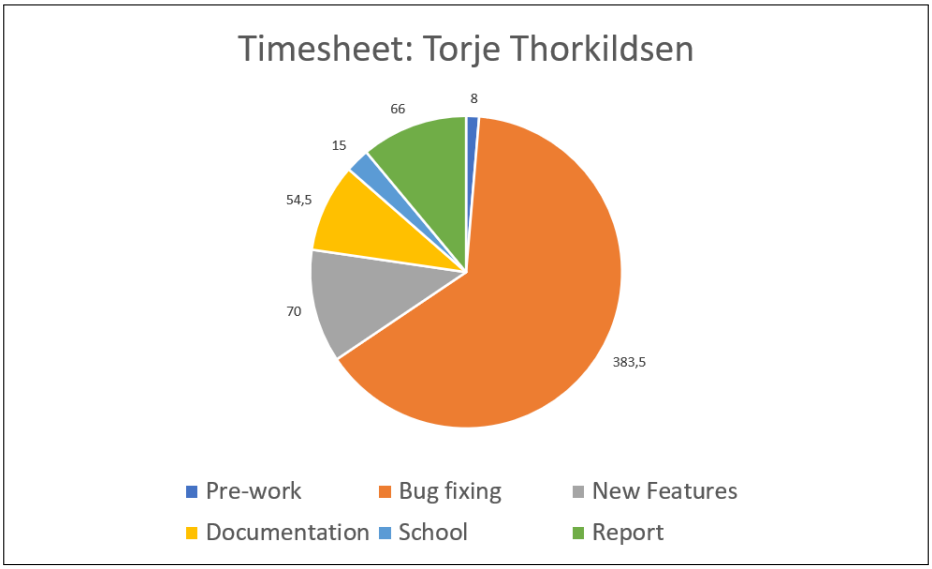


Figure 23: Torje Thorkildsen's distribution of work hours.

4.3.2 Planning and organizing

We implemented a kanban board on GitLab.

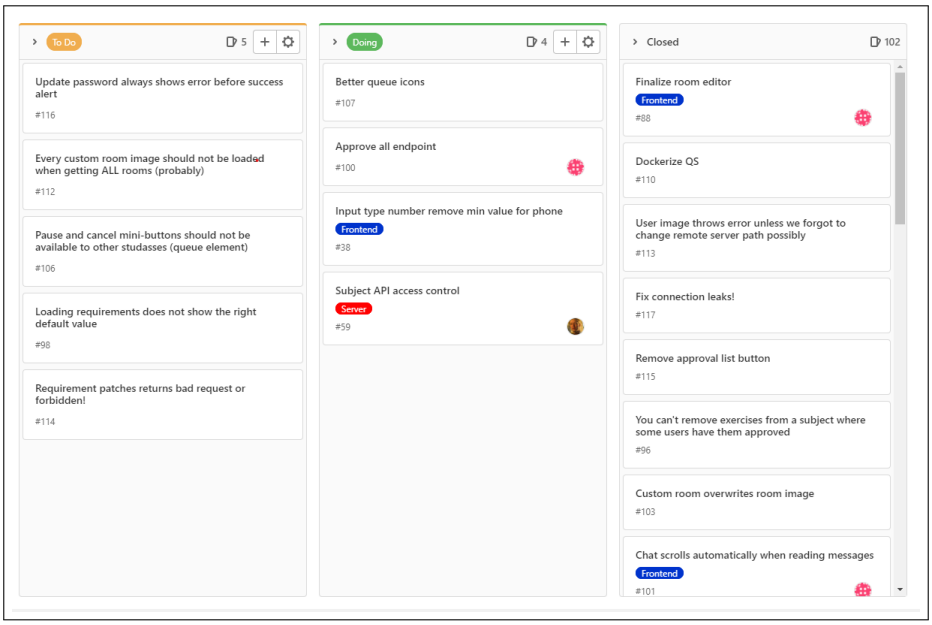


Figure 24: Kanban board on GitLab

The schedule plan was organized into a Gannt diagram, which can be found in the appendix.

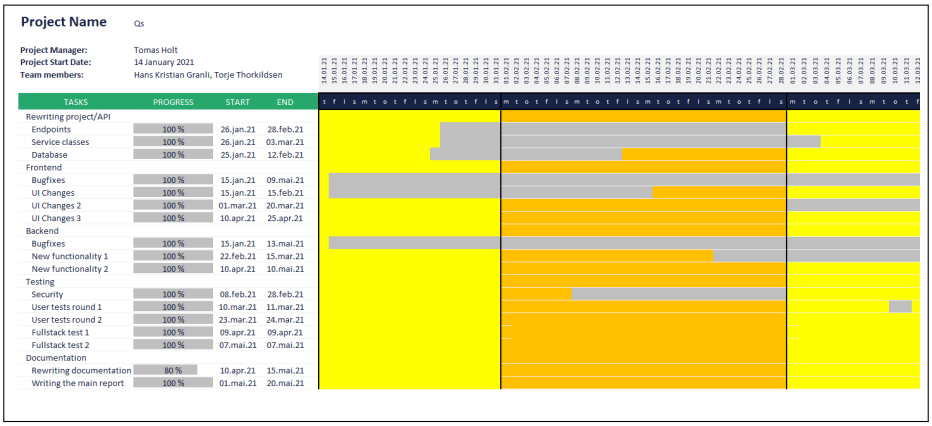


Figure 25: Part of the project's Gantt diagram.

5 Discussion

5.1 Scientific results

5.1.1 Back-end

5.1.1.1 Choice of Connection Pool Framework and Benchmarks

Hikari dwarfs the other Connection Pool frameworks in terms of performance, just like the developers claimed. However the error rate is significant compared to the other frameworks.

Hikari also has some non-performance properties called safe by default that makes it interesting [Bab]:

- Connections are rolled back when returned to the connection pool , so that uncommited changes aren't affected by the next user
- If the connection hasn't been used in the last 500ms, Hikari will ensure it's valid before handing it over
- If an exception occurs in the middle of a statement, Hikari will by default close the statement when the connection is returned to the pool, which eliminates potential connection leaks on error.

These properties are possible to achieve when using other frameworks as well, but frameworks with safe properties out of the box is very much sought after when working in a small team like we did.

5.1.1.2 Performance test

The tests we ran on the server showed extreme differences in the performance of the new system's server and our system's server. Both versions could sustain a heavy load of thousands of requests per second, but with very different response times. While analyzing the dependencies of the new system we quickly realized that its server didn't pool connections at all. It used a class called DriverManagerDataSource. Its documentation page clearly states: "NOTE: This class is not an actual connection pool; it does not actually pool Connections." ([Hoe]) and explains that it creates new connections for every request. It goes on to recommend Hikari as a connection pooler, which we took into consideration when deciding what we would use. The new system might have only used this for testing purposes, but either way it was a necessary change before production testing.

An even more costly mistake in the new system was how they created a new application context for each request the server received. This means that it loaded and created new objects of all the DAO-classes. These classes were meant to be singletons, meaning there only exists one object of each of the classes in the system. Some of the output logged for every request is shown below.

```
DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader
    - Loaded 2 bean definitions from class path resource [Database/Spring-Datasource.xml]
DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader
    - Loaded 1 bean definitions from class path resource [DAOs/Spring-Campus.xml]
.
.
DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
    - Creating shared instance of singleton bean 'org.springframework.context.support.
    PropertySourcesPlaceholderConfigurer#0'
DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
```

```
.
.
DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
    - Creating shared instance of singleton bean 'subjectDAO'
DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
    - Creating shared instance of singleton bean 'queueAdminDAO'
```

This resulted in new instances of all DAO classes which were supposed to be singletons. Redesigning their bean-structure and singleton usage to prevent this made the server much more responsive. From table 1 and table 2 we can see that the new system's beans structure came with a huge performance penalty. The server did not access the database during this the first test, and the bean/application context changes were the biggest changes when it came to request handling between the two versions of the server. This test also shows a small but noticeable difference when using temporary redirects instead of direct access to endpoints. Temporary redirects contribute to a constant delay to the response time since all they do is reroute the request (and respond to the client who sent the request with status 302 - temporary redirect). With this in mind, the new server would have suffered a far greater relative time delay if it still used temporary redirects. The average response time in 1 increased by approximately 8 seconds when using redirects, or a 10.6% increase. If we were to add this 8-second delay to the average response time of 10 seconds of our system's server, the average response time would increase by 80%. In other words, the server's response time would almost be doubled.

5.1.1.3 Database Structure

After analyzing the database of the current and the new system, it was clear to us that we needed to entirely restructure it. The current system's database was not even in the first normal form; it broke the requirement that says that every column in a table need to be atomic, meaning that is does not contain multiple values per cell. In multiple tables in the database they had comma separated values within the same cell stored as text, which would not only use more storage than storing numbers but also forced the system to split and parse the values at some point. The most obvious problem with this approach was how difficult it would be to extend the functionality of the system and join certain tables together.

The database redesign resulted in a much more maintainable and extendable system.

Many of the security issues mentioned in previous reports seemed to stem from poor database design. In our opinion designing a new database altogether would make the system more secure, increase performance and reduce error sources. The old database also made it impossible to cleanly implement certain features such as uploading exercises, setting a due-date to an exercise, and giving an exercise a description.

The new database is normalized and easier to maintain. Among other things we also allowed queue_elements to set their room_id to null/0. This is then interpreted as home approval on the client side, as apposed to the old system where home was set as a regular room.

Requirement logic was also improved. The system we inherited only had support for two types of requirements; "total" and "x of y". We added a third option "specific". The type is still stored in a single table row:

- Type > 0 : This is "total" and does not have any exercises tied to it
- Type $= 0$: This is "specific" and means that all exercises connected to this requirement in the requirement_exercise table must be approved
- Type < 0 :This is "x of y" and means that at least type of the connected exercises in requirement_exercise must be approved

5.1.1.4 API Design

When reading the report from [AW20], and looking at the code it was clear that the old API endpoints were a complete mess. Having 104 endpoints is not necessary. Additionally the API did not adhere to the REST-principles. We found that rewriting the API would make the system more intuitive, better to work with and easier to maintain.

5.1.1.5 WebSocket

The old WebSocket implementation was lackluster. It would listen in for incoming messages and broadcast this message to all connected sessions, even without any form of authentication. Additionally the event-messages were generic and would be sent out to all connected users. This means that if something happened in subject A, then all members in subject A and subject B would get a message saying something had changed and would reload their whole page.

Our implementation requires the user to send their authentication cookie in order to authenticate themselves. This authentication allows for features such as chat, and calling specific users. Additionally when an authenticated user fetches the queue or their subjects they are added to the related subject-map in the server. When something changes in the queue of this subject only the subject's users will receive a websocket message containing information about what has happened in JSON format. These messages contain no identifiable information. Any information that requires you to be a student assistant, or a teacher to access will not be sent, but rather a generic message, which means that the clients that know they can access this information can fetch the info from the HTTP-API, and the rest can ignore it.

This implementation however is not without flaws. The WebSocket service is still only mounted on the generic endpoint "/". Due to time constraints and the large workload we were unable to refactor this part of the service. This leads to extra overhead on the server side, additionally the Java API limits message handlers to one per session, this means that the chat and event messages can block each other and lead to worsened performance.

5.1.1.6 Code documentation

The current system's code documentation was mostly unfinished and did not help further development for the team. The back-end had seemingly auto-generated a lot of it. This, along with very unhelpful variable and method naming made us spend much more time than necessary getting started with our development. One example of important variables with horrible naming and documentation from the new system is shown in figure 26 below:

```
1     /**
2      * The Db.
3      */
4     PersonDAO db = (PersonDAO) context.getBean("personDAO");
5     /**
6      * The Db 2.
7      */
8     LoginDAO db2 = (LoginDAO) context.getBean("loginDAO");
```

Figure 26: The new system's naming and corresponding documentation of a few important variables in the back-end code.

Source: Line 43 in PersonRoute.java of the new system's server code.

This type of naming and documentation made it near impossible to effectively write code in these classes, and forced frequent look-ups to other parts of the code. This type of documentation has no effect on the readability of the code, meaning it could just as well have been left blank. These

are the types of documentation issues we tried to address in the new system. All methods in the API and DAO classes have been documented, making it easy for front-end developers to make use of them.

5.1.2 Front-end

5.1.2.1 Reactive design

One major focus for the website has been creating a smooth and reactive user interface that responds to changes and works on all kinds of devices and screens. Our end result was a website which could respond to any reasonably sized viewport by rendering all functionality in a fitting manner.

In order to make the new system usable we also had to make sure that the new system was perceived either as good or better than the current system by the user. Looking at the user tests conducted by [AW20] and trying the client ourselves we found that the elements in the page took too much space and looked unprofessional compared to the current system.

The biggest perpetrator of this was the help-component which was easily fixed by making the button float, instead of occupying a newline without any content. Another problem was the font

CSS on the old system was very difficult to work with and maintain. The old CSS-code referenced class names incorrectly:

```
/*
 Will apply on the inner element of <div class="btn"><div class ="qs-button"></div></div>
*/
.btn .qs-button{
  display : flex;
}

/*
 Will apply on element <div class="btn qs-button"></div>
*/
.btn.qs-button{
    display:flex;
}
```

This lead to the CSS-keyword !important being used 264 times, additionally the @media queries was written before the ”regular” code, which also forced usage of !important. In our system !important occurs 83 times, with those occurances being mostly fragments from the old system which we didn't seem necessary to change.

Additionally the old CSS exclusively used px as measurement. This is a problem when designing response webpages, since the components won't respond to bigger or smaller viewports natively. This leads to unnecessary repetition of code. We therefore changed many of the units to use the relative units like rem and vh. Additionally most of our CSS is written using grid. CSS grid by design makes it much easier to create responsive webpages.

5.1.2.2 Dark mode (theme option)

Different users have different preferences. The rise in popularity of giving users choices of how the systems are presented made us add an option to enable dark mode for the website. This changes the color theme of the whole website, while keeping the rest of the styling the same.

When revamping the style-sheets, we implemented CSS variables for colors. This made implementing a dark mode incredibly easy:

```
:root {
    --main-theme: #2656b2;
    --font-color : rgb(0,0,0);
}

:root .dark{
    --main-theme : #6d9ff5;
    --font-color : lightgray;
}
```

To activate the dark mode a parent div is set to the class dark. We created a button on the footer to toggle this class.

5.1.3 Code documentation

The new system had no front-end code documentation when we received it. During the process of developing the new system we have renamed many variables and methods, as well as written a lot of code documentation. Our system is first off all focused on using clear and self-explaining names to avoid unnecessary look-ups and code reading/interpreting. We tried to document and rename most of the methods and code we worked on, and made readability changes by splitting bloated code into more digestible smaller methods. While much of the front-end code from the new system still persists without documentation, it should be easier to read. The service classes (interfaces between the front-end and the REST-API on the back-end) now have sensible method naming and documentation, which is important for further work.

5.2 Engineering results

5.2.1 User tests

Due to Covid-19 restrictions testing scientifically was a challenge. We managed to test the system, but not as scientifically as we would have liked, this is something that should be considered for further work.

5.2.1.1 Production Test 1

The first test exposed a lot small unforeseen bugs. However these mostly occurred for the subject-management part of the site, and had little to no effect on the queue part of the system.

The biggest takeaway from the test was that the students really missed a way to watch the whole queue. Even tough this was a conscious design choice made by one of the previous groups, we decided the new system should implement this feature for the students again. We still kept the same security constraints, and made it so that a student in the subject only would get the queue element IDs and the status of the queue elements. The queue is ordered by the queue element IDs, and the status tells if the element is receiving help or not. Another addition was a button from the exercises-view to the queue.

Additionally, the navigation from the subject queue to the subject approval list seemed to be more clunky than necessary. We therefore added a new tab in the queue view which would bring up the approval list for all members of the subject, as well as adding a button to go to the approval list page itself. Another issue was that the student assistants felt like it was unclear whether the queue message had been altered or not. We therefore made it so that the queue message always renders.

Since this is a max 400 character message it won't break the user experience. They also had issues identifying what queue element they were currently serving. Therefore, a small icon was added to replace the queue element position for the queue elements the student assistant was currently assisting. The report of student assistants being able to remove other student assistants from the subject was a client side bug, as they would be allowed to press the button, but the back-end server would deny the operation. These buttons were removed for the student assistants altogether.

We felt like the issue of "stuff taking too much space" ultimately came down to the modal design. This was inherited from the previous group, but the component was both tedious and annoying to work with. Removing the modal design also made it easier to fix components such as the approval list, which was designed without the React design philosophy in mind.

5.2.1.2 Production Test 2

For the second test the system was stable and there were no big issues during the session. The users seemed pleased with the changes that had been made since last time, but we still received feedback on some details to work on. The new chatting system worked well and was used by the students to communicate with the assistants in the subject. A feature that was suggested repeatedly was the ability to mute the chat sounds, which was finished a few days later.

One of the bugs reported by the student assistants was that the video chat didn't work properly in the modal. For this reason we decided to scrap the built in video chat component, and rather open it in a new tab. As the website utilizes Jitsi as its video chatting service this turned out to be a simple fix, since the client would simply open the Jitsi meeting in a new tab. Another bug was that the queue position didn't update, this was due to an oversight where the WebSocket handler only updated the rendered queue and not the card view.

5.2.2 Qs in a system perspective

As many teams have contributed to this project, it has come far as a product and could be considered ready for production. There will always exist possible improvements, but it works well and can run on its own without assistance from developers. The system is capable of replacing the existing one, as it has about all of its core functionality, general improvements and a greater feature set.

The system is perfectly suited for any schools and universities who need a way to approve and help students effectively. It could be used for non-educational purposes as well, and might be sold commercially. Queues exist everywhere, and a simple way to organize them can be of great help to certain industries. Being able to chat with and video call users remotely is especially helpful during a world wide pandemic such as Covid-19. One could for example imagine this system being sold as a customer service organizer, where it would track the users' position in the queue. No users would be able to retrieve any information about other users in the queue, but the employees at the customer service would have everyone waiting for help in a well-organized list. The customer service could be split into different categories, each being its own "subject". Some changes would have to be made for the system, such as disabling exercises, but this could be done in a relatively short amount of time.

Our hope is that 3D Motion Technologies feels confident enough in the system to replace the current running version. It is also in a state where it could be introduced to other schools and universities with the hope of selling the product and generating revenue.

5.2.3 Professional ethics

Qs is a system meant to improve an otherwise tedious process of managing queues in a class like environment. We are confident that the system provides a better experience than the current running version. The new system has inherently greater respect for privacy, as the system follows

principle of least privilege, and a user cannot read anything from a user he or she doesn't share a subject with. It is also impossible for a user to read queue messages meant for student assistants and teachers. Students do not get full information about the queue in a subject, and cannot deduct what exercises another student is in queue for, or which exercises another student has approved.

In terms of privacy there are still some points of improvement. The chat functionality has no encryption neither when sending nor storing messages, so a database dump will show all messages in clear-text. Therefore, sending very private content in chat messages should be avoided. A database dump will not reveal information about the users who sent them however, as there is no information saved about anonymous users.

5.3 Administrative results

Both team members have been passionate about this project and dedicated much of our spare time trying to make the system production ready by the deadline. We overshot the expected workload of 500 hours each by around 100 hours each.

As the charts in figure 22 and and figure 23 show the majority of the work hours were spent fixing bugs and improving existing functionality. The time we spent creating new features was often followed up by even longer lasting bug fixing sessions targeted at those features. That is also a reason why the work hour distributions consist of mostly bug fixes.

The Kanban board in figure 24 was extremely useful during development. It was an easy way to check what the other team member was working on at any time, and what needed to be done. Our work never collided at ant point in the development process. Every time we found bugs or other issues we would immediately note it in the board, even if we did not work on a related issue at the time. This way, no discovered bugs were forgotten.

The Gantt diagram in figure 25 was an important tool for helping us keep a structured schedule. We always knew what issues we should be focusing on according to the plan. This diagram was formed and edited throughout the development process based on our current status. Is helped us plan how we were going to reach the goals of the finished project, and when we were behind schedule.

5.3.1 Group Reflection

We have both put in equal amounts of work, and stayed very consistent throughout the semester. All our conflicts were resolved with little strain. As both of us were passionate about the project, we tried to deliver as good of a product as possible. Our teamwork methodology also worked perfectly for us, as it had little administrative overhead and contributed to a smooth developing experience. All in all we are very satisfied with our team's effort, methods and results for this project.

6 Conclusion and Further Work

The project we are delivering fulfills all the requested core features of the assignment. The current Qs system has glaring design flaws which are either partly or completely removed from our version. We believe that our system, even with its flaws, is ready to replace the current system. Client-wise it might be worth making some more scientifically based studies on the UI after Covid-19, among the lines of what [AW20] performed. However, based on the feedback we received on our moderately large user tests, we feel confident that the new client provides a superior user experience to the system we inherited. With improved documentation, naming, and code it should be easier for a group to take up this project and iterate on it.

Our testing shows that the performance on the new back-end is massively improved compared to the inherited system. As the current system uses a single threaded framework for the server, Java's multithreading capabilities make it a much more scalable system at its core. The new normalized database contributes to a more robust server. However, there is still some work to do when it comes to protecting the server from spam-requests and DOS attacks.

In order to reach 3D Motion Technologies' goal of making Qs commercially viable we believe that the system can benefit from further scrutinize and testing.

Given the reasons stated we believe that the answer to our thesis *Can we complete new Qs so that it will be an improvement compared to the current system?* is yes.

We have outlined these suggestions for **Further work**:

- Refactor WebSocket service
- Scientifically conduct user tests
- As mentioned in [AR20] to port the React client to React Native to create a fully fledged mobile application
- The system lacks functionality to upload PDF files as exercises. For the project we are delivering this should be pretty straight forward to implement.
- Encrypt the chat messages to improve privacy
- Penetration testing
- Extend room editor
- Consider profanity filter on anonymous messages

Bibliography

[AR20] Vegard Andersson and Erling Roll. 'Front-end study and application of modern web-app technologies with the aim of improving an existing system'. In: (2020).

[AW20] Moe Adolfsen Jonson and Willa. 'A test of the new version of QS'. In: (2020).

[Bab] Nick Babock. URL: https://github.com/nickbabcock/dropwizard-hikaricp-benchmark (visited on 18th May 2021).

[Con] Babel Contributors. *Babel - A JavaScript Compiler*. URL: https://babeljs.io/docs/en (visited on 10th May 2021).

[Doca] MDN Web Docs. *HTTP request methods*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (visited on 11th May 2021).

[Docb] MDN Web Docs. *The WebSocket API (WebSockets)*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (visited on 10th May 2021).

[Fie00] Roy Thomas Fielding. 'Architectural Styles and the Design of Network-based Software Architectures'. In: Doctoral dissertation, University of California (2000). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (visited on 11th May 2021).

[For] Internet Engineering Task Force. *The WebSocket Protocol*. URL: https://datatracker.ietf.org/doc/html/rfc6455 (visited on 11th May 2021).

[Hoe] Juergen Hoeller. *Class DriverManagerDataSource*. URL: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/datasource/DriverManagerDataSource.html (visited on 19th May 2021).

[Mar03] Tom Poppendieck Mary Poppendieck. *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional, 2003. ISBN: 0321150783.

[MM20] Kevin M. Halvarsson Magnus Dahl and Fredrik Monsen. 'Reimplementasjon av QS'. In: (2020).

[Woo] Medina Wooldridge Qian. URL: https://github.com/brettwooldridge/HikariCP-benchmark (visited on 18th May 2021).

Appendix

A Old REST-API endpoints

login

login/checkToken/{token}

login/reset/{email}

login/logout

login/changePassword

buildingRoute

buildingRoute/{id}

buildingRoute/edit/{id}

buildingRoute/delete/{id}

campusRoute

campusRoute/{id}

campusRoute/edit/{id}

campusRoute/delete/{id}

room

room/{id}

room/edit/{id}

room/delete/{id}

room/editRoomImage

room/roomImage/{roomID}

person/getMe

person/editOtherMail/{othermail}

person/subject/{id}

person/exercises/{subid}

person/subjectstudents/{id}

person/subjectStudentsQueue/{id}

person/subjectExerciseGroup/{id}

person/mySubjects

person/subjectFromQueue/{id}

person/queue/{id}

person/element/{queueElid}

person/studentsElement/{id}

person/add/element/{subid}

person/element/edit/{queueElid}

person/element/delete/{subid}/{queueelid}/{queueElementPosition}

person/postpone/{subjectid}/{queueelementid}

person/getOwnerOfElement/{id}

person/persontToElement/{id}

person/regStudent

person/regStudent/{id}

person/regStudent/edit/{id}

person/regStudent/delete/{id}

person/employees

person/employee/{id}

person/employeeperson/employee/edit/{id}

person/employee/delete/{id}

person/roles

person/teacherRoles

person/getQueueElementExercises/{id}/{queueid}

person/getQueueElementID/{subjectID}

person/report

person/updateLanguage

person/search

queueRoute/subjects

queueRoute/{id}

queueRoute/disableQueue/{id}

queueRoute/startQueue/{id}

queueRoute/stopQueue/{subid}

queueRoute/closeQueue/{subid}

queueRoute/queueComment/{id}

queueRoute/updateQueueComment/{subid}

queueRoute/getStudentFromSubjectPersonID/{subPersid}

queueRoute/status/{subID}

queueRoute/statusSpecificList/subjectPerson/{subPersid}

queueRoute/exerciseNumber/{id}

queueRoute/element/{queueElid}

queueRoute/getElementStudents/{queueElID}

queueRoute/approveElement/{subPersid}/{exeNumber}

queueRoute/quickApproveElement/{subPersid}/{exeNumber}

queueRoute/quickApproveDetailed

queueRoute/disableElement/{queueElid} queueRoute/postpone/{queueElid}

queueRoute/{queueElid}/startElement

queueRoute/{queueElid}/stopElement

queueRoute/{queueElid}/pauseElement

queueRoute/approveAllExercises

subject/specific/{id}

subject/mySubjects

subject

subject/group/{id}

subject/group

subject/group/delete/{id}

subject/getPersons/{id}

subject/getTeachers/{id}

subject/getPersonFromEmail/{email}

subject/resetLearningAssistant

subject/addLearningAssistant

subject/addTeacher

subject/disablePerson/{id}

subject/addStudent

subject/addStudents

subject/regExerciseOnStudent/{subPersid}

subject/archive/{id}

subject/dearchive/{id}

subject/addSubject

subject/editSubject

subject/disableSubject/{id}

subject/subjectExercises/{id}

subject/getQueueLength/{subjectID}

Endpoints redirecting to the endpoints above:

loginForm

logout

resetPassword

reset:{token}

updatePassword

res/building

res/specificBuilding

res/addBuilding

res/editBuilding

res/deleteBuilding/{id}

res/campus

res/specificCampus

res/addCampus

res/editCampus

res/deleteCampus/{campusid}

res/changePassword

res/roles

res/teacherRoles

res/employees

res/employee/{personID}

res/addEmployee res/editEmployee

res/deleteEmployee/{id}

res/studentGetStudentId

res/editStudentOtherMail/{newAltMail}

res/subjectAndGroup/{subjectID}

res/studentGetCurrent/{subjectID}

res/getQueueElementExercises/{queueElid}

res/getQueue

res/studentSubjects

res/subject/{subjectID}

res/studentsInSubject/{subjectID}

res/studentsInSubjectFromQueue

res/subjectFromQueue

res/getQueueElement/{queueElementID}

res/addQueueElement

res/updateQueueElement

res/deleteQueueElement/{subjectid}/{queueElid}/{position}

res/studentPostponeQueueElement/{subjectid}/{queueelementid}

res/getQueueElementID/{subjectID}

res/addPersonToQueueElement

res/getStudentsInQueueElement/{queueElid}

res/studentGetWithSubjects

res/specificStudent

res/addStudent res/editStudent

res/deleteStudent/{id}

res/subjects

res/allSubjects

res/getQueueDetailed/{subjectid}

res/startQueue/{id}

res/stopQueue

res/closeQueue/{id}

res/restartQueue

res/getQueueComment/{subjectID}

res/updateQueueComment/{id}

res/getQueueElementTeacher/{queueElementID}

res/getStudentsInQueueElementTeacher/{queueElid}

res/deleteQueueElementSimple

res/approveQueueElement

res/quickApproveQueueElement/{subjectPersonID}/{exerciseNumber}

res/quickApproveQueueElementDetailed

res/postponeQueueElement/{queueElementID}

res/startQueueElement

res/stopQueueElement

res/pauseQueueElement

res/getStudentFromSubjectPersonID/{subjectPersonID}

res/getSubjectStatusList/{subjectID}

res/getSubjectStatusListSpecific/{subjectID}

res/getSubjectStatusListSpecificList/{subjectPersonID}

res/getExerciseNumber/{subjectID}

res/room

res/specificRoom/{roomID}

res/addRoom

res/addRoom res/deleteRoom/{id}

res/editRoomImage

res/roomImage/{roomID}

res/regSubjectGet

res/regSubjectGetMine

res/regSubjectSpecific/{subjectID}

res/regSubjectGetGroup/{subjectID}

res/regSubjectGroupAdd

res/regSubjectGroupDelete/{subjectid}

res/regSubjectGetStudents/{subjectid}

res/regSubjectGetTeachers/{subjectid}

res/regSubjectSearchStudents/{email}

res/resLearningAssistant

res/regLearningAssistant

res/regSubjectTeacherAdd

res/regSubjectTeacherRemove/{id}

res/regSubjectStudentsAdd

res/regSubjectStudentDelete/{id}

res/regSubjectArchiveSubject

res/regSubjectDearchiveSubject

res/regSubjectAdd

res/regSubjectEdit

res/regSubjectDelete/{SubjectID}

res/subjectExercises/{subjectID}

res/approveAllExercises

res/updateLanguage

res/getQueueLength/{subjectID}

res/searchPeople

B New REST-API endpoints

/login

api/logout

api/users/{user_id}/password

api/password-reset/token

api/password-reset/{email}

api/password-reset/token/{token}

api/campus

api/campus/{campus_id}

api/campus/{campus_id}/buildings

api/campus/{campus_id}/buildings/{building_id}

api/campus/{campus_id}/buildings/{building_id}/rooms

api/campus/{campus_id}/buildings/{building_id}/rooms/{room_id}

api/campus/{campus_id}/buildings/{building_id}/rooms/{room_id}/image

api/users

api/users/{user_id}

api/users/{user_id}/subjects

api/users/{user_id}/subjects/roles

api/users/{user_id}/subjects/{subject_id}/exercises

api/users/{user_id}/subjects/{subject_id}/exercises/{exercise_id}

api/users/email/{user_email}

api/users/{user_id}/photo

api/subjects

api/subjects/{subject_id}

api/subjects/{subject_id}/queue-meta

api/subjects/{subject_id}/exercises

api/subjects/{subject_id}/exercises/{exercise_id}

api/subjects/{subject_id}/users

api/subjects/{subject_id}/users/available

api/subjects/{subject_id}/users/responsible

api/subjects/{subject_id}/users/{user_id}

api/subjects/{subject_id}/users/{user_id}/photo

api/subjects/{subject_id}/users/roles

api/subjects/{subject_id}/users/exercises

api/subjects/{subject_id}/requirements

api/subjects/{subject_id}/requirements/{requirement_id}

api/subjects/{subject_id}/queue/{queue_element_id}/call

api/subjects/{subject_id}/chat

api/subjects/{subject_id}/users/{user_id}/chat

api/subjects/{subject_id}/queue/{queue_element_id}/chat

api/subjects/{subject_id}/queue

api/subjects/{subject_id}/queue/approve

api/subjects/{subject_id}/queue/messages

api/subjects/{subject_id}/queue/{queue_element_id}

api/subjects/{subject_id}/queue/{queue_element_id}/message

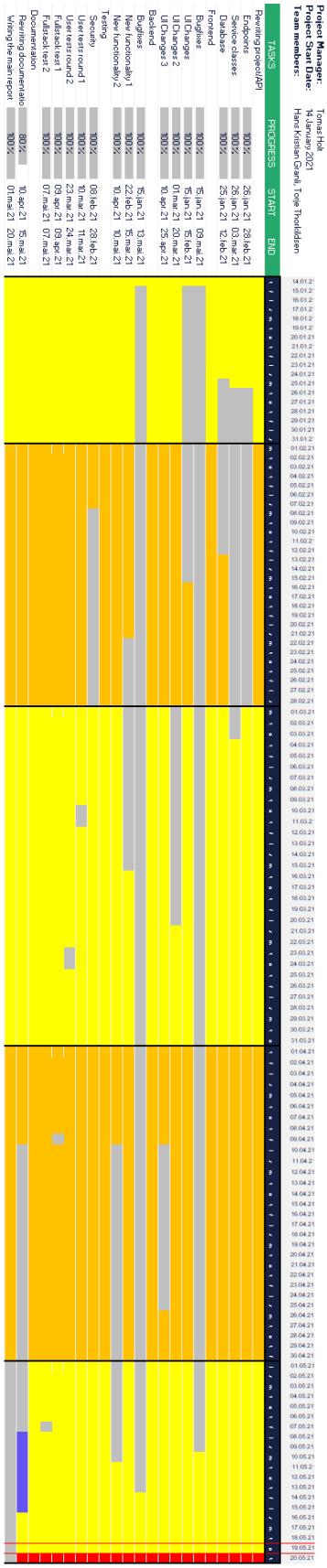
api/subjects/{subject_id}/queue/{queue_element_id}/users

api/subjects/{subject_id}/queue/{queue_element_id}/approve

api/subjects/{subject_id}/queue/{queue_element_id}/exercises

api/subjects/{subject_id}/queue/{queue_element_id}/exercises/{exercise_id}

C Gantt diagram



D Vision Document

Qs
Vision Document

Version <1.0>

Revision history

This is the revision history of the system after we inherited it from the previous group..

Date	Version	Description	Author
25/Jan/21	1.0	Specified all known needs and roles in the system.	Hans Kristian Granli Torje Thorkildsen
9/Apr/21	1.1	Added more user needs as a result of our production testing.	Hans Kristian Granli Torje Thorkildsen
7/May/21	1.2	Added more user needs after our second production testing. Also specified the solutions to our problems in the previous versions of this document.	Hans Kristian Granli Torje Thorkildsen
19/May/21	2.0	Translated the document to English, as we wanted all our documentation in the same language.	Torje Thorkildsen

Table of contents

Revision history 1
Table of contents 2
Introduction 2
Summary issue and product 3
Overall description of stakeholders and users 3
Product overview 6
References 8

Introduction

The purpose of this vision document is to give the reader an understanding of what we want to achieve with our product and any issues related to this. The document will describe what the product will achieve, with given assumptions and dependencies. It will also be mentioned what functional and non-functional requirements are imposed on the product. The product has already been partially developed, and we will finish it over a period of 5 months. Qs is a queuing system that connects students, student assistants and their respective subjects. The product was made in collaboration with 3D Motion Technologies, where it will be used as an assessment in the bachelor's thesis in TDAT3001.

Summary issue and product

- **2.1 Problem summary**

Problems with	The current system provides too much insight into other users' activity, and lacks some functionality that will make the system significantly more flexible and attractive. Among other things, the current system is not suitable for users who are not physically present, which over the past year has proved very important due to the fact that they are not physically present because of COVID-19 (home office and home school are very common).
affects	NTNU, students, course coordinators, student assistants
as a result of this	The biggest security hole in the current system is that a user who is a student assistant in one subject automatically becomes in all other subjects. This allows a former studass to open and close the queue however they want. Creating scripts is also a narrow matter that allows a user to always come first in the queue.
a successful solution will	Have few/no bugs, better user experience and virtual meeting options. A chatting system would also be helpful.

- **2.2 Product Summary**

For	3D Motion Technologies, Tomas Holt
as	New Queue System
with product name	Qs
which	Has few security holes, good user experience and a virtual meeting place
Unlike	Today's system that has major security holes and a lack of functionality
Our product is	Harder to crack, better security and virtual meeting place

Overall description of stakeholders and users

- **3.1 Summary stakeholders**

Name	More detailed description	Role in development
Project team	System Developers	Responsible for the development of the system
Supervisor	Subject teacher from the study	Get your project team on the right track, come up with tips and answer questions
System owner	The person who	Comment on progress, suggest features

	announced the mission	and fixes, and guide the project team in the right direction (Tomas Holt for 3D Motion Technologies).
End user	People from the target group: students, student assistants and lecturers	Provides user tests, input, etc.

- **3.2 Summing up users**

Name	More detailed description	Role in development	Represented by
Student	The student performs exercises and goes in queues for either approval or help	Testing, input	The developers, or a person we find for user tests.
Student-assistant	Student assistants help students who need help and approve exercises. They can also start and end the queue	Testing, input	The developers, or a person we find for user tests. In the production tests these will be real assistants from the subject.
Lecturer/teachers	Is responsible for obtaining student assistants for his/her subject, can in practice perform the same job as a student assistant	Testing, input	The developers, or a person we find for user tests.

- **3.3 User environment**

  The system consists of two parts; a server with all resources that are stored, and a corresponding web page which makes requests to this server. The end user only has to deal with the website that should work on both mobile, tablet and PC in all modern browsers.

- **3.4 Summary of users' needs**

Need	Priority	Affects	Today's solution	Suggested Solution

Secure login	High	All Users	Qs	New login solution for new Qs
Users can change email and password	High	All Users	Qs	New profile page on new Qs
Create a course	High	Lecturers who want to create courses and add exercises, as well as students and student assistants	Qs	Create subject option in the system.
Start Queue	High	Student assistants and lecturers	Qs	Buttons in the queue, only available for teachers and assistants.
Join the queue	High	Students, both for help and approval	Qs	"Go to queue" button for students in each subject.
Virtual video calls	High	Students who want approval/help	Nobody	Meeting place on the meet.idi.ntnu.no
Editing exercises and practice rules	High	Responsible for the subject	Partially supported in Qs	We want to have a system where the subject responsible can set deadlines and description of exercises
Off-the-queue practice approval	High	Student assistants and course coordinator	Qs	Approval list.
Way to communicate with teachers and assistants without needing to go to a video call	Medium	All users of the system.	None, or email.	A chatting system.
Way to create room images	Medium	Teachers in the system	Manually editing images in software like paint to match rooms	A web-based app as part of the website, specialized for drawing rooms.

- **3.5 Alternatives to our product**

The problem is already solved by time-consuming manual work in spreadsheets such as Google Sheets or Excel Online, as well as existing iteration of Qs. We have been in classes who use both, and the experience has often been quite bad. Using those solutions has been especially difficult since the Covid-19 pandemic started.

Product overview

- **4.1 The role of the product in the user environment**

The product's role is to connect students and student assistants through a queuing system that is easy to use, but at the same time robust enough that it cannot be abused. It should be as easy to use on mobile as it is on PC, and take as little time as possible to sign up for a queue, start queuing, or create subjects.

- **4.2 Prerequisites and dependencies**
  - User must have a browser and naturally be connected to the internet
  - The product owner must have a machine that can run the web server when the product is going into production

- **Functional properties of the product**

Functional properties	Description
Create a course	Creation of subjects can be done on one page, it is perhaps desirable that such information as a student list be obtained from the FS.
Add students to a course	These students will have access to which exercises have been approved and which are missing
Add student assistants to a course	Student assistants can be students who have had the subject in the past, they should not do exercises, but approve and help with them
Create practice rules for a course	Various subjects may have various rules met in order to be allowed to take the exam, for example 4 out of 6 exercises, or exercises 2, 3 and 5 approved.
Start Queue	A queue must be started for students to enroll
Add message	Sometimes student assistants may be given general notices regarding the approval process – for example, "Room 404 Approval Only"
Sign up for queues	Students must be able to join the queue, it is also desirable that a student can add fellow students to for a group approval
Virtual meetings	Virtual meeting rooms must be created if the student and assistant want digital approval.
See the queue as a student	To see their space in the queue, the student can either see the entire queue, or only see their space

Remove themselves from the queue	If a student discovers a fault in their assignment, it should be possible to leave the queue - it may not be desirable for each student to be able to pause their own participation in the queue as this can cause everyone to join the queue and immediately pause so that they do not have to wait until they finish, which would have created chaos.
Approve exercise	Once an approval process has been completed, a student assistant, or the lecturer must be able to approve the exercise
Approve exercise without the student being in queue	There may be situations where a student who is not in the queue gets an exercise approved, then it is fine if one gets the opportunity to approve students even if they are not in the queue
Room image creation within the website.	Drawing room setups through applications like Paint takes time and is difficult to organize for uploading. An integrated web-based solution could make this easier.
Chatting system	A way to discuss exercises or communicate with members of groups in the queue would improve the effectiveness and experience of the system.
Token timeout warning	Getting a warning about the user's token expiring so that he can perform an action and refresh it would good for the user experience.
Expiration date for exercises	Would help teachers keep track of when students get their exercises approved, and tell if they got them approved after the due date.
Option to upload user photos	Photos would help assistants to find the students they want to help if they are doing physical approval sessions.
Notification about being called	When an assistant or group member starts a video call, the users in the group should be notified about this.

- **Non-functional properties and other requirements**

demand	description
Good security	This is to safeguard personal data of the users, but also so that no one can sneak in the queue
Reliable website	The website should work independently of the browser and platform, as well as provide a good experience on multiple platforms

Stability	It should not require a lot of resources to operate the system itself after the end of development

References

These references were taken into consideration when writing the vision document, but are not directly correlated.

- Adolfsen et al. (2020). 'A test of the new version of QS'.
- Andersson, Vegard and Erling Roll (2020). 'Front-end study and application of modern web-apptechnologies with the aim of improving an existing system'.
- Magnus Dahl, Kevin M. Halvarsson and Fredrik Monsen (2020). 'Reimplementasjon av QS

E System Documentation

System Documentation - Qs

Hans Kristian Granli
Torje Thorkildsen

May 20, 2021

Contents

1 Revision History **3**

2 Introduction **3**

3 Architecture **3**

4 Project Structure **3**

5 Class Diagram **4**

6 Databasemodel **11**

7 Server Services **12**
7.1 HTTP . . . 12
7.2 WebSocket . . . 12

8 Security **12**
8.1 Cookie . . . 12
8.2 Password . . . 12
8.3 SQL-Injection and X-site scripting . . . 12

9 Installation and execution **12**
9.1 Frontend . . . 13
9.2 Backend . . . 13
9.2.1 SQL . . . 13
9.2.2 JAVA-Server . . . 13
9.2.3 Tomcat Maven . . . 14

10 Source code documentation **15**

11 Continuous integration and testing **15**

1 Revision History

Date	Version	Description	Author
09/03/2021	0.1	Wrote introduction, architecture and project structure	Hans Kristian Granli Torje Thorkildsen
29/04/2021	0.2	Created Database Model and Class Diagram	Torje Thorkildsen
14/5/2021	1.0	Wrote about source code documentation, CI setup, server services and installation and execution	Hans Kristian Granli Torje Thorkildsen

2 Introduction

This document is written in conjunction with a bachelor thesis in the subject TDAT 3001 spring 2021 at Norwegian University of Science and Technology. The task at hand was creating a new version of the internal queue system "Qs" - with the goal of improved security, functionality and performance. The point of this document is explaining the structure of the project, how it communicates and works together.

This iteration of QS undergone development by three other groups. Two previous bachelor theses for front and backend respectively, as well as one group who had the task of sewing it together. We have not altered the structure of the project itself in any major way.

3 Architecture

This project has three major components: a webpage (client), a java server (backend) and an SQL-database.

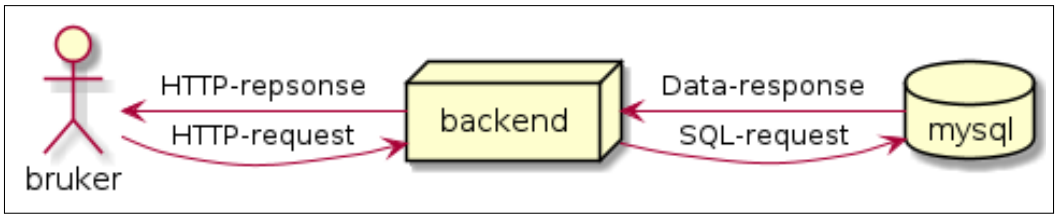


Figure 1: Entity classes

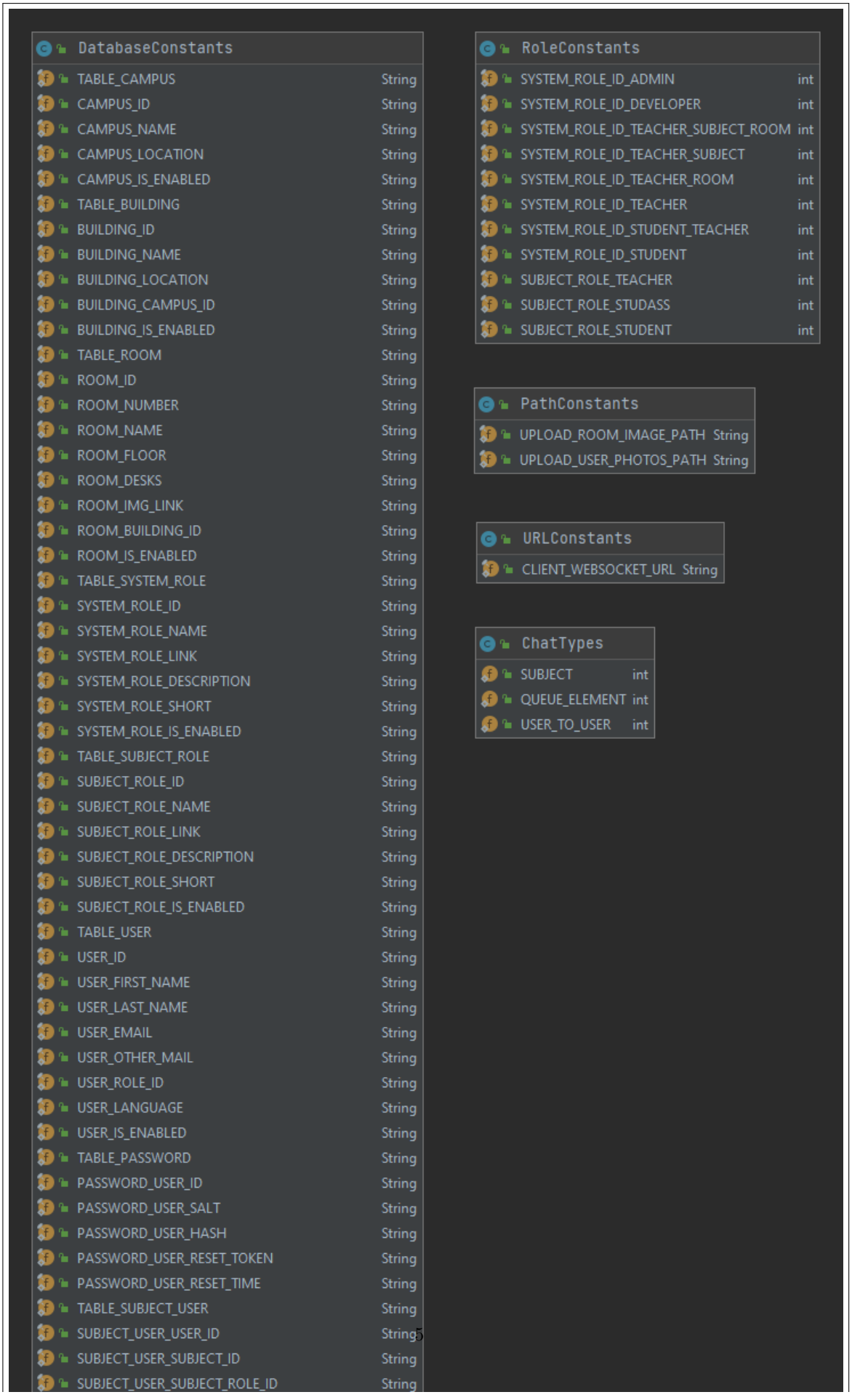
The user interacts with the client, which sends HTTP-requests to the server, which then in turn checks for access control and then queries the database for the relevant information and returns it in an HTTP-response.

4 Project Structure

Project Structure has already been documented by [Ado20]

5 Class Diagram

Figure 2: Entity classes


DatabaseConstants
TABLE_CAMPUS String
CAMPUS_ID String
CAMPUS_NAME String
CAMPUS_LOCATION String
CAMPUS_IS_ENABLED String
TABLE_BUILDING String
BUILDING_ID String
BUILDING_NAME String
BUILDING_LOCATION String
BUILDING_CAMPUS_ID String
BUILDING_IS_ENABLED String
TABLE_ROOM String
ROOM_ID String
ROOM_NUMBER String
ROOM_NAME String
ROOM_FLOOR String
ROOM_DESKS String
ROOM_IMG_LINK String
ROOM_BUILDING_ID String
ROOM_IS_ENABLED String
TABLE_SYSTEM_ROLE String
SYSTEM_ROLE_ID String
SYSTEM_ROLE_NAME String
SYSTEM_ROLE_LINK String
SYSTEM_ROLE_DESCRIPTION String
SYSTEM_ROLE_SHORT String
SYSTEM_ROLE_IS_ENABLED String
TABLE_SUBJECT_ROLE String
SUBJECT_ROLE_ID String
SUBJECT_ROLE_NAME String
SUBJECT_ROLE_LINK String
SUBJECT_ROLE_DESCRIPTION String
SUBJECT_ROLE_SHORT String
SUBJECT_ROLE_IS_ENABLED String
TABLE_USER String
USER_ID String
USER_FIRST_NAME String
USER_LAST_NAME String
USER_EMAIL String
USER_OTHER_MAIL String
USER_ROLE_ID String
USER_LANGUAGE String
USER_IS_ENABLED String
TABLE_PASSWORD String
PASSWORD_USER_ID String
PASSWORD_USER_SALT String
PASSWORD_USER_HASH String
PASSWORD_USER_RESET_TOKEN String
PASSWORD_USER_RESET_TIME String
TABLE_SUBJECT_USER String
SUBJECT_USER_USER_ID String
SUBJECT_USER_SUBJECT_ID String
SUBJECT_USER_SUBJECT_ROLE_ID String
RoleConstants
SYSTEM_ROLE_ID_ADMIN int
SYSTEM_ROLE_ID_DEVELOPER int
SYSTEM_ROLE_ID_TEACHER_SUBJECT_ROOM int
SYSTEM_ROLE_ID_TEACHER_SUBJECT int
SYSTEM_ROLE_ID_TEACHER_ROOM int
SYSTEM_ROLE_ID_TEACHER int
SYSTEM_ROLE_ID_STUDENT_TEACHER int
SYSTEM_ROLE_ID_STUDENT int
SUBJECT_ROLE_TEACHER int
SUBJECT_ROLE_STUDASS int
SUBJECT_ROLE_STUDENT int
PathConstants
UPLOAD_ROOM_IMAGE_PATH String
UPLOAD_USER_PHOTOS_PATH String
URLConstants
CLIENT_WEBSOCKET_URL String
ChatTypes
SUBJECT int
QUEUE_ELEMENT int
USER_TO_USER int

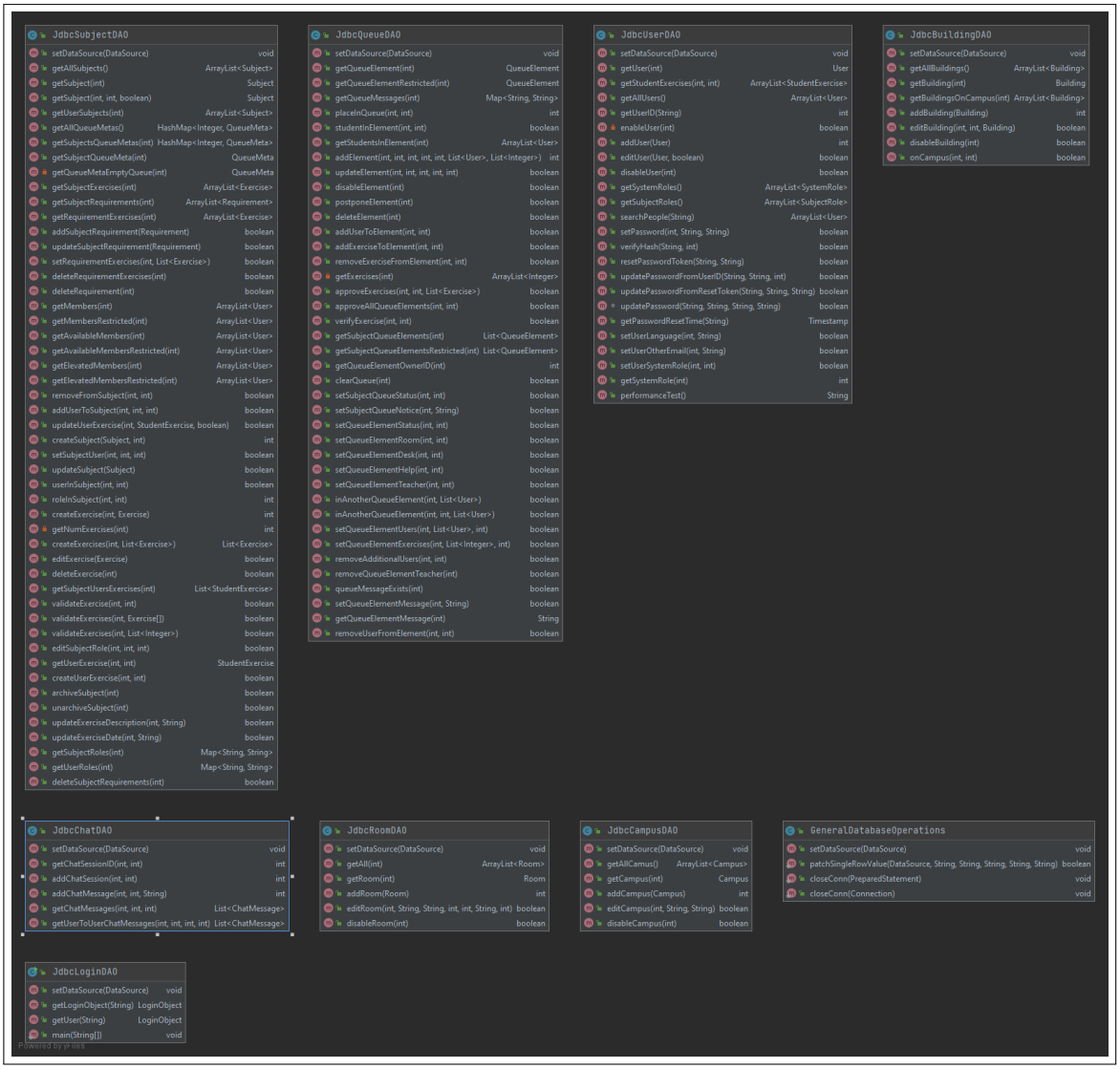


Figure 4: JDBC classes

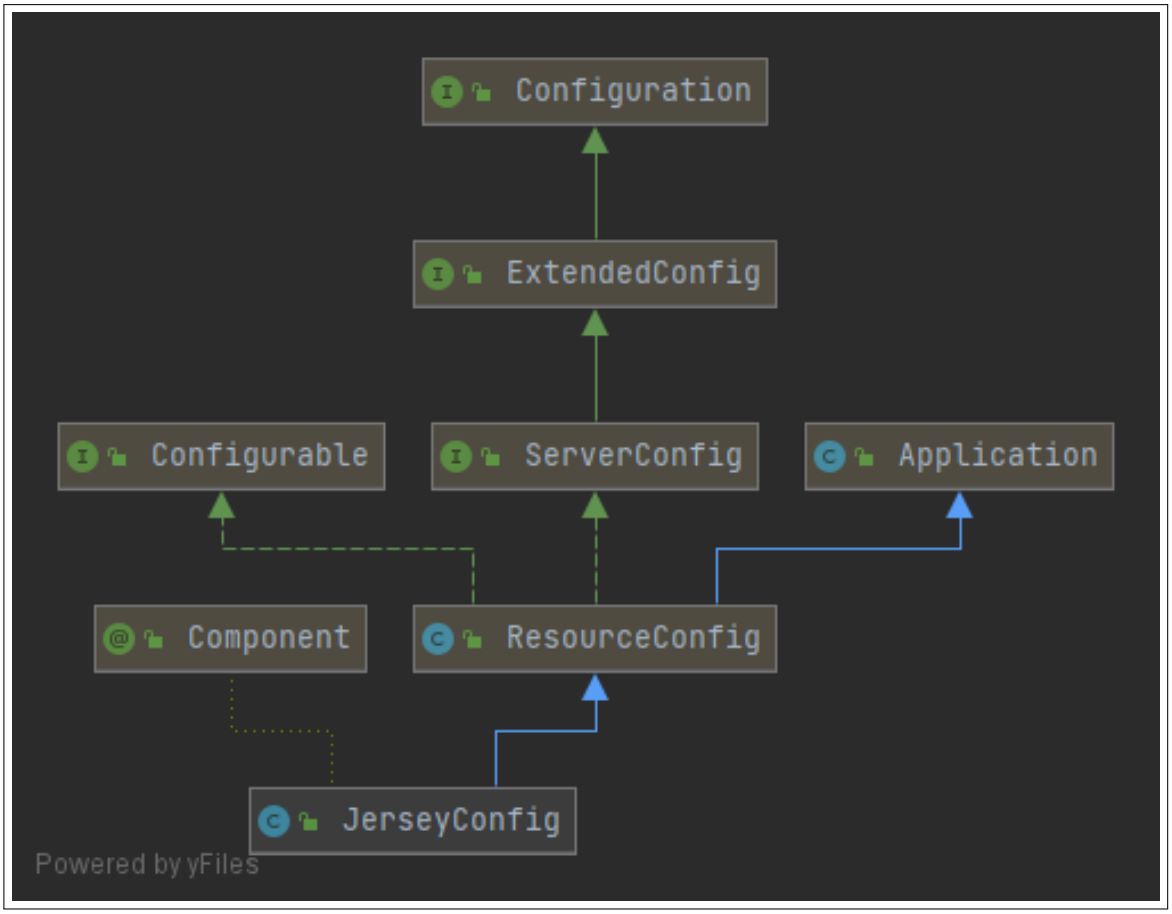


Figure 5: Jersey Config

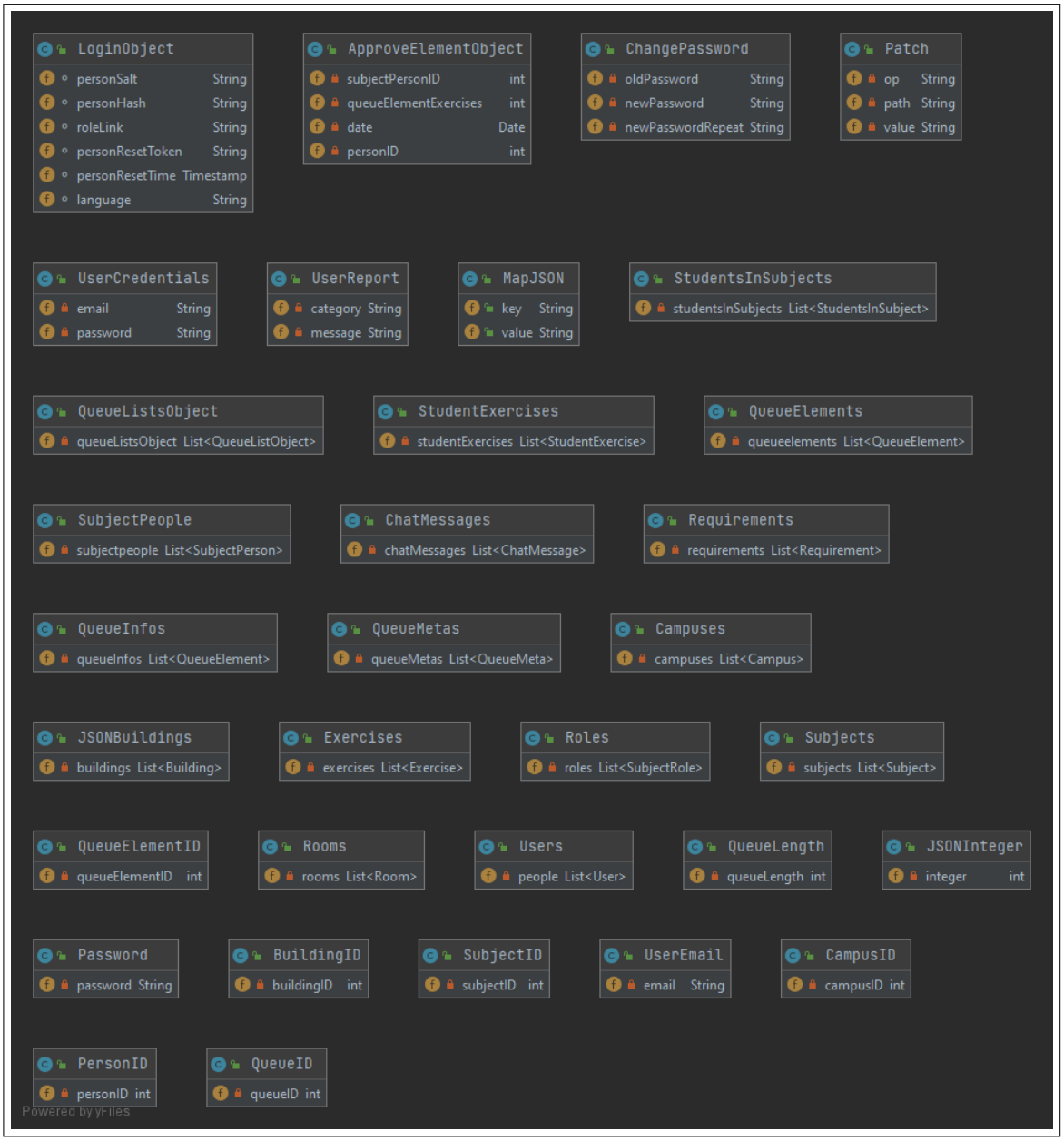


Figure 6: JSON Objects

Figure 7: Marshall Classes

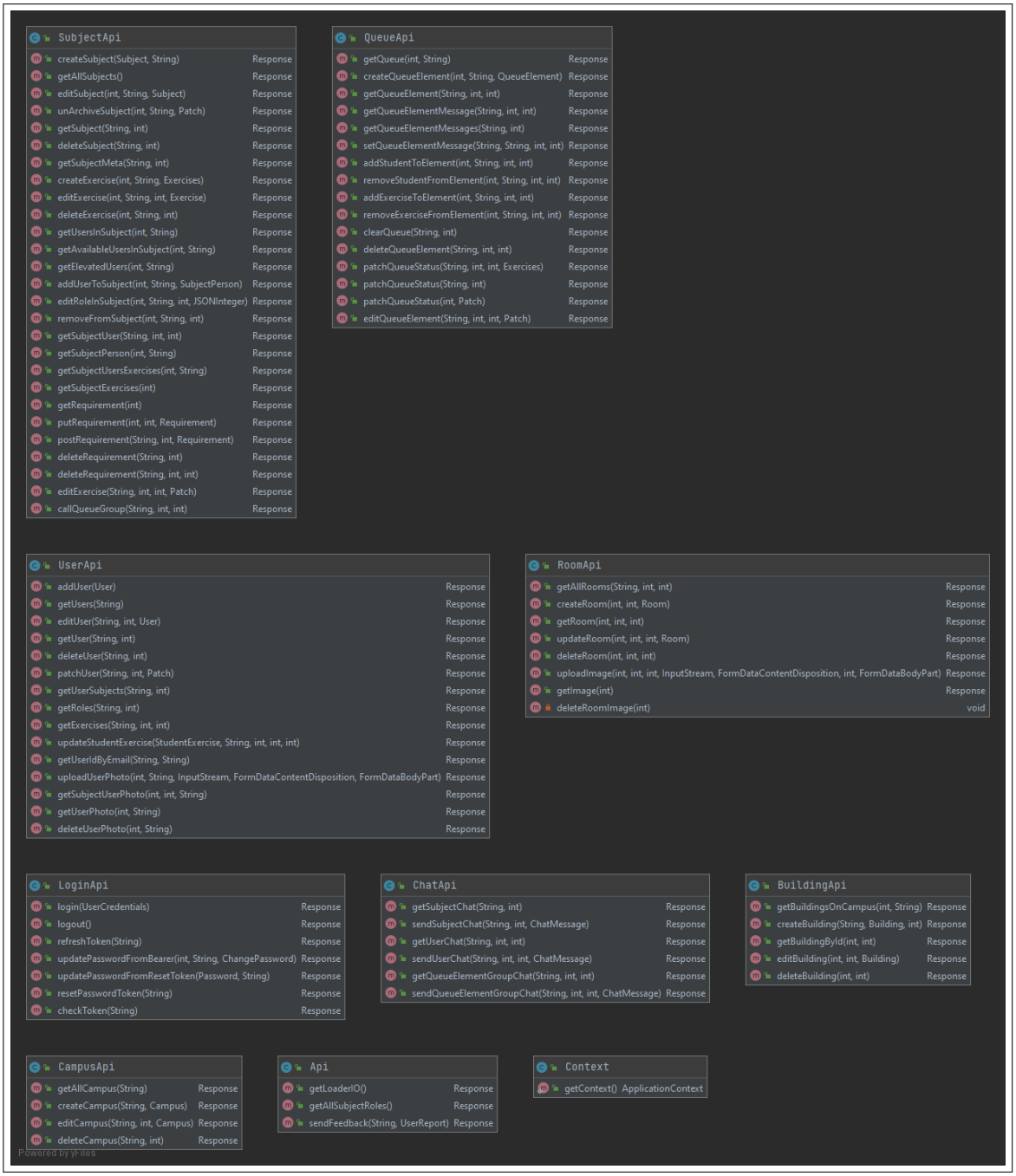


Figure 8: Route/API classes

6 Databasemodel

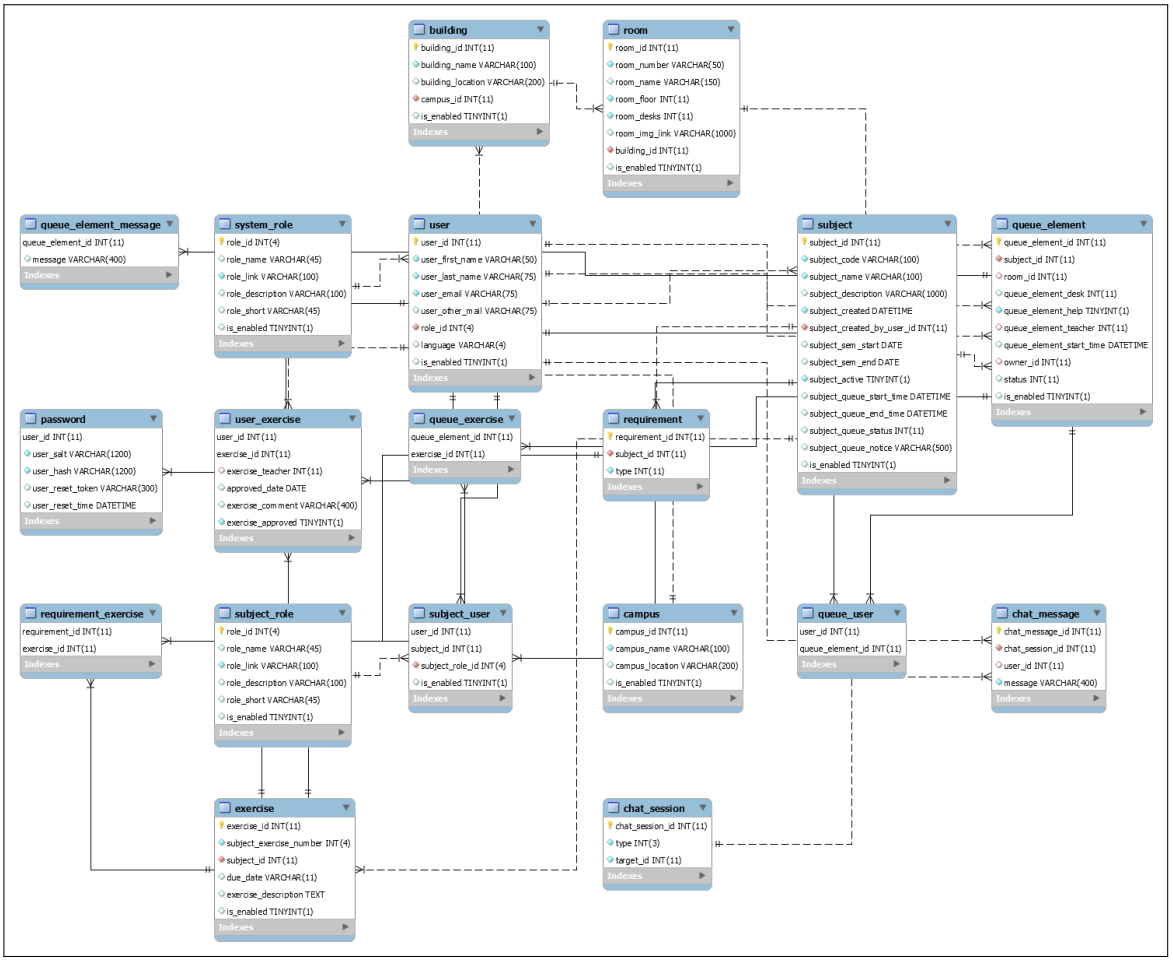


Figure 9: Database Model

The database is as normalized as possible. Most fields should be self-explanatory. For clarification the table "queue_element_message" is an optional message a group can set which only teachers and student-assistants may access.

Requirement has somewhat advanced logic. There are 3 possible types of requirements:

- X of y: which means that a given number of the connected exercises must be approved.Type value is negative.
- Total: which means that a given total of exercises in the subject must be approved – this should not have any exercises connected as its implicit that all exercises in the subject are connected. Type value is positive.
- Specific: which means that all the connected exercises must be approved. Type value is 0.

To support "home approval" the field room-id in the queue-element table may be 0. Normally an id cannot be 0, however allowing this means that we implicitly can interpret room-id 0 as home – without having to get the information from the database.

7 Server Services

7.1 HTTP

The REST Api is mounted on /api and is further divided into the following classes:

- Login-API
- Campus-API - Building-API - Room-API
- User-API
- Subject-API - Queue-API

7.2 WebSocket

The WebSocket-API has a single endpoint. To gain access to this resource, the client has to authenticate itself with the cookie given upon login. When the user is authenticated and either fetches its subjects, or fetches the queue of a subject, the user is added to the internal WebSocket subject-user-map. This is a data structure which connects a WebSocket-session to a user, so that we can broadcast subject-events only to the members of a given subject.

There are multiple different socket-events. When a queue-element is deleted on the backend, every connected and authenticated client gets a JSON-message in the socket connection telling what element has been deleted. This reduces the required HTTP-requests to the server. Not all changes are broadcast to all users, if a protected resource has been altered, a generic message is sent and the clients who know they can access the resource can make an HTTP-request to get the new information.

8 Security

Apache tomcat has built in support for encrypted HTTPS communication, but this is not configured on the project we are delivering. Resource authentication is being done server side. Without a cookie a user can only reach login and reset-password.

8.1 Cookie

The access cookie stores both user information, and a JWT authentication checksum used to verify authenticity. This checksum is generated using a server-side secret. Access cookies are used for authentication after login. This cookie has a timeout of one hour, which the client will automatically refresh if it detects activity.

8.2 Password

The security methods are unchanged from what we received from the previous group. But will be documented here for completions sake. All security methods are extensions of java memory safe methods. The current hash method is PBKDF2WithHmacSHA512 with a 1000 iteration key spec.

Passwords on the hashed and salted with a unique salt for every user. The salt is changed on password change.

8.3 SQL-Injection and X-site scripting

All the JDBC-methods are made using Java prepared statements. React has built in sanitizer against cross-site-scripting.

9 Installation and execution

To run the system, 2 files must be created. **config.js** for the frontend, and **prod.properties** for the backend, additionally **application.properties** must be configured on the backend. Tomcat's **web.xml** must be configured with a Cors filter to use cookies.

9.1 Frontend

The file config.js must be created under /client/config.js and should look like this:

```
export const config = {
    "server_url":"http://192.168.50.28:8080"
};
```

The value of server_url is the url of the backend server. Out of the box the backend will mount on port 8080.

Since the frontend is written in react it may be installed using node package manager (npm), to build this project python2 is also required (python3 will lend syntax error). During the development a new version of node was released. One of the project dependencies – node sass does at the time of writing not support node 16 without workarounds, therefore the stable build for this project requires node 15. There may be some more dependency conflicts in the future. You should however be able to install the project by running "npm install". If you run into mentioned issues then "npm install –f" will force an install. After installing you can start the development version of the client by running "npm start".

The development version however – is not suitable for deployment. In which case you should run "npm run build" which will create a deployable version of the project under /client/dist. From there you can use solutions like npx serve to host the project. In which case a deployment-script may look something like this:

```
#!/bin/bash
cd client
npm install
npm run build
cd dist
npx serve
```

9.2 Backend

9.2.1 SQL

The relevant SQL-scripts can be found under /server/build_test. For production all you really need is DatabaseSetup.sql. Beware that this file uses the qs-database by default. If you for whatever reason want to use a database with a different name than qs-simply change line 2 and set the database name you want.

As far as configuration goes – it is vital that your SQL-server has disabled the clause sql_mode=only_full_group_by. If this is enabled, the queue will only show queue-elements where there are multiple members. To disable this clause from the command line, you need to log into the SQL shell as root and type the following command:

```
SET GLOBAL sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY',''));
```

9.2.2 JAVA-Server

In order to connect the server to our MySql-databse a file by the name prod.properties is required. This configuration file should be stored under /server/src/profiles/prod/prod.properties and looks like this:

```
db.url=jdbc:mysql://*url*/*db-name*
db.username=qsman
db.password=password
```

Db.url references the address of the MySQL-server. This can be either localhost or an external server e.g. mysql-ait.stud.idi.ntnu.no. This URL usually requires a port number which by default is 3306 for MySQL-servers. The value *db-name* must comply with the database you installed the database script on earlier.. By running our SQL-scripts all tables will be created under a database by the name qs, but this can of course be altered.

Db.username and Db.password is the credentials you use when logging into the mysql server.

Application.properties serves mostly the same functionality as prod.properites. Make sure that the fields in prod and application match:

```
db.url=jdbc:mysql://localhost:3306/qs?serverTimezone=Europe/Rome
db.username=qsman
db.password=password
databaseName= qs

dataSourceClassName=com.mysql.cj.jdbc.Driver
hikari.maximumPoolSize=20
serverName=jdbc:mysql://localhost:3306/qs?serverTimezone=Europe/Rome
port=3306

logging.level.com.zaxxer.hikari.HikariConfig=DEBUG
logging.level.com.zaxxer.hikari=TRACE

create-jdbc-connection-pool --ping --restype javax.sql.DataSource --datasourceclassname com.mysql
   --property user=root:password=pass:DatabaseName=taman:ServerName=127.0.0.1:port=3306:useSSL=fa
  useUnicode=true:serverTimezone=UTC:characterEncoding=UTF-8:useInformationSchema=true:nullCatalo
```

9.2.3 Tomcat Maven

To run the backend server, **Tomcat 9** is required. Tomcat version 10 was released during development, but have not tried it, and cannot guarantee compatibility. Hence Tomcat 9 is stable.

We need to configure a Tomcat CORS-filter. Without this our browser won't allow us to communicate with the backend server. The file we need to change is relative to the installation folder of tomcat under /tomcat/conf/web.xml. The cors-filter looks something like this:

```
  <filter>
<filter-name>CorsFilter</filter-name>
<filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
<init-param>
  <param-name>cors.allowed.origins</param-name>
  <param-value>
    http://192.168.50.28:8081
  </param-value>
</init-param>
<init-param>
  <param-name>cors.exposed.headers</param-name>
  <param-value>
    Access-Control-Allow-Origin,
    Access-Control-Allow-Credentials
  </param-value>
</init-param>
<init-param>
  <param-name>cors.allowed.headers</param-name>
  <param-value>
    Content-Type,X-Requested-With,accept,
    Origin,Access-Control-Request-Method,
    Access-Control-Request-Headers,Authorization,
    dsName
  </param-value>
</init-param>
<init-param>
  <param-name>cors.allowed.methods</param-name>
  <param-value>POST,GET,DELETE,PUT,PATCH,HEAD,OPTIONS</param-value>
```

```
    </init-param>
    <init-param>
      <param-name>cors.support.credentials</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>CorsFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

The only value that has to be changed is the "allowed origins" param. This refferences the url of the running frontend. The filter supports an array of addresses, they have to be seperated by commas like this:

```
http://192.168.50.28:8081, http://qssystem.club
```

When all this is setup you can compile the server. Navigate to /server and run

```
mvn clean install -DskipTests=true {Ptest
```

This wil generate a file by the name of QsServer.war under /server/target. You now have to move QsServer.war to /tomcat/webapps. To make sure no other projects are running clear both the ROOT-directory and ROOT.war in /tomcat/webapps. QsServer.war should now be moved to webapps and be renamed to ROOT.war.

```
rm -rf ~/tomcat/webapps/ROOT*
mv ./target/QSServer.war ~/tomcat/webapps/ROOT.war
```

Now you can run the server either with tomcat or catalina:

```
~/tomcat/bin/catalina.sh run
```

```
~/tomcat/bin/start.sh
```

The difference being that start.sh wil start a tomcat service, and catalina will run as a regular application and give the program log in your terminal.

10 Source code documentation

11 Continuous integration and testing

During this project the CI service we have used is GitLab CI hosted by IDI. The Docker image we decided to use is ubuntu. We have not created any new tests, just altered the existing tests from the previous project. The tests cover the Api – there are no direct DAO-tests, however since the Api is directly communication with the DAO – an api test will cover both Api and Dao.

The tests require the project to be running - hence to compile the project one must always skip the tests. Here is the docker-configuration we used:

References

[Ado20] et al. Adolfsen. A test of the new version of qs. 2020.

F Requirements Documentation

Qs
Requirementsdocumentation

Version 1.0

Revision History

Date	Version	Description	Author
5/Feb/21	0.1	Added user stories	Hans Kristian Granli
10/May/21	1.0	Finished the document	Hans Krisitan Granli

Table of contents

Introduction.....3

User Stories.....4

Introduction

This document is written as part of the documentation for the bachelor's thesis (125) in the development of new QS. The purpose is to communicate the different requirements that are set and met by the new system. The document contains user stories for the project.

User Stories

As an administrator
Do I want to create user accounts
So I can give users access to my system
Scenario: user account creation
Scenario: saving user account creation

As an administrator
Do I want to be able to create locations
So users can easily tell where they're sitting when they sign up for the queue
Scenario: location creation
Scenario: Upload room image
Scenario: Create room with built in room editor

As a teacher
Do I want to create a course with or without exercises, as well as students and other course coordinators
So my students can access the system
Scenario: creation of subjects

Given that I'm a teacher in a subject
When I press "New course"
Should I be able to enter the subject name, subject code, start and end date, as well as exercises, exercise-rules, students, student assistants and any other course coordinators.
The list of students should be able to be sent as a csv in the format "Last Name", "First Name", "email"

Scenario: saving new course
Provided that all required information is filled in
When I press "save"
The course will then be created – the students should be added, if the students do not have an account in the system, this must be created
And then I will be sent to a page with an overview of the subject

As a teacher
Do I want to change the subject – add students, change their name, number of exercises and exercise-rules.
So that the students have access to the exercise requirements in the subject.

Scenario: changing subjects
Given that I'm a teacher in a subject
When I press "update subjects"
Should I be able to change all fields that were filled in during the creation of subjects

Scenario: saving changes
Provided that all required information is filled in
When I press "save"
Should the information in the database be updated
And I'm going to be sent back to the page with an overview of the subject.

As a teacher
Do I want to add students afterwards
So that any student assistants, or students who are subsequently enrolled, can also use the system.

Scenario: adding students
Assuming I'm a teacher
When I tap into the course page, I'll be able to add students to my course
In the same format as in creating the subject, I should be able to enter a csv that does the job of creating users

Scenario: Save changes
Given that the information is filled in correctly
Should users who do not have an account in the system be able to create an account, and those who are already part of the system should be added to the subject
And I'm going to get a list of all the students in the course.

As a teacher
I want to be able to add exercises with deadlines and descriptions, as well as rules for which exercises must be approved in a subject.
So that the students know what is expected for each exercise and what exercises need to be done

Scenario: adding an exercise
Assuming I'm a teacher
When I change the subject, I should also be able to change exercises related to the subject
I should be able to set a description of the exercise and any deadline for the rehearsal.

Scenario: save an exercise
Given that the information is filled in correctly
Should the information be stored in the database
And students should have access to up-to-date information

Scenario: adding exercise rule
Assuming I'm a teacher
When I change the subject, I should also be able to change the rules related to exercises and the subject – there should be 3 different types of rules; "x of y", "total" and "specific".
I'll be able to add more "x of y" rules, but only a "total" rule and a "specific" rule.

Scenario: saving rules
Given that the information is filled in correctly
Should the information be stored in the database
And students should have access to up-to-date information

As a student
Do I want an overview of exercises in the subject, exercise rules, and which exercises I have been approved for.
So that I have an overview of how I am doing in the exercises

Scenario: See exercise overview as a student
Given that I am a student in a subject
Should I be able to go into rehearsal overview to be able to see all exercises, as well as rules and my approved exercises

As a student assistant and teacher
Do I want to be able to open and close the queue to a subject, as well as clear the queue - both through approval and deletion of queue items
So students can join the queue

Scenario: opening the queue
Given that I am a student assistant or teacher in the subject
Should I be able to open the queue for students

Scenario: closing the queue
Given that I am a student assistant or teacher in the subject
Should I be able to open the queue for students

Scenario: authenticate the entire queue
Given that I am a student assistant or teacher in the subject
Should I be able to authenticate everyone in the queue at once
And the students in the queue should get information about this

Scenario: clearing the queue
Given that I am a student assistant or teacher in the subject
Should I be able to delete all queue items at once
And the students in the queue should get information about this

As a student
Do I want to be able to join the queue, either for authentication or help and see what place I am in the queue

Scenario: join the queue
Given that the queue is open and I am a student in the subject
Should I be able to enroll in either approval or help, and with associated exercises. I should also be able to add fellow students.

Scenario: waiting in the queue
Given that I'm a student and signed up for the queue
I want to be able to see what place I'm in the line. I should also have the opportunity to see limited view of the entire queue - that is, a queue with no information about the different queue items.

As a teacher or student assistant
Do I want a full overview of the entire queue, and have the option to select a queue item and change the queue message

Scenario: add/change queue message
Given that I am a student assistant or teacher
Should I be able to change queue message
So I can give practical information to students before they sign up for the queue

Scenario: delete queue message
Given that I am a student assistant or teacher and a queue message has been set
Should I be able to delete the message
So there's no message anymore

Scenario: selecting queue item

Given that I'm a student assistant or teacher, and the queue item I select is free
Should I be able to activate the queue item and the students in the queue item should get information that I'm helping them
So I can get in touch with students

Scenario: authenticate queue item
Given that I am a student assistant or teacher and have selected a queue item
Should I be able to approve one or all of the exercises associated with the queue item
So that the students get approved the exercise

Scenario: delete queue item
Given that I am a student assistant or teacher and have selected a queue item
Should I be able to delete the item without approving any of the exercises
So that can students jump out of the queue, without getting approval for the exercises

Scenario: reserve queue item
Given that I am a student assistant or teacher and have selected a queue item
Do I need to be able to reserve a queue item
So I can get back to the group later, without deleting the queue item, while other student assistants don't have to check the item

Scenario: remove reservation
Given that I am a student assistant or teacher and have reserved a queue item
Should I be able to remove the reservation
In order to activate or delete the item

Scenario: calling the group
Given that I am a student assistant or teacher and have selected a queue item
Should I be able to call the group
So I can talk to them through jitsi meet.

As a teacher or student assistant
Do I want a list of all students with associated exercises in the course so that I can easily see if they have received an exercise approved or not, and remove or approve an exercise without the students joining the queue

Scenario: approving exercise
Given that I am a student assistant or teacher
Should I be able to approve a student's rehearsal

Scenario: approving a exercise for all students in a course
Given that I am a student assistant or teacher
Should I be able to approve a rehearsal for all students in a course at once

G Project Handbook

Project Handbook - Qs

Hans Kristian Granli
Torje Thorkildsen

May 20, 2021

Contents

1 Gantt Diagram **2**

2 Meetings **2**
2.1 1. Meeting . . . 2
2.2 2. Meeting . . . 2
2.3 3. Meeting . . . 2

3 Weekly reports **2**
3.1 Week 2 . . . 2
3.2 Week 3 . . . 3
3.3 Week 4 . . . 3
3.4 Week 5 . . . 3
3.5 Week 6 . . . 3
3.6 Week 7 . . . 3
3.7 Week 8 . . . 3
3.8 Week 9 . . . 4
3.9 Week 10 . . . 4
3.10 Week 11 . . . 4
3.11 Week 12 . . . 4
3.12 Week 13 . . . 4
3.13 Week 14 . . . 4
3.14 Week 15 . . . 5
3.15 Week 16 . . . 5
3.16 Week 17 . . . 5
3.17 Week 18 . . . 5
3.18 Week 19 . . . 6
3.19 Week 20 . . . 6

4 Time Sheet **6**

1 Gantt Diagram

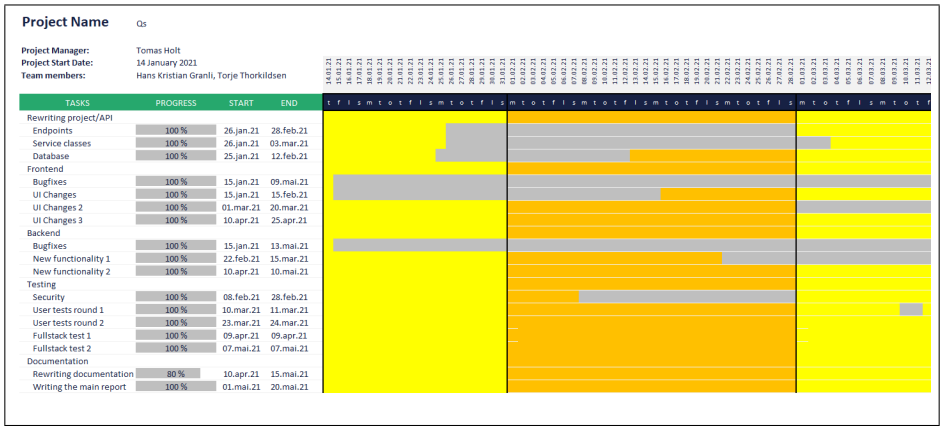


Figure 1: The project's Gantt diagram.

The excel file containing the Gantt diagram may be found in the attachment.

2 Meetings

2.1 1. Meeting

Date: 13.01.2021

- Received the project
- Talked about vision for the project
- Talked about most important issues

2.2 2. Meeting

Date: 29.03.2021

- Got up to speed on the status of the project
- Planned user test with TDAT2105

2.3 3. Meeting

Date: 20.04.2021

- Talked about feedback from the user test
- Talked about future work
- Discussed approval list CSV format

3 Weekly reports

3.1 Week 2

This week we had a meeting with the project leader and contact person Thomas Holt on Thursday. We went through the project's status and what the goals were for the project. We got the project files, which we spent the Friday testing and understanding the code.

3.2 Week 3

In week 3 Torje got back a bacterial infection he had a month earlier, so he was unable to work for most of the first part of the week. Hans Kristian worked on creating a reliable CI test-environment.

After Torje recovered we looked through the user tests from the previous groups who worked on the system and made changes to the user interface based on the testers feedback. The system had some platform specific bugs with the file system which we fixed. The ui improvements were mostly making the elements smaller to reduce scrolling.

3.3 Week 4

This week we started rethinking the API, the database and the server classes. We spent the first few days redesigning endpoints and database tables. The server needs a lot of refactoring in both code and documentation. The code is written in a very unnatural way, with especially senseless use of inheritance. The server has more than 100 endpoints and has hundreds of useless methods, with extremely bad naming (short names which do not describe the object nor its purpose). The documentation almost seems to only be written so the programmers can say "We have documented all our code". One example is shown below:

3.4 Week 5

This week we continued implementing the project's database and REST API. The database was not normalized and had plenty of design flaws that forced the server to implement unnecessary logic. This database design was inherently slow and very prone to bugs, and we are quite honestly in a state of shock. Our redesign addressed these issues but came with the cost of changing a lot of the server code as well.

The old REST API was not at all RESTful, and we had a lot to do since we decided to redesign it from scratch the week before. The server used to have more than 100 endpoints. Our redesign changed this to about 30 for now, is RESTful and very easy to use and extend. The redesign of the database, server and REST API is forcing us to change a lot on both the client and server, but we are confident that this is essential refactoring for the project, and that it will pay for itself when we are finished.

3.5 Week 6

The redesigning process continued this week and is taking longer than expected. The server code proved itself very difficult to understand. The documentation is very poorly written, and most of it seems to be automatically generated by a program. It doesn't tell us any more than the method or variable names. We also haven't used Jersey, Java Beans or automatic JSON marshallers before, which were difficult to get used to.

This week we also finished creating the database-setup scripts and generated test data for the project, and worked on refactoring the server tests.

3.6 Week 7

This week we finished the most important endpoints on the server, so most of the functionality is restored to the webpage. The server is up and running, and most of the requests work as intended. We still lack some functionality, like creating requirements for a subject.

We are finding a lot of bad and bloated code, bad security measures and lots of unnecessary resource-heavy operations. All this needs to be fixed if the project is ever going to be used in a real uncontrolled environment.

3.7 Week 8

We finally finished the refactoring and redesigning process, and most of the project is working as it should. The security of the server is better than ever, and all endpoints are working as expected. We know how everything in the project works and we are finally ready to move on to new functionality and user tests. There is still a lot of bloated/unnecessary code and classes, but more refactoring is not going to be the main focus moving forward. During the redesigning process we also set the project up

for new features we knew we were going to need, like estimated queue time and "do at least 2 out of exercises 5, 6 and 7"-type requirements.

3.8 Week 9

We were satisfied with the backend to the point where we merged it into the master branch. We had no merge conflicts!

Torje fixed the connection pool implementation as well as the bean sharing. It turns out the server isn't supposed to create a new singleton for each incoming request, and rather reuse the objects (what a surprise...).

Backend now uses hikari pooling, which seems to make wonders. We have not yet merged this to the main branch because it needs more testing.

3.9 Week 10

We started implementing the backend endpoints onto the frontend. Turns out the frontend is quite a mess as well. CSS is a complete disaster and will need quite some work. A lot of the React components have bugs that will stop them from rendering at all in certain cases, and some even crash the site.

Hans Kristian created a new subject component which uses less space with less code and looks better.

We tried to perform a couple of informal online user tests with some friends and family to get some input on how the site looks. The feedback was mixed, but mostly positive.

3.10 Week 11

Since we are refactoring the CSS we were able to easily integrate a dark mode which turned out fantastic. We continued progress on the frontend, fixing bugs and improved multiple components.

We decided to change some of the websites basic design. As circles and round edges usually is correlated with creativity and fantasy, we decided to sharpen edges to make the system look more structured and professional.

We also implemented async and promise-based polling on the frontend for a faster website. Very happy with how it turned out.

3.11 Week 12

Started looking at the WebSocket implementation. It broadcasts ANY message send to the websocket-server to ALL connected users without any authentication no matter what action is being performed. We really had to fix this, as this is not scalable at all.

Now you need to authenticate yourself to use the websocket-server, and you cannot just broadcast whatever you want.

As this was the last week before the Easter holiday, we tried to get some input from some friend on the look and feel of the website. These tests were pretty informal but gave us valuable feedback.

We also agreed with Tomas Holt on a date for testing in a real school class after the easter holiday.

3.12 Week 13

Easter holiday.

Did some work on the websocket implementation. Now you can call users!

We also started working on creating a room editor where you can design rooms within the browser.

3.13 Week 14

Got a VM and managed to set it up after a lot of tinkering. It is now possible to access and use the website as long as you are connected to an NTNU-network or to the NTNU-vpn.

The room editor is now at the point where it is an MVP (minimal viable product). It can serialize and deserialize rooms so that you can save rooms and load them later. There is still missing functionality, but it works well enough for now.

We got the mailer working as well now. This can be used in conjunction with the CSV functionality to add users to a subject.

Since we wanted backward compatibility with the current Qs-system we decided to create a database-conversion script which converts the data in the current systems' format to work in our database. This can either create INSERT-sentences or insert the data directly for you.

We had the first real user test with a school class this week. We had to make a crucial hotfix during the test regarding web socket – and the backend didn't crash. Also, we don't believe anyone managed to get access to – or perform an action they weren't supposed to. The users sent some fantastic feedback, and we had a long ongoing conversation with our product manager via email about bugs and suggestions. We noted all of this and added most of their input as issues in our Kanban board on Gitlab.

3.14 Week 15

At the start of this week our focus was mainly on fixing the bugs that appeared during the testing we did with the class the last Friday. Most of these were easy fixes and we could then start working on the suggestions we got during the testing from the class and our product manager. Among these were a chatting system, which we got working by the end of the week. This still needs some testing and adjustments, and we are not sure yet who you should be able to chat with. We also spent some time discussing the security around this.

Also, we did more optimization work on the WS-server. Now we have a more event-based setup which MASSIVELY reduces the http strain.

The other Qs-group asked us for our backend. We sent it to them so they could use it for their frontend project. They are reporting a few bugs which we have not considered.

3.15 Week 16

This week we finalized the student queue view and stopped using modal in queue view. This enables us to utilize more of the screen space, and it a game changer for mobile devices.

The chat system is still a work in progress both on front and backend, but it is working quite well. The system is starting to come together.

3.16 Week 17

We finalized the chat system, which is stable and fun to use. It can save and load recently active chats from local storage. Tomas told us we could have another chance at using the system for approval in a school class next week, and we are excited to see how this functionality will be used.

The approval list is now its own proper component. Now we could enable it as its own tab in the queue for studas and teachers, so they don't have to navigate far from the queue to find it.

Added another endpoint to approve the whole queue – to improve performance.

We also added a system to automatically refresh access token – this will only trigger if the user is active – if they are inactive for too long, they will get a prompt asking if they are still there. If they don't respond within a certain amount of time, they will be automatically logged out.

3.17 Week 18

Even though we thought we were finished with the chatting system for now, we decided to reimplement the backend-side of it. The chat messages are now sent through HTTP-endpoints instead of through the websocket-server as JSON-objects. We did this because the HTTP-server is extremely quick and multithreaded, as well as that the JSON-parsing could cause trouble with certain characters. This change was also made for the calling system, where users now will be notified about teachers and group members calling.

The students in group chats will also get a warning about teachers and assistants being able to read what they say, which was an important ethics-related function.

Friday this week we had the second real class user test. Users seemed to love the new chatting functionality, but it was also where most of the feedback was targeted. They especially wanted a way to mute the notification sound, something which was already in progress but not finished. We also

had a couple of instances of explicit/offensive language in the subject chats, sent with the anonymous functionality. At least we can say they trusted our implementation of anonymous messaging, and we do not have any way of tracking those users.

3.18 Week 19

Although we had started working on a few parts of the report over the semester, this is where we really got into it. Most of our time was spent writing. However, we did manage to fix a severe chatting bug which allowed some users to read other chats' messages. The chat also has a mute option now.

We created and ran some stress tests for the server to see how it compares to the server we inherited at the start of the semester. Knowing we had made big improvements, the results still shocked us. It has an average response time of about 10

In the weekend we finally finished the room editor. It can definitely be extended and improved, but it works very well.

3.19 Week 20

This week we finished the report and documentation. The system is still lacking some code documentation, but we do not have time to write documentation for all the code we have not touched since we inherited the system (especially on the frontend). We fixed a couple of bugs as well.

4 Time Sheet

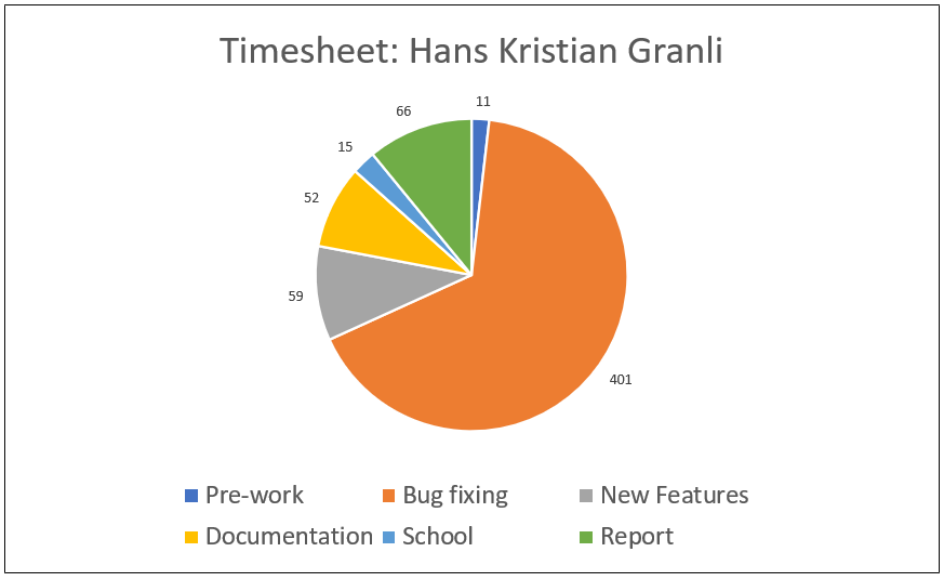


Figure 2: Hans Kristian's distribution of work hours.

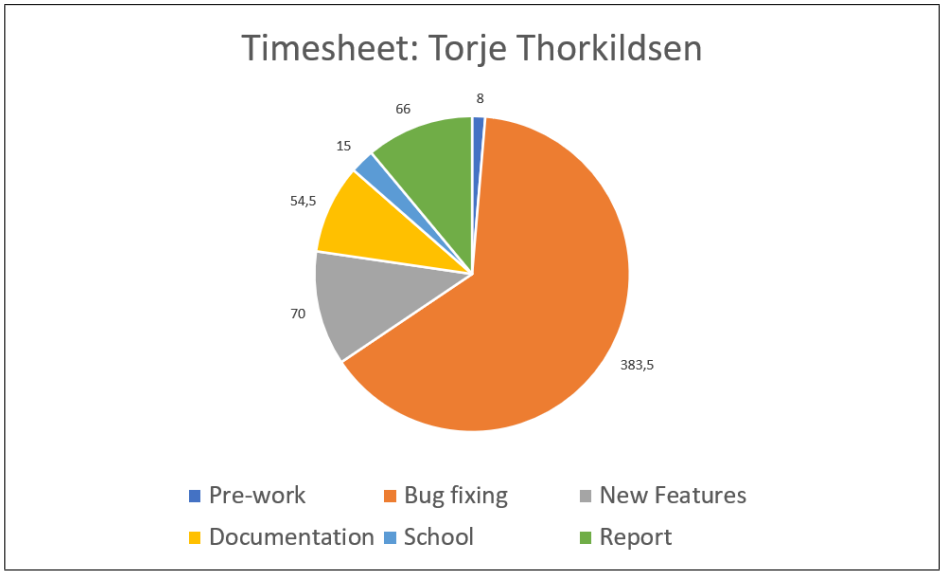
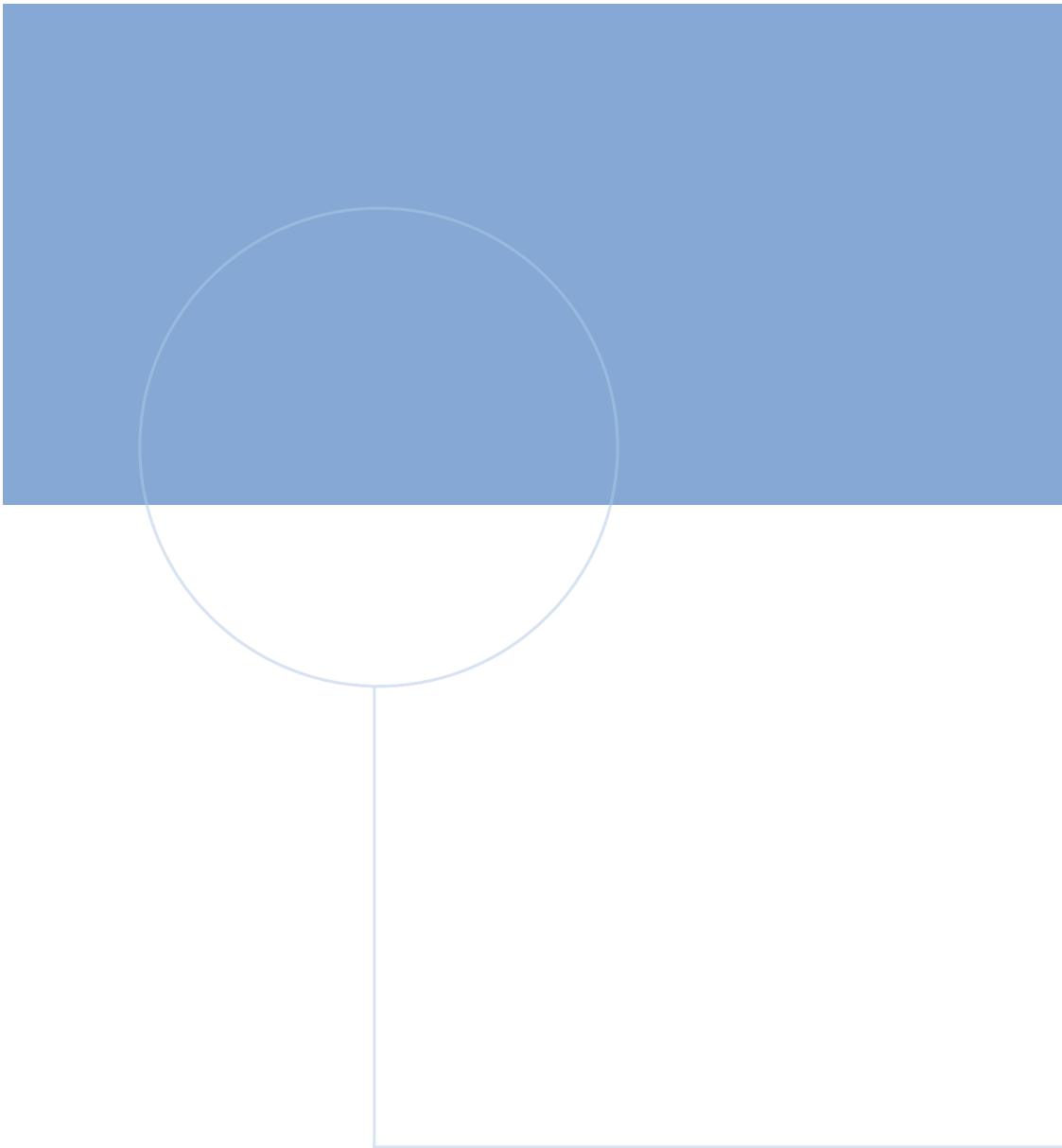


Figure 3: Torje Thorkildsen's distribution of work hours.

The full spreadsheet is a part of the attachment.

References



NTNU

Norwegian University of
Science and Technology