Bachelor's project

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

Vegard Andersson
Erling Roll

Front-end study and application of modern web-app technologies with the aim of improving an existing system

Bachelor's project in DATAING
Supervisor: Tomas Holt

May 2020

NTNU
Kunnskap for en bedre verden

Vegard Andersson
Erling Roll

Front-end study and application of modern web-app technologies with the aim of improving an existing system

Bachelor’s project in DATAING
Supervisor: Tomas Holt
May 2020

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

NTNU
Norwegian University of
Science and Technology

Front-end study and application of modern web-app technologies with the aim of improving an existing system

Vegard Andersson
Erling Roll

May 2020

Preface

We chose this assignment because it was a system we have a lot experience using and something we truly believed could be improved in several ways even before we knew the assignment was available. This assignment also required exactly the knowledge, technical skills and methods that our education has given us. It has therefore been a more than adequate test to assess and develop ourselves as computer engineers

In our case the assignment ended up being slightly different than what first described. Due to the circumstances, we were therefore also an active party in the discussion of what the assignment should be and how high we could set our expectations. The core of the assignment was still to evaluate the previous version of a product which is the basis of our process and foundation for our results.

We would like to thank our supervisor and the other team who did the counterpart to our assignment for giving us valuable advice about and insight into our work. We'd like to acknowledge our teachers and peers in supporting us through everything that has led to this work.

Problem description

The original assignment was to create a secure and well working version of Qs with the hopes that the solution would be adequate for commercial usage. The task was however split between two groups where our group was given the assignment of evaluating and reimplementing the system frontend in a new framework.

Summary

The first part of this assignment was to get familiar with the previous version of the Qs system. The two main goals of this task was to, first of all, figure out what kind of improvements and changes needed to be made. The second was to analyze the time it would take to implement the changes and set the appropriate requirements and goals for the whole assignment. This process was done with the help and approval of the supervisor.

The agreed upon assignment was primarily to reimplement the frontend of Qs. This led to research about frontend frameworks, how they worked, what support they had and if they were suitable for the system. The purpose of the research became to note down the most important features to look for when evaluating each framework. The most important qualities being performance, functionality and sustainability.

Based on these guidelines, the framework that fitted the most of the system requirements was React. All of the functionality was reimplemented and added to in the React framework. While nearing the end of our development both the frontend and backend team had the choice to collaborate to merge their work. The result ended up being a one directional merge where the backend team received the new frontend but did not push their work to the other group. There now exists two versions of the new system. One with the old Express backend and a new one written in Java. The express version is more tested but any future work will be done to the Java version.

Contents

1 Introduction and relevance

The client who put forth the assignment is 3D Motion Technologies. The Queues (Qs) service has been in use since its developments started in 2016. Qs is based on an even older version of the same purpose system by the name of SmartKøSystem (SKS). The reason for the existence of the system and its service is to provide a way to approve student exercise on site by letting teachers and their assistants, which will henceforth be referred to as student assistants, check and approve the work of a singular or group of students in a sequential manner. This is done by having the students who are ready to have their work approved place themselves in a virtual queue on a web based platform for a chosen subject with specified information such as which assignment they want to have checked, what group members they have and at what location they are currently at.

In the assignment description there are two main reasons why the system needs to be reviewed and remastered. First of all, it is based on the framework AngularJS. This is problematic because the performance of AngularJS is not satisfactory when it comes to the use case of the application. There are quite a few pages in the current application that are slow due to how AngularJS handles rendering. There is also proof that the system contains bugs and security holes. The result goal will be to create a secure system that can easily handle a migration of users from the old system and has improvements when it comes to user experience, functionality and stable versions for future work.

A secondary result goal of the project was to create a product that could be commercialized. There are multiple requirements that needed approval for this to happen. The system needs to be secure, able to handle a large user base, easy to maintain and work in a desired environment.

This report and its assignment is based around the reimplementation of the client side user interface and experience. The point of this being to maintain the core functionality of Qs while also fixing and adding features desired by the current users. As previously mentioned the use of the framework AngularJS is not deemed sufficient enough. This leads to the thesis: What framework, library or combination would best suit the Qs service considering performance, functionality, usability and sustainability?

There are two parts to this thesis. On one end there is analysis of requirements that make up the core functionality of Qs. The other end being research into what capabilities and qualities define the use of frameworks, libraries and platforms. A solution can contain a combination of multiple components made up of different technologies and methods. Therefore the assessment of each technology and method both individually and paired together becomes the foundation that the results, discussion and conclusion is based upon.

The performance of a system is the most concrete and measurable aspect. The system is a web based solution and as such there are several problems it should handle. These issues include handling a user base of at least a few hundred to a few thousand and providing good accessibility. One of the problems with AngularJS is the outdated rendering engine. For example, if there are too many listeners on a page it will become slow even on good hardware. A major part of the user environment is mobile usage. This means that the application should run well on smartphones and tables which tend to have worse hardware than laptops and desktop computers. The load times of the pages should also be adequate and the application should not use excessive memory or computing power.

With phones and tables comes another challenge. Another part of accessibility is responsive design. This means that the application should be equally usable on all mediums. For example, adapting for small screen sizes is essential when it comes to responsive designs. Other functionalities that are important when comparing system components are security and use case coverage. It is not only up to the developers to implement the system in the correct manner but also what each component has to offer, how compatible they are with other components and their versatility, resilience and interchangeability. These are some qualities that the current system is missing and are one of the main subjects of further discussion.

The last aspect of the system, more specifically related to the codebase, is sustainability. This is not the first time the system has been rewritten and might not be the last. This also means that adding new features and doing maintenance are important to include when choosing things like programming language, frameworks, libraries, documentation, file structure and other implementations. There are many challenges that come with continuing previous work. This report will therefore describe what these challenges are, how to deal with them and how to reduce the amount of unnecessary friction between project handovers.

There are more than a few technical aspects to the discussion section of this report. All the technical terms and methods will be explained to ensure the full comprehension of the subject. The explanations will be thorough and detailed enough to complement the justifications of our decisions. Further subject matter lies outside the domain of this report and will not be included but is recommended for those who wish a deeper understanding. The theory chapter will include important concepts such as the difference between front-end and back-end when it comes to an application.

The report structure is as follows, in order:

- **Theory**, this chapter contains concepts and technologies that require more detailed explanations in order to be of any value when discussed and analysed, and the chapter aims to give the reader the necessary groundwork to be able to understand the arguments posed in the report.
- **Choice of method and technology**, this chapter includes the choice of technology which introduce and briefly describe the most important technologies that contributed to the creation of the end-result, and choice of methods which briefly describes the different methodologies used in developing the product.
- **Results**, this chapter contains all the data, knowledge and tangible results that we were either able to gather or create. These results are the basis for the discussion in taking place in chapter 5.
- **Discussion**, this chapter contains our thoughts, assumptions and review of the results we were able to procure during the entire process.
- **Conclusions and further work**, this chapter contains our conclusions that we were able to draw from the thoughts we were able to formulate in chapter 5 and the initial conditions for the project that exist in the vision document and in our thesis. It also includes what we consider to be relevant work that can be conducted in the future based on the work we have done during this project.

Our advised strategy for reading the report is as follows: Our thesis mainly made us conduct our project as a two-step process. The first part involves using research, evaluation of the system and argumentation to arrive at a choice of the best suited technology to use for the application, while the second part involves actually developing the product using the chosen technology and approach. For the report to make the most sense, reading it chronologically will make the most sense. We therefore advise reading the chapters 4.1-4.4, the results from our study of the technologies, and 5.1-5.3, our discussion of and final choice of technology, before you read the rest of the report. This way, you will have established and understanding of why we chose what we chose before you read the results of the implementation of that choice. This is only a recommendation, but we at least advise you to be aware of the chronological order of the work we have done when reading the report.

1.1 Abbreviations

The abbreviations used in this report with a short explanation.

- **CSS**: Cascading Style Sheets Used to style and describe layout of HTML.
- **DOM**: Document Object Model World Wide Web Consortium standard for Object trees created from HTML.
- **ES5/ES6**: ECMAScript 5/6 Standardized scripting language specifications.
- **MVC**: Model View Controller An application architecture that splits all contents into three seperate components.
- **HTML**: Hypertext Markup Language Standard markup language for web browser documents.
- **I/O**: Input/Output Information sent between a computer and external sources.
- **NPM**: Node Package Manager Manages packages for Node.js from an online database called the NPM registry.
- **JS**: JavaScript Programming language used mainly in web development.
- **JSON**: JavaScript Object Notation. Open standard file format primarily based in JavaScript.
- **JSX**: JavaScript XML JavaScript that can be injected with inline XML.
- **XML**: Extensible Markup Language Markup language with standardized rules for data structures and document formatting.
- **CSV**: Comma Separated Values. Non standardized text file that uses commas to separate values in a tabular format.
- **RAM**: Random Access Memory Computer memory used to hold information of current processes.
- **SEO**: Search Engine Optimization Visibility of a web page for search engines.
- **(G)UI**: (Graphical) User Interface Allows the user to interact with a system through graphical indicators or textual symbols.
- **CI**: Continuous Integration A practice of running all work through a pipeline before merging work.
- **UX**: User Experience How a user feels about using a service or product.
- **SQL**: Structured Query Language A computer language used to interact with relational database management systems.

2 Theory

2.1 Framework and Library

In almost any instance of developing a system or product through code, there will be the need to resort to already pre-built and pre-written code made by previous developers. The use of already written code might not always be obvious, but when you create a system or product you are most likely going to be using a computer, a compiler or the internet among other things. These pre-built services are a greater part of development than usually visible. Therefore, these things are easily taken for granted, but the borrowing and usage of other people's work becomes much more obvious when you directly have to rely on it in your own systems. A collection of such pre-written code is called a library.

Almost all programming languages and similar environments have the ability to draw strength from libraries to make development of systems easier and more efficient in terms of both time-investment and system-complexity. Libraries usually contain loads of classes, functions and even quite extensive subsystems. Some libraries are small and only contain essentials like sorting-algorithms, connections to I/O-equipment etc. while others are large libraries containing almost everything needed to build a certain kind of system, and in so much detail that you can build an entire system just by piecing together different parts of the library.

Frameworks are similar to libraries and the two terms are often confused when discussing computer science, especially web development as the field is rather flooded with different libraries and frameworks. Still there is a distinct difference between the two and they serve different purposes in the development of your system. While a library, as stated earlier, is just a collection of pre-written code like classes and functions, a framework also contains pre-written code, it has its own library, but a framework is a much stricter and more extensive collection of not just classes and functions to be used, but also flow-controlling code written to utilize the code contained in the library. A framework can be thought of as a skeleton or shell of a system with all the necessary ground-work of an application or system already in place and where connections and calls to library-classes and functions are already a part of it. Your role in the development process when utilizing a framework is simply to build upon the already existing template-application and specialize it to your own desires. A framework usually has specific areas of itself where it can be modified, and others where it does not allow change. This can make for a stricter experience, where the developer has less control over the final product, but in turn it provides a safer environment for development and a lot of the responsibilities of the developer, such as memory-management, responsive web-design, security, code-efficiency are all taken care of by the framework and usually, if it is a good framework, these things are executed better than your average developer could write on his own. (1)

2.2 MVC design-pattern

MVC is an acronym referring to a specific architecture used when developing applications for both the web, as well as other devices and environments. It is a model describing the different parts that make up such an application and in what manner the parts are to be connected to each other. The main philosophy behind this architecture is to achieve a modular design in which either part of the system can be replaced without heavily affecting the other parts. It contains the following parts:

- Model (M) The model is the part of the system which holds the data. Almost any kind of application is in need of data which it either displays to the user of the application or utilizes in the background to achieve different types of tasks and operations. When designing an application with the MVC-structure in mind, the model is the part which actually holds the data, in most cases a collection of references to data stored in the memory.

- View (V) An application most often needs a way of communicating with the user. This is primarily done using a graphical user interface (GUI). The View, when speaking of the MVC-architecture, is the part of the system that displays the application's data to the user through such an interface. The view does not need to care about what happens to the data before or after it has been displayed, only of how the data is to be displayed. There exist a number of different types of user-interfaces, not just graphical, but in web-development, it is the most common way of thinking about the View.

- Controller (C) If the View only displays data, and the Model only holds data, there needs to be a part of the system that actually utilizes the data and has the ability to mutate the data. This role is filled by the Controller. The controller is the part responsible for the flow of the application and what happens with the data. It's task is to supply the View with data from the model to display, as well as receive and handle data and events coming from the View in the form of input-fields, clicks and similar interface-operations, handle the data, and then give it to the model to be stored. The controller can be thought of as the middle man that makes communication with the View, displayer of data, and the model, holder of data, possible.

This design-pattern forces a lot of good design-practices to be used in the development of applications. Separating the View into its own component that is not interconnected to the application-flow and the logic, makes it easy switch between different types of Views as well as utilize modular, reusable View-components that can be used several places in the same application. As long as one ensures that the View contains the right connection-points to the controller, and the right slots for the data coming from the model, any View can be used.

The design of the application also becomes quite object-oriented as the parts of the system are made very modular and when the parts are to speak with each other, it is easiest to design them as objects with fields and methods. The controller represents the object's possible actions, the view represents the object's appearance and the model represents the object's attributes. The name "Model" is even derived from the idea of thinking about the data as models of objects, height, weight, color being attributes that can model the appearance and behaviour of a stone. By building your application using these MVC object-like components, the intuitiveness of objects automatically makes the application's structure and architecture more organized and readable and the connections between each part of the system become more clear.

2.3 Data binding

When working with the MVC design-pattern, the View is the displayer of the data and the model is the holder of the data. When we discuss in which way the application handles data-communication with the View and what a View and a Model can and cannot do with this data, we are talking about the properties of data-binding.

When designing the relationship between the View and the Model and how they interact with the data that flows between them, data-binding becomes an important subject. It describes the relationship between the two and provides routines and protocols for how data is to be connected to the parts of the View, as well as how changes to the data in the View affects the Model. Different types of data binding can lead to applications having to be designed in completely different ways. There are primarily two types of data binding that are relevant when speaking of the MVC design pattern and application development.

One-way binding

As the name implies, one-way data binding is a relationship between the Model and the View where data is only able to flow one way. The most common way in this case is for data to flow from the Model to the view, but not being able to flow in the opposite direction. This leads to the Model being able to display data to the view, and if the data in the model changes, the data displayed in the view will also change. This however, does not work in the other direction. If data is changed in the view, usually by a user interacting with the user-interface, the data will not be changed in the Model. This ensures that the application is the only one allowed to change the value of its variables and control the flow of data.

Two-way binding

In two-way data binding, data has the ability to flow and be mutated in both directions, both from the holder to the displayer, and from the displayer to the holder. When employing this type of data binding, the user will be able to make changes to the data that is displayed in the view, and these changes will lead to an update of the data kept in the Model. This way often leads to data and updating of displays being easier to manage in applications using two-way data binding, but in turn they lose a layer of protection against the actions of the user and has to have good protocols in place for sanitizing and controlling a user's actions.

One-way data binding is usually preferred when dealing with more static systems. This is because in static systems data does not need to be updated as frequently and users have limited ability to make changes to what is displayed in the application. Two-way data binding often handles dynamic systems more easily, where data and display often needs continuous updating and users can heavily affect what the application displays.

2.4 Static and Dynamic pages

In the realm of web-development there exists two main types of web-pages. These are Static and Dynamic pages. The names refer to how a web-page behaves and to what extent users can interact with the web-page and at what rate the web-page changes in behaviour and appearance. Although the exact difference between static and dynamic web-pages are not always clear there are some key-features setting them apart:

If a web-page's main purpose is to display non-moving, textual or graphical data and the user's primary role is to view the data and not interact with it, the page in question is most likely a Static page. It will often have the following qualities:

- The content being displayed to the user will most likely be the same every time the user tries to access the content.
- It will mainly be written in HTML, and there will be little to no extensive functionality provided by a scripting language such as JavaScript.
- Each request from the client usually receives the same response from the server and content is only updated when changes are made to the content stored on the server-side

Dynamic pages are web-pages which offer a great number of ways for the user to interact with and change the data being displayed, or the application itself makes continuous changes to the data. These pages will often contain animations, changing graphically displayed data, buttons, sliders and options for users to make changes, as well as generally more complex functionality than a static page. It will most likely have the following qualities:

- Content may continuously change and be updated by scripts running on the server side, and clients might need to update often to always stay up to date with the content stored in the database. Examples of this might be stocks or other graphed data that change regularly. Web-sockets are often used to ensure that clients stay up to date with new content.
- Displayed content on the client side may have large opportunities for being changed and affected by the user's actions using client-side scripting. An example of this would be options for users to choose in which way they want to view graphical data(Pie-chart, bar-graph etc.) or a web-based photo-editing software.

Again, there is no absolute line between Static and dynamic pages but by studying the way the page behaves and how the server and the client behaves, one can draw conclusions on whether or not it is closer to being static or dynamic. (2)

2.5 DOM and the Virtual DOM

The Document Object Model (DOM) is the standardized API that represents HTML. All HTML elements are nodes in tree which makes up the DOM. When changing the data in the DOM there are several steps that happens. First the HTML is parsed to construct a new DOM tree. Then the DOM tree constructs the layout of the render tree by applying CSS. Finally the render tree is painted by the browser. The process is fast for a single change but once you want to make multiple changes this becomes a problem. This is because this whole process is repeated for each change. The whole render tree is recreated each time.

The virtual DOM is not a replacement of the DOM but rather a middleware between the DOM change and the re-render. The virtual DOM is a JavaScript object tree that that represents the real DOM. The virtual DOM can be translated to the real DOM and shown in the exact same way as previously described. On the surface this just seems to add extra steps but there are further optimizations that can be done when utilizing the virtual DOM. Since it is only in memory, changes to the virtual DOM are efficient and does not require re-renders. This means that multiple changes can be done to the virtual DOM without translating it to the real DOM. Only when all the desired changes are done to the virtual DOM is the real DOM changed.

Another problem with regular re-rendering is that the whole render tree is updated again. One application of the virtual DOM is to use it to only update the parts that are affected. Since the virtual DOM is a JavaScript object tree, a diffing algorithm can compare the object trees before and after the changes. This will then produce a result of what is different and change that in the real DOM. This can heavily reduce the workload surrounding DOM manipulation. (3)

2.6 Agile Software development

Development methods are based on different philosophies, project management and project requirements. A core part of the subject lies within the contrast between an adaptive and a predictive method. An adaptive method is defined by flexibility around milestones and ability to change the milestones during the process. A predictive method on the other hand is based around analysing time and risk for each task before making an estimate for when each task will be finished. This allows a much more defined long term development plan with specific tasks narrowed down to predicted time frame.

The traditional waterfall model is based around singular building and testing phases for each feature. In the extreme case, each feature is thoroughly completed with documentation before moving on. This means that any change or reimplementation of a feature will contribute to a lot of work wasted. One aspect of agile development is less commitment to each task. This removes excessive documentation and planning from being discarded too frequently.

3 Choice of technology and methods

3.1 Choice of technology

React

Our primary development library where we gather most of our crucial View-management functionality like routing, DOM-manipulation and data binding from is Facebook's React library. It is a very popular and in-demand software library used in the development of many websites and applications found on the internet today. Since the primary scientific endeavour of this bachelor is the choice of which library/framework best suits our application, the reasoning for the use of React is explained in much greater detail in the following chapters, so we will make no further attempt at justifying React in this chapter, but will focus on the accommodating technologies we use together with React to make everything function in the best possible manner. https://reactjs.org/

NPM

Node Package Manager (NPM) is the most universally used packagemanager for web-development regardless of what library or framework one primarily uses. Node package manager is a software giving a developer access to the largest collection of source-code in the world today, the NPM-registry. The registry contains libraries, frameworks and technologies uploaded by many different users and it provides tools for keeping your technologies organized and up to date. Every technology used in this project, including the React-library, is gathered from and maintained by the Node package manager. https://www.NPMjs.com/

React-simplified

React-simplified is a JavaScript-library written by Ole Christian Eidheim that acts as an extension to the React-library, with the main goal of it being the simplification of many aspects of React. It removes much of the boiler-plate code needed for state-management in React and replaces it with much simpler functions that can be used instead of duplicating large portions of code. We have experience using this library from our previous projects as we were introduced to it by Ole Christian himself during our initial work with React and we have used it ever since, as it is quite useful for making already compact React-components even more readable. https://www.NPMjs.com/package/react-simplified

TypeScript

To compensate for classic JavaScript's lack of static type-checking which often leads to severe run-time issues, we quickly decided we were going to use some kind of external technology or extension to JavaScript that provided us with static typing. We mainly had two options to choose from: Facebook's Flow service or Google's superset to JavaScript, TypeScript, which compiles directly to JavaScript. We have experience with using Flow from previous projects, but our experience is that Flow rarely works perfectly and its reliance on the Flow-server to execute real-time static type checking, which often failed to run, made Flow contribute negatively to work-flow. We concluded that an actual statically typed superset would be a much more stable option for achieving this task than Flow, and so we ended up using TypeScript for the majority of our project.

JSX

JSX is a syntax-extension to JavaScript that allows JavaScript-objects to be written and represented using HTML-markup syntax. When a virtual DOM is in use as the one in React, you are working with a virtual approximation to the real DOM and not the actual DOM, so the elements created for the Virtual DOM are not actual HTML-elements but just JavaScript-objects impersonating HTML-elements. JSX is the technology that makes creating these JavaScript-HTML-elements possible. JSX is heavily used and endorsed by the React library as it mixes markup and logic together instead of keeping them apart, which is one of React's main philosophies. JSX uses very similar syntax to regular HTML, so developing with it is very easy if one has experience writing HTML.

Store

To be able to store application-wide states in your client you need a store. A store's main purpose is to hold all application-wide states and give components in the application the ability to access, read and change based on the application state. Examples of such states can be what page the user is currently on, if the user is logged in or not etc. In this project, we had a goal of reducing our product's dependencies as much as possible so instead of using the widely used JavaScript-library Redux for state-management, we chose to create our own store through the combination of regular TypeScript and the use of the sharedComponentData-component found in the react-simplified library. Our application has a very small amount of application-wide states, which further strengthens our reasoning behind creating our own store as importing Redux would provide unnecessary overhead if not compensated by extensive usage. The store is further detailed in the System-documentation.

Widgets.tsx

One of the strengths of React is its ability to easily create reusable View-components using the component-class, which enforces an object-oriented design where View-components become JavaScript-objects that can be used as markup-elements when combined with JSX. The more we utilize this object-oriented design to create generalized View-components and use them in several places, the less code needs to be duplicated and our project, overall, becomes much more readable and easy to understand and maintain. Our file named Widgets.tsx is therefore a core part of our design. It is a file containing all reusable components declared as JavaScript-objects using React's component-class. Every component in this file is exported, and other components in the application freely import from this file as they see fit. A good React-application often comes down to how well the relationship with the Widget-collection and the rest of the application is in terms of number of reused components and how well the application has created room for reusing code. A rich widget-file also makes further work on the project and the addition of new features and functionalities easier as much of it can be made by pairing together already designed components in new ways

App.tsx

The main architecture of the application is derived from the file named App.tsx. This file includes all the routers responsible for rendering the right components at the right time. All the individual components are tied together here to create the full application, and restrictions in the form of checking of roleID's on the custom route-component ProtectedRoute in the routers ensure that content is only visible to users with the right level of authentication.

i18next

To achieve the ability to change the displayed language of the application , the JavaScript-library i18next was used. i18next is an internationalization library that provides methods, functionalities and guidelines for creating and loading language files into your application and using references to language files instead of hard-coded strings to display language-specific text in your application. Language files are JSON-files found in the translations-directory consisting of a collection of references describing language-specific text in the application. Each language-file includes the exact same set of references in them, but the values connected to these references will vary depending on the language that the file represents. The library provides simple methods for switching between languages through the use of language string-codes like 'en' for english and 'no' for norwegian and the use of such a library to make the application multilingual greatly increases the degree of accessibility that the application can offer to its users. https://www.i18next.com/

Webpack

Webpack is an extremely useful JavaScript-tool for bundling and optimizing the application build. Webpack takes all the modules in the applications and their dependencies and bundles them all together into static files. After this, we use NPM's build-function to compress webpack's bundle-files even further, making the project as small as possible while still remaining lossless, ensuring that we offer the lightest possible application to our users. Results in chapter 4 clearly show that this bundling and compression has a significant impact on download size.

Not only does webpack provide this bundling service, but it also comes with useful developer tools. The one in which we draw the most benefit from is webpack's development-server and its hot-reload functionality. The development-server compiles and runs the application whenever we save or make changes to the project, making it so we never have to restart our application in order for changes to be applied. The hot-reload functionality also ensures that the browser reloads each time webpack recompiles the project. The application displayed in the browser is now always up to date with the code which greatly increased development work-flow.

Since Webpack uses babel and TypeScript it goes through an internal compilation process. This can also provide us with compiler-warnings and errors when they occur. Regular JavaScript being an interpreted language and not a compiled language, makes it so we have to run the code in the browser and only receive errors and warnings during run-time. With Webpack and the typing of TypeScript, we now get an environment where we discover errors through compilation which we otherwise would have to enter the browser to discover. This significantly reduces the risk of runtime errors. https://webpack.js.org/

Babel

Babel is a JavaScript-compiler that is able to compile JavaScript syntax-extensions and newer versions of JavaScript syntax like ES5 and ES6 into normal, runnable JavaScript that browsers are able to interpret. Babel is a crucial component in the ability to use supersets like TypeScript and the syntax-extension JSX together with regular JavaScript, and is therefore a must-have when working with React, because React strongly depends on JSX-syntax. https://babeljs.io/

3.2 Choice of methods

Our methodology consists primarily of two different types of methods: Methods used for developing the product and backing up our design-choices, and methods used for gathering empirical scientific data to back up our technological choices. We begin by describing our scientific methodology for collecting empirical data. To gather data from multiple sources and in multiple forms, we had to employ several different tools and technologies to achieve this:

Empirical

The browser

Modern browsers are starting to become extremely powerful in terms of what developer-tools they provide that can be used to monitor your application. These tools are quite extensive and include monitoring of several performance-parameters for your website, as well as a developer-console for monitoring JavaScript-logging and an inspect-tool for studying the DOM and the css-styling. The inspect-tool and the JavaScript-console are tools primarily used during development of the application, while monitoring can be done both during and at the end of development to benchmark your application on several different qualities. In attachment B we provide detailed benchmarking of our application and the old application and compare them in order to conclude whether or not we have met our goals for the application stated in the vision document. We primarily used the browsers Google Chrome for this data-gathering. Their development-tool in the browser is called DevTools which you can do further reading on at https://developers.google.com/web/tools/chrome-devtools. The main tools we employed were:

- Memory The memory tab is a tool designed to monitor the current size of the application that is loaded into the browser. It can provide a very detailed overview of each individual part of the client and the sizes of each of them. By monitoring with this tool, you can identify both how much total space your application takes up in the browsers memory, which again is taken from the device's RAM, as well as see which parts of the data make up the largest pieces.
- Network The network-tab is designed to monitor the flow of data going between the client and the server. It displays both the time data takes to travel and the size of the data being transmitted. The tool is great for assessing how efficiently the client and the server communicates, which is an important factor in optimizing resource-usage across the application and making the user's experience as fast and seamless as possible.

Lighthouse

Lighthouse is an open-source extension to Google Chrome that acts as an analysis tool for collecting meaningful data on your website in the categories performance, accessibility, best practices and Search Engine Optimization (SEO). When run on a website of choice, it runs through multiple tests in all 4 categories and displays results, as well as ratings in each category from 1 to 100. It is a great tool for discovering weaknesses in your application and learning what you should do to fix them, as the software provides detailed explanations for each weakness and in what manner they can be dealt with. Seeing as it is only an add-on to Chrome and open-source means that we cannot guarantee its correctness or effectiveness but in our experience using it on not only our own websites but any other website out there and seeing it produce meaningful data has given us confidence that this tool is quite useful in giving an overall impression of what fields you website is most lacking in https://github.com/GoogleChrome/lighthouse.

Usability testing

One of the most effective ways of identifying if the user-experience you've created is actually usable and easy to understand, is to conduct usability-testing. These are tests usually consisting of different use-case scenarios that users are to complete, and observers make observations on how easily and effectively users are able to achieve these tasks. They are a great tool for discovering what works and what doesn't work in a particular design, as the developer often has a totally different understanding or view on the design than a randomly selected user has.

When conducting usability tests, considering the user-group and the user-environment during testing is also very important. Different user-groups often have very different skill sets and predispositions to understanding designs and scenarios, especially if these are very domain-specific designs. Testing on users with different levels of domain-specific knowledge and experience is therefore quite important. We therefore took steps to assure that we conducted tests in different environments and in different groups. Still, we explain in chapter 5 that finding a pool of test-subjects diverse in both size and experience proved difficult because of the unique situation we ended up during the execution of this bachelor's.

Development

We will in the following section explain the different methods we chose to deploy to aid our development of the product and why we chose them specifically.

Scrum

When developing a product, one usually follows some sort of executive, all-encompassing plan that aims to organize and follow the whole developing-cycle from start to finish. Such plans can be found in a large variety, some with more success than others. As was explained in chapter 2, usually all software-engineering projects nowadays are conducted with the use of an agile development-model. This ensures flexibility and control over the process and the possible changes that can occur during development, as well as leave room for testing and redesigning based on test-results. We no doubt wanted to go for an agile model, and our agile model of choice was Scrum.

The main reasoning behind our choice of using the Scrum development-framework is mostly based on experience and previous results. We have already, during our education, executed a relatively large-scale project using Scrum which proved a great success. Scrum provides methods for both the planning and structuring of the work, as well as the execution of the development itself. Scrum's use of back-logs and sprint-logs align very well with the requirements in the vision document, making the planning of the work organized and easy to grasp. Executing large projects can often be difficult due to time estimation, work-load estimation and delegation of tasks, but through the use of Scrum's great progress-tracking tools like the burndown-chart and the Scrum-board, it is possible to achieve a great level of understanding and overview of the process during development. Seeing as the model is agile and flexible, finding deviations or unexpected time-usage during analysis of the scrum-board, burndown-chart or Gannt-chart creates the possibility to adjust development plans, rearrange tasks and reevaluate expected progress in order to meet deadlines. Traditional waterfall-methods fail in adapting to new circumstances because each feature has very large time and work investment. Scrum shines in its ability to maintain control over the process while still being able to change greatly due to the iterative and incremental philosophy and practice. Seeing as we have never executed a project of this scale and with so much individual responsibility, choosing a flexible method like Scrum is especially important if we want to ensure that a good product is produced despite the eventual short-comings on the planning- or execution-front. Our usage of this methodology is documented in chapter 4. We advise further reading on all that Scrum contains here https://www.scrum.org/resources/what-is-scrum.

Wireframing

Considering a large part of our focus when developing the new system falls on user-experience, wireframing becomes extremely important. Wireframing is a time-cheap option for creating mock versions of product. This means it is great for creating many iterations of mock-ups. By scheduling meetings specifically to discuss wireframing with the employer, we can get valuable input on what works and what doesn't, in order to be sure we create the product that the client actually wants.

Very soft usability-testing is also very useful to conduct on wireframes. The will not be nearly as accurate or as interactive as the actual design would be, but a user still makes up very definite thoughts on how he/she expects the application to behave through the design, so hearing their thoughts and observing their actions when interacting with the wireframes is very important. Spending a lot of time wireframing is much cheaper in time and resources than spending a lot of time redesigning the application through programming, so this method is very powerful in ensuring the flexibility we want when employing an agile method like Scrum in the cheapest possible way.

Our main tool for wireframing used in this project is the interactive and collaborative software mockflow found on (26).

Gitlab and CI

As our tools for version-control, we used Gitlab. Gitlab is a version-control service where we are able to work on different version of the same project locally on our own computers. Then when we wish to synchronize our local work with the work of others, we push our code to Gitlab which then handles the mergin of our code into one unified source-code.

Continuous integration (CI) is the methodology used for ensuring that a product that receives consinuous updates, newly integrated features and changes always aligns with certain requirements as efficently as possible. When developing software, these requirements are usually that the software continues to pass certain tests when being worked on. To achieve continuous integration, we used Gitlab's CI-friendly testing pipelines which is a feature of Gitlabs that allow you to run a pipeline of tests each time someone pushes code to the repository. The pipeline will reject the request to merge if the tests fail and only accept integrations if the tests pass. This way one ensures an efficient development-process as Gitlab handles the testing on its own without our need to spend time on it, and we ensure that our software stays on par with our requrements.

4 Results

The wishes of the employer that were described earlier in the report clearly state that there is a need for an evaluation of the current system, the technologies used to create the current system and current relevant technologies and use this evaluation to make an informed decision on whether or not the current system and its technologies hold up or if the need for an upgrade or reimplementation is present. Our thesis also reflects this, and a big part of our bachelors and our scientific endeavour is therefore to research what makes different relevant technologies of current and previous web-development differ from one another and use these differences to decide on what approach to take and what technology to make use of when working on the system.

The plan of action for arguing which strategy will benefit the system the most, and which technologies to use will be as follows:
First we must achieve an understanding of what system the employer wishes to have made, and what parts of the system will have the most influence on our choice of technology as well as which part of the system that will be most affected by our choice of technology. To identify these qualities and attempt to label them, we must study the vision document.

When we are confident that we have reached an appropriate understanding of the system in question, we must decide on the right course of action in terms of which technology is best suited for the system and if the benefits of a possible change in technology is large enough for it to be worth it to make large changes to the system. To achieve this, we will compare different technologies to the different theoretical aspects described in chapter 2 and look at which types of these theoretical properties each of the technologies possess.

After identifying the technical needs and properties of the system and which technical properties each of the different technologies possess, we are in a position to identify, based on these technical properties, the technology that is most suited to use in developing the system. As stated earlier, the decision must not just be based on which choice theoretically looks the best but we must also account for all realistic, real-world challenges that will follow from the choice and together with these make a decision. The obvious example of this is that even though we might argue that one specific technology is superior to another, re-implementing the whole system from scratch might not be realistically possible and we must therefore consider other options, even if said technology theoretically is the optimal choice.

4.1 Evaluating the system

We begin by evaluating the needs of the system based on the requirements found in the vision document and draw on the theory from chapter 2 to decide what type of application it is and in what categories it will fall in order to be able to choose the right technology for it later.

By looking at chapter 3.2 in the vision document, we see what users the system will consist of. This gives us an idea of the demographic the application is meant to be catered towards. When looking at the users of this application, there is a clear distinction between each user-group and each user-group are to be given different parts of the application. It will make sense to evaluate each subpart of the system on its own as functional and technical requirements might differ between the parts.

Firstly, we can look at the part of the application that is meant for the students to see. The vision-document which defines the user needs gives us an idea of what the system needs to be able to do in order to satisfy the needs of the users, as well as the employers wishes for the application. We look at all the rows related to the user-group “Students” to understand the needed capabilities of the student-page.

1. Students need to be able to see an overview of the subjects they are a member of.
2. Students need to be able to look at their exercises for a chosen subject and what their status' are (approved or not approved)

3. Students need to be able get their exercises approved by teachers or student assistants

4. Students need to be able to see and edit their user-settings

We can start off by using this information to identify some of the properties of the system related to the theory. We begin by looking at whether the application is a static or a dynamic website (Chapter 2). We can try to understand and answer this question by looking at the different requirements of the system listed above and identifying if these requirements and functionalities contribute to a more static or a more dynamic experience, considering both the client and the server side.

Both requirement number 1 and number 2 describe lists of data-objects that are to be looked at. There is no need for the user to be able to interact with the data contained in the described lists in any way other than to simply observe it. By comparing this with what defines static and dynamic pages (Chapter 2), it becomes clear that these are static components. The requests for these pages and the data contained in them will be mostly the same all the time. The only time they are different is when the data is updated on the server side (No client-side mutation of data by the user). The frequency at which these data are updated in the database is also quite low. A teacher rarely ever registers subjects more than a few times at the start of a semester and exercises are approved at maximum a few times per week. The lack of user-interaction and low frequency of change in data makes these static pages.

The same can be said about the users needing to look at as well as change their settings. The settings a user is able to change in this system are alternate email, password and language-preference. These are all items that very rarely need changing and if they do, at most get changed a few times in the course of a year. There is no changing of data happening on the server side that is out of control of the user, and only minor changes can be made by the user on the client side. The lack of extensive amounts of serverside and clientside data-mutation and the low frequency of change in the data also makes the settings page a static page.

The students' ability to get the attention of a supervising instructor and being able to have their exercises approved are functionalities that are covered by the queue-component page. Our evaluation of the systems needs on this particular matter is based on the specific design that we chose to go with for this page. The reason for this design and how it came about is described in (Chapter 4 5) as well as the details of the design, but the effect of the design-change led to this page going from a dynamic to a static page.

The page displays all data relevant to the students entry in the queue. The data displayed includes things like location, which exercises to approve and the students position in the queue. Of the data displayed, only the displaying of time spent in the queue can be considered to be truly dynamic. This data updates every second, which in turn forces the displayed value to update every second. This can be considered as a high rate of change for displayed data, and is therefore dynamic, but as it is the only thing which updates at a high rate on the whole page, the page itself can be considered to be mostly static.

From our findings on how the different pages of the student-application behave, we can conclude that the student part of the application is mostly a static website with little to no severely dynamic activity on neither client nor server-side.

A student assistant will have mostly the same experience with the application as a regular student would, aside from some extra functionality that a teacher also has. This means analyzing the needs of the teacher-side of the application will be sufficient in order to understand the technical requirements needed for both the teacher and the student assistant's experience with the application. The main requirements that need to be met in order to satisfy the user group "Teachers(and student assistants)" are:

1. Teachers must be able to have an overview over and to manage students, subjects and exercises.

2. Teachers must be able to approve the students' exercises, as well as recognize who needs help.

If we look at the first requirement, we can see the keywords overview and manage. Overview and management implies that there not only needs to exist a way of displaying information to the user, but also a way for the user to interact with the data and make changes to the data or to the displayal of the data. The nature and behaviour of the displayal mainly relies on what data is to be displayed. In this case the data are subjects, students and the students' exercises. Displaying this data will most likely be split into several pages, each page displaying one type of data. To decide whether or not this displayal will be dynamic or static, we look at the behaviour of the data.

The rate of change of subjects were described earlier concerning the students, and although a teacher might be part of more subjects than a student, the rate of change of this data is still quite low. Subjects are usually only created one time each semester, and are rarely changed. This data is therefore quite static. Managing a subject will include editing the information of a subject through an edit-page. Aside from members of the subject, it does not hold a huge amount of data and a page that lets you edit general info on a subject will not require extensive client-side functionality other than some buttons and input-fields. Although such a page is not purely static, it is not heavily dynamic either.

Managing and having an overview over students' exercises in a given subject can become a heavy task. Each student will approximately have 10-20 exercises and there will be anything from 50 to 500 students in a subject. If one wishes for the functionality to interact with each individual exercise in a meaningful way, this page needs to include a lot of interactable objects. The data itself may not be very dynamic from a server side perspective as it only gets updated once or twice a week, and so the heaviest of dynamic functionality lies in the client-side scripting allowing for interaction with the data. We can employ clever tricks to reduce the amount of dynamics needed and still retain the same amount of functionality, as will be shown in Chapter 4 5 describing our implementation, but there is still no doubt that this page will need to be rather dynamic.

The management of students in subjects requires interactable lists with well over 500 elements and such a page falls under the same category as the one described above. There is little dynamic change of the data on the server side, as the students in a subject mostly remain static throughout the subject's lifetime, but some dynamic client-side functionality is needed in order to let teachers smoothly manage the students.

Probably the part of the whole application that is going to be the heaviest in terms of dynamic data updating is the queue. The queue is necessary for teachers and student assistants to be able to manage students that are in need for help or approval of their exercise. It is the main attraction of the application and it is where most teachers and student assistants will be spending their time. The queue will consist of a list of students sorted by the time they've spent in the queue, and by which type of assistance they require, help or approval.

Such a queue might consist of several hundred students and each of the elements in the queue must display some information about the entry of the student. The most dynamic part of this information is the elements diplayal of time spent in the queue. Displaying time spent in a queue for hundreds of elements and updating this every second is a very dynamic property.

The students themselves, the data in the list, is also quite dynamic. Students can come and go as they like, they can edit their information (location, type of assistance etc.) and several teachers and student assistants will be interacting with the students at the same time. Websockets are required to keep every client up to date with what is stored in the database, and the more activity there is, the more often each client has to update. The queue will therefore include both extensive client-side functionality and scripting (local time-keeping, ability to manage each queue-element) and server-side communication (constant websocket updates, data changing at a decently fast pace in the database). We consider this component to therefore result in the most dynamic page of the application.

The final piece of the application that can be considered its own part, is the administration pages. An administrator that is logged in is, by the vision document, in need of the following functionalities:

1. The administrator must be able to manage all types of users of the application
2. The administrator must be able to manage locations such as campuses, buildings and rooms.

We have already discussed what type a user-managing page is in terms of static or dynamic when we talked about teachers managing lists of students and an administrator managing several groups of users would simply require the same functionality as this, only spread over more pages. There will be a decent amount of client-side functionality needed through scripting, but not much dynamics in terms of the data filling the page.

Managing locations such as campuses, buildings and rooms will be of the same nature as managing users. The data-set will definitely be much lower than that of the users, considering there are thousands of students attending a campus and not nearly as many rooms. This means that such a page, aside from client-side functionalities needed to be able to manage the rooms, will be experienced as mostly static to the user because of the low frequency of updates on the data. Rooms and buildings are seldom updated after they've been created.

4.2 Summary of the evaluation of the system

If we are to easily draw lines between the needs of the system we are to build, and the technologies we are studying, in order to make a choice of the best suited technology, we should summarize our findings here. This will provide an easy way of referencing later, as well as a better overview of the combined requirements of the system.

Students

- Overview of subjects
  - Static levels of client-side functionality
  - Static data, low rate of change on server-side
- Overview of exercises
  - Static levels of client-side functionality
  - Static data, low rate of change on server-side
- Display of queue-entry
  - Moderate levels of client-side functionality
  - Static data, low rate of change on server-side
- Conclusion: **Mostly static**

Teachers and assistants

- Managing of subjects
  - Moderate levels of client-side functionality
  - Static data, low rate of change on server-side
- Managing of exercises
  - Considerable levels of client-side functionality
  - Static data, low rate of change on server-side
- Managing of students in subject
  - Moderate levels of client-side functionality
  - Static data, low rate of change on server-side
- Queue

    - Moderate levels of client-side functionality
    - Dynamic data, considerable rate of change on server-side

- Conclusion: **Mostly static, Queue is rather dynamic**

Administrators

- Managing of users
    - Moderate levels of client-side functionality
    - Static data, low rate of change on server-side
- Managing of locations
    - Moderate levels of client-side functionality
    - Static data, low rate of change on server-side
- Conclusion: **Static data, client-side is somewhat dynamic**

As can be deduced by reading our evaluation of the system above, we focus heavily on identifying what are static components and what are dynamic components. The reason for this is that a lot of other theoretical principles such as data binding, framework/library and the MVC design pattern often tend to rely on whether or not a system or webpage is static or dynamic. We will discuss these different types of principles and theoretical properties and which of them are suited to solve what kinds of problems in the chapter below, but many of the properties that set the different web-development technologies apart is often very dependent on the static/dynamic qualities of the system.

Identifying how dynamic or static the system needs to be, the complexity of the front-end client functionality and the frequency of change in data in the database tells us how much and what functionality we wish to see in our chosen technology and what methods for View-management, data-binding and server-communication our chosen technology must be able to do.

The vision document also states some governing qualities that the employer wishes to see in the system that will affect the technology we find to be best suited for the system. Some of these qualities were also found and agreed upon through discussion and thinking about the applications use-cases, user-base and user-environment. The most important governing qualities that will affect our choice of technology is as follows:

- Several complaints by the employer were made concerning how unusable some of the components of the old system were on mobile and smaller devices, both in lack of a responsive user interface, as well as components and pages being very slow on mobile devices
  Implication: A technology must be chosen with regards to it being lightweight and have the ability to include only what is necessary for each part to function (little overhead). In the event of reimplementation, design changes must also ensure that components are as responsive and light as possible to render and have function.

- The user-base will in numbers almost entirely consist of students using the student-side of the application. Considering a class of 500 students with 15-20 teachers responsible for it, there will be roughly 90% to 95% students using the application. A student's normal day attending a university involves a lot of travel from place to place, work and the use of a computer for other things than browsing the web and will benefit from having our application be as easily accessible and usable as possible.
  Implication: Because of this and the fact that the largest portion of the user-base are students, making the application as lightweight and mobile-friendly as possible becomes a high priority and will certainly guide our choice of technology, maybe even more than the technical requirements of some of the heavier components found in the administrator and teacher sides of the application.

4.3 Comparing technologies

Now that we have identified which technical needs the system has and what aspects of it is important to take into account when choosing the right technology, we must establish an overview of all relevant technologies that are eligible for use in developing the system. These technologies must be categorised and researched based on several different parameters, derived from the principles and theories described in chapter 2, as well as other important factors like market and support, dependencies and what the technologies are intended for. We will make no attempts to argue which of the different qualities and properties of the technologies are better or worse in this chapter, but only focus on identifying and explaining what they are.

When creating such an overview, it becomes apparent that it is impossible to discuss and analyse every possible technology out there that somewhat fits the bill of what we are looking for. This is especially true in the realm of web-development, where frameworks, libraries, technologies and trends change all the time and technology that is new and revolutionary now suddenly becomes deprecated or outdated merely a few months or years after its release. Therefore, for this to be a meaningful and in-depth analysis of the most relevant technologies found in the field right now, we must identify the few key technologies that we believe are the best possible options to choose from and focus on these.

Figure 1: AngularJS logo

To begin with, it is obvious that considering the old system is written using the JavaScript web-development framework AngularJS, we should analyse this technology and identify its key features. Considering that one of the main requests of our employer was to get an expert opinion on the current state of the system and answer the question of whether or not to reimplement the whole system or continue development on the current version, it becomes especially important to explore how well AngularJS holds up with modern web-development and if AngularJS really is the best match for such a system.

Figure 2: Angular logo

Another choice of technology for analysation is Angular. The inclusion of this technology follows naturally as it is very similar, and considered the successor of AngularJS. The names of these two technologies might seem similar, but they are in fact considered to be two different frameworks. At one point during development of AngularJS, the development team made severe changes to the framework that deviated so much from the original framework that it made the most sense to separate the two. We will be analysing the newest stable version of Angular, Angular 8.

Figure 3: React logo

React will be our next technology to include. This technology is created by facebook and has seen widespread use across the web. When we have been studying and working in web-development during our education, we have primarily developed using React. Considering it is a very popular framework and it is the one we have the most knowledge of and experience with, it makes sense to include this as a possible technology for the project.

Figure 4: Vue logo

Another quite popular and up-and-coming technology is Vue. Vue is a technology that employs much of the same techniques and ideas that React does, but tweeks them and builds on them to better fit the creators wishes. Vue is special because it is not developed by a huge company or organization like React and Angular are. Rather, it is being developed by one singular person. This can result in both strengths and weaknesses as we will discuss later, but considering Vue's likeness to React and its very fast growth in the web-development community, it was a good candidate for discussion when choosing the right technology.

Market research

When reducing the amount of technologies to consider in order to have a small enough list to be able to analyse in a meaningful test, an important factor was studying the state of the web-development community and the market share of each of the most used frameworks and technologies. Although such a study cannot determine which technology suits our needs the best, as it tells us nothing of their technicalities or features, it provides us information on which technologies are popular, which of them has great support from the community and possibly good support and documentation online. The more popular and well-used a technology is, the more likely it is that the technology has enough support to be sustainable in the future and will keep on receiving updates, fixes and staying relevant and up to date.

Considering the fact that this project had gone through 4-5 reworks already and the fact that technologies in the web-development are so fleeting and short-lived, we wanted to make it a priority to choose a technology that was sure to not be outdated soon and one that is flexible in terms of updates and dependencies. From (4) we found the following data on web-technology usage across the field (We urge you to look at (4) yourself as it contains a lot more data and metrics on the different technologies, we just show some important excerpts here). The data in (4) is from 2019:

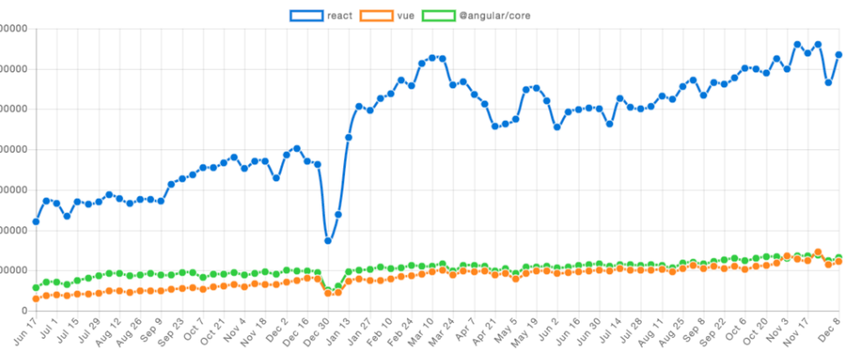


Figure 5: NPM-downloads over the past 2 years (4), the drop at 30 december is mutual between the technologies and should not be used to compare them

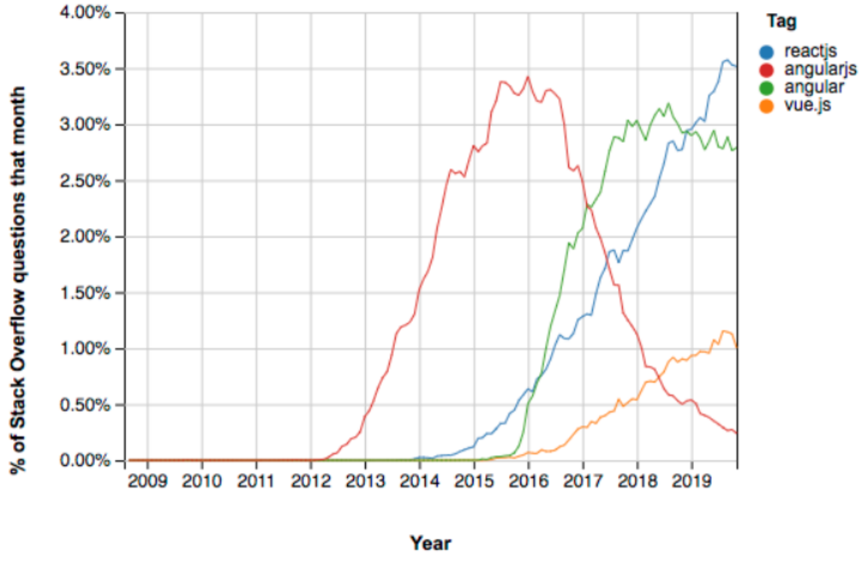
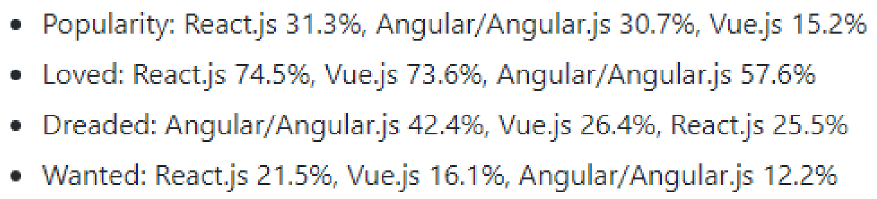


Figure 6: StackOverflow-trends (number of issues with the tags react, angular, angularjs or vue) (4)



- Popularity: React.js 31.3%, Angular/Angular.js 30.7%, Vue.js 15.2%
- Loved: React.js 74.5%, Vue.js 73.6%, Angular/Angular.js 57.6%
- Dreaded: Angular/Angular.js 42.4%, Vue.js 26.4%, React.js 25.5%
- Wanted: React.js 21.5%, Vue.js 16.1%, Angular/Angular.js 12.2%

Figure 7: StackOverflow-users opinions on the different technologies based on a survey conducted with roughly 90,000 developers

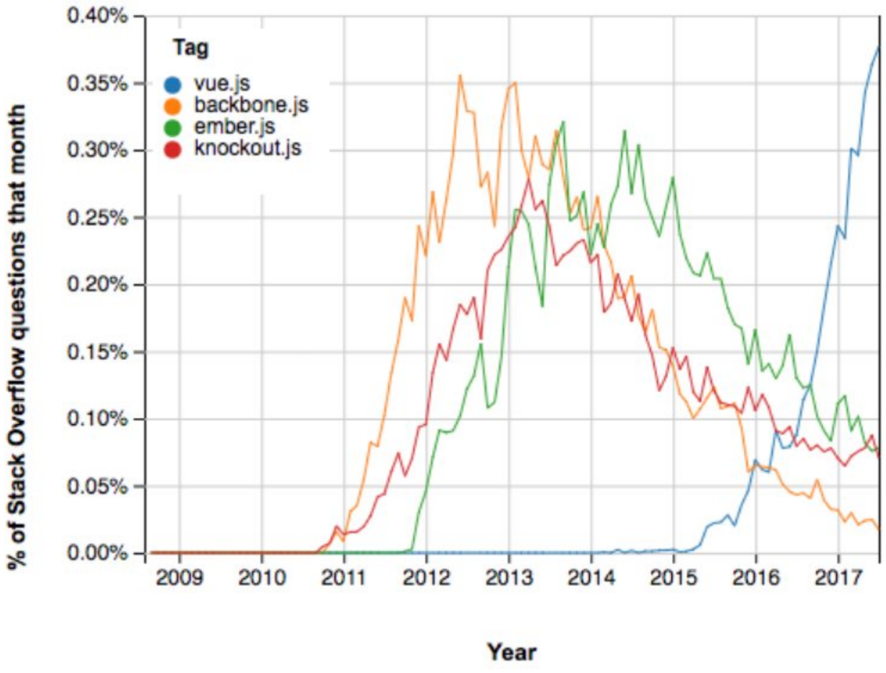
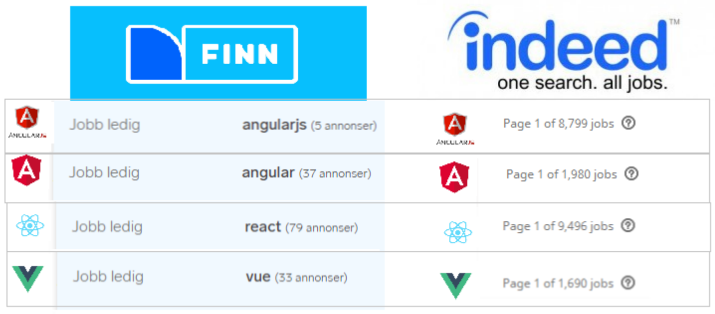


Figure 8: Vue versus other similar technologies that have fallen off the market, measuerd by stackOverflow-questions per month



FINN		indeed one search. all jobs.
Jobb ledig	angularjs (5 annonser)	Page 1 of 8,799 jobs
Jobb ledig	angular (37 annonser)	Page 1 of 1,980 jobs
Jobb ledig	react (79 annonser)	Page 1 of 9,496 jobs
Jobb ledig	vue (33 annonser)	Page 1 of 1,690 jobs

Figure 9: Number of job-applications for each technology. Data collected from finn.no and indeed.com, data visualised by us

Framework and library

AngularJS and Angular 8 (Frameworks)

By reading their own documentation and how they describe themselves on their website, we find that AngularJS is a framework and not just a library. It is a framework built as an extension to HTML with the intention of making HTML more flexible and easier to manipulate via scripting.

Their goal is to make the development of new web-applications as easy as possible by providing an extensive and detailed application-skeleton for the developer, containing built-in methods for everything form rendering, data binding, deep-linking, form-validation and security as well as css-resources and scripts making it very easy to create good looking, responsive components.

Although Angular and AngularJS differ in many places, this area is not one of them. The general direction that the developers of Angular are headed and their idea is still very much the same as it was during AngularJS. Angular 8 is also a comprehensive framework containing large amounts of pre-written libraries, scripts for responsive web-design and dynamic components like dropdowns and navbars, css-libraries and an application-schema that very much guides you in how you build your application. (15)

React and Vue (Libraries)

The idea and vision of React is quite different from that of AngularJS and Angular. Their vision is not to create a predetermined recipe for application-development containing large amounts of support, libraries and scripts making it extremely easy to create large-scale applications. Rather, their goal is to provide the developer with a library made up of code covering all the basic needed functionalities for building an application, like data binding, DOM-manipulation etc. but leaving the actual structure and design-choices of the application in the hands of the developer.

Another distinction that can be made between React and other frameworks is that React is only concerned with the UI and the View. Its main features, like the data binding and DOM-manipulation are all part of providing an easy way of manipulating the View and how it renders, but how you structure the rest of your project, like communication with the server, styling, modelling and logic is up to the developer to design.

The freedoms that come with using a library like React is the ability to pick and choose exactly which features and functionalities to include in your project, and which to exclude. In a framework like Angular, you have little choice of what to include and don't include. Since the framework takes care of so many features for you, they do not trust you, the developer, to know which resources are needed to run the project and which are not. They relieve you of this responsibility, but in turn this often results in the project containing a lot more resources than it effectively needs to, making the build less lightweight.

The library Vue falls in the same category as React. It is not a framework containing a detailed description and structure of the application to build, but rather a library containing loads of useful features at your disposal to make developing the application less time-consuming but at the same time preserving your freedom to choose how you structure your app and what you want to include.

Just like React, it is a library mainly concerned with the UI and the manipulation of the View and its relationship to data in the model. It contains methods for manipulating the DOM and binding data to the View, but leaves most of the other parts of the application untouched and up to your choice. (15)(16)

Data binding

One of the techniques described in chapter 2 is data binding. This is a feature that every new web-development library or framework should provide support. It is the act of binding the data contained in the model with the view, making the data displayable, as well as handling how the view or the model is affected by changes to the data. Support for data binding in these frameworks and libraries save the developers a lot of time as one need not worry about manipulating the DOM or updating the view manually when data changes, as this is done automatically.

We already know that there are primarily two types of data binding, one-way and two-way. The way in which data binding is implemented can vary, but any implementation of it is either one-way or two-way.

AngularJS, Angular and Vue sets themselves apart from React in that they all offer the ability to use two-way data binding, whereas React does not. React only has the ability to create one-way bindings between the View and the data held by the model. If React is to handle inputs and actions from the user which transforms or affects the data in the application, it needs to handle this using events. Instead of giving the user the ability to directly change the data like in a two-way binding, the user is only given the opportunity to trigger an event. Such an event informs the application of the user's wish to interact with the data, and it is up to the application to decide how it will respond to the event. By binding data this way, the application creates an extra layer of security where the user never has access to the data directly and with every user-interaction, there can exist middleware to handle interaction with the data.

In AngularJS, Angular and Vue, you have the opportunity to create two-way bindings between the data and the View. Not only can the data manipulate the View, but the View can also make changes to the data directly. This lets a user directly interact with data and affect it without any middleware or application-control disrupting the interaction.

Two-way bindings can be very useful by providing the shortest possible route between user-interaction and data-manipulation, and might be very beneficial for complex and dynamic applications where a lot of data needs to be interacted with, and where the event-triggering and middleware execution would slow things down or complicate it. Still, two-way binding is often very error-prone as the application has less control over how the users actions affect the data, and errors might be more difficult to trace.

Both Vue and Angular offer good implementations of both one-way and two-way binding of data, while AngularJS primarily requires the use of two-way bindings and React only offers One-way data binding. Identifying which needs an application has and what type of behaviour it exerts(Dynamic, Static) affects which type of data-binding that should primarily be used. (5)(6)(7)

MVC relationship

The MVC design pattern is described in chapter 2 and is a blueprint for creating web-applications and it concerns the way we model the relationship between the different parts of the application, specifically the model(Holder of data), the controller(Manipulator of data) and the View(Displayar of data). It is a very popular design pattern and a lot of frameworks and libraries employ it. The degree to which different technologies enforce the idea of MVC or any other design-pattern for that matter, decides how much freedom the developer is left with to create his/her own architecture.

AngularJS and Angular 8 (Strict MVC)

AngularJS and Angular being frameworks, is already expected to be more strict in what design choices one can and cannot make, and this is also true for how one wishes to construct their applications architecture. AngularJS and Angular employ a strict MVC-pattern with clear separation of each component.

When creating AngularJS and Angular apps the Model, the Controller and the View are all separated into their own respective files. The view is made of what angular call Templates that consist of only markup(HTML) containing data bindings. No logic or manipulation of data occurs in the View-file. The controller consists of pure JavaScript(AngularJS) or TypeScript(Angular) and is concerned only with logic and orchestrating the communication between the model and the view. The model consists only of references to the data and is not concerned with its behaviour or displayal at all. By placing each part of the architecture into its own concealed environment, the framework achieves a clear separation of concerns and each part is only responsible for doing its own job.

Such a clear separation forces a developer to think through exactly what responsibilities each part is supposed to have and it often automatically results in a more organized design and file-structure.(8)(9)

React and Vue (None or optional MVC)

Where Angular is quite strict in its design-ideas, React and Vue are more loose. Instead of forcing separation of concerns by dividing each part of the MVC-pattern into different components in different files, React and Vue rather invite the idea of keeping them together and separating them as little as possible. React and Vue are very similar on this subject, but still differ some.

The developers of react have stated that they do not align with the idea that concerns in a web-application should be separated as heavily as they are in frameworks like Angular. They believe that each part of the application, the view, the model and the controller(logic), are so tightly interconnected and dependent on each other that it makes more sense to keep them together than to keep them apart.(14)

React therefore uses JavaScript classes representing components, and such a class contains the view, the model and the logic all in one. To achieve this, React uses a technology called JSX. JSX is a syntax extension to JavaScript that allows the developer to create React-elements that simulate HTML-elements of the DOM, but have the ability to work together with JavaScript. It allows for JavaScript-logic to be written directly into the markup that makes up the elements of the component and so creates a strong relationship between the logic and the view. These JSX-elements are what make up the Virtual DOM. The model in this context becomes the fields that are declared in the class. Because this data is declared in the same class as the JSX and the JavaScript-logic, the controller and the view can make direct use of it.

Vue is a bit more flexible in terms of design-choices than React is, as it allow for both strict separation of concerns using templates like Angular, as well as the use of the Virtual DOM using JSX-elements and baked-in logic in the HTML-markup. Because of this, the developer is free to choose what kind of technique best suits the kind of application that is to be created. Choosing a strict separation of concern or a more interconnected design like React uses, depends on the needs of the application. Vue is quite flexible in this area as it provides the use of the real DOM and the Virtual DOM. Using a combination of the two is also possible, and where templates fall short in reflecting a complex behaviour wanted in the component, the render-function and JSX can be employed instead, to achieve the needed dynamic logic without the need for overly-verbose templating like seen in (19). Vue and Angular 8's MVC is often thought of as being more (MC)V as the model and the controller(the logic) is combined into one component most of the time. (10)(11)

Visualising the different design-patterns of these libraries and frameworks looks roughly like the following:

Typing

Typing in programming languages refers to a categorization and limitation of a data representation. Strict typing means that there are rules for how each type can be used that cannot be circumvented. A result of this is that it limits the syntax errors produced when using a programming language. The cost of strict typing is, in a lot of cases, the freedom to write code in certain ways. There are situations where a languages forces unpractical implementations because typing will not allow a much more efficient and readable solution.

In JavaScript variables are not initiated with explicit types. An explicit type means that when a variable is created, it might point to a number but is also assigned a type that is number.

TypeScript(12) is a superset of JavaScript. TypeScript can be configured to have different levels of strictness. A notable difference between JavaScript and TypeScript is that TypeScript is a compiled language. This means that it is easily possible to discover errors before runtime in the compiler. TypeScript compiles to plain JavaScript. The advantage of having reduced errors in the code greatly outweighs the compilation time and the boilerplate code needed for the setup.

Angular is dependent on and enforces the usage of TypeScript. This has been built into the framework since Angular 2. This means that Angular is not backwards compatible and upgrading from AngularJS to newer version of Angular requires that all JavaScript is exchanged for TypeScript. Both Vue and React can be used with either JavaScript, TypeScript or a combination.

Another option is to use static type checking. Flow(13) is an open source library created by Facebook. Flow utilizes a Flow server to check types live while in an editor or IDE with the correct environment setup. Through testing we found that there is a problem with the Flow server where it will stop processing the code for longer periods of time. This interrupts workflow to a degree in which makes it a burden to use.

Dependencies

The amount of dependencies a framework, library or any other type of technology has refers to the amount of resources linked to the use of the technology that is absolutely necessary for it to function. The more dependencies something has, the less flexible it becomes. You are left with less choice in what to include and what to exclude as most of the resources tied to the technology are mandatory and finding out which are not can be a hassle in and of itself. The upside of heavy dependence is less responsibility put on the developer, as well as a lot more built-in features and functionalities to choose from. The amount of dependencies that come with a technology is often tied whether we're talking about a framework or a library.

AngularJS and Angular 8 (Highly dependent)

AngularJS and Angular 8 being full-blown, complete frameworks naturally makes them highly dependent technologies that rely on the full inclusion of all of its resources and built-in features if one are to utilize the technology to build an application.

React and Vue (Mildly dependent)

React is the least dependency-heavy technology of the four in discussion. It provides you with the necessary features to handle rendering and DOM-manipulation, but leaves almost all of the remaining parts of the application to be decided by the developer. It puts little to no restrictions on what to include in the project, and one is free to add and remove features as one sees fit.This puts a lot of responsibility on the developer to identify relevant technologies and resources to include in the project, and because of this, compatibility often becomes an issue. When frameworks like Angular provide you with a large amount of dependent resources, they are all managed by the framework and ensured to be compatible, functioning additions to the project. Choosing whatever you want to include when using React is no walk in the park, as React is only concerned with their own technology contained in their own library, and external resources you include to achieve wanted functionality is not guaranteed to be quality-ensured or compatible with React or any other resources you've added.

Vue is more of a combination of the two. Vue is also a library where you can choose only to include the most necessary parts for creating an application, but gather other needed resources from other sources. It is, like React, very flexible in letting itself be used and included with other technologies, unlike Angular which is difficult to integrate into other technologies. Despite this, Vue contains a lot more built-in features and resources than react, and is often thought of as being in the middle of a library and a framework. The developer has the choice of including tons of features and functionalities from the standard library that are highly dependent on each other, effectively creating a "framework" of sorts, or he/she can include only some of it, leaving it more flexible, and combining it with other technology. This makes Vue more easily cater to any kind of developer, both the ones that want the stability and ease that comes with a framework and the ones that want the freedom and flexibility that comes from something like React.

One can get an impression of how many dependencies a technology relies on and what they offer by looking at the libraries' and frameworks' sizes:

Updates and support

How frequently updates are pushed to the technology and how great of a support one can expect from the developers of the technologies and/or the community tells us how much assistance and helpful resources one can expect during use of the technology. This is an important factor to discuss when considering the longevity of the technology and how easy it will be to hand over the project to other developers after we've completed development on it.

It also gives us insight into how easily the system can be maintained and for how long of a period the system can stand on its own without having to update it or make changes to it. We also find it an important factor to look at, as one of the wishes of our employer was for the system to be able to stand on its own feet as much as possible with little to no interference required to keep it running.

AngularJS

Concerning the update and support of AngularJS, which the old system is written in, there is not much to discuss. AngularJS is an outdated framework left behind by the developers after the framework diverged into the new Angular 2.0. Their own home-page at angularjs.org states clearly that AngularJS has gone into long-term support, and unless severe security issues are found, will not receive any sort of updates any time soon. This puts AngularJS in an obvious spot of being deprecated and outdated in comparison to newer technologies, and is a huge factor in judging this technology in relation to our thesis.

Angular 8

As angular is one of the most popular web-frameworks out there, there is bound to be a lot of support from the community and one is sure to find good solutions and tips to most of the problems that one can run into. Since it is developed and maintained by a huge tech company like Google, there are resources to provide great support from the developer side as well. Great documentation ensures great communication and support between the developers and the consumers of Angular.

One of Angular's traits that sets it apart from React and Vue is its update policy. They way the development team of angular has decided to handle updating and patching of the framework is by deploying large, batched updates every 6 months. These updates can contain everything from fixes, to changes to new additions to the framework. They've chosen this type of deployment-schedule over continuous and disjointed updating whenever changes and bugs present themselves.

As we discussed earlier, Angular is quite a dependency-heavy framework. Since many of the parts are dependent on each other and mandatory for the framework to function properly, this means that it often is necessary to always stay up to date with the current version of the framework. Since Angular deploys extensive updates every 6 months which almost render the previous version outdated or incompatible with the new version, a company almost always have to choose to either constantly stay up to date by applying changes and rewritings that align with new updates, or ignore it and accept that their systems slowly but surely becomes deprecated and unable to adopt new features. This kind of update-schedule is one of many traits that make Angular a better fit for large, enterprise applications maintained by companies of considerable size and manpower rather than smaller applications without much maintenance, as they often do not have the resources to be able to keep up with the constant, large changes made to the framework. (22)

React and Vue

React and Vue being libraries and having a great ability to be used together with other types of technology, projects using one of these two are often very independent and consist of a wide spread of different technologies and packages. This makes it so that any updates or changes made to these types of libraries most of the time have very little impact on a project using them. One often has the choice to update only parts of the libraries, while keeping other parts unchanged and since the technologies are designed with compatibility and flexibility in mind, huge changes are rarely made and if so, older versions are still kept as part of the library as seen in (25), where React promises that even though they introduced hooks as a new way of creating functional components, they would never render components created as classes a deprecated technology. React also states that they commit to keeping breaking changes to the minimum to ensure updating the library is as painless as possible. By doing this, maintaining a React or a Vue app for a longer period of time is a lot easier and resource-efficient than doing the same with Angular. (20)(21)

Support-wise we have seen from the market-research that React holds a larger market-share than Vue, Vue being a considerably new technology, and one can therefore expect React to have a better foundation in the web-development community and you will probably find better support for React over Vue in the community, although Vue has also grown quite a bit so there is a chance the support of Vue will no longer be a problem when using it in development. Both technologies, just like Angular, present well-written and easy-to-use documentation that make the road to mastering either of them a smooth and comfortable road to travel.

4.4 Summary of technologies

AngularJS

- Strict Framework containing loads of features and resources for producing a fully fledged application.
- Not very compatible with other technology, provides everything on its own, not very flexible
- Two-way and One-way data binding
- Strict separation of concerns using the MVC-pattern.
- Dependence-heavy
- Entered long-term support, considered to be deprecated

Angular 8

- Strict Framework containing loads of features and resources for producing a fully fledged application.
- Moderately compatible with other technology, provides most resources on its own, not very flexible
- Two-way and One-way data binding
- Semi-strict separation of concerns using the MVC-pattern.
- Dependence-heavy
- Heavy updates are rolled out every 6 months, can be breaking changes, resource-heavy to maintain
- Developed by a large company

React

- Library containing only the most necessary parts for building a basic web application.
- Very compatible with other web-technology and almost relies on it to build good and rich applications. Very flexible, but provides much less on its own.
- Only One-way data binding
- Little to no separation of concerns, but instead uses integrated MVC where all parts are present in one component.
- Not very dependence-heavy
- Updates when needed, seldom breaking changes, easy to maintain
- Developed by a large company

Vue

- Library containing a moderate amount of parts for building a web application.
- Very compatible with other web-technology, but includes a lot in its library for the developer to have the choice of whether or not to use Vue's technology or others. Moderately flexible, but can also provide much on its own.
- Two-way and One-way data binding
- Optional MVC-pattern where both strict separation of concern and integrated concerns are possible
- Moderately dependence-heavy, optional dependencies
- Updates when needed, seldom breaking changes, easy to maintain
- Developed and maintained by one person

4.5 Scientific results

DevTools

B.1.1 and B.1.2

Memory snapshot of the student subject page. The different parts of the memory usage is split up into the DOM, scripts, strings and others.
B.1.1 React
B.1.2 AngularJS

B.2.1 and B.2.2

Memory snapshot of the teacher subject page.
B.2.1 React
B.2.2 AngularJS

B.3.1 and B.3.2

Performance snapshot of the student page. These are taken after the pages have cached what is possible. The bottom section of the DevTools show the amount of resource requests the page made, how fast the DOM was loaded (DOMContentLoaded) and how fast the browser load event started (load).
B.3.1 React
B.3.2 AngularJS

Lighthouse

B.4.1 and B.4.2

Performance section of the lighthouse report generated for each version of the system. See the rest of the report in the HTML file located in the project handbook.
B.4.1 React
B.4.2 AngularJS

4.6 Engineering results

UX and Responsivity

We will now present brief explenations for what each of the snapshots found in the attachments under chapter 8.1 contain and what part of the application they display. For all of the snapshots, the old system is found to the left in the image and the new system is found to the right. A.1 is for mobile devices, A.2 is for desktop or other.

A.1.1 and A.1.2

Approval lists, functionality like list-download and global approval disappears in the mobile version of the old system

A.1.3 and A.1.4

A students overview of his/her exercises in a subject, status-information moved to the top in new version as to not disappear when there are a lot of exercises.

A.1.5

Location-management: campuses, buildings and rooms are managed together and simultaneously

A.1.6 and A.1.7

The login page, we've simplified it greatly, added the logo, restructured and removed unnecessary text.

A.1.8 and A.1.9

The main queue-component that the teachers and student-assistants have access to.

A.1.10 and A.1.11

The part of the queue that a student sees, the old one includes the whole queue where other students are visible as well, the new version contains only data on the student's own entry in the queue.

A.1.12 and A.1.13

How the queue-component that the student has access to looks when the queue is both paused and includes a message from a teacher or student assistant.

A.1.14 and A.1.15

How the queue-component conveys to the student that a teacher or a student assistants have activated their element and plan to help them.

A.1.16 and A.1.17

The part of registration of a subject where you add exercise-rules.

A.1.18 and A.1.19

The main overview of subjects that a teacher sees. The old version does not show a subject's status and does not contain option to start or stop queue, this is done on a separate page.

A.1.20 and A.1.21

How the navigational components of each of the systems look when accessed via mobile. New version is a vertical drop-down as opposed to the old system's horizontal pop-in.

A.2

The approval lists for students in a subject that teachers have access to.

A.3

A students overview of exercises in a subject.

A.4

The footer that was added in the new system able to hold important information and the "Bugs og forslag"-button which provides an opportunity to send feedback on the system to the developers and administrators. Both the desktop and the mobile version is displayed.

A.5

The components responsible for giving the administrator the ability to manage locations like campuses, buildings and rooms. Three separate, but very identical pages seen to the left have been combined into one page in the new application.

A.6

The login page

A.7

The part of the registration of subjects where one chose the duration of the subject. In the new application, we removed the need to enter an end date and instead calculated and displayed this automatically based on the start-date and duration chosen.

A.8

The main view of the queue that teachers and student assistants have access to.

A.9

The component that contains all the actions that a teacher or student assistant can take on a queue-element in the queue after it has been activated.

A.10

User-settings. The new application includes the ability to set a language-preference, which the old one does not have.

A.11

The first pair of old and new components is the selection of which exercises a student wishes to get assistance or approval of. The new version includes icons to display already approved exercises. The second pair of old and new components is the selection of which type of help a student wishes to receive. Changed from check-boxes to toggle-buttons. The last one is the component that allows you to add other students to your queue-element.

A.12

The part of the queue that a student sees. In the old version a student sees the whole queue as opposed to seeing only his/her own queue-element in the new system.

A.13

The main overview of subjects that a student has access to.

A.14

The homepage of a subject that a teacher can use to maintain the subject and add or remove students or users in the subject.

A.15

Overview of subjects that a teacher sees, 4 components have been compressed into 1 in the new system, where positioning of elements to describe their nature has been replaced with different status-colors and text.

A.16

Overview of users, which looks the same for users as well.

A.17

The first pair of components show an example of difference in hierarchical UX-design between the old and the new login-page. The second pair of components show a comparison of an alert-box from the old and the new system.

Security

The current status of the system is that we believe we have fixed every major security hole that was present in the old application. The details of exactly how we implemented the security-measures and what measures we took to ensure security in the application is described in the system documentation and so we advise you to read this. The implications and justifications of the measures we took to ensure security are discussed in chapter 5.

Testing

The reasons for our great lack of testing and why we have failed to achieve our wanted level of testing as engineering-goal is discussed in chapter 5. The available testing that we were left with was unit-testing through scripts and jest that were maintained by the CI gitlab-pipeline. The only other external testing that we were able to conduct were a few usability tests that we managed to run on people we live with. We did not get hold of a lot of inexperienced users to test on as everyone we live with are students and have experience using such software. Still, they provide valuable data. The user-tests can be found in the folder "userTests" in the project-folder.

4.7 Administrative results

Documentation

For the administrative documentation we refer you to the project handbook document found in the project-folder which will lead you to the documents related to timesheets, status-reports, our Gantt-chart, work-logs and work-agreement. Our ours and total ours spent on each task and on the entire project are shown in the timesheets, but as a summary for easier access:

- **Erling Roll**: 513 total hours, within requirement of 501,5 hours
- **Vegard Andersson**: 526,5 total hours, within requirement of 501,5 hours

Scrum

We refer you to the project handbook document found in the project-folder, which will guide you to the sprint-backlogs, the product-backlog and our visual documentation found in the ScrumSnapshots document.

Meetings and communication

We refer you to the project handbook document found in the project-folder, which will guide you to meeting-inquires with summaries of each point from the meetings written in them.

5 Discussion

5.1 Choosing the right technology

We begin this study by evaluating the system in terms of being static or dynamic.We followed this by taking a closer look at each of the four technologies we saw fit to discuss and identifying which key traits that set them apart from each other. We did this being as objective as possible, not saying which traits are better or worse, but simply stating them and defining them, as the question of them being worse or better often is a subjective question based on the environment we plan to use them in. We're now going to discuss subjectively which one of the technologies is best suited for our system, based on our evaluation of it.

Let's think about what wishes our employer has together with what we know of the system:

1. Application is not going to be maintained by a large company, as of now it is only being maintained by one person

2. The employer has a wish of making the application needing as little maintenance as possible in order to run.

3. Employers had several complaints concerning the speed and general feel of the application with regards to performance and optimization.

4. We found that the application is mostly going to consist of pages manipulating static data, as well as many of the pages themselves being static display of data

5. The app includes some more dynamic components on the administrative side of the application

6. The level of mobile-friendliness currently found in the application was also a concern voiced by the employer, where mobile-friendliness concerns both user-interface and performance

7. Roughy 90% of the user-base, if not more, is made up of the students, which are going to be using the lightest, least complex version of the application.

To identify which technology best satisfies requirement 1 and 2, we take into account what we have found concerning their update-cyclus, support and how dependency-heavy they are. If the wish is for the application to need as little maintenance as possible and for the application to stay up to date as easily as possible, React and Vue present themselves as better options than AngularJS and Angular.

AngularJS is an obvious bad fit as the framework is considered deprecated and receives only long-term support from the developers. Because of this, there is of course no need to continuously update the application with the latest versions of the framework in use, but since the framework is already outdated and lacking in providing a lot of necessary features and the wanted performance, keeping the framework is not an optimal choice. AngularJS is also rather dependency-heavy as we found in chapter 4, meaning that inserting new features and assets from say a newer version of Angular to compensate for AngularJS's shortcomings is not going to be easy, as new technologies are not likely to be compatible with it. We even explored this when trying to migrate from AngularJS to Angular 8, where we had very little luck in getting AngularJS-code to work properly in Angular 8 as even the architecture had changed drastically from one to the other.

Because of Angular 8 being quite dependency-heavy as well as having the 6-month update-cycle, it is not a very good choice when considering maintenance either. The update-cycle which brings with it extensive changes, often breaking, will make keeping the application up-to-date and compatible with new technology a more tolling job than it needs to be. You do not have the freedom to pick and choose new features and technologies to include as freely as in React and Vue, meaning that if you want to integrate something new into your framework you most likely have to upgrade the entire framework to the new version, which again will result in you having to change a lot more than only the feature you wanted.

Angular is still very well suited for large, enterprise applications as they often have more than enough man-power to keep the application up-to-date and can draw on all the features and resources that the framework offers much more effectively. The truth is that the application we're going to work on, according to our evaluation of it, will not be large enough and complex enough for it to be able to utilize all the strengths of Angular. The price of having to constantly update and maintain will then be higher than what we are able to draw out of the framework in return, and the relationship between framework and application will not be very beneficial.

In this case, React and Vue are much better matches. Since they are libraries, they are very modular and flexible in terms of what can and cannot be included. They do not follow the same scheduled update-cycle that Angular does, but instead deploy fixes and updates as they see fit when they need to. React even stated that they would avoid breaking changes as much as possible.

Since projects created in React and Vue often consist of many different 3rd party technologies and only the core is made up of technology from React or Vue, this means that maintenance and keeping the system up-to-date becomes a distributed job between each technology. If a new feature is announced or a crucial update has been applied to one or more technologies that you have included in your project, tending to this mostly becomes tinkering with only the technologies in question. There will be no need to update an entire framework and apply huge changes, because as long as the technologies are compatible with one another, unrelated packages and parts of the project don't need to be touched. This kind of relationship between the different parts of the project will fit the situation described in requirement 1 and 2 better than AngularJS or Angular will. React might even fit these conditions better than Vue, as Vue's complexity and dependencies stem from how much of the library you include and how much you want to rely on Vue's own technology beyond the core necessities. The responsibility of making the app as flexible and easy to maintain as possible then becomes more difficult in Vue than in React, as you have more options to choose from.

Requirement 3, 4 and 5 concern the general nature and behaviour of the application. The main trait that we wanted to identify during our evaluation of the system was which parts were static and which were dynamic in terms of data and the level of user-interaction. The reason for this is that this trait determines how we want a lot of the other traits like data binding, size, structure to behave.

By looking at our evaluation of the needs of the system that we conducted in chapter 2.1, the overall conclusion was that the system consists of mostly static pages. The teacher and the administrative parts of the application are the ones that contain most of the dynamic parts, primarily the queue and the managing of exercises, which are the two heaviest components.Taking into consideration the fact that 90% of the user-base are students that will take advantage of the static part of the application, and will most likely utilize a lot of mobile-devices, we conclude that having a focus on our chosen technology being fast, lightweight and suited for static applications is going to be our main mission. We will still ensure that the technology can handle all parts of the application smoothly, even the dynamic ones, but choosing the lightweight, flexible option will no doubt be hugely beneficial for the application considering its user environment and user-base.

When it comes to technologies being lightweight, flexible and having well suited for static applications, React and Vue again beat out AngularJS and Angular. The main factors that make React and Vue, especially React, well suited for handling static applications is because of its data binding, its modularity when it comes to reusable components, its architecture in terms of MVC, its size and its lack of dependencies. Let us look at each part separately:

Data binding

When dealing with static applications or generally applications of little dynamic activity and low complexity in its functionalities, One-way data binding is often the preferable way of binding data to the View. One-way data binding, if implemented correctly, always provides an extra layer of security and control over the user's interaction with the data displayed. Two-way data does not provide this same kind of security since user's can have direct contact with the data and change it without going via a listener or any other middleware. The upside that two-way binding brings to the table is, as we spoke of earlier, an easier time interacting with the data. This freedom over the data, however, only becomes beneficial when there is a lot of data to be interacted with and this usually only occurs in heavily dynamic and complex applications. In static applications, One-way data binding is often enough to provide a seamless experience for the user, while still maintaining superior control over the data.

We have seen that all of the technologies include a way of performing One-way binding. This should mean that neither of them are to be preferred over the other. While this is technically true, there can still be some benefit in choosing React, which is the only technology to not include Two-way bindings. Firstly, if you remove the ability to perform two-way bindings it is going to mean that the library will have one less functionality to provide for. This is going to result in less complexity and a smaller size of the library or the framework in question. Another benefit is that it forces a distinct pattern of coding across the application, which will ensure that all bindings are of the same type and that the security of One-way bindings are achieved across the whole application. When having the option of two-way bindings like in Angular and Vue, more responsibility is put on the developer to ensure the right bindings are made and security-measures are taken wherever they choose to perform two-way bindings. Two-way bindings might also be chosen over One-way bindings in places where they are not needed, so having only One-way bindings in React removes some of the responsibility from the developer and has a higher chance of producing a secure application when dealing with user-interaction. We believe that this coherency is worth giving up on two-way bindings as we do not see the need for them based on our evaluation.

Architecture

If we have come to the conclusion that our application is not a heavy one, with it being mostly static, and that our wish is for it to be lightweight, we look for an architecture that will provide as little boiler-plate code as possible and be as readable as possible. The less boiler-plate we have, the less code we are going to have to write, and the lighter our application becomes.

How one achieves readability and good architectural structure in a project depends on how comprehensive the project and the code is. If we are dealing with large, complex and logic-heavy components composed of many different parts, readability means making it as modular as possible and separating concerns so that each sub-part of the component becomes more readable on its own. Angular's structure and design that we design ideas that we discussed in relation to the MVC pattern earlier, is perfect for these kinds of large, "messy" components consisting of a lot of different functionalities. Their focus on separation of concern into seperate parts for the View, the Model and the controller gives them the ability to keep JavaScript-logic and HTML-templating in their own files and effectively making each part more readable.

Although this is beneficial for large components, it can work against itself when creating smaller components. If components of too small a size are separated into different parts and put into separate locations, they might become less readable than they would be as one whole piece. The overall purpose of the component might be difficult to make out without seeing it assembled, and having separate files for all the separate parts comprising the components of the application can make the project messy, unorganized and difficult to maintain when doing things like bug-fixing, as responsibility becomes scattered and errors might be difficult to trace through the project.

React actively uses the non-separation design described in 4 which makes use of the JSX-technology to intertwine HTML and JavaScript together and not separating them. When creating small to medium-sized components, this way of writing creates really compact, easy to read components that are just as easily debugged as the relationship between markup and logic is more obvious when viewing them together and React can provide error messages that concern both parts at the same time.

Although Vue has the ability to choose between the two, separation of concern is the most well supported way of structuring components. Vue provides the user with the option to separate concerns into their own parts while still remaining in the same file through their own, special file-extensions called .vue files.

The main weakness of such a design is when components become large and complex. Combining HTML and JavaScript can become quite unorganized and difficult to understand if too much logic or conditional rendering needs to be packed into the component. The HTML might bury the JavaScript making the logic difficult to understand or HTML-elements might be filled with too many conditional clauses and complex event-functions, making it difficult to understand exactly what the HTML-element is supposed to look like or how it's supposed to behave. Still, if components' complexity are kept to a minimum using good modular design, React's compact design is going to provide a lighter application and development-environment than Angular, AngularJS or Vue.

Size and dependencies

The employers wish for the app to be self-sufficient and not in need of a lot of maintenance implies that the chosen technology should not be very dependence-heavy, should not force the application to need maintenance due to frequent, heavy updates and new features should be easy to incorporate without changing much else.

By looking at our findings on dependences, updates and support under chapter 4, we can quickly conclude that Angular and AngularJS is not a very good choice here either. AngularJS's deprecation and Angular's maintenance-inducing update-schedule make both frameworks badly suited for upholding the problem of creating a self-sufficient application.

React and Vue both have similar update-protocols with problems being fixed when they arise and the same effort to avoid breaking changes. Again, this comes naturally when being a library and not an entire framework as things much less often break when things are not as tightly connected to each other.

5.2 Concluding thoughts and final verdict

An immediate conclusion we can draw from the arguments above is that in many areas that we have discussed, Angular and AngularJS do not meet the requirements we seek. Many of AngularJS's and Angular's best qualities and strengths are often most suited for applications of a considerable size and complexity or systems run and developed by large companies. Its abundance of features and pre-written code, its strict but robust separation of concern and its many under-the-hood technologies are almost all catered towards dynamic, heavy and complex applications that would greatly benefit from having a lot of the work put on Angular's shoulders instead of the developers so they can focus on the distinguishing features of the system. Since we have already judged QS to be an application not in need of all the functionalities Angular bring at the cost of size and efficiency, but looked for lighter, more flexible possibilities that are more easily maintained, we think that Angular 8 and especially AngularJS (Deprecated) are not good matches for the system. Instead we look towards React and Vue.

We also talked with an employee at our university under the computer-science faculty that is well versed in web-development frameworks. He presented many of the same thoughts as we had, especially concerning Angular's heavy update-schedule which he too saw as a big disadvantage when a small group is to maintain a system. He also quote on quote said "If you've come to the place where you're deciding between Vue or React I already think you've come a long way, either of these is a good choice over Angular" which only made us more confident in our conclusion that we should focus on React and Vue.

React and Vue are in many ways very similar. They are both libraries and lighter in size than Angular, they are both mainly concerned with the View, they both have support for the Virtual DOM and JSX-syntax and they are both less dependency-heavy than Angular. Still Vue is in some ways heavier than React. Although they support Virtual-DOM manipulation, they're main technology has for the longest time been templating and direct DOM manipulation like Angular. The way in which you take greatest advantage of the Vue-library is by separating the concerns of your components into different parts and making use of technologies like two-way data binding which we believe is not necessary. React enforcing JSX and keeping concerns woven together as well as only using One-way data binding that guarantees an extra layer of data-control makes it a preferable choice when discussing View-management.

Still, there is not a whole lot of arguments that can be made in favor of React over Vue on a technical level. They both perform very well and certainly are capable of fulfilling our and the employers wishes for the application. To make a decision, we therefore have to consider more non-technical arguments as support for our final choice. Firstly, we must acknowledge that we are the ones that are to develop and work on this project and if we are to choose the option of reimplementing the whole front-end from scratch, we must be very sure that we are capable of achieving this task during the given time-frame. We must therefore consider our own experience with the frameworks and with web-development in general.

During our education we have completed several web-projects, both personally and in a group. Applications and websites that are quite similar in nature to the current system in question have been developed, and they have all been made with the use of the technology React. We definitely have the most experience with using React and especially with the use of the JSX-syntax and the mixing of HTML and JavaScript. We have not used any other framework besides this one, and in doing so, have not developed a project using heavy separation of concern with templating and splitting into several files. This means that we not only are most familiar with React, but also with the design-choices and the ideas that this technology brings with it. If we come to the conclusion that React and Vue both are very fitting technologies for the system, we choose to value our own knowledge and experience very highly as this certainly will increase our chances of creating a satisfying product. This is not to say that we are not capable of adapting to new technologies and learning new frameworks, but our argumentation has led us to the conclusion that React and Vue are similar enough for us to include our own bias into the equation as will be very beneficial for development.

Another, non-technical argument that helps push us in the direction of React is Vue's market-status and support-status. The market-research conducted in chapter 4 showed us the current state of technologies in today's web-development community. The data shows us that React has much more tightly connected roots in the industry than Vue does. If the relevancy and ease of maintenance of the system is a concern of the employer, considering the technologies' support and longevity is important. The graphs in chapter 4 show us that not only has React been relevant for a longer time and has an obvious larger following than Vue in terms of the job-market and the developer-community, but it has also grown at a slower rate.

Figure 8 shows that Vue has had a very sharp increase that resembles the increase of its fellow technologies that have faded over time. Of course this does not mean that Vue will become irrelevant or drop out at the same pace as its predecessors, but it might be an indication that Vue's longevity is more uncertain than Reacts. Again, one should usually not put too much emphasis on this data, as choosing the right technology in terms of technical capabilities is the most important, but considering our arrival at the conclusion that React and Vue both are very fitting, we find that this data helps reassure us that React might be a less risky option to choose in terms of longevity. This is also coupled with the fact that Vue only has the support of the open-source community and its singular creator to back it up, while React is maintained by a huge company, meaning that as less people are invested in Vue, it does not have the same type of security-net as React does.

All in all, we find that based on both technical capabilities, longevity and our own abilities and experience, React is the technology we think will best ensure that we not only create a good system during our bachelors but also build the system using a technology that will make the future of the system and its maintainers as smooth as possible.

Choosing a technology in the web-development business is not an easy task. Seeing as almost every framework or library can achieve the same tasks, only in different ways, and that almost everyone in the field has their own opinions on what technology is best and they almost always differ, finding clear data on what is best is difficult. Still, we believe we are quite confident in our final choice of using React, as we think it fulfills the task of creating a lightweight, mobile-friendly and easily accessible system better than the other frameworks are able to do and the system will surely be easier to manage in the future than it was when AngularJS was the chosen technology.

5.3 Summary of verdict

- React's relaxed update-protocol and promise of few breaking changes make it more flexible and easily maintained than Angular.
- React is best suited for Static applications, which we evaluated the system to mostly be, as it enforces compact, low-complexity components and One-way data binding ensuring good control over the user's actions.
- React's component-class structure makes making reusable components and widgets very easy, further decreasing the amount of code needed to fulfill a task.
- Being a very small library, React is lightweight and filling your project with the necessary functionality to back up react becomes the developers responsibility instead of a framework choosing for you. If the developer is skilled, he can achieve a lot with use of very little external dependencies and technologies, making for a very compact project.
- React is heavily backed in both the community and by its creator, making it a very risk-free technology to use in terms of longevity and relevancy, which are both important factors in the ever-changing field of the web.
- Manipulation of the Virtual DOM using JSX is often much faster than manipulating the real DOM that Angular and Vue to some degree does.
- We have a high degree of knowledge and experience with this technology compared to the other technologies.

With the data we found and the discovery of just how poorly optimized the old application was and how bad the state of their chosen technology, AngularJS, was we ultimately concluded that re implementing had to be the way to go and that the option of staying with AngularJS was close to unacceptable. Still, we had to be sure that the task was possible, that we were able to handle re implementing the whole system and that our chosen technology could handle the most dynamic parts of the odd system. We therefore conducted some quick prototyping of the system using React. We created fast prototypes of the navigational component, the queue with scripts that simulated students interacting with queue and the approval, where we also stress tested with roughly 4000 listeners on one page (equivalent to around 350 students with 12 exercises).

The prototypes served as both opportunities to measure our speed and ability in writing the application in React, as well as seeing if the technology of React would handle what we considered to be the most challenging parts of the application for React to tackle. This was a fast and messy process that was not entirely necessary as our decision still remained that AngularJS could not be used, but if we had discovered something truly breaking during this prototyping it would have been very valuable in our decision-making. Still, the fact is that the prototyping went smoothly and we quickly concluded that re implementation was the correct approach.

5.4 Discussion of scientific results

The first part of both the results-chapter and the discussion-chapter aimed at answering the first part of our thesis, which as to identify the best and most relevant web-technologies out there today and conclude which of them would be best suited for making the system the ebay it can be in regards to the wishes and the requirements presented by our employer. This was essentially a question that we had to answer on a theoretical level, by comparing our scientific findings and our conducted testing together with theory, but there is no way to know for certain that the technology we chose indeed was the best choice until we can look at practical data concerning the actual application. We gathered such data via the methods described in chapter 3. We start by discussing the non-functional, technical parameters of the application.

DevTools

A part of the thesis was to choose the technology that would create a lightweight and fast application. The old version of the system was described as slow and unresponsive and our data gathered on this subject will tell us if we succeeded in achieving this part of the thesis. The results from the usage of Google's DevTools are presented in chapter 4 and we will discuss each important parameter displayed in these screenshots.

An important factor to look at when describing the application as lightweight is its general size, how much data that is actually being downloaded when the browser loads the application. We put a lot of effort into including technologies in our project that greatly assist in compressing and shrinking the size of our application to make it accessible on all devices, regardless of bandwidth and browser-power. We used technologies like Webpack and NPM-build described in chapter 3 with the aim of achieving this task, and looking at the results displayed in chapter 4 we consider our application to have succeeded greatly in this area.

The reduction in size should be viewed both in what travels over the network and what is loaded locally in the browser, the actual size of the client. By looking at the actual size of the client in the memory-tab, there is a clear difference. Our application's size is reduced by well over 50% compared to that of the old application, with it being roughly a 12 MB difference. If one looks more closely at each of the parts making up the application, one gets a better understanding of where the difference in size is greatest. In doing so, we find that data making up the application's scripts differs greatly from the old to the new system.

The AngularJS-version has a large amount of scripts that take up a considerable amount of space relative to the rest of the application. By looking at the scripts, a lot of them have the word angular in them as well, meaning they are scripts that specifically belong to the framework. In our choice of technology, we talked a lot about frameworks being highly dependent and often forcing you to include a lot of things that you won't necessarily have much use for. This is an example of that. Qs is not nearly large enough or complex enough to make use of all of the features contained in the scripts and if one looks at the usage of these scripts in the source-code it becomes clear that a lot of these scripts are imported because of one specific feature, resulting in a few MB's being loaded into the browser while the application in reality only uses a few KB of it. This is true for several of the other files as well including angular style-sheets.

We find that although this coincides with our conclusions of the benefits of light libraries over strict frameworks, we still believe that the old framework could have been used in a more efficient way and that much of the unnecessary files being included is because of lazy front-end programming. The source-code shows that the old system is made up almost entirely of angular-specific components, scripts and styling. Their own style-sheets are extremely small compared to what is being shown on screen, meaning they use almost only angular's own styling. The drop-downs, input fields and buttons found in the application are entirely angular's own resources as well. This means that not only does the application seem very unoriginal and indistinguishable from other angular applications as very little creative touch is added by the developers, but it effectively makes the application very large in size as shown in attachment B.

We believe that one of the core strengths of our application, which was one of our main goals during development, is the level of independence our application has that comes from having written almost all of the styling and the scripting ourselves. By scripting and styling our own dropdowns, inputs, search bars etc. we effectively remove the need for importing certain heavy dependencies like jQuery for View-scripting or large css-libraries for responsive design. By writing most of our assets and scripts ourselves, we gain numerous benefits, the most important ones being: Reduction in size, Low dependency on other technologies which result in low need for maintenance, easier to make changes as it is your own technology you are changing instead of using unchangeable third-party technology, More specialized and unique look and feel of the application making it more attractive to users and possible clientells.

The most obvious weakness of choosing a low-dependency design is the fact that the developer is given a much larger responsibility. A general rule of thumb when developing is that if you're trying to create a difficult algorithm or a large, comprehensive functionality, someone else has probably already done it better and faster than you are capable of. When the majority of our styling and scripting are self-made there is the risk of our application being more inconsistent and buggy than it would have been using pre-made assets, but we ultimately have to believe in our own abilities to write good code and seeing as the most security- and resource-sensitive features are covered by React already, we are confident that we are able to create a very pleasant and secure experience for our users without the use of much of the pre-written code found in cookie-cutter applications, and the results from the DevTools at least show that the benefits of this choice absolutely are present.

Lighthouse

The Lighthouse addon is most likely our least reliable source of information on the quality of our application and the results it presents should not in any way be seen as absolute truths and using it to compare different applications should be done with utmost care and doubt. The reasons for this are a few. Firstly, it is a third-party, open-source software. Unlike Google's DevTools which is maintained and developed by a large company and has an enormous user-base providing feedback and increasing validity of the tool, Lighthouse does not have any really reliable source that can guarantee its correctness, other than its much smaller user-base.

Another factor working against its reliability is what the tool presumes to measure. For example, one of the measured categories is accessibility. Accessibility is a very complex and comprehensive topic including loads of different parameters and though many of them are actually measurable, like checking for the presence of unique keys, labels and alt-texts, things like language-support, general design and placements, feedbacks etc. are qualities not easily measured in numbers or by programmatically scanning a HTML-document.

The third unreliability present in the tool is its attempt to put numbers on the different categories. Rating a quality like accessibility or best practices from 0 to 100 is very difficult as weighting each parameter correctly will not always make sense. Saying the lack of an image alt-text reduces the score by 8, while a missing label reduces it by 4 is actually a very inaccurate way of measuring quality as there really is no way to numerically identify just how big of an impact a missing alt-text or label actually has on the user-experience nor does it make sense to compare the numbers in relation to each other as there is no way of telling if a missing alt-text is twice as impactful as a missing label even though the numbers 8 ¿ 4 imply this relationship.

Although we now have presented you with several weaknesses of the data gathered with this tool, we think it still serves as a good way of getting an overall impression of the strengths and weaknesses of an application. The numbers should not be taken as scientifically accurate numbers, but rather numbers that imply an idea of how good or bad the application is. It is also a very useful tool in identifying obvious failings overlooked when developing like the missing alt-texts and labels described above, which then can be fixed. We therefore used this tool during development to aid in discovering such failings.

With the appropriate context established, let us discuss the results. Chapter 4 displays lighthouse's results when analysing both the old and the new application. We can quickly spot some obvious differences between the two. By lighthouse's judgings, the old application scores extremely low on performance compared to ours. The relative numbers should be ignored as the event of our application being 20 times more performant than the old version is not very likely, but the score still indicates the presence of some clear weaknesses in performance in the old application. If the numbers are somewhat accurate, this means that many of these performance-issues have been resolved in our new application, further pointing at our success in answering parts of the thesis. Generally our application receives high scores in all fields and is deemed a good application in lighthouses eyes which, even though lighthouse might be unreliable, certainly shouldn't be viewed as something negative but rather points at our application likely being quite good in these categories. Still, we will not dwell long on these results as their truthfulness is uncertain, but as quantitative analysis-tools for web-applications are hard to come by, we feel these results still hold some worth and deserve to be mentioned.

The above results and discussion of Reacts performance almost entirely mention the positive sides of using React. We can with confidence admit that this is because the technology proved to be almost entirely positive and achieved what we wanted from it very successfully. We also feel that were able to implement the technology and write the code to our best abilities and create good, reusable components and a organized and logical project-architecture and structure, as can be seen in the system documentation.

We feel we only really found one obvious weakness of React that impacted the quality of our code. It was a weakness that we anticipated and did our best to avoid and work around, but it did show itself in some select areas. This weakness is the fact that React's compact component-structure and its philosophy of not separating concerns by mixing logic and view into one entity can sometimes cause very complex and large components to become very difficult to read. To make such components work, conditional rendering is often needed and the more conditional statements there are, the messier the code becomes to understand. This also becomes worse with React's inability to use if-else statements together with JSX, but only accepts ternary-operators(?:) syntax which does not have an extensive track-record of increasing readability either.

We were able to avoid these components for the most part as most of the application was not in need of comprehensive logic or dynamic components (As expected from our evaluation of the system in chapter 4), but we admit that a few components are more unreadable than we are satisfied with. We believe that this is both a result of React's general way of structuring components and its low separation of concern, but also of our own coding and how we chose to utilize React in those components. The way to avoid such components in React is to, instead of separating concerns, you separate the component itself into smaller, more understandable components and use them as building blocks to create a large and complex component. This is also the whole idea behind the widgets.tsx file described in chapter 3 and we were very diligent with this in most of the cases, but for some components we predicted too poorly how complex the component would end up being, as it grew over time, which resulted in components that should have been separated into smaller pieces even more. Still, we believe this is just as much the fault of our implementation as it is the limitations of React and measures can be taken to avoid letting these limitations impact the application. It is still important to mention that this is the case only for a few of the largest components, but it is still worth mentioning as weakness in the relationship between the technology React and this system.

5.5 Discussion of engineering results

The engineering results contain the results of our attempts at creating an application that coheres with the requirements stated in our system requirements and vision document. In regards to the thesis, these results will mainly answer the question of whether or not we were able to create a user-friendly, secure and robust solution when employing the technology we previously decided upon. These results are of a more qualitative nature.

UX and Responsivity

The user-experience and the responsibility of our application was a big priority of ours from the start. Considering the employer's complaints about the responsibility on mobile-devices, the performance and the fact that we were assigned front-end as our main focus for this project, we knew this priority would greatly benefit the final product. We will now discuss and justify the design choices we made to ensure that ours and the employers goals were met and the thesis answered.

The results chapter for UX and responsibility contains references to a large number of screenshots of the different pages and components of the new application in comparison to how they looked in the old application. We advise you to study these snapshots to get a feel of the overall look and design of the new application and see how it compares to the old design. When creating the new UI, Universal design was one of our key concerns. We wanted to make the app as expressive and readable as possible, and we wanted every component to be mostly self-explanatory and easy to grasp. To achieve this, we used several different UX-techniques. By studying the screenshots you might be able to notice a lot of the techniques we will discuss. We do not have the ability to discuss every part of the application in detail as that would double the length of this report, and so we will instead quickly discuss our design-ideas and why we made certain decisions, and provide a few examples of those ideas and leave you with the ability to study the rest of the application with our design-choices in mind.

Contrast is one important factor for achieving readability. We made our best efforts to always ensure good contrast between adjacent components and their background. Where contrast is in risk of becoming too small and components might lose readability because of it, one should always use supplementary indicators like icons or positioning to make sure the lower contrast does not affect readability. Looking at attachment A.11, you can see an example of this approach. To inform the user that an exercise is already approved and is not eligible for selection, lowering the contrast signals unavailability of the component. Still, only doing this can lead to low readability as seen in the old application. We supplement this by also including a checkmark to show the approved status, which strengthens the understanding of the component.

Size is also an important factor. Attachment A.11 shows our version of the component that lets you choose which exercise-type you want. Although there is nothing inherently wrong with the old design as check-boxes clearly display what is chosen, the design is still very small, inexpressive and can become difficult to read on certain devices. We chose to make the whole component larger since it only serves one purpose and that purpose should be quickly and easily understood. The design uses size, extreme contrast and icons to clearly indicate what has been selected.

Icons is also a very important component that greatly increases readability and ensures that the actions and properties of objects on the website are understandable even when other factors like text might fail in doing so. We have therefore ensured that almost every single meaningful label, button or display includes not only text but also icons. Examples of this can be seen in many of the attachments, but looking at the new version in attachment A.9 it is clearly displayed in the buttons, that contain color, text and icons.

Colors are important for displaying the different meanings components can have and what nature the results of interacting with the components are of. An example of this is that green often indicates positive actions and results, why red indicates negative actions or results. Colors are also used heavily in our application to indicate the qualities described above. The alert shown in attachment A.17 is red because it indicates a negative response, and the approval(godkjenn)-button in A.9 is green because it indicates that the outcome of the action connected to the button is positive and that this button is the one that fulfills the primary action that the component in question is supposed to fulfill, which in the case of attachment A.9 is to approve the exercises.

Hierarchy and correct relative sizing is also important, as the hierarchy of components on your webpage inform the users of which parts hold the highest importance and it subconsciously directs the focus of the user to the parts highest up in the hierarchy, which is an important tool the developer can use to guide the users actions. An example of a place where we deployed this technique in order to improve the old design is seen in attachment A.17. On the login page, the login-action is obviously more important than the forgot password action, as the forgot password action is needed maybe once every 200th login. The components should therefore be sized appropriately so as to make the user find the action he is most likely looking for, the login button, more easily. The old design uses the same-design for both buttons, which can cause confusion and the user will be forced to actually read the text on the buttons, while the function of our login-button is implied almost entirely by its size and position.

Most of our designs are based on the old system's design and we have simply improved them in terms of the UX-techniques described above, and made attempts to make the system feel as unique and expressive as possible. We consider that making the design of the application unique through the use of our own css-libraries and clever design is actually an important factor, considering that the employer had future plans for possibly making the system commercially available, and the more unique and interesting an application is, the more attractive it is to other clients.

We also made some breaking changes to the design with regards to performance, privacy and mobile-friendliness which require further justification beyond just looking at the screenshots:

The student-queue

One of the most impactful changes we knew we wanted to make was the transition from having students be able to see the whole queue to students only being able to see their own, personal entry in the queue. We discussed this redesign with the employer early in the wireframing process to be sure that making such a breaking change was in alignment with the employers wishes, and it definitely was. The design change can be seen in attachment A.12 for desktop and A.1.10-A.1.11 for the mobile version. The reasons for making this change are described below:

- Privacy reasons The fact that a single student had access to the whole queue in the old system posed privacy issues. In a university-setting, your achievements and the progress you make in a subject can be considered sensitive information. The queue displays just how many and which exercises students plan to have approved during that session and looking at this number can tell you how that particular student is progressing. A student might be very late and register for approval of an exercise that was due last week, and it is very likely that this is information he has no wish to disclose to others. Reducing the queue to only displaying your own element in the new system removes this possibility and you have no way of gaining any information on other student's progression.

- Performance With the new design, which required changes to the back-end in order to work, you now only load in data concerning your own element from the server instead of the whole queue, as you did in the old system. You only render data concerning your own element as well. If you estimate an active queue to have 300 students in it, this redesign will effectively reduce the amount of data transferred from the server to the client by around 30,000% as well as reducing the amount of data the client needs to load by about 30,000%. Although this seems like a silly number to bring up, the fact is that the component in general becomes drastically lighter and faster because of this change, and considering we wanted to make the student-page as light as possible in accordance with the thesis, this change was very important to incorporate.

- User-friendliness With the new design, the data that actually concerns you becomes much easier to read and interpret. You only care about your own entry in the queue anyways, so displaying any other information is irrelevant and only clutters the understanding of the component. If you compare the display of your own position in the queue, which is the information you most often check regularly while waiting for approval or help, in attachment A.1.10 and A.1.11 you can see a clear difference in the readability of that information. Our design also gives us an ability to even more clearly signal to the user that their element is activated, as seen in attachment A.1.15.

Compression of pages

The old application consisted of components that were similar in design and functionality, but were split into several pages because they had a singular functionality that set them apart. There were mainly two examples of this, the subject-views and the location-management.

A subject can have different actions taken on it and so is in the need of different components that provide different interactions with the subject. A subject can also mainly have two statuses when it comes to the queue, it can either be active or inactive. The old system chose to visualize these different properties of a subject by only utilizing positioning. The actions for managing a subject's queue were put into its own page, while the actions for managing them were put in another. The subject's active-status was also visualised using positioning. The problems with this kind of design is mainly two: one is that you create a lot of unused space and a lot of distance between information which increases the level of navigation and actions needed to access content, the other is that the same subject-component is used all over the place while still serving other purposes so as an example, in attachment A.15 and A.13 the button associated with the subject in the old system serves two different purposes while still remaining identical, which can cause confusion as different actions should always be associated with different looking components.

We solved this by compressing 4 separate components into one singular component. The actions of starting/stopping a queue and managing a queue were both put on the same component using different buttons, and the subject's status was indicated using color, text and hierarchical placement instead of horizontal positioning. This way we can more clearly display the information we want to convey while at the same time keeping the information as compact and easy to get to as possible.

The redesign of location-management follows the same idea. The redesign can be seen in attachment A.5. Managing buildings, campuses and rooms are three actions that are very similar and relate to each other and so it makes sense to combine them instead of spreading them over three separate pages requiring three separate navigational buttons to access, as seen in the old system.

As stated earlier, justifying every single design-choice takes too much time and the descriptions provided in the results-chapter for the different designs is all that we can say for most of the designs, but we again advise you to look at all the snapshots while considering the different UX-techniques described earlier as justifications for the design-choices we have made and try to identify where the different techniques have been applied to increase user-friendliness and readability in order to align with the thesis.

As for the responsivity, we believe that every component and part of our application is fully responsive and usable on every type of device realistically found in the real world. During all of development, we spent as much time viewing the page on mobile-devices as we did in desktop-view and for every new feature added or change made, we always made sure that components did not produce unwanted behaviour on any type of device. For this purpose, Google's DevTools is again extremely useful. Their inspect-tool which displays the DOM also comes with the functionality of simulating the displays of many different devices like tablets, mobile-phones and different resolutions of desktop-monitors. This tool makes workflow when developing responsive web-applications extremely efficient. The snapshots in attachment A.1 include almost the entirety of the application on a mobile device and attachment A.1.16 and A.1.17 show a great example of an area where we severly improved on the responsivity of the component, making the application more user-friendly on mobile. This specific component part of the register-subject form which the employer expressed great annoyance towards, as mentioned earlier, and so we see it as a success that we were able to ease this annoyance by making the whole application as equally user-friendly and readable on mobile devices as it is on desktop.

Approval-list

One of the first components of the old application that our employer showed us and expressed annoyance with, was the approval-list. This component's purpose is to display all exercises for all the students in a subject and give the teacher the ability to be able to edit any one of these exercises. In the old application, the page was extremely slow and inefficient and this was due to one primary reason, which was how they had chosen to implement the page. Their choice of implementation was to represent each exercise as a listener, which resulted in cases of having to load 3000-4000 listeners on the same page at once. Even if AngularJS hadn't already been badly equipped to handle this task and contributed greatly to it being slow, this is still a very poorly optimized solution when considering that devices vary greatly in processing-power and memory.

If you look at attachment A.2, you can see that not much has changed in our visual design, but we did make one design change that drastically improved the performance of the component, which was to remove all the listeners on the main page and render them only when requested instead. You still receive the same amount of visual information as the old version, but the exercises are only graphics and not buttons with listeners. In order to access the listeners, you have to click on a student to receive a pop-up window with the ability to edit the exercises. This way, we reduce the amount of listeners loaded from 3000-4000 to around 10-20 at once. This is a huge improvement to performance with a very low loss in functionality as a teacher is not able to physically interact with more than one listener at once anyways, so loading 4000 is extremely unnecessary.

Language

The addition of the ability to change language preference was a considerable boost to the application's accessibility. Our user-base is mainly students in a university-environment where lots of the students are exchanging and many of them do not speak Norwegian. Providing the option to switch language not only greatly increases the amount of users that can access the application but also makes the application easier to adjust to different environments by adding more languages, which is a necessary feature to have considering the possibility of making the product commercially available.

The language functionality was achieved using the library i18next described in chapter 3 and the UI for switching between languages can be seen in the new version of the user-settings in attachment A.10.

Security

At the beginning of the project, it was made clear that the system had several security flaws present, and there was a strong wish for these security flaws to be eradicated. These security flaws were mostly present in the back-end of the application and concerned communication with the server. This meant that the majority of these security issues would be the responsibility of the group working on the new server-implementation. Still, it became clear that as connecting the new server and client proved to be an effort in and of itself, the best option would be to still have a fully functioning node-server based on the old system as well. If we were going to have a functioning node-server, we had to take it upon ourselves to fix the security-issues in the node-server as well.

By looking at the results in chapter 4, we believe that we were able to eliminate the most obvious security holes present. During our evaluation of the system and the old source-code, we quickly identified which parts of the server caused the unwanted functionalities and they proved to be rather easy to fix. The exact measures that were taken to fix these security flaws are described further in the system-documentation, but in short, the main issue was whenever a client requested a modification of an item, the server only checked if you had the appropriate authorization to access the item, but didn't check if the item belonged to you. This resulted in security issues where students were able to modify other students' queue-elements, since they had the right authorization level, but never got checked if they were the owners of the elements. We have also taken care of the obvious security flaw where the old system sometimes sent user-sensitive data through the plain-text request instead of accessing the token.

The ProtectedRoute-component responsible for limiting access of the front-end View to only authorized users is also described in greater detail in the system documentation. The components main purpose is to filter out any users that do not have the right role and prohibit them from being able to render certain parts of the system. This provides an extra level of security as the server-endpoints are already being checked for authorization as well as the front-end client. There is even a third level of authentication in place in each sensitive-component where if you still manage to send a render-request for a page you are not authorized to enter, the component itself will check your role when it mounts and redirect you to the home-page. For each form and any kind of input that the user is able to interact with, we also have detailed input-sanitation and alerts that help decrease the possibility of faulty user inputs as much as possible.

When considering the security holes identified and fixed in the node back-end, the ProtectedRoute-component, the redirects when mounting a restricted component and the sanitizing of user-input we believe the security goals for the application are met. From the client side it is now extremely difficult to access any page or functionality you do not have access to and faulty user-input is very hard to get away with as a user. If faulty requests manage to get to the server anyway, we have covered the security-holes that were present in the old system and see it as very unlikely that the same types of unwanted functionalities that could be done in the old system are possible in the new one.

Still, security in systems-engineering is a very delicate matter and unless developed and maintained by people with extensive knowledge and experience in application- and server-security one can never be entirely sure that there does not exist security holes or other flaws that might induce unwanted behaviour when interacting with the system. Although we are quite certain that we have eradicated the security-flaws that were known when we took on the project, we do not have extensive knowledge in server-security and considering the front-end was our main priority we have not dedicated a large enough portion of time tending to this matter either. Another factor greatly impacting our confidence in the system's security and general robustness is the lack of extensive testing, which we discuss in the proceeding section.

Testing

Although we made great efforts to avoid letting the situation of the corona-virus and the inability to leave our homes affect our bachelors and the final product through the use of digital collaborative and communicative tools, we would say that testing is the area in which this whole situation had the greatest negative impact.

Testing is an extremely important part of creating almost any type of product or service, and this is especially true for an application, considering the amounts of different, connected components it consists of and the levels of user-interaction it needs to handle. Guaranteeing a user-friendly and secure experience for the users of your application greatly relies on the amount of, and quality of the testing you do.

When we discussed plans for the development cyclus and the application earlier in the process, we agreed that we would try to create a fully functioning version of the student-part of the application, as aligns with the idea of Scrum, and try to host it in a real user-environment. We had plans to let one or more student-groups participate in the usage of our new system in their regular day-to-day and use it as they would the old system. For this exact purpose, we also quickly implemented a discussed functionality of giving users an easy way of sending feedback on the system, the bugs- and report-button which is shown in attachment A.4.

This functionality was to go hand in hand with our deployment of parts of the application to users during development. It would have hopefully provided us with invaluable user feedback during development of the application, which is the time we are most flexible in terms of making changes and adjustments. We believe that such a functionality and the deployment of the application to a real environment for testing purposes align perfectly with the ideas of agile development as we would have remained very open and flexible to what the users say and would have left great room for change during development.

A number of features of the application also rely especially heavy on testing. For example, a big part of making the synchronization of the queue-activity possible is support for good real-time action-management and the handling of possible race-conditions and parallel communication. This is especially hard to test alone in a development environment as the only real possibility for such testing that requires a large number of users, is by writing scripts. The downside to scripts is that they are not human and so are rarely inconsistent or unpredictable, which makes testing parallel communication or otherwise confusing interactions hard as such scenarios most often arises from human-like interaction not found in scripts. Such problems also arise from communication between several different types of browsers, devices and users of different environments with different internet-connections and/or protocols and none of these qualities are easily simulated through scripts either.

Security is also something that is most easily identified when testing on real users of different environments and parameters, as security-holes most often are security-holes for a reason, meaning they are difficult to spot when doing local testing and local interaction as one would have already fixed such security-holes if they were known. By just thinking of the old system as well, we are pretty sure that the reason those security-holes existed in the old system was not because the previous developers were terribly unskilled, but more so that they didn't notice the security flaws until they were discovered by ingenuitive users. They may seem obvious to us now, but only because we had access to real user-data that they did not. We even had to acquire some of the system's users in order to gain more insight into the security holes and how they were exploited, further showing how important of a resource users are.

The best we could muster for testing during this project were locally written unit-testing, personal user-testing, scripting and a few real usability tests conducted on outside individuals. The front-end unit-tests are a good way of identifying if there are breaking, functional faults in any components in the View, making them render incorrectly. They are also able to mimic user-input to see if forms and inputs function correctly and together with the usage of CI, these unit-tests can be run everytime we push code to ensure that our project always remains error-free in this area.

Still, such unit tests only measure functional and technical qualities of the application and on a rather shallow level as well, but logical errors, usability-faults, security-holes and other failings that emerge through interaction with the system do not show themselves through simple unit-testing but often require actual interaction and observation in order to be discovered. Such interaction is also infinitely more valuable when coming from an outsider as well, as the developers have a totally different understanding and expectation when it comes to the system, often making them unable to discover or spot faults that an outsider might discover instantaneously.

Therefore, it is a great sadness to us that the situation we found ourselves in the latter 3 months of development, resulted in us only interacting with roughly 5-6 other human beings which led to usability tests being extremely sparse and the possibility for large-scale system-testing in real environments being nullified. This lack of testing results in a reduction in confidence of the system as a whole, both in its robustness to handle large amounts of users at once as well as bugs that might be quickly discovered by users when the system is set into play. Still we believe that we have designed the system well enough for changes to be quickly applied and errors and bugs to be easily traced, but it is definitely to be noted that the application cannot be trusted to same level as if it was heavily tested so future maintainers of the system are advised to heavily test it before actual deployment.

5.6 Discussion of administrative results

Our original plan for the project is shown in the Gannt-diagram and it describes how we spread out the workload and how much time we chose to set aside to each task. Since such a chart is made early in the process, and agile development-projects are prone to undergo changes along the way, the reality often deviates from what is predicted in the chart. This was true for our case, as true as for any other project that plans 6 months ahead in time.

We had initially planned for both being able to reimplement the system using a new technology as well as work on some big, new features that would benefit the system and that had been discussed with the employer. These new features were electronic delivery of exercises online and a chat-functionality.

The reality that met us was that we were not able to implement either of these new functionalities. In re implementing the system, we were able to make severe improvements and add new, smaller features like the ability to reserve queue-elements among other things, but we were not able to follow the Gannt-chart perfectly as we only ended up having time to re implement the system. This was due to a few different factors:

Firstly, we knew that adding the implementation of the new features could be considered slightly overshooting our own expectations. We knew that re implementing the whole system from scratch in a new framework would require a great deal of work, and so understood that taking on both reimplementing the new system AND adding new features was going to be a tough challenge. Despite this, we knew that it was a possibility that we could make it and so wanted to incorporate everything into the Gantt-chart. After completing the project we can conclude that the main reason we weren't able to complete everything according to plan was that things simply took longer to complete than originally anticipated.

When we planned and made estimations on workload and duration we focused mainly on the pure development of each part of the system and the testing of it afterwards, but we didn't really consider too heavily that something usually doesn't let itself develop as smoothly as we imagine in our minds. For many of the components and features we developed, there would usually always follow a fair share of debugging, in-development testing, discussion, re-designing and other things that would result in development of a component always taking a few days longer than expected. When these extra, unplanned days were added to each component during development they accumulated to so much time that the Gantt-chart ended up being shifted a considerable amount, ultimately pushing the new features out of the way and beyond the deadline of the project.

We realized that we wanted to make the core system as good as possible and decided to scrap the addition of the new features. We imagined the scenario where we forced the addition of these new features anyway, which would result in less time to test, debug and finalize other components. This could result in a project that would be riddled with equal amounts of bugs, lackings and security holes as the old one did because we would prioritize quantity over quality. This would not align with the employer's wishes as he prioritized making the application as smooth, bug-free and secure as possible over adding new things. We therefore knew we would be more complacent with the quality over quantity approach, and felt that the exclusion of the two new features from the Gantt-chart was not a very large catastrophe. With the exclusion of those two features, we followed the Gantt-chart quite accurately and were able stop the heaviest part of the development and begin writing the main-report and other documentation when planned in the chart.

Timesheets and Status reports

We came to an agreement with the supervisor to send in weekly timesheets and status reports. This was achieved by giving the supervisor access to our google drive files containing the sheets and reports, and we would update them every end of the week. To organize and document our own work during the week to be able to write the status-reports and timesheets at the end of the week, we kept our own, private work-logs where we logged what we did and the time we spent working on the project each day. These personal logs ended up affecting the quality of our timesheets and status-reports as keeping the same information in two separate places sometimes becomes unorganized.

We were quite diligent in keeping up with the timesheets and status reports during most of the project, but became a little less diligent by the latter part of it. As development became more and more all-encompassing and less modular, and we no longer easily could say which component we worked on but rather worked on several components at once, worked with bug-fixing and testing it became more difficult to easily separate our time and label it with what we worked on. Noting in greater detail what we worked on and what we achieved in our worklogs instead of labeling the time with a certain component-name or activity in the timesheet, and so we ended up spending more time on our worklogs and less time translating them to the timesheets and status-reports.

This resulted in timesheets and status-reports being delivered later than once a week during the end of the project, as we would often fill out timesheets and reports in bunches every 2 or 3 weeks as it made more sense with what had actually been done. Status-reports were still pretty detailed as we had source from the work-logs to pull from, but as abstract activity-labels became harder to divide our work into as it became very varied by the end, the timesheets became much less detailed and served more as only hour-keeping by the end of the project with the activity just being programming. Because of this, we consider the timesheets to be the weakest part of our administrative work as they ended up not serving much purpose other than keeping hours, but as the status-reports in greater detail explain exactly what we did and the scrum burndown-chart doing a better job than the timesheets at keeping track of the progress of each task, we still felt that we documented what needed to be kept tracked of, but maybe just not in the correct places.

Scrum

The execution of the Scrum method is documented in the project-folder(instructions in chapter 4). We managed to complete 2 successful sprints during development, each with complete sprint-logs, burndown-charts, standups and scrum-boards created with Trello. We chose to follow the fully-fledged Scrum-methodology with all its artifacts and processes as close as possible with the exception of only a few parts of it.

The main part missing in our execution of Scrum is the Retrospective. The retrospective is an important part of scrum as it is the main platform for evaluating one's own performance and usage of Scrum. Instead of discussing the project, one discusses the process. The retrospective is where meta-opinions on the work-process are shared and it often becomes more important to execute retrospective in great detail and following all the correct steps, the more people are involved in a project. The more people there are collaborating, the more problems, unspoken opinions and general factors halting the development process there is going to be. We felt that as we are only two people with an extremely easy access to discuss anything, anywhere at all times, simply talking about the process and the project when it becomes necessary satiates our need for meta-discussion. We often discussed the Scrum-process and how the sprints went during their completion which could be thought of as retrospectives but we never executed them in such detail that there would be any meaningful documentation to be displayed in this report, which is the reason that the retrospective seems to be missing in our usage of scrum.

Generally, our implementation of Scrum went very well and assisted us greatly in delegating and keeping track of work and the status of the system throughout development. Trello is a great tool for mimicking a physical scrum-board and we used it actively to keep track of which tasks were being worked on, who worked on them and what their status was. Trello even gives notifications when tasks are moved around on the board, always keeping us informed on what the others are working. We also held daily stand-ups either in physical space when we worked together in the same room, or digitally by using voice-communication. The stand-ups were especially helpful for keeping track of work and making sure we not only stayed on track, but also avoided working on the same things or made clashing changes. The stand-up serves as a supplement for the scrum-board but is extra helpful, as it is more direct and often brings out details of each other's work that are not reflected in other parts of scrum. We initially thought of the standups as being unnecessary and awkward when working as a group of only two people but we became used to them after a while, and in retrospect we believe they were probably the most powerful synchronization tool we had in our arsenal.

Thanks to the very handy burndown-chart scripts written in google sheets, we were able to always dynamically update the chart each time we had made progress on tasks. The burndown provided a good visual representation of our progress and we would regularly make visual predictions of the future using a line and the burndown's slope to see how well we were on our way to reach the sprint's goals.

When we conducted sprints, they were affected by the same characteristics we described when talking about the Gantt-chart above. Unplanned for extra hours of debugging, tweaking, discussion and quality-assurance made every task often trail a few extra hours than planned. In our scrum-board this would relate to the fact that items were kept on "for review" longer than we anticipated, as we were always busy developing new features and completing new tasks as well. The result of this was that the burndown-chart often reached its desired endpoint, but the sprint still had to trail on longer as there was work to be done before we could start another sprint. We found ourselves in-between sprints where we had work remaining that neither fit well in the old or the new sprint, and so spent some time before we started a new sprint after the end of the previous one.

During the latter part of the project we worked disconnected with scrum and without the use of a sprint.This was because the meat of the development was mostly finished and there remained a lot of superficial work like testing, integration, communication and collaboration with the other group, that could not be easily modulated and expressed by a sprint-log or a burndown-chart. We therefore chose to stick with our two sprints and not include a third one, as we felt Scrum absolutely had served its purpose in providing a great framework for organizing our work during development, but now wouldn't serve the same purpose any longer. All in all, we feel that the usage of scrum went very well and had many great benefits to us, and we are only disappointed that the frameworks agile power of flexibility and change was not put to the test as the virus-situation and the lack of testing and user-feedback put us in a situation where we experienced very little factors that prompted the need to change or be flexible.

Meetings and communication

As for the meetings and general communication with the supervisor and the other group concerned with the system, our documentation of the meetings, as shown in the project-handbook, does not display a whole lot of extensive meeting-history. We conducted meetings regularly up until the point where the virus-situation began to develop at a quickening rate and the administration of our university began taking measures. Because the wish was to have as little physical contact as possible at a very early stage, we cancelled our first meeting as early as late february. We agreed together with the supervisor at this point, to keep working and send emails if things came up, but we mostly kept working independently from this point on.

Then, in the middle of march, we set up a video-call with the supervisor and the other group using the video-call tool Jitsi.org. We simply told of our current status and shared general thoughts on the project so far and how we would move on. This video call was when we initially began more extensive communication with the other group with the intention of connecting their back-end and our front-end together. We simply gave them access to a copy of our system which they would try to connect with theirs, and since this could be a difficult task, we decided in plenum to hold a video-meeting of the same nature with all parties present every wednesday from then on. This mainly became our primary meeting-schedule until the very end of the project, where both groups decoupled into mainly working on documentation and further inquiries from the other group were handled casually through discord. The nature of the

Wednesday meetings were casual, and we entered each meeting without much of an agenda other than summarization of the previous week's work and current status. This lack of an agenda or a proper structure came about because neither parties initially took command over the meetings and so the responsibility was mostly mutual. This type of flat meeting-structure will surely contribute to meetings being a bit slower and more unorganized than regular meetings, but since what we discussed was not extremely comprehensive and we generally knew each other's roles and objectives, the meetings were still productive, especially through the use of screen sharing-functionality that allows source-code and front-end work to be very easily navigated and displayed to everyone in the meeting. There is no doubt that video-call tools are an invaluable resource in times like these.

This casual approach also explains the lack of proper meeting-documentation from these video-calls like meeting inquiries and meeting-summaries as they were not long or comprehensive meetings and we mostly noted the important things locally. The things mentioned in the meetings were also mostly technical matters that the groups handled through discord afterwards, and were not products of the meetings itself.

The project being split

We believe that there was one very impactful factor in this project that affected the final products of both us and the other group. This is briefly discussed earlier in this chapter under the UX and Responsivity section, but the issue mainly concerns the way both groups ended up working on this project. This is a somewhat all-encompassing issue that takes both groups working on this product into account and might not directly relate to our bachelor, or it might not be our business to comment on the outcome of the other groups work but we believe what we learned concerning this issue is important to remember in the future.

The issue is the one that arises when groups responsible for two different parts of a product work too disjointed from one another and with too little communication and dependence on each other. This was especially true for our case, where the goal was to eradicate old faults and create a new, better system. We can discuss exactly why it happened later, but we must first talk briefly of what consequences this issue brought with it for our project.

One of the main reasons that we were given this project to begin with was because the old system was slow and riddled with bugs and security holes. When we began work on reimplementing the system we slowly began to understand that a lot of problems, bugs and unwanted came not from simple mistakes easily fixable, but because of the way the whole thing was architectured and wired together. The database and the server had a lot of weird, inefficient SQL-queries and endpoints, and we realized that the front-end was also buggy and inefficient not only because of bad front-end programming, but because a lot of inefficient implementations had been made to fit with the already peculiar choices that were made on the server-side. The reality was that the system as a whole consisted of many inconsistent, buggy and inefficient qualities as a result of the relationship between the front-end and the back-end. You rarely found a bad implementation that was not reflected in both the front-end and the back-end and things were tightly connected, especially since they had built the client and the server as common entities with the same express-instance.

With this knowledge, knowing that the project was split into two separate bachelors might not sound the optimal choice for improving the product and it turns out that it indeed was not. For most of the project, we(the two groups involved) worked entirely separated and we were concerned only with our respective parts of the product, in this case front-end and back-end. In practice, this meant that with the knowledge that the two final products had to be compatible and would eventually be connected, both groups made sure to make as few breaking changes as possible. Still, some breaking changes had to be made regardless of consequence as some things were just too inexcusable to include in the new product, and we as front-end devs actually made a decent amount of these changes considering design-changes had to be made. Despite our great wishes to make changes, we had to be conservative in order to reduce the amount of work needed to make the two parts compatible and reduce the amount of extra work the other group had to spend rewriting.

The situation described above resulted in the new system still containing a lot of the faults and inefficiencies that existed in the old system. Although we tried to strike a balance in order to change as much as possible, there was a lot that was kept the way it was because the system simply wouldn't have functioned together with the other groups back-end otherwise. If you find yourself studying our source code and thinking "That was an odd and inefficient workaround", it most likely is exactly that, an odd and inefficient workaround because the back-end is odd and inefficient. The other group didn't change much either because they probably thought as we did, that compatibility was crucial, and so rewrote an odd and inefficient back-end into an equally odd and inefficient back-end, only in a different language.

The absolute best approach to improving this system would have been to rewrite and develop the backend and frontend simultaneously and together, and when faults and oddities were discovered, it would have been immediately improved both in the frontend and the backend. Compatibility wouldn't have been an issue as both parts would have been created with the same information and the same agreements on what to change and how to change it. This would most likely have resulted in the system being even better than the final end-result of this project is. The final product we created is absolutely better than the old one in many ways, and especially the design, but what lies under the design and the UI could probably have been optimized even further together with the backend.

The reasons for the situation becoming like this and the project being developed in separate parts is unknown to us, but we know that the project originally was a single project but became split somewhere along the way and we only knew that we suddenly were responsible for the B-part of the project instead of the whole project as it was originally shown. A lot of factors probably forced the project into becoming split and we cannot argue with that, but we can give a reason as to why it stayed split, even though we present you with our knowledge of the system and the situation.

The fact is that such a detailed understanding of the system and its implementations presents itself only after working a great deal with the system. In our initial evaluation of the system, we looked at it in a more abstract way, how it was structured, the nature of each component etc. and did not study specific implementations too hard as this would have taken a long time and would not have contributed to our thoughts on what technology suited it best very much, and so wouldn't have answered our thesis. It was when we had already worked with the system and reimplemented large portions of it that we realized that the system, and especially the database and the server actually had such a great number of oddities that they should have been tended to collaboratively. By this time, both groups had done so much work that there was no time to begin the extensive process of redesigning the server, the database and the relations of the system.

Therefore, when we look back at the beginning of the project with the knowledge we possess now, our informed decision on what we should do with the system should not only have concerned which technology to use, but also that the whole system, and especially the database, should have been redesigned and recreated with both groups collaborating and we should have worked together with the other group much more tightly than we did and it is a shame that we did not realize this earlier. We think that logistic reasons were probably the biggest reasons that this project was split into two, and maybe we should have motioned harder for a more collaborative process and that this might be on us, but individual bachelors are entitled to their own tasks and their own work and so we mainly focused on our own work as well wanted to make our part of the product as good as possible.

As stated earlier, our product still ended up really good and introduced great improvements from the old system, but seeing how this kind of split development affected what workarounds and assumptions we had to make and how both groups' concerns with being compatible with the other group made us weary of breaking changes made us learn a lot about how we should organize work on larger projects, how collaboration and communication is really important for a project to become the best it can be. The split development might have been exceptionally impactful when we worked on an already existing project and our aim was to improve it, which meant that two groups had different versions of the same system and had to ensure that they were compatible while still improving the system, but collaboration is undoubtedly important when creating products from scratch as well.

Working in a group

There are several qualities that come with working in a group. This widely depends on what the group is tasked to achieve but more importantly who the members of the team are. Some of these qualities are negative to the work process and product but can be minimized if handled in a proper manner. The first part of this is to analyze what working in a group means when it comes to this project. The question was whether to assign different roles to each person to try to separate work or try to collaborate as much as possible. Close collaboration gives the benefit of concurrency when it comes to design choices, work methods and review. The cost is that this takes a lot of extra time and there might be duplication of work which means the planning of each part will take longer. On the other hand if you split all tasks and give each person sole responsibility then it will remove a lot of need for extra communication and overhead information. The downside now being that different parts of the project might be created on different terms. The way this project was structured featured a hybrid solution where each task was given a owner who was given the main responsibility but the planning and review being done together. This was done for most tasks.

There is also an importance of utilizing the strengths of the different team members. Upon agreement some major aspects of the project such as the graphical design choices were allocated to a single person as planning and review from the other person gave little value. This also removed the necessity to align different ideas and production.

One major challenge in many group projects is to manage expectations and create an environment in which all team members are comfortable. The proposed and used solution was to create a work agreement. The work agreement was rarely referred to in the working process and not used but it is a safety for ensuring that conflicts that could potentially arise would be possible to handle.

Professional ethics

The product developed in this project is a new version of a product that is already in use. As described before, the purpose of the application is to provide a service where students get their assignments approved. The usage of the information about who has what assignment approved is crucial to the way the school handles validation of subjects and in some cases decides whether or not a student can participate the subject exam(s). This is one of the only, but still immensely important, reasons why the product produced and the developers behind it must be wary of the ethical problems that lie within the system service and what could potentially cause these.

One of the main problems to look out for is the validation of the approval. For example, the system needs to be accurate when generating CSV files of the approval lists because these might later be use to automate the pipeline between the database holding the information and the school's organ responsible for passing students. The database also needs to be updated correctly and contain information only set through conscious and authorized means. Unintentional editing or hacking of the system could potentially lead to wrong data being used to evaluate students. There's also a question of if a user inputted mistake is made, who is then responsible? Is it the developers who made a faulty user interface or the user who did something they shouldn't have.

There is no surefire way to securely use a system. There is always a possibility where something goes wrong but there are ways to reduce the chance. The potential cost of a system failure could be fatal when it comes to the education of an individual. The best way to prevent such a failure is to test the system thoroughly. Due to the circumstances it has not been possible to test the system to an extent that is acceptable. As such it is very import to include the state of the stability of the product when documenting and to underline that if the system is to be used it should be notable care and under close surveillance.

Comment on sources

If one studies our collection of sources and references that we base most of our research on concerning the different web-development technologies, one surely notices that a large portion of them are blogs. This is no mere coincidence or a lack of effort in trying to find scientific papers of more credible nature, but it is simply a result of what the field of web-development consists of and how people share their knowledge.

Blogs are a very modern way of conveying information, opinions and knowledge and web-development is considered a very modern and hip field of study. We have stated earlier that technologies and trends change very rapidly in the field, and the community therefore tends to value a medium of knowledge-sharing that is equally able to adapt to rapid change and trends. This medium must be very easy to consume and very fast to write and blogs fit this description perfectly. Knowledge on these technologies are primarily spread using blogs and finding credible, extensive scientific works conducted on web-technologies is very difficult as the technologies simply vary so much and change so rapidly that investing countless ours in research papers and review-articles that are invalidated or irrelevant 2 years from now simply isn't very attractive to a great number of people.

That is why we primarily draw data and research from blogs. We are aware that this format is much more subjective to people sharing their own thoughts, opinions and biases and objectivity becomes harder to find, but blogs still reflect the most updated opinions and knowledge found in the web-community and will often be more helpful than outdated articles, even if they require more critical reading and one can expect more hours sifting through countless blogs having slightly different views and opinions on the same technology.

6 Conclusions and further work

Our thesis consists of different parts which we have tried to answer to the best of our abilities throughout the work we have done and in the thoughts and discussions we've shared in this report. We've managed to reach some meaningful conclusions about the questions posed by the thesis based on what we have discovered which we will present in this chapter.

The primary scientific endeavour of our work was to compare, analyze and make a choice on which current web-development technology was best suited for the system in question. If you've read the rest of the report, you will know that our final choice of technology landed on Facebook's View-library React. The reason we chose this framework is explained in greater detail in chapter 4 and 5, but the main goal was for us to choose a technology that would be able to answer the part of the thesis which refers to the qualities we wanted to achieve in the new system. The vision document describes these qualities in further detail, but our main goal was to achieve an application that was lighter, faster, more user-friendly, more responsive and more secure than the old system.

Based on the data collected by DevTools that displays the application's performance in several different areas, we can definitely conclude that we were able to create an application that was significantly lighter and faster than the old system. It showed to be much smaller in size, require significantly fewer requests per load and rendered 4-5 times quicker than the old system. It is important to note that even if we can conclude that React was a good choice of technology, one must also point out that React was not solely responsible for the application being as light and fast as it is.

As React is a library only responsible for the View-part of an application, one needs to employ several different external technologies to make it work. We talked about webpack earlier and how it offers great tools for bundling and compressing your application to decrease its size, and its usefulness cannot be overstated. The workflow-benefits of having hot-reload and compilation when used together with the babel-compiler are also extremely helpful. Setting up a webpack-configuration can be a lot of work as there is an immense amount of configuration that can be applied, and knowing what is useful for you application and avoiding unnecessary overhead or incompatible configurations is important, but if you set some time aside to make it work, we definitely recommend anyone attempting web-development to learn and utilize webpack together with almost any kind of chosen technology.

Still, React was the technology that mainly provided us with the ability to make the application fast, with the use of JSX to manipulate the Virtual DOM and providing a design-pattern that enforces compact code and structure that removes a lot of slow communication between components and unnecessary separation of responsibilities that can lead to cluttered and slow code.

Despite React's power, we also learned that implementation is just as important as choosing the right technology. Our presented research and the sources that we were able to amass on the subject will surely leave you with somewhat of the same impression as us, that the field of web-development is very cluttered with technologies that are difficult to rate above or below each other and that in the end, almost any technology can achieve anything, just not necessarily as perfectly as another, but still at an acceptable level. Although AngularJS is an exceptionally slow and outdated framework, the reason the old system was slow and unoptimized also relied heavily on the previous developers' implementations. They could have avoided a lot of unnecessary weight if they had not relied so much on using the frameworks functionalities to achieve trivial tasks. We believe that if one had enough knowledge and experience on how to use any of the technologies of React, Vue or Angular efficiently one could probably have made the Qs-system work pretty well in either technology. Almost all the technologies provide the same core-features needed to create an application and there exists great applications out there written in all three technologies. When choosing a technology for your own project, we advise you to make it a delicate process where you focus on correctly identifying specific needs and wishes for your application and look for the technology that satisfies these needs. You shouldn't try to answer the question of "Which technology is best for making an application" as any technology can do that, but rather choose the technology that can achieve your specific goals the best. In our case, we chose to make low weight and fast speed our primary goals with regards to making it very mobile-friendly for students and looked for a technology that fit this bill. We might have had a more difficult time adding advanced functionality and spent more time writing our own components like dropdowns, searching etc. than we would have in technologies like angular, but because of this, we in turn managed to achieve our main priorities of weight and speed at a very satisfying level and that result is the most important result. Our conclusion is that defining which technology is the overall best is probably impossible, but if you carefully define exactly what makes your app unique and what it requires in order to be the best at what it is intended for, you can find the technology that answers those specific questions better than others, and we feel that React absolutely answered our requirements for the application defined by the thesis and the vision-document the best out of all the technologies.

We can definitely conclude that our choice of technology made us able to create a responsive application as well, as all of the application is fully usable on almost all devices. The part that is more difficult to answer is if we successfully created a well-designed and user-friendly user-interface using our chosen technology. The quality of a user-interface is often a more intangible and difficult quality to measure and requires real input from real users. Although tools like lighthouse can measure the tangible parts of a user-interface, saying whether or not it is actually friendly, feels good to use, feels good to look at, feels rewarding to use are all subjective parameters that can not be measured by two developers stuck inside their own apartments for 3 months straight. We do definitely believe that we were able to create a user-friendly and good UI based on our own standards and sparse input collected from a few users and our supervisor, but we must conclude that this part of the requirement and the thesis cannot be answered confidently before a lot more testing is conducted and feedback is processed.

We can make further conclusions and recommendations on the process involving the development of a web-application, and especially improving on an old system. As described in greater detail in chapter 5, we believe that we gained enough insight to be able to conclude that work on a system, and especially work that aims to improve and eradicate bugs, security-holes, faults and bad practises from a product should absolutely be done collaboratively and in synch, in order to eradicate things completely, from both front-end and back-end, instead of the fear of incompatibility between work leading to bad workarounds still having to be made along the way.

As for process, we can conclude that we were able to lead a good-enough work-practise and plan that allowed us to develop a good product in the end, but we have definitely learned a lot on organizing work and which challenges arise in doing so. We found that striking a balance between lean and non-lean development is key. Although agility is modern developers' favorite word, and indeed it is extremely powerful for ensuring that time is not spent on unnecessary documentation, still having some process-artifacts in place to not make things too loose and unorganized is very important. We found that our process got more and more unhinged the longer we went into development.

Managing everything in the beginning was easy, but the more complex the system got, which resulted in more time needed for each new addition of code, the harder it became to stay aligned with the Gantt and with our Scrum-logs. Our takeaway is that the smaller a project is, the more agile one can be, but when the system reaches a certain level of complexity relative to the number of people working on it, process-tools and meta-work needs to be taken a little more seriously. We feel that we striked this balance of agility relative to system complexity right up until the end, where the system's complexity outweighed our process, resulting in more unorganized work than we were comfortable with.

In such a situation, a work-agreement and mutual trust becomes important as we rely heavier on people's own sense of responsibility to do work when the process doesn't account for it any longer. We therefore say from our experience that a work-agreement is really convenient to have in place early. This ensures that everyone is on board with the same kind of mindset and relationship with the work-ethics and what to expect, which makes people more able to work independently and a greater trust is built. We advise anyone, no matter the number of people or the circumstances, to write at least something resembling a work-agreement, even if it feels like overkill, because it will surely only be beneficial.

As a final reflection, we can conclude that the thesis's requirements for choosing a fitting technology and using it to create an application that aligns with the vision document and greatly improves on the old system were met at a satisfactory level. We must conclude that there is uncertainty surrounding the design and the stability of the application because of factors described in chapter 5, but we believe that some further work that we discuss below will resolve this and that not much is required to bring the system to an acceptable state, and be left with a very pleasing product.

As for further work relating to our project and the system in question, there are several areas in which other developers definitely could contribute that would greatly benefit the application.

Firstly, there are very new technologies and coding-practises being added all the time, which is also true for React. One very influential, new technology that is being practised in parts of react-development is the introduction of Hooks. Hooks are basically ways of declaring View-components in React that differ from the classic way of declaring them as classes. We primarily use class-declaration in our project, as it is the most common and is the technology we have the most experience with, but Hooks are proven to be quite powerful in certain situations. Hooks are basically React-components written as functions instead of classes. They allow you to access states and other React-features without the need for a class. Their main purpose is to simplify code even further, and make Reacts already compact component-architecture even more compact. Hooks will definitely be the most powerful for declaring small components that have little to no complex logic related to them and are mostly static. Classes might still be the best way of creating larger components, as the risk of components becoming too compact and unreadable when using hooks is present. Still, React-Hooks are absolutely worth looking into and seeing if parts of our application, especially the smallest and most trivial components, can be written as functions instead of classes to further reduce the applications weight and complexity can be considered valid further work.(23)

Another, smaller argument in favor of React over the other technologies that might not play a huge role now but can be relevant in the future, is Facebook's React Native technology. React-Native is a powerful library created for developing mobile-applications. It's syntax is very similar to React's syntax and it adopts a lot of the same ideas and techniques that React does. We think that if there ever arises a wish for the employer to expand the project into the mobile-kingdom even further, which we think is a valid wish to have seeing as we have designed our application with the knowledge and assumption that people wish easy and fast access to the application as to not be intrusive during day-to-day activities and work, converting the application to mobile-app will be much easier from React to React-Native with the relationship they have, than it would have been with other technologies. Further work might therefore consist of researching, considering and if so, developing a mobile-version of the app using React Native.(24)

Some more immediate further work to be done that is actually very important with regards to deployment of the application, is ensuring that the application is tested heavily. Since we were not able to get nearly enough testing done as we initially wanted and never got to utilize the feedback-button or stress-test on large amounts of users as described in chapter 5, we are not able to give this project over to the supervisor with the level of confidence that we would like to have in a finished project. Seeing that the system gets adequately tested and robustness is ensured before deployment of the application to user-environments is immensely important for work that should be conducted on the system. The fact that we must deliver a product that is in need of this extra work in order for the systems robustness to be guaranteed is indeed saddening, but the situation concerning the virus which we ended up in simply did not allow for the process to yield any other result.

7 References

[1] **Author:** Ekta Goel,
Title: *Software Framework vs Library*,
Link: `https://www.geeksforgeeks.org/software-framework-vs-library/` [Last accessed: May 2020]

[2] **Author:** SpiderWriting,
Title: *Static v Dynamic Website Design*,
Link: `https://www.spiderwriting.co.uk/static-dynamic.php` [Last accessed: May 2020]

[3] **Author:** bitsofcode,
Title: *Understanding the Virtual DOM*,
Link: `https://bitsofco.de/understanding-the-virtual-dom/` [Last accessed: May 2020]

[4] **Author:** tkrotoff,
Title: *Front-end frameworks popularity (React, Vue and Angular)*,
Link: `https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190` [Last accessed: May 2020]

[5] **Author:** Nit A,
Title: *Data Binding in Angular*,
Link: `https://dzone.com/articles/data-binding-in-angular` [Last accessed: May 2020]

[6] **Author:** openclassrooms,
Title: *Understand one-way data bindings*,
Link: `https://openclassrooms.com/en/courses/4286486-build-web-apps-with-reactjs/4286721-understand-one-way-data-bindings` [Last accessed: May 2020]

[7] **Author:** Creators of Vue,
Title: *Data Binding Syntax*,
Link: `https://v1.vuejs.org/guide/syntax.HTML` [Last accessed: May 2020]

[8] **Author:** Okta,
Title: *MVC in an Angular World*,
Link: `https://scotch.io/tutorials/mvc-in-an-angular-world` [Last accessed: May 2020]

[9] **Author:** Google,
Title: *Introduction to components and templates*,
Link: `https://angular.io/guide/architecture-components` [Last accessed: May 2020]

[10] **Author:** Facebook,
Title: *Components and Props*,
Link: `https://reactjs.org/docs/components-and-props.HTML` [Last accessed: May 2020]

[11] **Author:** Creators of Vue,
Title: *Single File Components*,
Link: `https://vuejs.org/v2/guide/single-file-components.HTML` [Last accessed: May 2020]

[12] **Author:** Google,
Title: *Documentation*,
Link: `https://www.TypeScriptlang.org/docs/home.HTML` [Last accessed: May 2020]

[13] **Author:** Facebook,
Title: *Documentation*,
Link: `https://flow.org/en/docs/` [Last accessed: May 2020]

[14] **Author:** Facebook,
Title: *Introducing JSX*,
Link: `https://reactjs.org/docs/introducing-jsx.HTML` [Last accessed: May 2020]

[15] **Author:** Tomas Holas,
Title: *Angular vs. React: Which Is Better for Web Development?*,
Link: `https://www.toptal.com/front-end/angular-vs-react-for-web-development` [Last accessed: May 2020]

[16] **Author:** Creators of Vue,
Title: *Vue homepage*,
Link: `https://vuejs.org/` [Last accessed: May 2020]

[17] **Author:** Lokesh Sardana ,
Title: *Understanding DOM manipulation in Angular*,
Link: `https://medium.com/@sardanalokesh/understanding-dom-manipulation-in-angular-2b0016a4ee5d` [Last accessed: May 2020]

[18] **Author:** Facebook,
Title: *Virtual DOM and Internals*,
Link: `https://reactjs.org/docs/faq-internals.HTML` [Last accessed: May 2020]

[19] **Author:** Creators of Vue,
Title: *Render Functions and JSX*,
Link: `https://vuejs.org/v2/guide/render-function.HTML` [Last accessed: May 2020]

[20] **Author:** Facebook,
Title: *Updating to New Releases*,
Link: `https://create-react-app.dev/docs/updating-to-new-releases/` [Last accessed: May 2020]

[21] **Author:** mdbootstrap,
Title: *Vue 3 - Release date, tutorials, latest news*,
Link: `https://mdbootstrap.com/vue-3/` [Last accessed: May 2020]

[22] **Author:** Google,
Title: *Angular versioning and releases*,
Link: `https://angular.io/guide/releases` [Last accessed: May 2020]

[23] **Author:** Facebook,
Title: *Introduction to Hooks*,
Link: `https://reactjs.org/docs/hooks-intro.HTML` [Last accessed: May 2020]

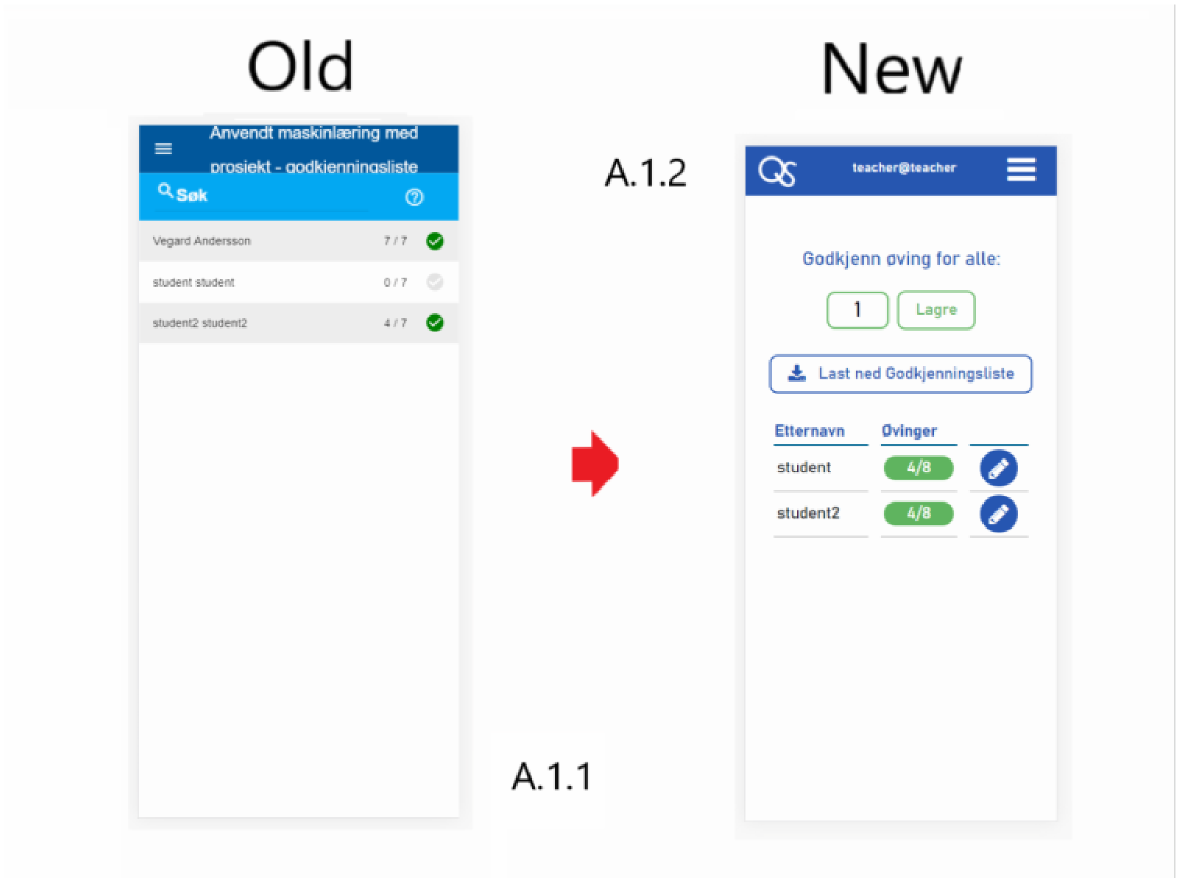
[24] **Author:** Facebook,
Title: *React Native homepage*,
Link: `https://reactnative.dev/` [Last accessed: May 2020]

[25] **Author:** Facebook,
Title: *Hooks FAQ*,
Link: `https://reactjs.org/docs/hooks-faq.HTML#do-i-need-to-rewrite-all-my-class-components` [Last accessed: May 2020]

[26] **Author:** Creators of mockflow,
Link: `https://mockflow.com/`

8 Attachments

A.1



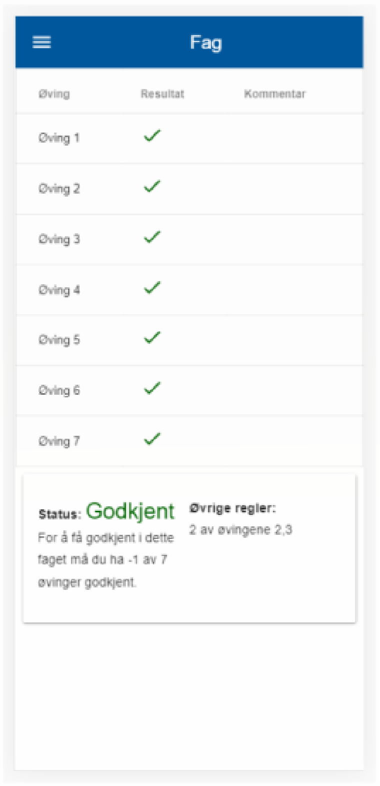
Anvendt maskinlæring med prosjekt - godkjenningsliste

Vegard Andersson	7 / 7	✓
student student	0 / 7	
student2 student2	4 / 7	✓

Godkjenn øving for alle:

1

Etternavn	Øvinger
student	4/8
student2	4/8



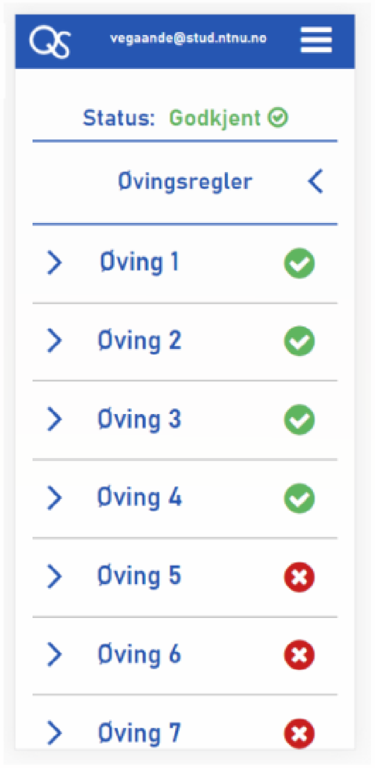
Fag

Øving	Resultat	Kommentar
Øving 1	✓	
Øving 2	✓	
Øving 3	✓	
Øving 4	✓	
Øving 5	✓	
Øving 6	✓	
Øving 7	✓	

Status: Godkjent
For å få godkjent i dette faget må du ha -1 av 7 øvinger godkjent.

Øvrige regler:
2 av øvingene 2,3

A.1.4



vegaande@stud.ntnu.no

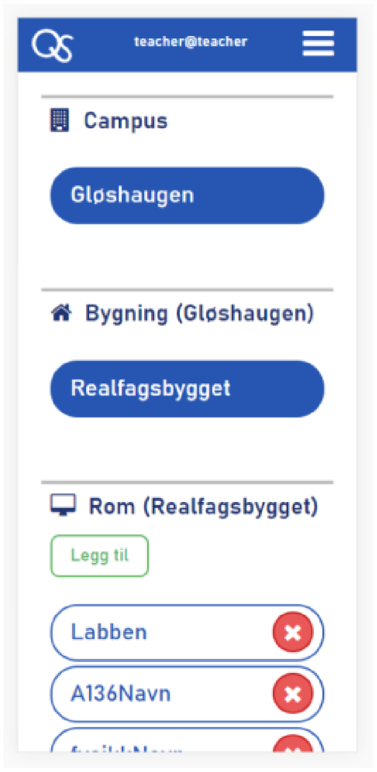
Status: Godkjent

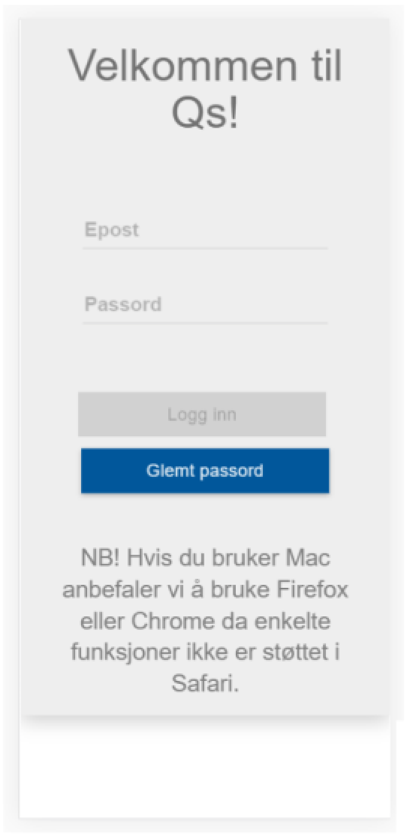
Øvingsregler

Øving 1	✓
Øving 2	✓
Øving 3	✓
Øving 4	✓
Øving 5	✗
Øving 6	✗
Øving 7	✗

A.1.3

A.1.5


teacher@teacher
Campus
Gløshaugen
Bygning (Gløshaugen)
Realfagsbygget
Rom (Realfagsbygget)
Legg til
Labben
A136Navn


Velkommen til Qs!

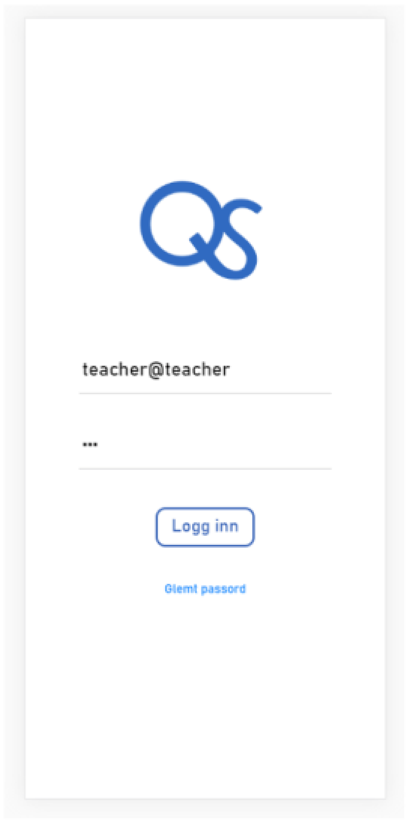
Epost

Passord

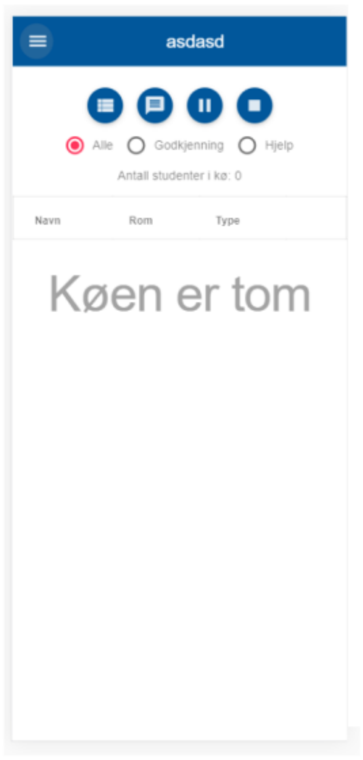
NB! Hvis du bruker Mac anbefaler vi å bruke Firefox eller Chrome da enkelte funksjoner ikke er støttet i Safari.

A.1.6

A.1.7

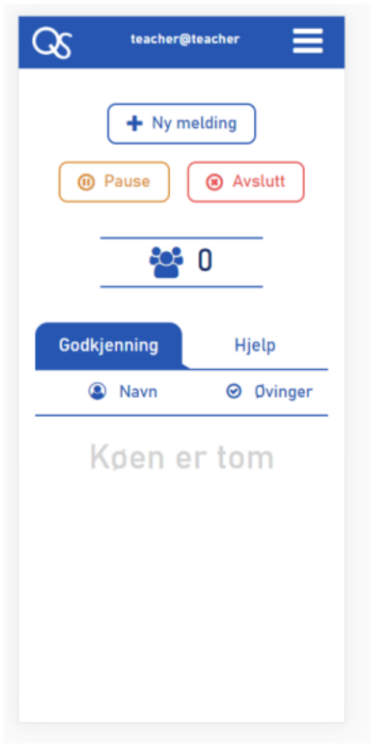

teacher@teacher

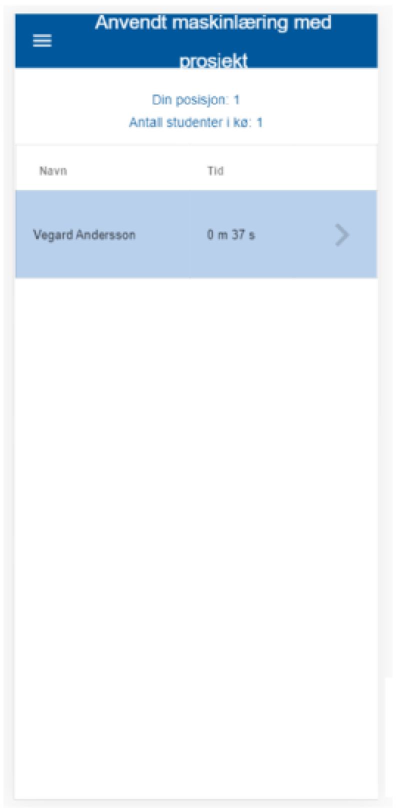
•••


asdasd
Alle
Godkjenning
Hjelp
Antall studenter i kø: 0
Navn
Rom
Type
Køen er tom

A.1.8

A.1.9


teacher@teacher
Ny melding
Pause
Avslutt
0
Godkjenning
Hjelp
Navn
Øvinger
Køen er tom

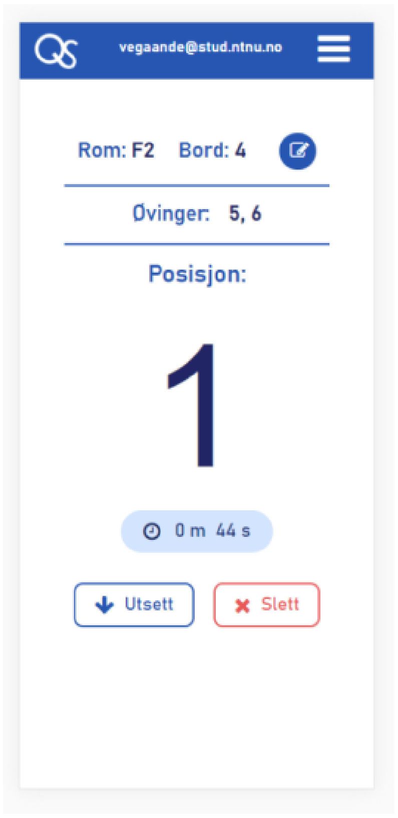


Anvendt maskinlæring med prosjekt

Din posisjon: 1
Antall studenter i kø: 1

Navn	Tid
Vegard Andersson	0 m 37 s

A.1.11



vegaande@stud.ntnu.no

Rom: F2 Bord: 4

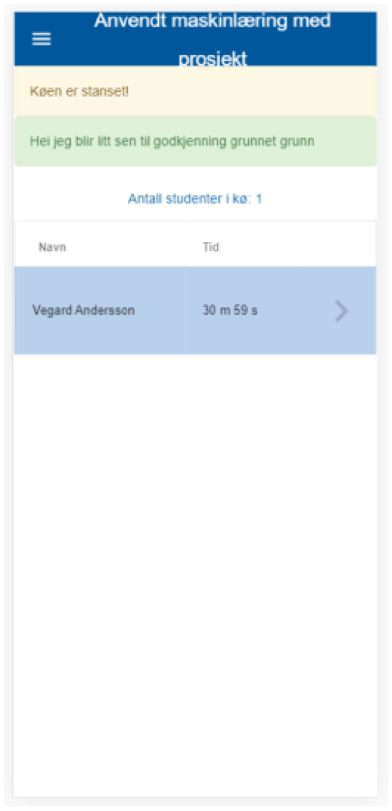
Øvinger: 5, 6

Posisjon:

1

0 m 44 s

A.1.10


Anvendt maskinlæring med prosjekt

Køen er stanset!

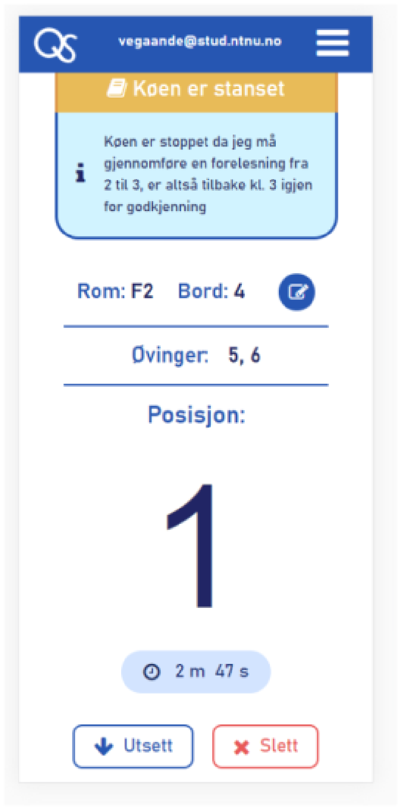
Hei jeg blir litt sen til godkjenning grunnet grunn

Antall studenter i kø: 1

Navn	Tid
Vegard Andersson	30 m 59 s

A.1.12

A.1.13


vegaande@stud.ntnu.no

Køen er stanset

Køen er stoppet da jeg må gjennomføre en forelesning fra 2 til 3, er altså tilbake kl. 3 igjen for godkjenning

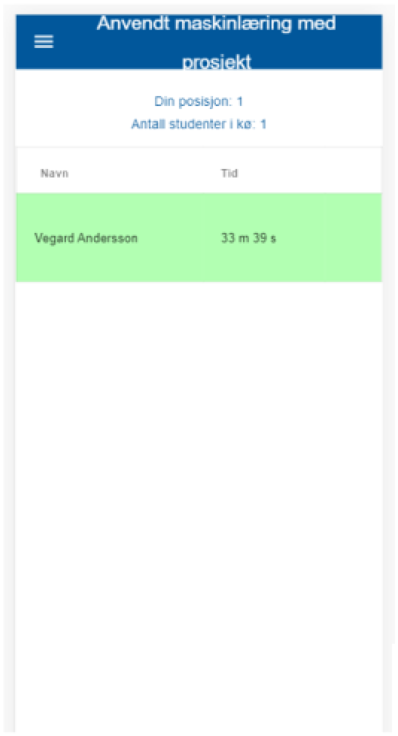
Rom: F2 Bord: 4

Øvinger: 5, 6

Posisjon:

1

2 m 47 s

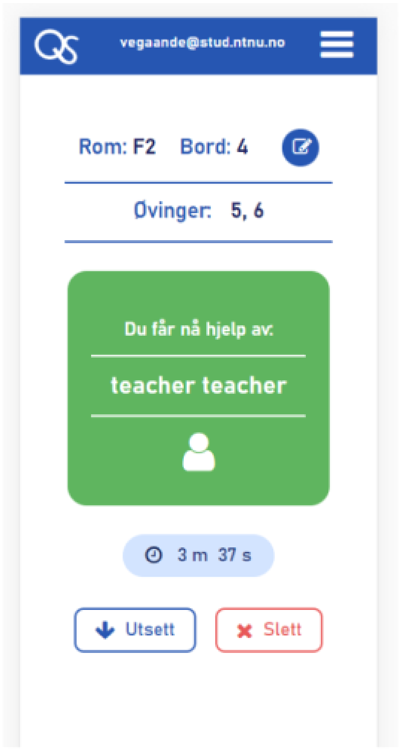


Anvendt maskinlæring med prosjekt

Din posisjon: 1
Antall studenter i kø: 1

Navn	Tid
Vegard Andersson	33 m 39 s

A.1.15



vegaande@stud.ntnu.no

Rom: F2 Bord: 4

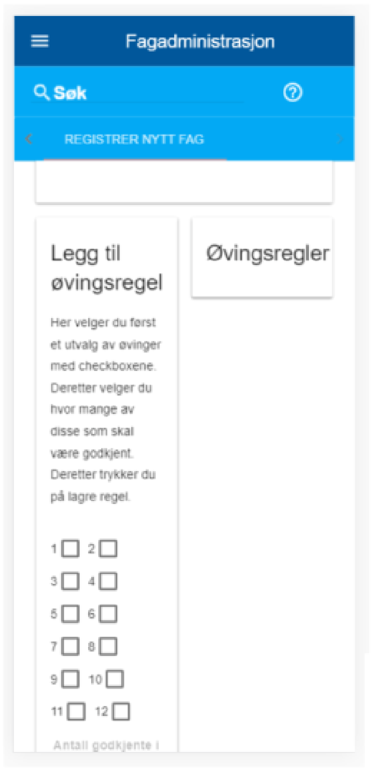
Øvinger: 5, 6

Du får nå hjelp av:

teacher teacher

3 m 37 s

A.1.14


Fagadministrasjon

Søk

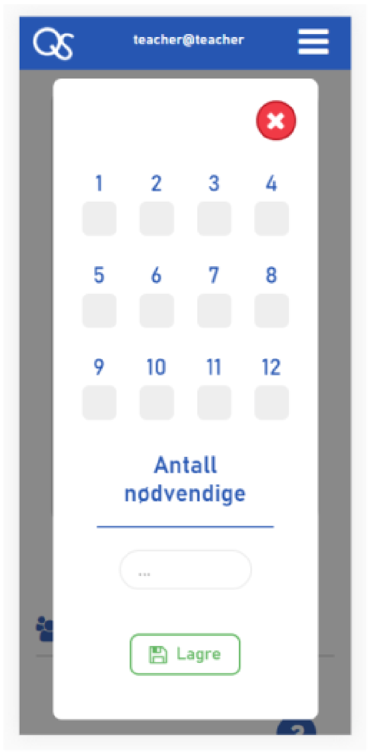
REGISTRER NYTT FAG

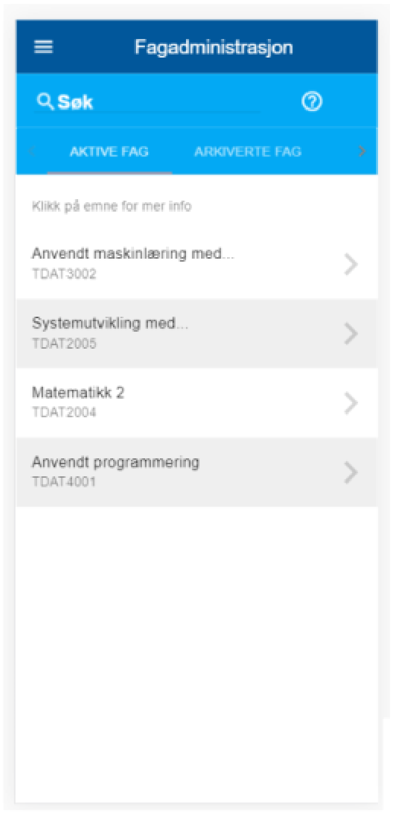
Legg til øvingsregel

Her velger du først et utvalg av øvinger med checkboxene. Deretter velger du hvor mange av disse som skal være godkjent. Deretter trykker du på lagre regel.

A.1.17

A.1.16




Fagadministrasjon

Søk

AKTIVE FAG ARKIVERTE FAG

Klikk på emne for mer info

Anvendt maskinlæring med...
TDAT3002

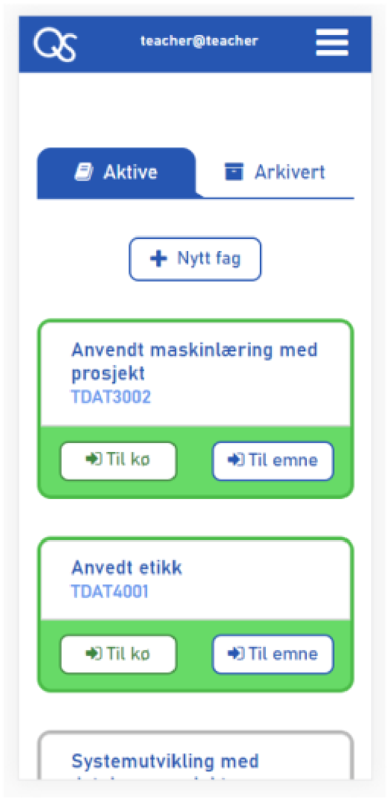
Systemutvikling med...
TDAT2005

Matematikk 2
TDAT2004

Anvendt programmering
TDAT4001

A.1.18

A.1.19


teacher@teacher

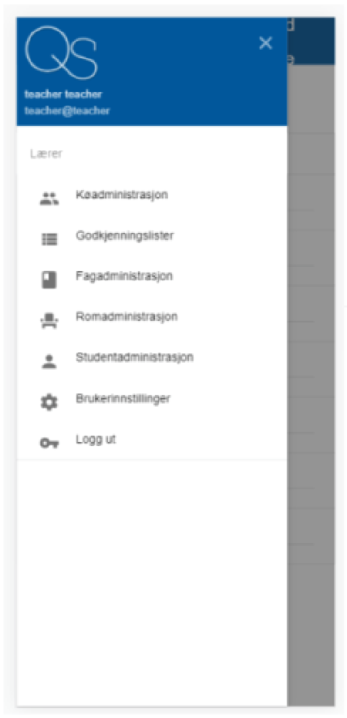
Aktive Arkivert

Nytt fag

Anvendt maskinlæring med prosjekt
TDAT3002
Til kø Til emne

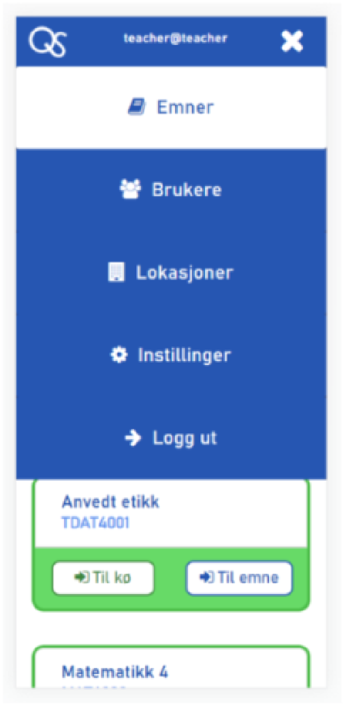
Anvedt etikk
TDAT4001
Til kø Til emne

Systemutvikling med



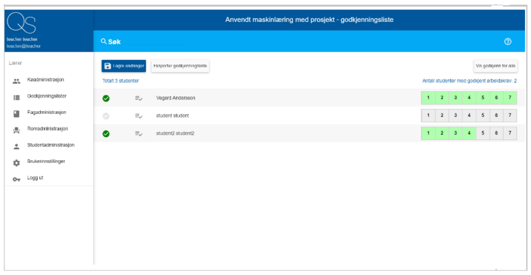
A.1.21

A.1.20



A.2

Old



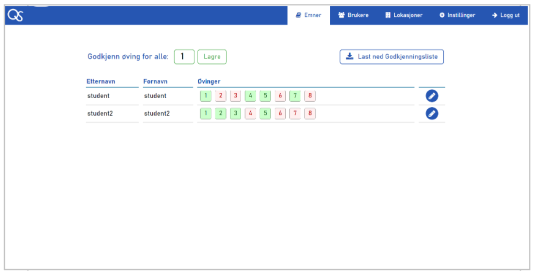
Anvendt maskinlæring med prosjekt - godkjenningsliste

Totalt 3 studenter

Antall studenter med godkjent arbeidskrav: 2

✓	Vegard Andersson	1	2	3	4	5	6	7
	student student	1	2	3	4	5	6	7
✓	student2 student2	1	2	3	4	5	6	7

New

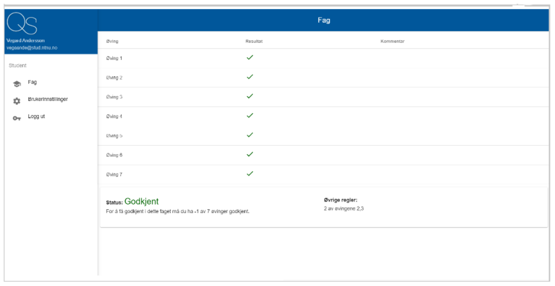


Godkjenn øving for alle: 1

Etternavn	Fornavn	Øvinger							
student	student	1	2	3	4	5	6	7	8
student2	student2	1	2	3	4	5	6	7	8

A.3

Old

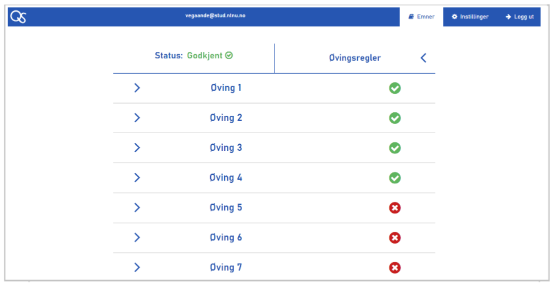


Øving	Resultat	Kommentar
Øving 1	✓	
Øving 2	✓	
Øving 3	✓	
Øving 4	✓	
Øving 5	✓	
Øving 6	✓	
Øving 7	✓	

Status: Godkjent
For å få godkjent i dette faget må du ha -1 av 7 øvinger godkjent.

Øvrige regler:
2 av øvingene 2,3

New



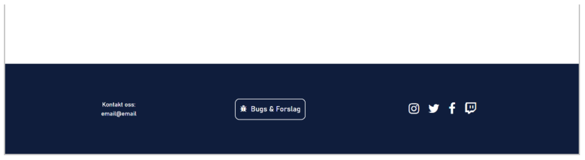
Status: Godkjent ✓ | Øvingsregler

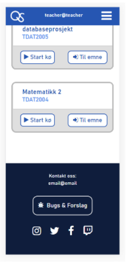
Øving 1	✓
Øving 2	✓
Øving 3	✓
Øving 4	✓
Øving 5	✗
Øving 6	✗
Øving 7	✗

A.4

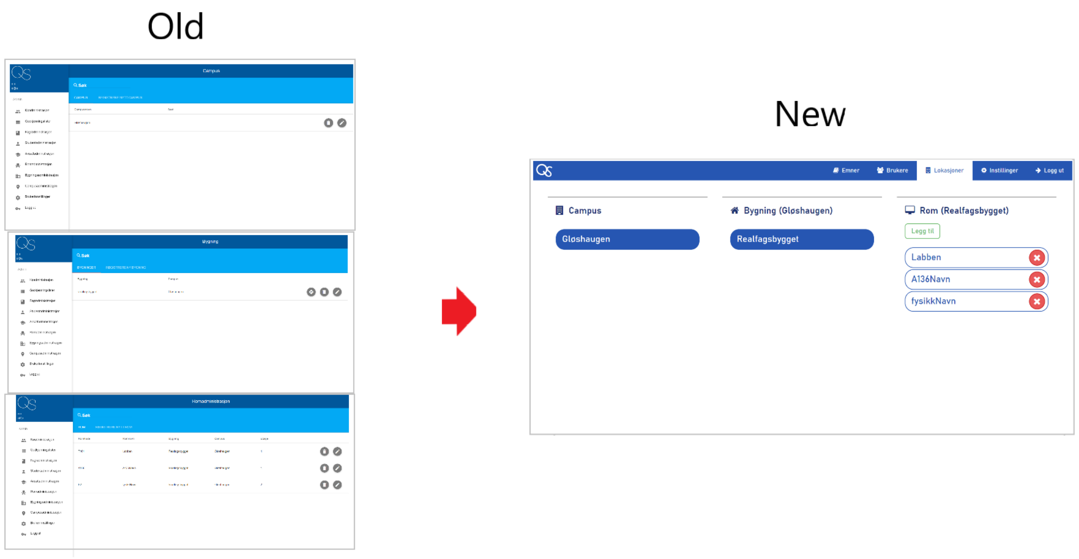
Old

New

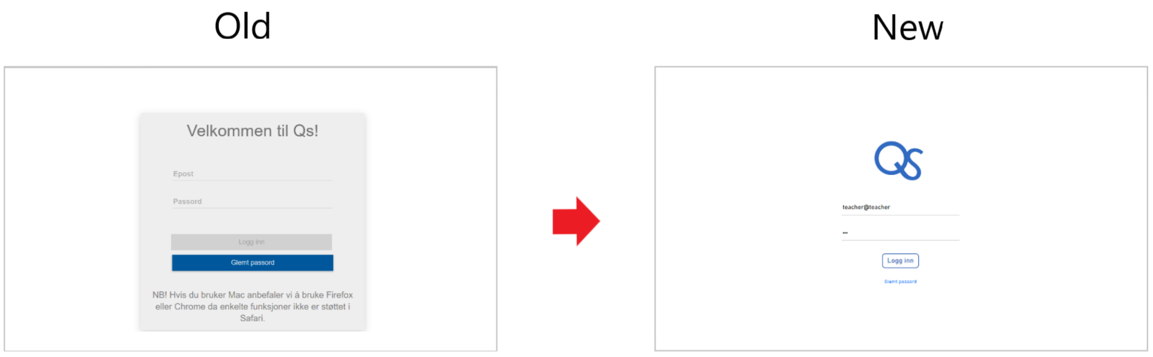




A.5


Old
Campus
Bygning
New
Emner
Brukere
Lokasjoner
Instillinger
Logg ut
Campus
Gløshaugen
Bygning (Gløshaugen)
Realfagsbygget
Rom (Realfagsbygget)
Legg til
Labben
A136Navn
fysikkNavn

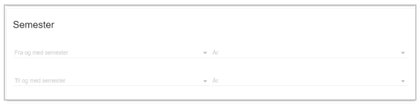
A.6

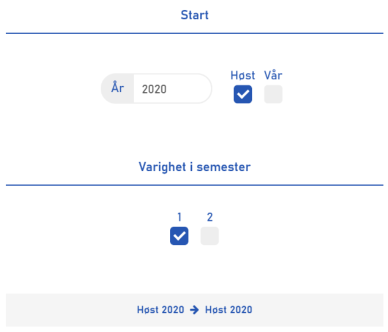

Old
Velkommen til Qs!
Epost
Passord
Logg inn
Glemt passord
NB! Hvis du bruker Mac anbefaler vi å bruke Firefox eller Chrome da enkelte funksjoner ikke er støttet i Safari.
New
teacher@teacher
Logg inn
Glemt passord

A.7

Old

New

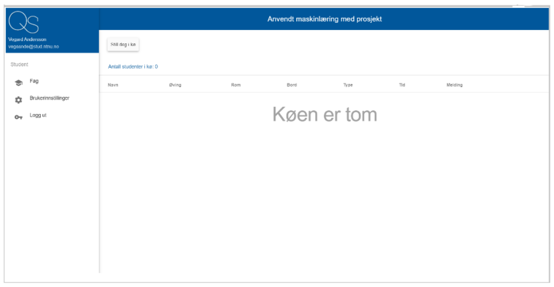


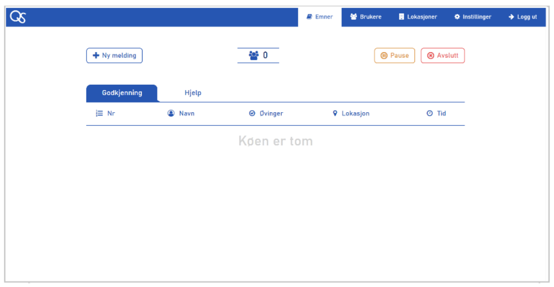


A.8

Old

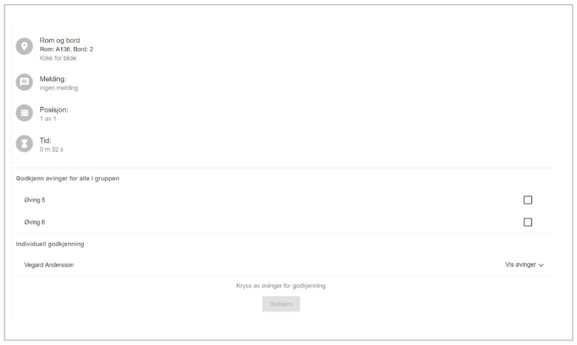
New





A.9

Old



Rom og bord
Rom: A136, Bord: 2
Klikk for bilde

Melding:
ingen melding

Posisjon:
1 av 1

Tid:
0 m 32 s

Godkjenn øvinger for alle i gruppen

Øving 5 ☐

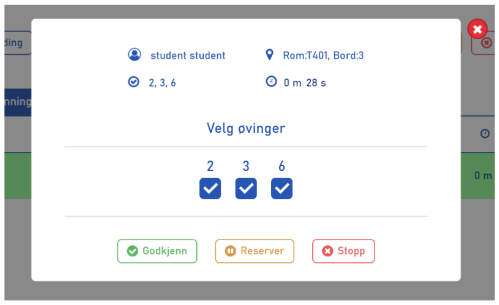
Øving 6 ☐

Individuell godkjenning

Vegard Andersson — Vis øvinger

Kryss av øvinger for godkjenning

New



student student — Rom:T401, Bord:3

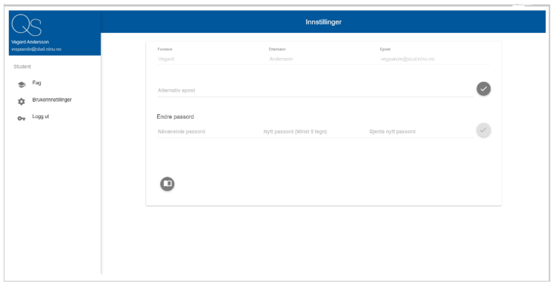
2, 3, 6 — 0 m 28 s

Velg øvinger

2 ✓ 3 ✓ 6 ✓

A.10

Old



Vegard Andersson
vegaande@stud.ntnu.no

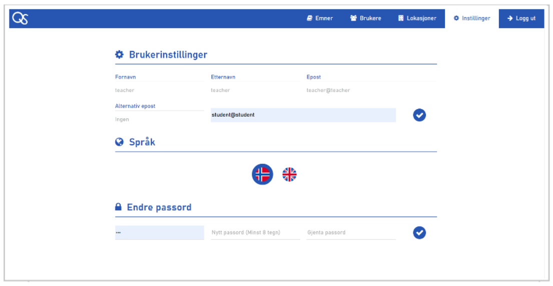
Fornavn	Etternavn	Epost
Vegard	Andersson	vegaande@stud.ntnu.no

Alternativ epost

Endre passord

Nåværende passord — Nytt passord (Minst 8 tegn) — Gjenta nytt passord

New



Brukerinstillinger

Fornavn	Etternavn	Epost
teacher	teacher	teacher@teacher

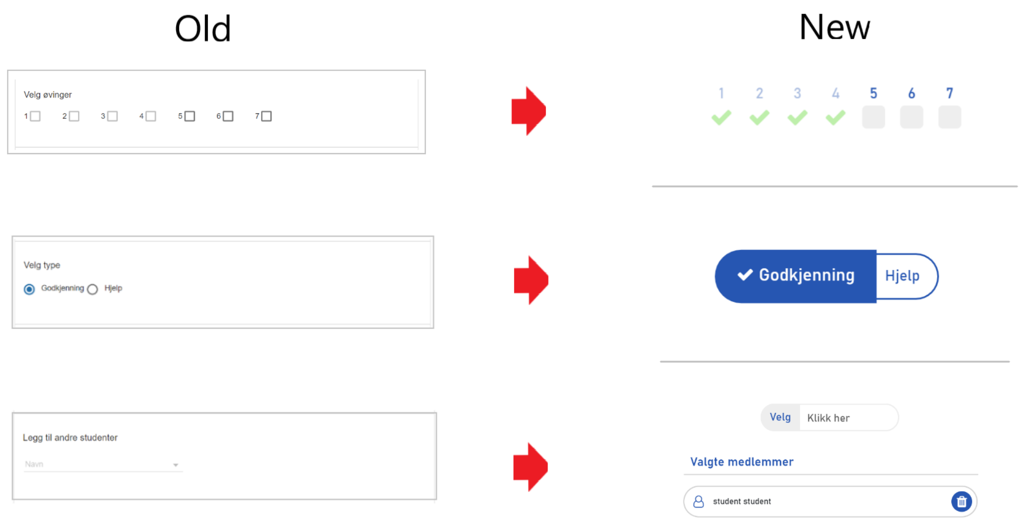
Alternativ epost
Ingen
student@student

Språk

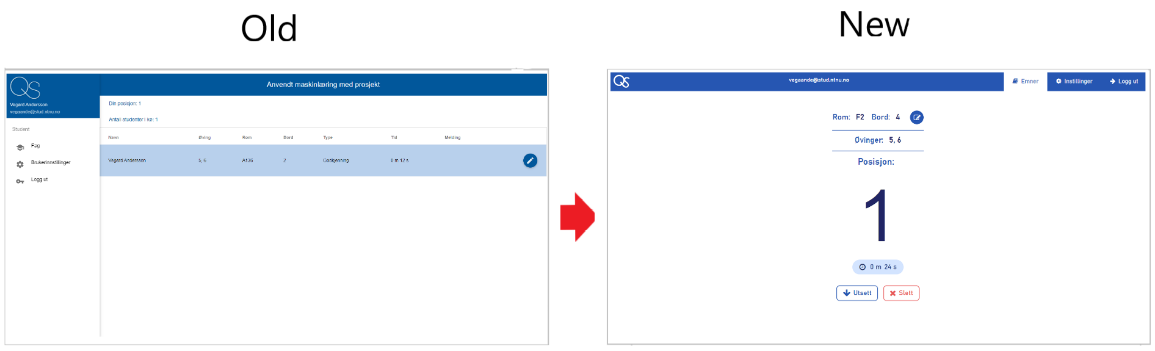
Endre passord

••• — Nytt passord (Minst 8 tegn) — Gjenta passord

A.11

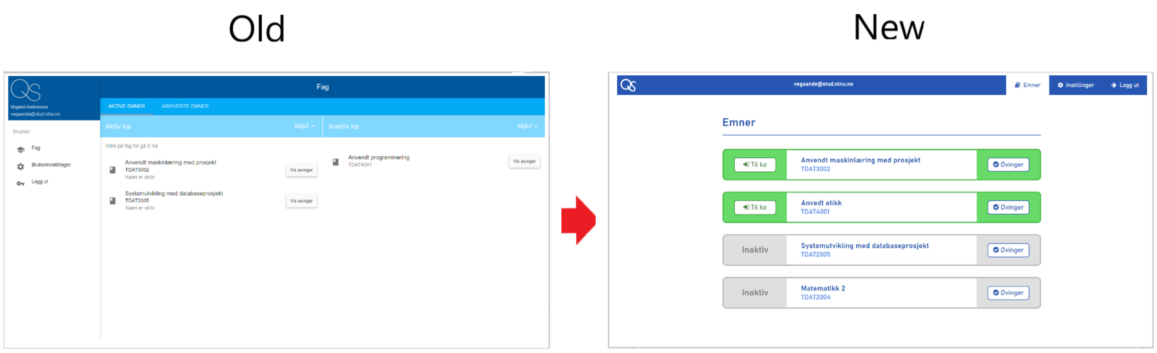


A.12

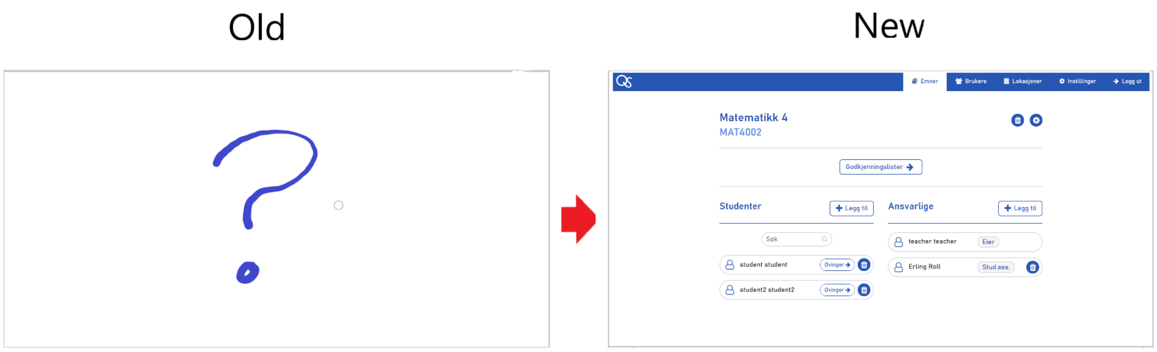


Navn	Øving	Rom	Bord	Type	Tid	Melding
Vegard Andersson	5, 6	A136	2	Godkjenning	0 m 12 s	

A.13

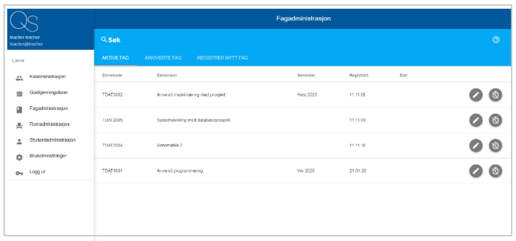

Old
New
Fag
Vegard Andersson
vegaande@stud.ntnu.no
AKTIVE EMNER
ARKIVERTE EMNER
Aktiv kø
skjul
Inaktiv kø
skjul
Student
Fag
Brukerinnstillinger
Logg ut
Klikk på fag for gå til kø
Anvendt maskinlæring med prosjekt
TDAT3002
Køen er aktiv
Vis øvinger
Systemutvikling med databaseprosjekt
TDAT2005
Køen er aktiv
Vis øvinger
Anvendt programmering
TDAT4001
Vis øvinger
vegaande@stud.ntnu.no
Emner
Instillinger
Logg ut
Emner
Til kø
Anvendt maskinlæring med prosjekt
TDAT3002
Øvinger
Til kø
Anvedt etikk
TDAT4001
Øvinger
Inaktiv
Systemutvikling med databaseprosjekt
TDAT2005
Øvinger
Inaktiv
Matematikk 2
TDAT2004
Øvinger

A.14

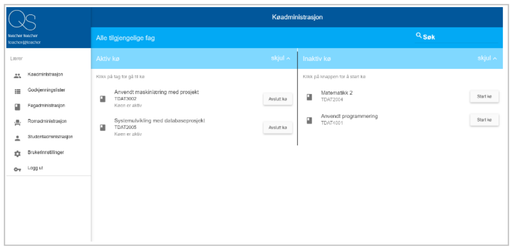

Old
New
Emner
Brukere
Lokasjoner
Instillinger
Logg ut
Matematikk 4
MAT4002
Godkjenningslister
Studenter
Legg til
Ansvarlige
Legg til
Søk
student student
Øvinger
student2 student2
Øvinger
teacher teacher
Eier
Erling Roll
Stud.ass.

A.15

Old



Emnekode	Emnenavn	Semester	Registrert	Eier
TDAT3002	Anvendt maskinlæring med prosjekt	Høst 2023	11.11.08	
TDAT2005	Systemutvikling med databaseprosjekt		11.11.09	
TDAT2004	Matematikk 2		11.11.18	
TDAT4001	Anvendt programmering	Vår 2020	21.04.20	



Aktiv kø

Klikk på fag for gå til kø

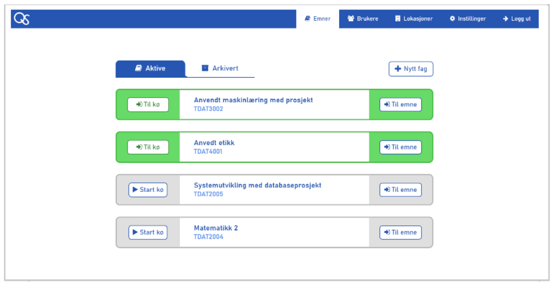
- Anvendt maskinlæring med prosjekt, TDAT3002, Køen er aktiv — Avslutt kø
- Systemutvikling med databaseprosjekt, TDAT2005, Køen er aktiv — Avslutt kø

Inaktiv kø

Klikk på knappen for å start kø

- Matematikk 2, TDAT2004 — Start kø
- Anvendt programmering, TDAT4001 — Start kø

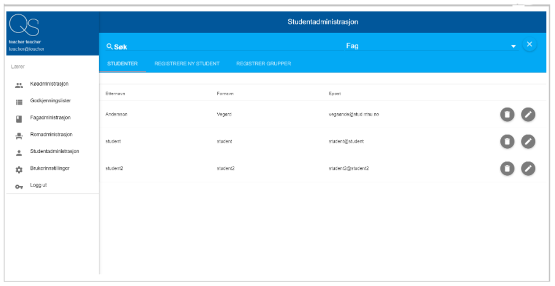
New



Til kø	Anvendt maskinlæring med prosjekt TDAT3002	Til emne
Til kø	Anvedt etikk TDAT4001	Til emne
Start kø	Systemutvikling med databaseprosjekt TDAT2005	Til emne
Start kø	Matematikk 2 TDAT2004	Til emne

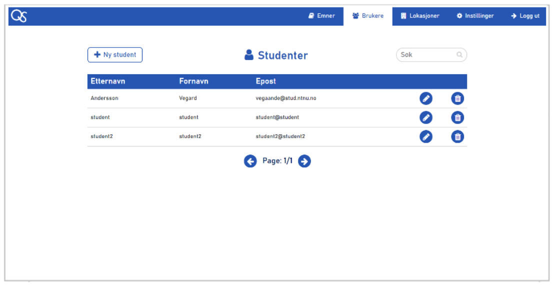
A.16

Old



Etternavn	Fornavn	Epost
Andersson	Vegard	vegaande@stud.ntnu.no
student	student	student@student
student2	student2	student2@student2

New



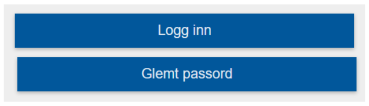
Etternavn	Fornavn	Epost
Andersson	Vegard	vegaande@stud.ntnu.no
student	student	student@student
student2	student2	student2@student2

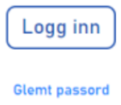
Page: 1/1

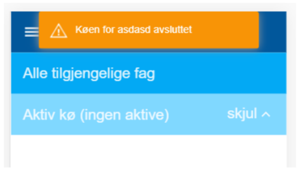
A.17

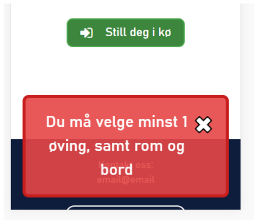
Old

New



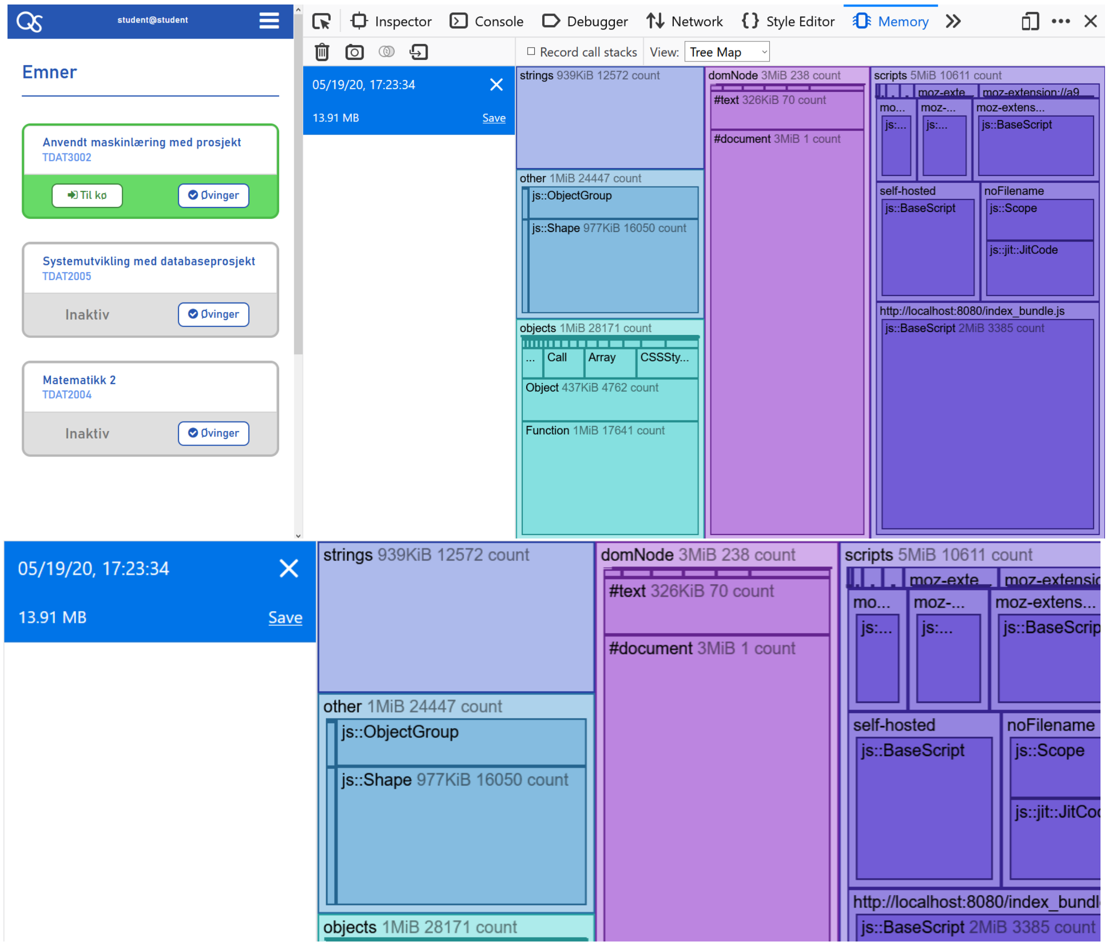



Køen for asdasd avsluttet

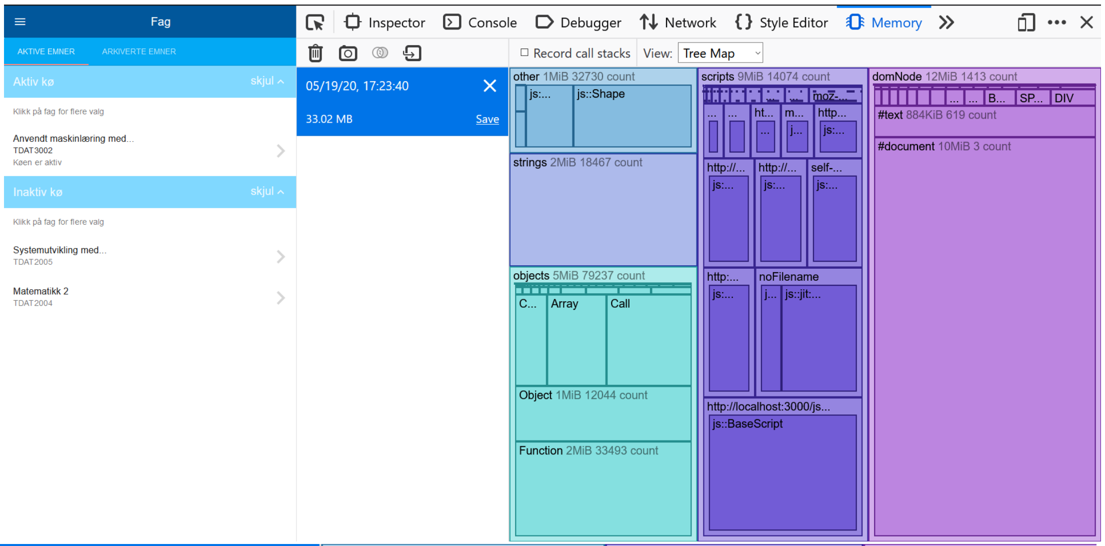


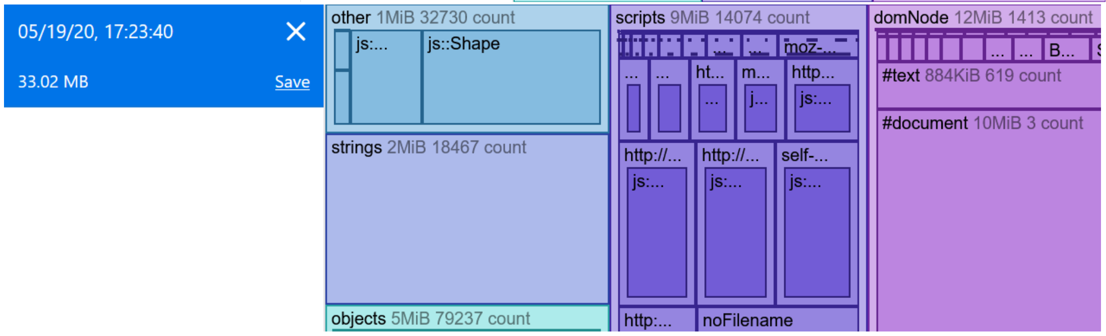
Du må velge minst 1 øving, samt rom og bord

B.1.1

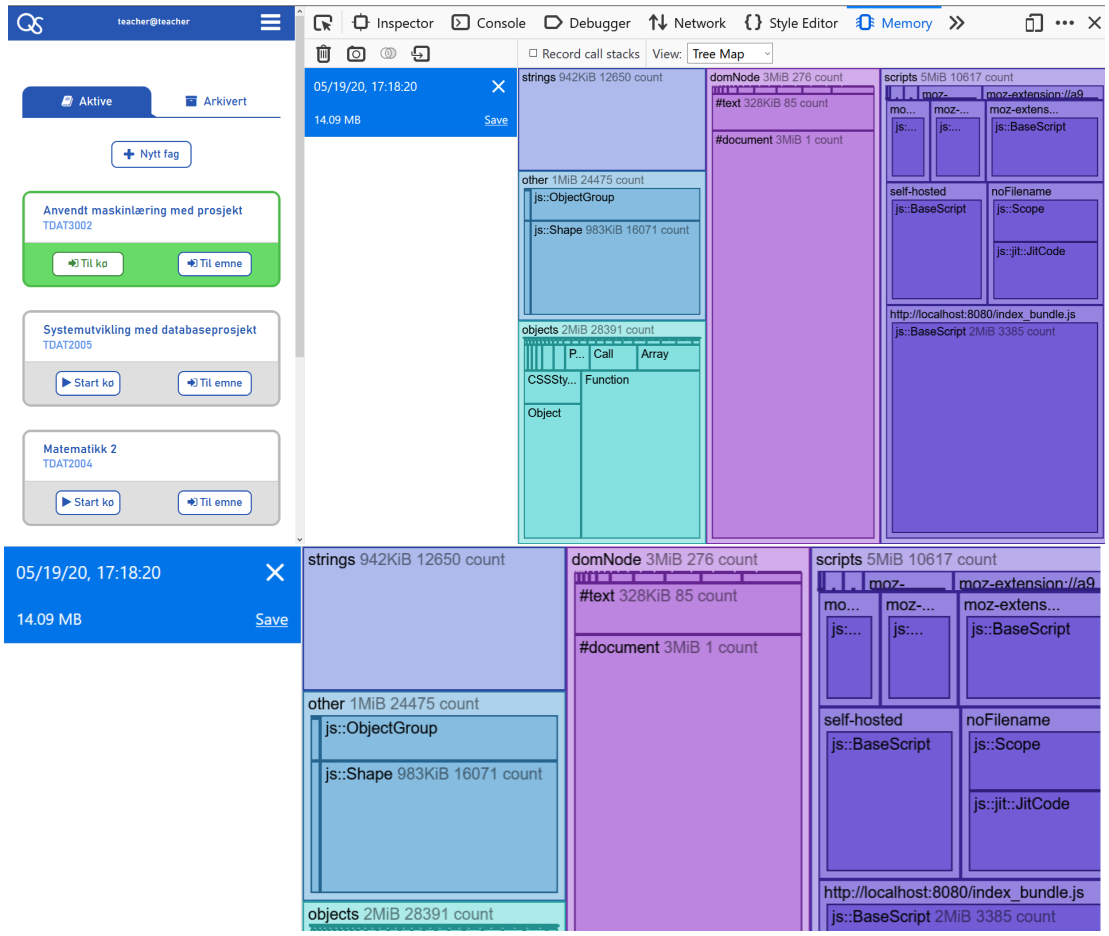

student@student
Emner
Anvendt maskinlæring med prosjekt
TDAT3002
Til kø
Øvinger
Systemutvikling med databaseprosjekt
TDAT2005
Inaktiv
Øvinger
Matematikk 2
TDAT2004
Inaktiv
Øvinger
Inspector
Console
Debugger
Network
Style Editor
Memory
Record call stacks
View: Tree Map
05/19/20, 17:23:34
13.91 MB
Save
strings 939KiB 12572 count
other 1MiB 24447 count
js::ObjectGroup
js::Shape 977KiB 16050 count
objects 1MiB 28171 count
... Call Array CSSSty...
Object 437KiB 4762 count
Function 1MiB 17641 count
domNode 3MiB 238 count
#text 326KiB 70 count
#document 3MiB 1 count
scripts 5MiB 10611 count
moz-exte...
moz-extension://a9...
mo...
moz-...
moz-extens...
js:...
js::BaseScript
self-hosted
js::BaseScript
noFilename
js::Scope
js::jit::JitCode
http://localhost:8080/index_bundle.js
js::BaseScript 2MiB 3385 count

B.1.2

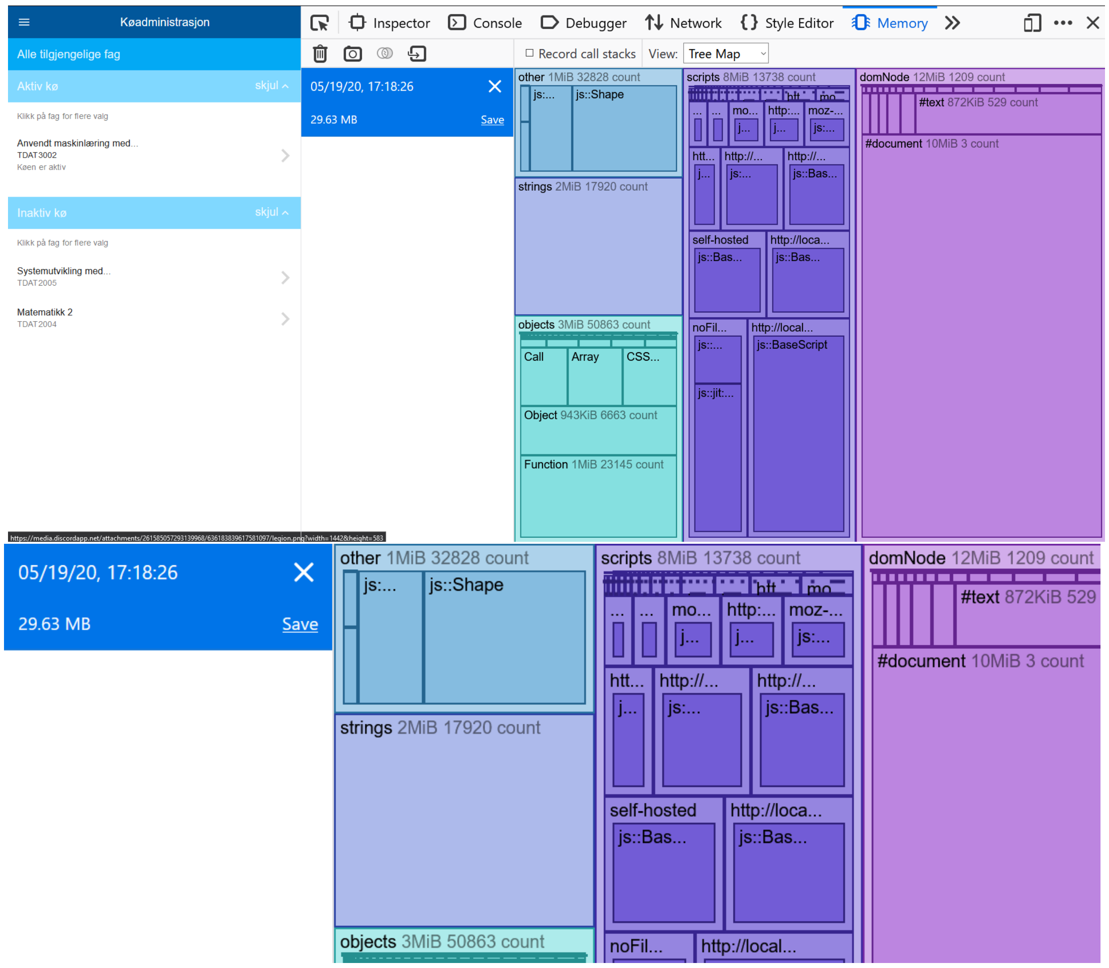

Fag
AKTIVE EMNER
ARKIVERTE EMNER
Aktiv kø
skjul
Klikk på fag for flere valg
Anvendt maskinlæring med...
TDAT3002
Køen er aktiv
Inaktiv kø
skjul
Klikk på fag for flere valg
Systemutvikling med...
TDAT2005
Matematikk 2
TDAT2004
Inspector
Console
Debugger
Network
Style Editor
Memory
Record call stacks
View: Tree Map
05/19/20, 17:23:40
33.02 MB
Save
other 1MiB 32730 count
js:...
js::Shape
strings 2MiB 18467 count
objects 5MiB 79237 count
C...
Array
Call
Object 1MiB 12044 count
Function 2MiB 33493 count
scripts 9MiB 14074 count
moz-...
ht...
m...
http...
j...
js:...
http://...
self-...
http:...
noFilename
js::jit:...
http://localhost:3000/js...
js::BaseScript
domNode 12MiB 1413 count
B...
SP...
DIV
#text 884KiB 619 count
#document 10MiB 3 count


05/19/20, 17:23:40
33.02 MB
Save
other 1MiB 32730 count
js:...
js::Shape
strings 2MiB 18467 count
objects 5MiB 79237 count
scripts 9MiB 14074 count
moz-...
ht...
m...
http...
j...
js:...
http://...
self-...
http:...
noFilename
domNode 12MiB 1413 count
B...
#text 884KiB 619 count
#document 10MiB 3 count

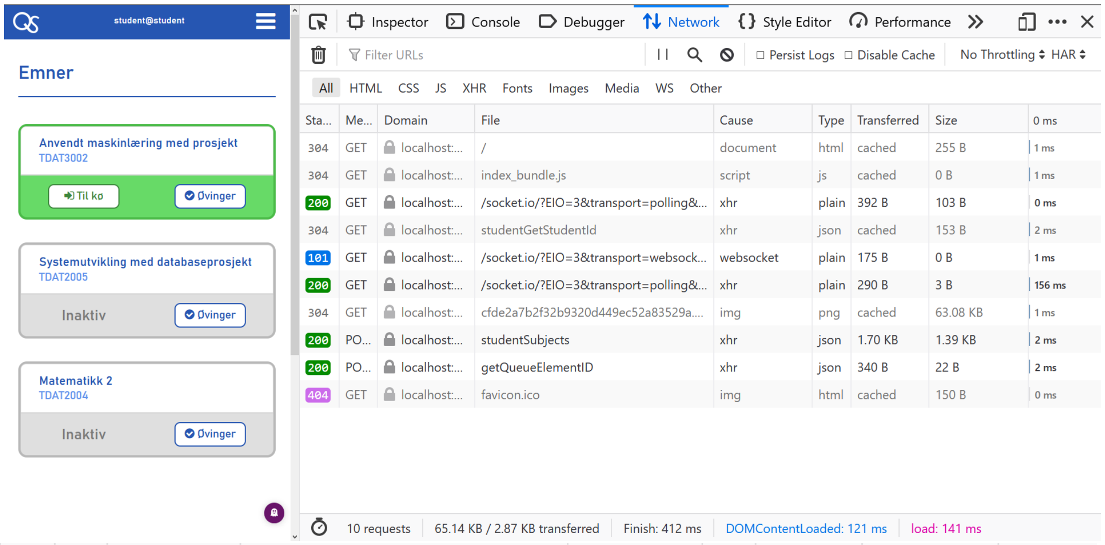
B.2.1


teacher@teacher
Aktive
Arkivert
Nytt fag
Anvendt maskinlæring med prosjekt
TDAT3002
Til kø
Til emne
Systemutvikling med databaseprosjekt
TDAT2005
Start kø
Til emne
Matematikk 2
TDAT2004
Start kø
Til emne
Inspector
Console
Debugger
Network
Style Editor
Memory
Record call stacks
View: Tree Map
05/19/20, 17:18:20
14.09 MB
Save
strings 942KiB 12650 count
other 1MiB 24475 count
js::ObjectGroup
js::Shape 983KiB 16071 count
objects 2MiB 28391 count
P...
Call
Array
CSSSty...
Function
Object
domNode 3MiB 276 count
#text 328KiB 85 count
#document 3MiB 1 count
scripts 5MiB 10617 count
moz-
moz-extension://a9
mo...
moz-...
moz-extens...
js:...
js::BaseScript
self-hosted
noFilename
js::Scope
js::jit::JitCode
http://localhost:8080/index_bundle.js
js::BaseScript 2MiB 3385 count

B.2.2

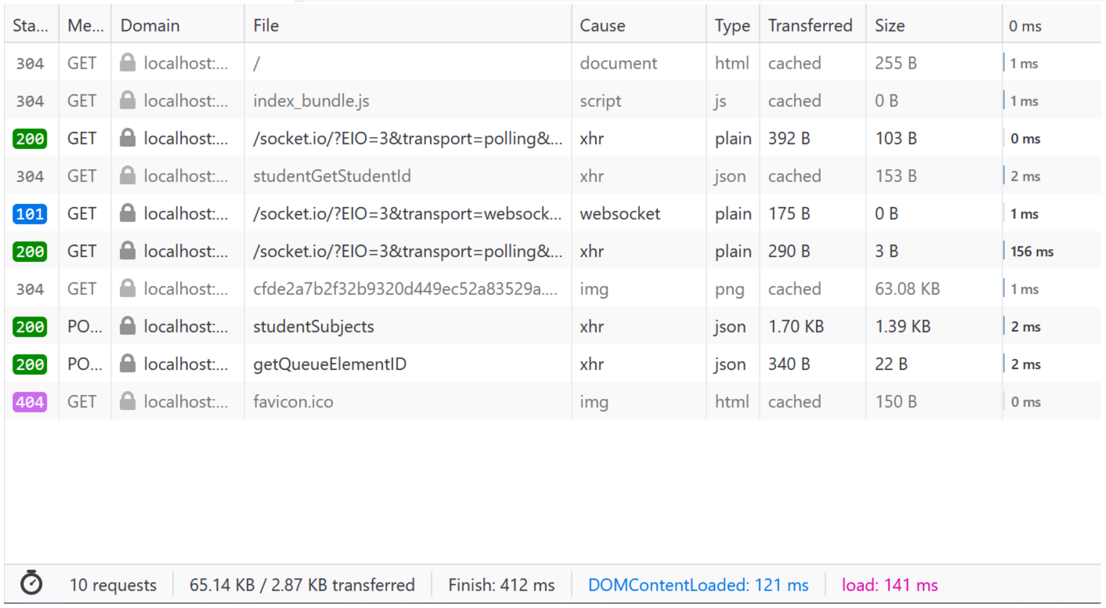

Køadministrasjon
Alle tilgjengelige fag
Aktiv kø skjul
Klikk på fag for flere valg
Anvendt maskinlæring med...
TDAT3002
Køen er aktiv
Inaktiv kø skjul
Klikk på fag for flere valg
Systemutvikling med...
TDAT2005
Matematikk 2
TDAT2004
Inspector
Console
Debugger
Network
Style Editor
Memory
Record call stacks
View: Tree Map
05/19/20, 17:18:26
29.63 MB
Save
other 1MiB 32828 count
js::Shape
strings 2MiB 17920 count
objects 3MiB 50863 count
Call
Array
CSS...
Object 943KiB 6663 count
Function 1MiB 23145 count
scripts 8MiB 13738 count
self-hosted
http://loca...
js::Bas...
noFil...
http://local...
js::BaseScript
js::jit:...
domNode 12MiB 1209 count
#text 872KiB 529 count
#document 10MiB 3 count

B.3.1



Sta...	Me...	Domain	File	Cause	Type	Transferred	Size	0 ms
304	GET	localhost:...	/	document	html	cached	255 B	1 ms
304	GET	localhost:...	index_bundle.js	script	js	cached	0 B	1 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling&...	xhr	plain	392 B	103 B	0 ms
304	GET	localhost:...	studentGetStudentId	xhr	json	cached	153 B	2 ms
101	GET	localhost:...	/socket.io/?EIO=3&transport=websock...	websocket	plain	175 B	0 B	1 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling&...	xhr	plain	290 B	3 B	156 ms
304	GET	localhost:...	cfde2a7b2f32b9320d449ec52a83529a....	img	png	cached	63.08 KB	1 ms
200	PO...	localhost:...	studentSubjects	xhr	json	1.70 KB	1.39 KB	2 ms
200	PO...	localhost:...	getQueueElementID	xhr	json	340 B	22 B	2 ms
404	GET	localhost:...	favicon.ico	img	html	cached	150 B	0 ms

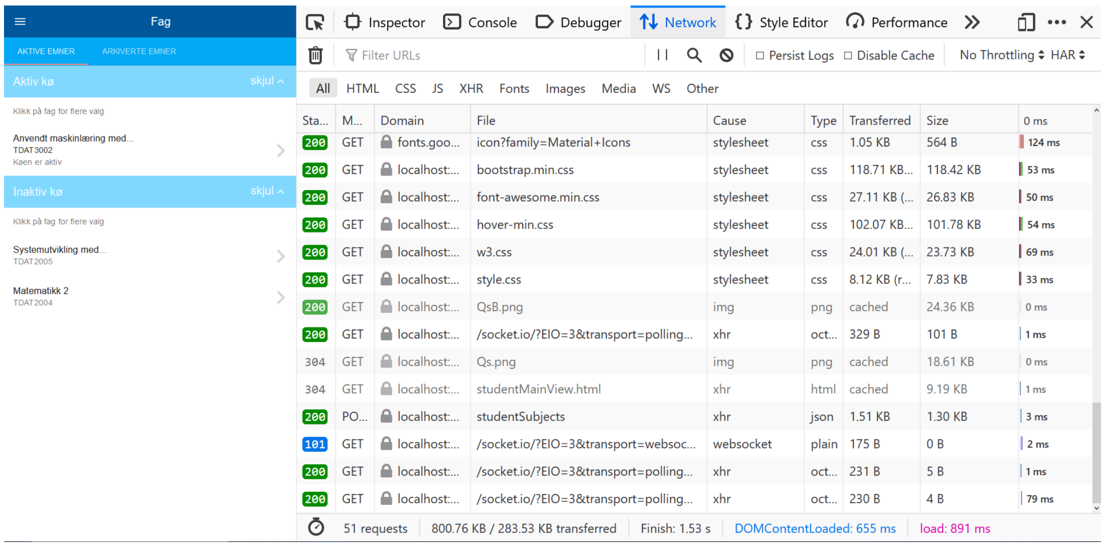
10 requests | 65.14 KB / 2.87 KB transferred | Finish: 412 ms | DOMContentLoaded: 121 ms | load: 141 ms



Sta...	Me...	Domain	File	Cause	Type	Transferred	Size	0 ms
304	GET	localhost:...	/	document	html	cached	255 B	1 ms
304	GET	localhost:...	index_bundle.js	script	js	cached	0 B	1 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling&...	xhr	plain	392 B	103 B	0 ms
304	GET	localhost:...	studentGetStudentId	xhr	json	cached	153 B	2 ms
101	GET	localhost:...	/socket.io/?EIO=3&transport=websock...	websocket	plain	175 B	0 B	1 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling&...	xhr	plain	290 B	3 B	156 ms
304	GET	localhost:...	cfde2a7b2f32b9320d449ec52a83529a....	img	png	cached	63.08 KB	1 ms
200	PO...	localhost:...	studentSubjects	xhr	json	1.70 KB	1.39 KB	2 ms
200	PO...	localhost:...	getQueueElementID	xhr	json	340 B	22 B	2 ms
404	GET	localhost:...	favicon.ico	img	html	cached	150 B	0 ms

10 requests | 65.14 KB / 2.87 KB transferred | Finish: 412 ms | DOMContentLoaded: 121 ms | load: 141 ms

B.3.2



Aktiv kø skjul

Klikk på fag for flere valg

Anvendt maskinlæring med...
TDAT3002
Køen er aktiv

Inaktiv kø skjul

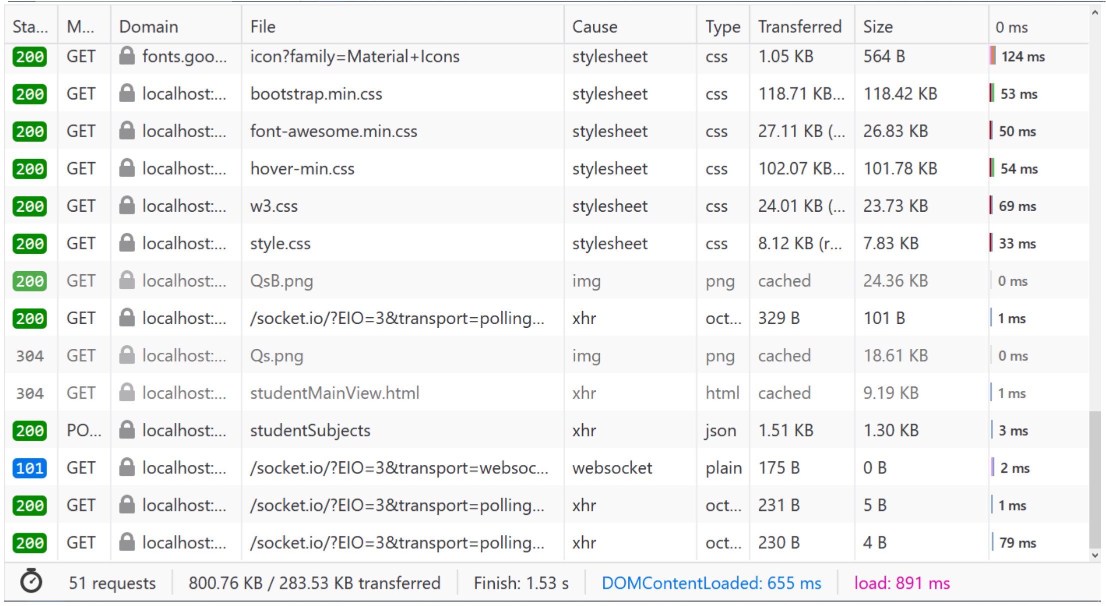
Klikk på fag for flere valg

Systemutvikling med...
TDAT2005

Matematikk 2
TDAT2004

Sta...	M...	Domain	File	Cause	Type	Transferred	Size	0 ms
200	GET	fonts.goo...	icon?family=Material+Icons	stylesheet	css	1.05 KB	564 B	124 ms
200	GET	localhost:...	bootstrap.min.css	stylesheet	css	118.71 KB...	118.42 KB	53 ms
200	GET	localhost:...	font-awesome.min.css	stylesheet	css	27.11 KB (...	26.83 KB	50 ms
200	GET	localhost:...	hover-min.css	stylesheet	css	102.07 KB...	101.78 KB	54 ms
200	GET	localhost:...	w3.css	stylesheet	css	24.01 KB (...	23.73 KB	69 ms
200	GET	localhost:...	style.css	stylesheet	css	8.12 KB (r...	7.83 KB	33 ms
200	GET	localhost:...	QsB.png	img	png	cached	24.36 KB	0 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling...	xhr	oct...	329 B	101 B	1 ms
304	GET	localhost:...	Qs.png	img	png	cached	18.61 KB	0 ms
304	GET	localhost:...	studentMainView.html	xhr	html	cached	9.19 KB	1 ms
200	PO...	localhost:...	studentSubjects	xhr	json	1.51 KB	1.30 KB	3 ms
101	GET	localhost:...	/socket.io/?EIO=3&transport=websoc...	websocket	plain	175 B	0 B	2 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling...	xhr	oct...	231 B	5 B	1 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling...	xhr	oct...	230 B	4 B	79 ms

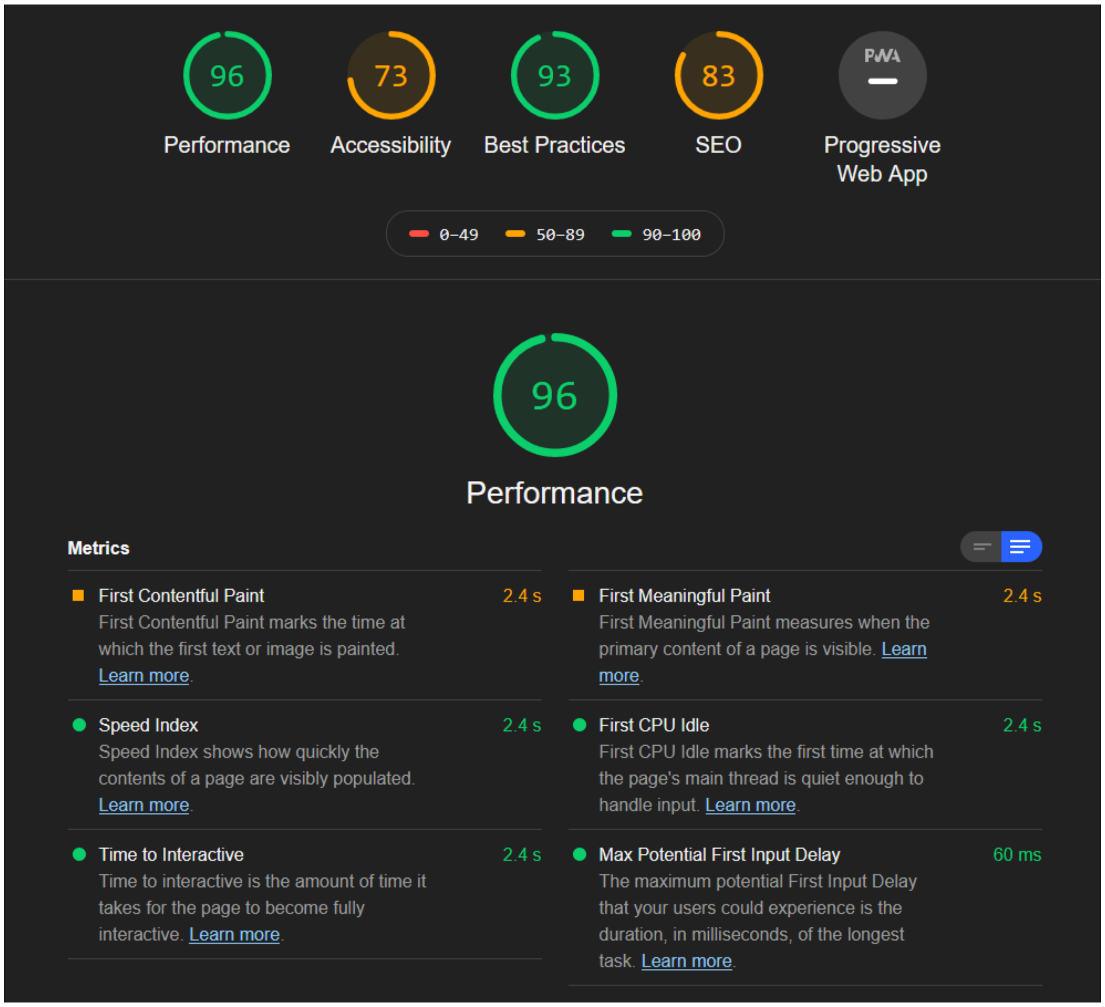
51 requests | 800.76 KB / 283.53 KB transferred | Finish: 1.53 s | DOMContentLoaded: 655 ms | load: 891 ms



Sta...	M...	Domain	File	Cause	Type	Transferred	Size	0 ms
200	GET	fonts.goo...	icon?family=Material+Icons	stylesheet	css	1.05 KB	564 B	124 ms
200	GET	localhost:...	bootstrap.min.css	stylesheet	css	118.71 KB...	118.42 KB	53 ms
200	GET	localhost:...	font-awesome.min.css	stylesheet	css	27.11 KB (...	26.83 KB	50 ms
200	GET	localhost:...	hover-min.css	stylesheet	css	102.07 KB...	101.78 KB	54 ms
200	GET	localhost:...	w3.css	stylesheet	css	24.01 KB (...	23.73 KB	69 ms
200	GET	localhost:...	style.css	stylesheet	css	8.12 KB (r...	7.83 KB	33 ms
200	GET	localhost:...	QsB.png	img	png	cached	24.36 KB	0 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling...	xhr	oct...	329 B	101 B	1 ms
304	GET	localhost:...	Qs.png	img	png	cached	18.61 KB	0 ms
304	GET	localhost:...	studentMainView.html	xhr	html	cached	9.19 KB	1 ms
200	PO...	localhost:...	studentSubjects	xhr	json	1.51 KB	1.30 KB	3 ms
101	GET	localhost:...	/socket.io/?EIO=3&transport=websoc...	websocket	plain	175 B	0 B	2 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling...	xhr	oct...	231 B	5 B	1 ms
200	GET	localhost:...	/socket.io/?EIO=3&transport=polling...	xhr	oct...	230 B	4 B	79 ms

51 requests | 800.76 KB / 283.53 KB transferred | Finish: 1.53 s | DOMContentLoaded: 655 ms | load: 891 ms

B.4.1



Metrics

■ First Contentful Paint — 2.4 s
First Contentful Paint marks the time at which the first text or image is painted. Learn more.

■ First Meaningful Paint — 2.4 s
First Meaningful Paint measures when the primary content of a page is visible. Learn more.

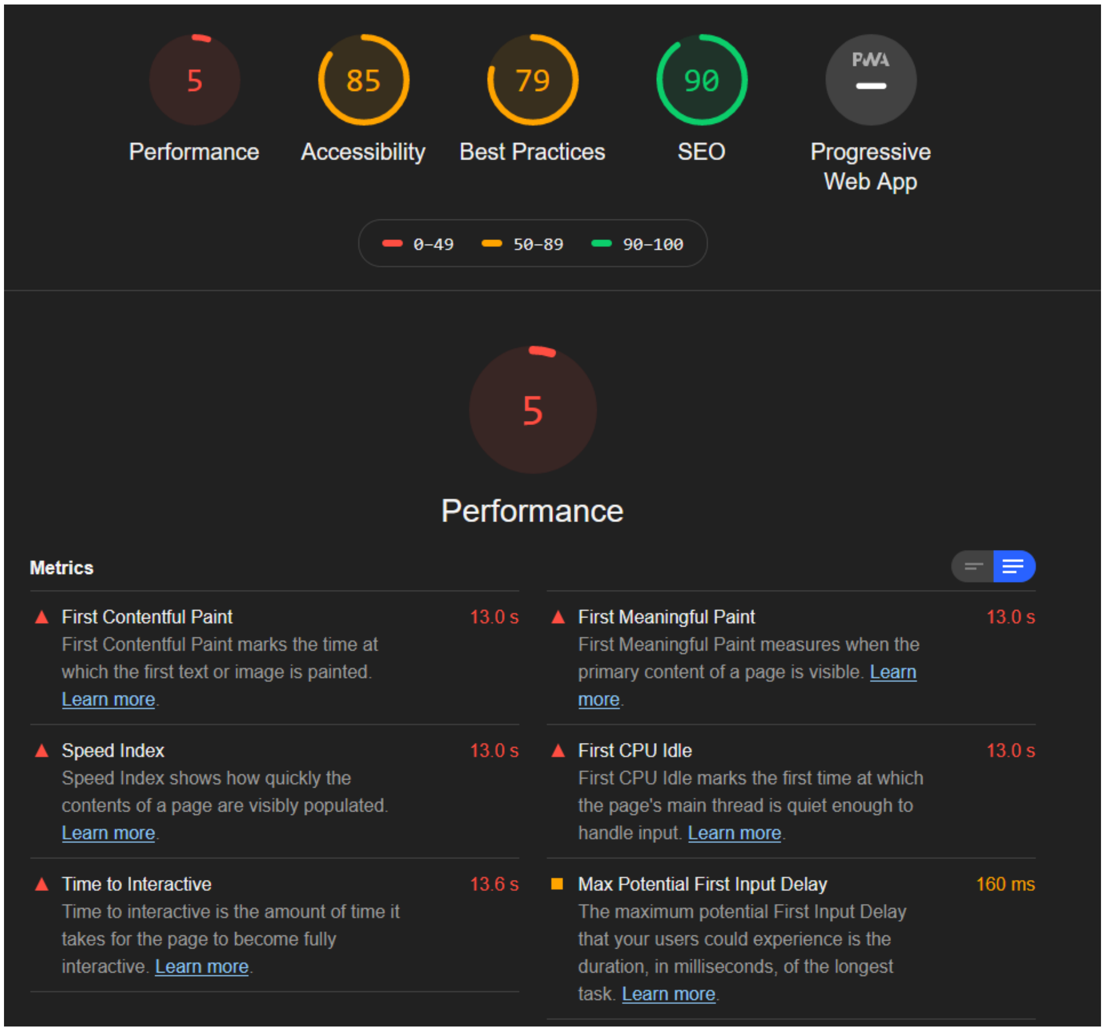
● Speed Index — 2.4 s
Speed Index shows how quickly the contents of a page are visibly populated. Learn more.

● First CPU Idle — 2.4 s
First CPU Idle marks the first time at which the page's main thread is quiet enough to handle input. Learn more.

● Time to Interactive — 2.4 s
Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.

● Max Potential First Input Delay — 60 ms
The maximum potential First Input Delay that your users could experience is the duration, in milliseconds, of the longest task. Learn more.

B.4.2



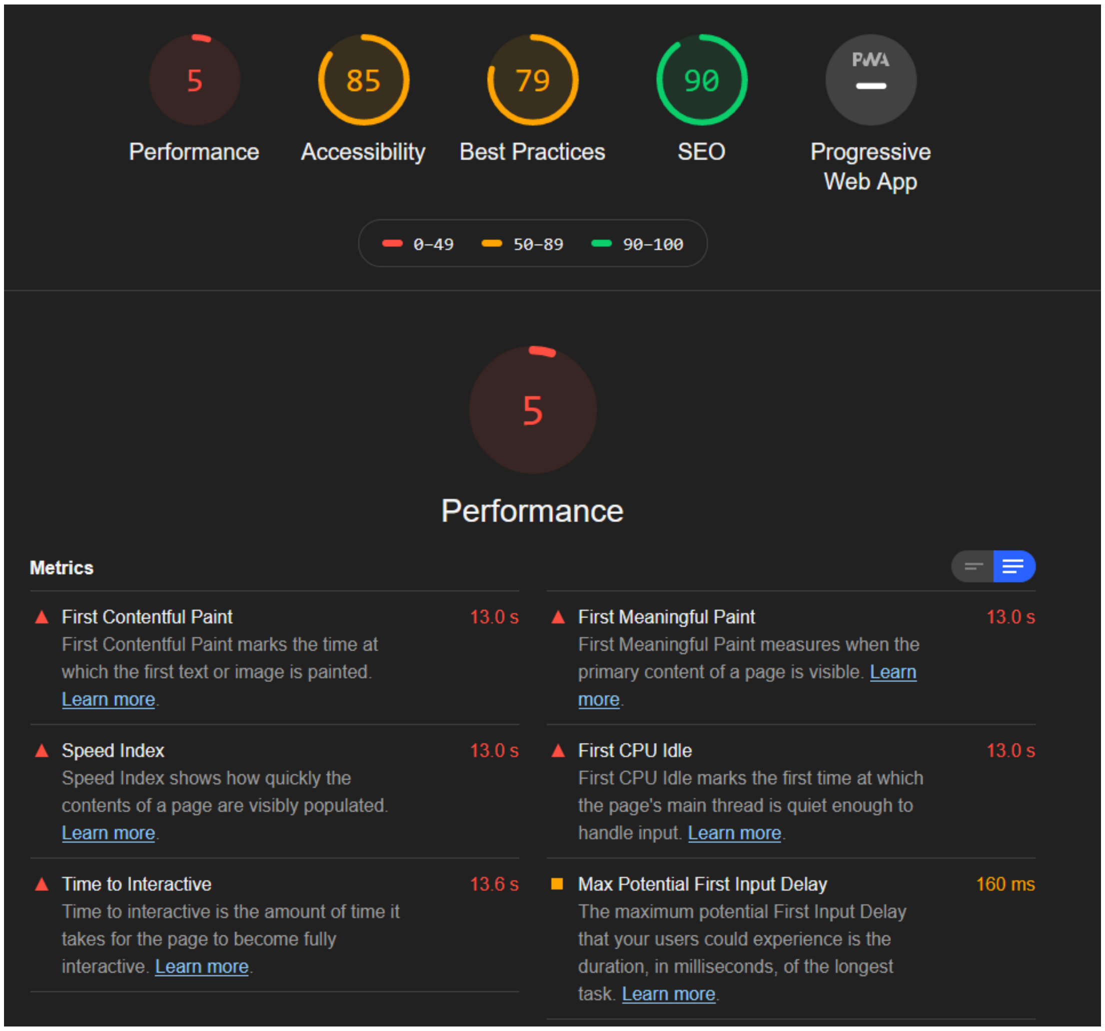
5

Performance

Metrics

▲	First Contentful Paint — First Contentful Paint marks the time at which the first text or image is painted. Learn more.	13.0 s
▲	First Meaningful Paint — First Meaningful Paint measures when the primary content of a page is visible. Learn more.	13.0 s
▲	Speed Index — Speed Index shows how quickly the contents of a page are visibly populated. Learn more.	13.0 s
▲	First CPU Idle — First CPU Idle marks the first time at which the page's main thread is quiet enough to handle input. Learn more.	13.0 s
▲	Time to Interactive — Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.	13.6 s
■	Max Potential First Input Delay — The maximum potential First Input Delay that your users could experience is the duration, in milliseconds, of the longest task. Learn more.	160 ms

C.1


5 Performance
85 Accessibility
79 Best Practices
90 SEO
PWA — Progressive Web App

🟥 0–49 🟧 50–89 🟩 90–100

5

Performance

Metrics

▲ First Contentful Paint 13.0 s
First Contentful Paint marks the time at which the first text or image is painted. Learn more.

▲ First Meaningful Paint 13.0 s
First Meaningful Paint measures when the primary content of a page is visible. Learn more.

▲ Speed Index 13.0 s
Speed Index shows how quickly the contents of a page are visibly populated. Learn more.

▲ First CPU Idle 13.0 s
First CPU Idle marks the first time at which the page's main thread is quiet enough to handle input. Learn more.

▲ Time to Interactive 13.6 s
Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.

■ Max Potential First Input Delay 160 ms
The maximum potential First Input Delay that your users could experience is the duration, in milliseconds, of the longest task. Learn more.