Høgskulen på Vestlandet

My Pension

Bachelor, Computer Engineering

Department of Computing, Mathematics and Physics

Faculty of Engineering and Science

Submission date: 03. June 2019

Number of words: 12127

Kristoffer-Andre Bakkland Kalliainen

Preben Haukebøe

Arne Engelsen Flatekval

I confirm that the submitted work is independently produced and that all references and sources are clearly stated according to *Forskrift om studier og eksamen ved Høgskulen på Vestlandet*, § **9-1.**

Stacc Insight AS 181192, 181394 and 181194 03.06.2019

BACHELOR THESIS TITLE PAGE

Report title: My Pension	*Date:* 03. June 2019
Author(s): Kristoffer-Andre Bakkland Kalliainen, Preben Haukebøe and Arne Engelsen Flatekval	*Number of pages:* *41* *Number of appendix pages:* *6*
Education programme: Computer Engineer	*Number of CDs/DVDs:* None
Contact person at Western Norway School of Applied Sciences: Svein Ivar Lillehaug	*Classification:* None
Remarks:	

External Company: Stacc Insight AS	*External company reference:* -
External contact person: Martin Larsson	*Phone:* 920 53 606

Summary:

This bachelor project is about creating a progressive web application with VueJS and GraphQL that calculates and simulate national insurance, collective arrangement, individual saving and present a savings plan.

Dette bachelor prosjektet handler om å lage en progressiv web applikasjon med VueJS og GraphQL som kalkulerer og simulerer folketrygd, tjenestepensjon, individuell sparing og presenterer en spareplan.

Keywords:

Progressive Web Applications	GraphQL	VueJS	TypeScript

Western Norway University of Applied Sciences, Faculty of Engineering and Science

Postal address: Postboks 7030, 5020 BERGEN *Visitation address: Inndalsveien 28, 5063 Bergen*
Phone. 55 58 75 00 *Fax 55 58 77 90* *E-mail: post@hvl.no* *Home page: http:/hvl.no*

Høgskulen
på Vestlandet

Table of Contents

List of Figures

1 INTRODUCTION

1.1 Briefly on the project owner, Stacc Insight AS

Stacc Insight AS is a software/fintech (financial technology) company with its base in Bergen.

They provide a module-based platform solution, called *Rådgiver Privat Marked* (RPM), providing banks and their financial advisors with a powerful tool in the day-to-day work. RPM delivers precise and detailed analysis of a customer's financial situation, which can in a simple way be used to offer concrete advice and execute plans regarding credit, savings plans, pensions or other financial decisions.

In RPM there are two modules called *Saving and investment* and *Pension*. The *Saving and investment* module provide an investment plan calculator with goal-based saving and target outcome. It offers a customized portfolio recommendation based on risk and timeline. Through the *Pension* module, a bank's financial advisor can calculate future pension payments for a customer, experiment with a customer's pension age and see consequences and recommend savings plans.

This project will expose the underlaying calculation engine of RPM that is primarily used by internal systems and make it available through a progressive web application. This application will be accessible by anyone with a web browser and an internet connectivity.

1.2 Goal and motivation

The goal of this project is to create a standalone pension calculator that will be able to take national insurance, collective arrangement and individual savings into account, and recommend a savings plan.

A survey *Opinion* has carried out for KLP shows that only 2 out of 10 under the age of 30 have familiarized themselves with what they'll receive in retirement income (KLP, 2018). By completing this project, the project team want to enlighten people that don't know much about pension.

1.3 Context

Banks using RPM today are dependent on their customers to seek out information about their pension savings through the bank's financial advisors. This project will remove the customer's need to consult an advisor to learn about her or his pension, and rather make the information needed through the application.

The project team have decided to create an application that will be deployed in a cloud infrastructure, thus removing the need for internal servers and firewalls. Cloud platforms offer the ability to scale the resources needed in real time, resulting in the banks being able to only pay for actual usage and not worry about unused resources that would be prevalent with an on-premise managed solution.

1.4 Limitations

RPM is a very complex system, with a lot of dependencies and services integrated in the solution. The primary focus for this project will be to simplify the complex API's and to use these new APIs to implement our own pension calculator.

The team's knowledge about banking is very limited, with emphasis on, but not limited to processes, regulations, laws and current systems such as MiFID II (Markets in Financial Instruments Directive) and ESMA (European Securities and Markets Authority). MiFID II is the EU legislation that regulates firms who provide services to clients linked to 'financial instruments' (shares, bonds, units in collective investment schemes and derivatives), and the venues where those instruments are traded (MiFID, n.d.). ESMA works in the field of securities legislation and regulation to improve the functioning of financial markets in Europe and strengthen investor protection and co-operation between national competent authorities (ESMA, n.d.). The project team will do the best to make sure that the product is compliant with both MiFID II and ESMA. To do so, resources at Stacc Insight will be used.

A potential limiting factor is the team's lack of knowledge about the different frameworks and technologies that is going to be used in this project. The team will use knowledge available among developers at Stacc Insight as well as reading up on subjects through documentation and video courses etc.

1.5 Resources

The most important resource for this project is Stacc Insight and their combined competence. Stacc Insight can contribute with knowledge and experiences within and about banking, processes, RPM and technological solutions.

The project will require the use of many different frameworks and tools. A more detailed walkthrough will come later in section 4, but as an overview, the finished project will be deployed in Azure on Azure Kubernetes Services using Helm, the Kubernetes package manager. The pipeline will follow the GitOps methodology and use Bitbucket Pipelines together with Flux to provide a CI/CD (continuous integration/continuous development) pipeline. The client and the server will run on NodeJS and use the Apollo GraphQL framework to communicate between each other. VueJS framework will be used to build the user interface.

To manage tasks in the agile process, *Trello*, together with the *Elegantt Chrome plugin, will* provide both a Kanban styled task board and an integrated Gantt diagram, which will provide a better overview of the management process. Slack, OneDrive and Microsoft Office 365 Word will be used for communication and collaboration. Slack will be the main channel for communication and will facilitate sharing of important information, files and code snippets. In order to communicate with resources at Stacc Insight (i.e. Developers, advisors and product managers) the team can use the chat function inside Slack. Microsoft OneDrive makes it possible to share and store important documents, whereas Microsoft Office 365 Word facilitates collaboration on the BSc thesis in real-time.

Through the use of the Live-Share plugin in Microsoft Visual Studio Code it is possible for several programmers to write code in real-time on the same project. It also makes it possible to observe each other's code and see outputs from the host's terminal. This makes the Live-Share plugin a great tool to practice pair-programming and/or XP (extreme programming), even if the collaborators are not in the same room.

2 PROJECT DESCRIPTION

2.1 Practical background

Stacc Insight AS have no previous collaborations with HVL on bachelor projects. They have not put any requirements on the project team, and the team is free to come up with their own ideas and to use whichever technologies they want.

2.1.1 Project owner

Stacc Insight AS provides the module-based platform solution, RPM, as mentioned in section 1.1, enabling financial advisors to give precise and reliable advice based on unique customer insight. It is partly owned by the employees, who own 49 percent of the shares, while the mother company, Stacc, owns the remaining 51 percent. Today, there are approximately 30 employees at Stacc Insight, working together to provide the *RPM* solution to over 150 banks, primarily in Norway, but also in Denmark, Sweden and the Faroe Islands.

One of Stacc Insight’s biggest customers is the Eika-group which consist of Sparebank1, Sparebanken Vest, SDC, Danske Bank, Sparebanken Sogn og Fjordane, Sparebanken Sør, Sparebanken Møre, Helgelands Sparebank, Fana Sparebank and DSS.

Stacc Insight is part of the Stacc group, which is composed of Stacc Insight and three other companies:

- *Stacc Core* who provides a solution to administrate loans, leasing and savings.
- *Stacc Flow* who delivers the next generation *Business Process Engine* that provide solutions for loan applications and processing.
- *Stacc X* who delivers configurable standard components for better customer experience.

2.1.2 Previous work

Stacc Insight has for quite a while provided the “RPM” solution for its advisors, and it is this solution, particularly the *saving and investment* module, that will be the basis for this project. However, there has never been an attempt to extract the module and present it as a business to customer (B2C) service. There has been an attempt to deliver an external service to one of their customers based on a financial fund application.

Although no previous work has been done in creating a solution like this, Stacc Insight has a comprehensive system for calculating all the figures that this application needs with the underlaying infrastructure of RPM. Figure 9.8 shows a screenshot of how this solution works internally in RPM today.

2.1.3 Initial solution idea

The initial solution idea was to focus on the Pension part of the *Saving and investment* to scope the project. The deliverables should include an application available on mobile and desktop devices Too achieve this the application should be written in a framework that allows cross-platform compatibility. The functionality of this application is to be identical on all platforms.

By utilizing the power of Progressive Web Apps (PWA) the application would be developed to run on all devices. PWA is great in that it supports different types of architecture, which results in that once developed it will run natively on iOS, Android and in a web browser. By publishing the solution in the cloud, the scalability of the application will be handled by a third-party vendor.

The new solution should be modular, and to achieve this the application should be developed in coherence with the principles of REST. This will be accomplished by introducing a gateway/proxy that can access any internal resources that are required. The application will do requests towards this gateway by using the lightweight GraphQL framework to query and mutate data as it needs.

After a few meetings with resources at Stacc Insight, the project team decided to expand the project, from a pension savings calculator, to a full worthy pension calculator. The calculator should provide estimated results from national insurance along with the default case of 2% in defined contribution pension and the RPM engine to calculate based on stock- and fund savings.

The project expansion is done in order to better reach our end goal, mainly to increase awareness for pension savings for people born after 1963.

2.1.4 Initial requirements specification

The initial requirements specification is to make API calls cheaper for devices running on limited bandwidth. The solution should be user friendly for the end-user and provide a well-documented solution. It should also lay the groundwork for further development and expanding the solution.

A study performed by two Norwegian scientists, Anniken Hagelund and Anne Skevik Grødem, discussed how newspapers present the topic of pension saving, they found that it is mainly the individual consumers' responsibility to make sure that they gain the knowledge they need to handle the consequences of pension reform responsibly (Hagelund & Grødem, 2017). The *My pension* application will try to attract the interest among younger people, mainly those born after 1963, so that they become more interested in their own pension. As of now, the current solution that Stacc Insight provides requires the end users to consult an advisor. This presents a barrier that our application intends to tear down. The project team decided to focus on developing a retirement savings calculator application This is done to be able to greater increase the awareness surrounding this.

3 PROJECT DESIGN

This section presents and discusses some of the design approaches evaluated by the project team and describes the specifications of the chosen approach in detail. Details about resources such as tools, languages and methods are also described. The development and evaluation methods will also be described.

3.1 Specification

3.1.1 MoSCoW Analysis

To be able to prioritize and reach a common understanding of what to deliver, the project team used the MoSCoW analysis. After learning in the courses ING101 (Teknologiledelse, økonomi og nyskaping) and ING102 (Ingeniørfaglig yrkesutøvelse og arbeidsmetoder for datafag) that projects are easily scoped too broad, the project team decided that all the *Must have*, *Should have* and *Could have* requirements will initially be met.

Must have

As mentioned in section 2.1.4, one of the initial requirements is to create an application. To better explain the Norwegian pension system the application must include all three elements described in the pension triangle (Figure 3.1 Pension Pyramid), national insurance, collective arrangements and individual savings.

Some of the key features have been split up into smaller, more attainable features to fit the "must have" category and will be developed further in the should-, could- and would have categories.

The main components in the "Must have" category is:

- Calculating the national insurance based on active working years in Norway and expected yearly salary until age 67.
- Calculating the collective arrangements with the standard 2% of a person's salary as is enforced on businesses by Norwegian laws.
- Calculating individual savings based on what the end-user wants to achieve or wants to save and give a recommendation to a savings plan.
- Making a Lightweight application with mobile devices in focus.
- Displaying the pension triangle graphicly.

These main components have a few sub-requirements in order to work, such as a risk tolerance analysis, and the option to choose a risk-level (small, medium, high).

To make this a lightweight application, the project team have decided that it needs to be stateless. This means that it doesn't save any data to a database, but rather in the cache, local storage or cookies. The application must also represent the pension triangle graphicly to educate the end-users on how the Norwegian pension system works and because it is the basis for the application and the project.

Should have

- A "flow" to build a personal pension triangle, step by step, in any order the end-user wants.
- Fetch products for the recommendation dynamically.

  - This enables the user to switch between high, medium and low risk and the different products will be retrieved based on that.
- Graphicly represent an interactive graph of a savings and withdrawal plan, where the user can check the expected "balance", with upper and lower bounds, at any given time in the savings-plan.

Could have

- Ability to change the type of collective arrangement, from normal 2% bank deposits to consider multiple options like shares and funds.
- Graphical view to show the differences in the possible savings methods.
  - This may encourage the user to select the best saving method based on the user risk tolerance.

Wish/Would have

- Provide both an input and an output calculator.
  - This would mean that you can provide both the scenarios where you put in *X* money per month/year how much would you end up with, and if you want *Y* money at the end of the savings period how much do you need to save per month/year.
- Gather user's actual collective arrangement from services like the customers bank or NAV.
  - Will require BankID authentication implementation.
- Transfer the individual savings plan to a real savings plan in a bank.
  - Will require an option for the Banks to connect to the application and be able to parse the individual savings plan result.
- Provide an offer from the banks based on the savings plan result.
  - The bank would then need to be able to provide their offers on shares and funds via API's and the application could recommend the different options and provide it to the user.

3.1.2 User stories

An end-user wants to start saving money for retirement. As the user is very busy it's not easy to find time to see an advisor. A mobile web application allows for easy access to powerful saving tools. The end-user should be able to get an overview over their pension based on some input supplied. The result presented to the user should be enough for them to start thinking about their pensions and what they can do about it. The end-user shall be able to see a pension estimate, and furthermore get recommendations based on the end-user's risk tolerance score. Risk tolerance is a metric used to determine how susceptible the user is to fluctuation in the market.

Saving for pension have three major parts, all build on top on each other as shown in Figure 3.1. The national insurance (Folketrygd) depends on how much a person have earned in total working years in Norway before retirement. The collective arrangement (Kollektive Ordninger) is what one's employer has arranged on the behalf of the employee, whereas the individual savings (Egen Sparing) is what a person has saved by themselves.

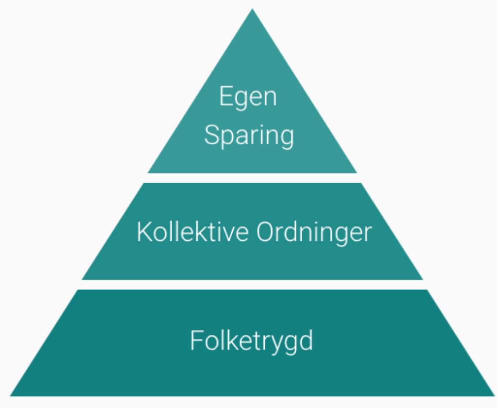


Figure 3.1 Pension Pyramid

3.1.3 Functional requirements

Functional requirements define specific behavior or functions in a system. The plan for implementing the functional requirements will be detailed in the section 4.

- A user should be able to create a saving goal the first time the user visits the application.
- The user must provide two out of three key figures: initial capital, monthly savings or monthly payout.
- The application sets various values as standard, including retirement age (67 years), payout period (10 years), risk profile set to medium (unless they have performed a risk tolerance analysis) and a selection of founds.
- The user will then be redirected and shown key figures and expected savings requirements.
- If the user is already registered and has created a savings goal, the user will be redirected to an overview of the user portfolio.
- In the portfolio section of the application the user should be able to view key figures, historical numbers, overview of savings goal and returns.

3.1.4 Non-functional requirements

The non-functional requirements specify criteria that can be used to judge the operation of a system, rather than specific behaviors. Non-functional requirements define how a system is supposed to be (NFR, n.d.)

- The project shall deliver an API to interact with data located on the Stacc Insight beta environment. The API will be a wrapper around the existing API using the endpoint *visningsrom.stacc.com*.
- One of the goals is to reduce the number of API requests between the client and the server. This can be solved in several ways by utilizing caching techniques on both the server and the client and combine multiple request to bigger ones to reduce the RTT on unstable networks.
- It should be easy and intuitive to fill out the initial values when the user first starts the application. The information required for the user to use the application for the first time must be kept to a minimum. Also, the information the user provides should be easy to change and modify.
- The key figures of the graph provided for the user should be easily read and its' content understood. The graph should display worst case, best case and predicted case for the saving goal.

3.2 Possible approaches

3.2.1 Location of the application

The RPM solution heavily depends on having a customer object in the solution. To create a customer object, a lot of information is required about the end user. This is not ideal for the solution as it should only be used for “simple” calculations. As the customer object in Stacc Insight’s source code is being used for almost everything, the project team is dependent on some parts of this being rewritten. A possible workaround for this is to have a “dummy” customer in the database that can be used to interact with the rest endpoints. Again, this is not ideal and will be used as a fall back.

The product will be very independent of the systems that the banks are running, meaning that this will be made generic and then easy to tailor to each customer. The goal is to have the application be stateless enough to be placed in the right most application box shown in figure 3.2. Figure 3.2 is showing suggestions on where the application can belong; It can be highly dependent on the solution the bank is running, shown to the left, or totally separated from any backend banking solution, shown to the right.

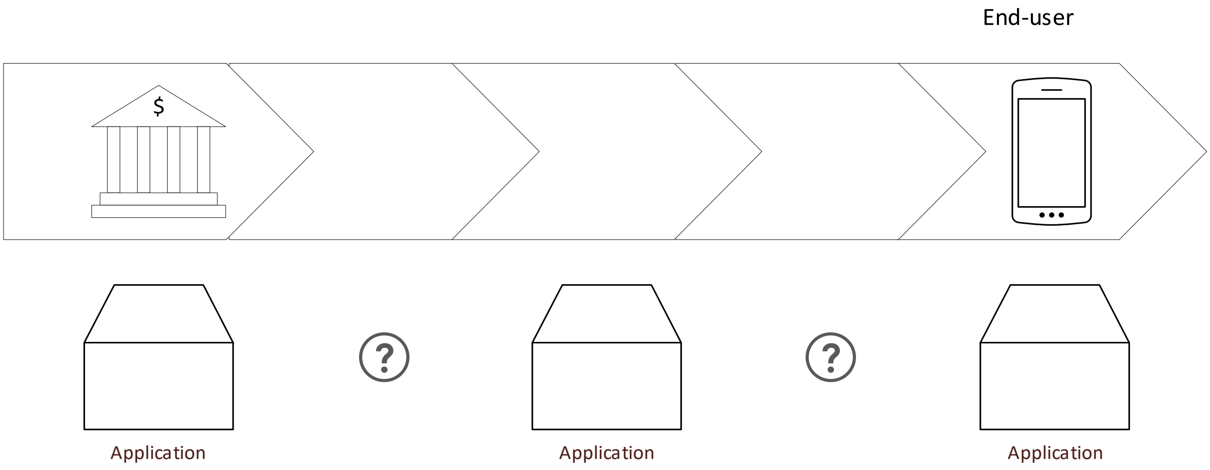


Figure 3.2 Location of the application

3.2.2 Client/Server communication

Rebuild the solution, and make a better Rest API

The team had an initial idea to rewrite the solution to create a better Rest API. This would include a lot of work as the framework that Stacc Insight has developed is massive and complicated. A total rewrite and design of the API from scratch would increase speed of fetching data and would make the clients do fewer requests. The value Stacc Insight would be getting out of this would be high, but the project team decided that this was too big of a task to handle for a BSc project.

Proxy server with REST API against visningsrom URL

Stacc Insight has exposed the beta environment on visningsrom.stacc.com to be used as a showcase for customers. In this approach a proxy server is located externally and queries the endpoint with REST calls. The proxy server transforms the data to a proper format, making it easier and lighter for a mobile device to fetch. This results in making a wrapper API doing the heavy lift before serving it to the client. The existing code can live as it is, as changing the existing code base will result in a conflict with the existing architecture described in 3.2.1.

In order to make this work, the design of the wrapper API is crucial. REST API's tend to return more data than needed or making more requests if the data returned is not enough. This is a disadvantage on devices with low network bandwidth or an unstable connection.

A big benefit of using REST APIs is that it's just normal HTTP-requests and a lot of programming languages have native libraries for supporting HTTP-requests.

Proxy server with GraphQL against visningsrom URL

This approach solves the same issues as in the approach described above but handles REST biggest problem with over fetching and multiple calls, by using GraphQL. With GraphQL clients can dictate exactly what they need from the server and receive that data in a predictable way. You only receive the data you explicitly requested. By nesting queries, GraphQL makes it possible to retrieve many resources through a single request.

This makes it a very useful tool for providing devices that run on low network bandwidth and unstable connections a better user experience. The most crucial part with devices running on poor connections is the RTT (Round-Trip-Time). By only fetching the data needed and making as few requests as possible, it will improve the responses of the application. Mobile devices can handle bigger HTTP-requests well, but many small requests will result in a lot of overhead and unnecessary data usage, as well as being very time-consuming with limited bandwidth.

Designing good GraphQL queries is just as important as designing good REST-APIs. Queries that are nested need to be supported by the server, otherwise the queries can result in computations for the server that are so heavy it crashes. GraphQL provides a more backed stability, an application-backend can be overhauled without necessitating a complete restructuring of the client application.

This approach will also use a proxy server transforming GraphQL queries from the client and making normal REST calls in the backend against the visningsrom.stacc.com endpoint.

3.2.3 Client

The client will be a Progressive Web Application (PWA). A PWA is a web app that uses modern web capabilities to deliver an app-like experience to users. If a browser doesn't support PWA the application will still work but without the progressive enhancements.

PWA advantages over native apps are many. According to comScore's 2017 U.S Mobile Apps Reports, more than half of Americans smartphone users download exactly zero new apps a month (Chang, 2017). A PWA application is accessible from a URL that makes a low friction of distribution and easy discoverability. A PWA application will only have one code base for both web and mobile devices. So, in the battle of PWA vs Native, it's really PWA vs (web vs native vs native) (Dascalescu, 2018).

The VueJS framework together with Typescript will be used to create the UI. VueJS is a progressive framework, which means it adapts to the needs of the developer. It doesn't require you to rewrite an existing application to use it. VueJS lets you extend HTML with HTML attributes called directives. It provides built-in directives and user defined directives that offers functionality to HTML applications. VueJS will make it simple to get started on the project and produce results early, routing and data management is covered by official libraries like VueRouter and Vuex. Fast rendering is ensured by the virtual DOM implementation resulting in minimal load time that is important on a mobile device to give feedback as soon as possible to the consumer.

Apollo Client will be used to connect the client to the GraphQL endpoint. This is a lightweight module that allows for quick and easy access to the data needed. In order to not run unnecessary queries to the GraphQL endpoint, the team will also utilize the Apollo Local State Management which allows for storing and retrieving data in the client cache.

3.2.4 Discussion of alternative approaches

RPM is a solution running on an internal system. Including the development environments at Stacc Insight. This makes it a difficult dilemma on how to solve the project. By using the visningsrom.stacc.com we avoid interrupting the internal system. Stacc Insight has started the process on moving RPM to the cloud, by making it possible to sell their solution as services instead of a standalone application. By designing the project as described in the GraphQL and Client section above and using the external endpoint, it makes the application decoupled, and it can also be used when RPM is available in the cloud.

Making the API lightweight and only returning what is necessary is the primary goal in this project. REST and GraphQL solves this in different ways, it's all down to how the API is designed. By using GraphQL you get more control of the data returned from the server with its query structure.

Even though there are several different approaches mentioned in this thesis, they are all dependent on RPM as the source of data and engine for processing heavy calculations. In appendix 9.3 there are several figures showing how a bank is set up with the correct levels of risk tolerance, correlation matrixes between the different funds, product distributions etc. All the values are standardized and thus are the most common ones to use. Each bank using RPM may have a different setup of this which could yield different results then what the "My pension" app delivers.

3.3 Selection of tools and programming languages

3.3.1 Project management tools

Trello will be used for project management. With Trello a user can sort the various tasks into buckets to easily separate tasks that are not started yet, ongoing or completed. With the plugin *Elegantt* it's also possible to generate a Gantt diagram from the tasks.

Git and *Bitbucket* are needed for having multiple developers working in parallel on the same codebase. Git ensure that there are no code conflicts and provides a version control system that allows developers to revert and go back to older versions of the code if needed. Bitbucket is a repository hosting service by Atlassian and provides Bitbucket Pipelines.

3.3.2 Tools

Visual Studio Code is a powerful lightweight cross-platform source code editor developed by Microsoft. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion and refactoring. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) (VSCode, n.d.).

Apollo Playground is a graphical, interactive, in-browser GraphQL IDE, created by Prisma and based on GraphiQL. Apollo Playground is a feature in Apollo Server and is enabled by default in development mode. It is hosted in the same URL as the GraphQL server itself (Apollo, 2019).

Postman is a powerful HTTP client for testing web services. Postman makes it easy to test, develop and document APIs by allowing users to quickly put together both simple and complex HTTP requests.

Bitbucket Pipelines is an integrated CI/CD service, built into Bitbucket. It allows you to automatically build, test and even deploy your code, based on a configuration file in your bitbucket repository (Bitbucket, n.d.).

Docker is a container technology that packages code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings (Docker, n.d.).

Microsoft Azure is a cloud computing service for building, testing, deploying and managing applications. It provides SaaS, PaaS and IaaS solutions. This project will use Azure Kubernetes Service for running the application.

Kubernetes is a container orchestration system for automating application deployment, scaling and management.

3.3.3 Programming languages

JavaScript/Typescript

JavaScript is a high-level, interpreted, object-oriented programming language based on the ECMAScript specification. Interpreted means that JavaScript is being executed at runtime, without compiling directly to the target machine. The advantages on interpreted languages is that they are easier to implement and can execute code directly on the fly. JavaScript is the programming language for the Web. It can update and change both HTML and CSS and calculate, manipulate and validate data. JavaScript normally runs on the client side, but with NodeJS, that provides run-time environment. You can execute JavaScript code server-side.

For managing packages in the NodeJS run-time environment Yarn will be used. It's a light node package manager (NPM) client developed by Facebook. It uses a *yarn.lock* file that ensures that the exact same package gets installed on every device.

TypeScript is a superset of JavaScript which primary provides optional static typing, classes and interfaces. One of the big benefits with typescript is to enable IDEs to provide an environment for spotting common errors as you type the code. VS Code is written in TypeScript and provide excellent support and code completion. Coming from a static typed language like Java from the university, TypeScript is easier to learn, understand and work with.

This project will use the ESNext specification of JavaScript. ESNext is the future version of ECMAScript which is yet to be released. To provide backward combability against older browsers, TypeScript will be transpiled down to ES5 with the help of the TypeScript compiler.

GraphQL

GraphQL essentials is described in 3.2.2 and 3.2.3. To summarize GraphQL has three main characteristics. It lets the client specify exactly what data it needs. It makes it easier to aggregate data from multiple sources, and it uses a type system to describe the data. With GraphQL, the user can make a single call to fetch the required information rather than to construct several REST requests to fetch the same.

A very nice feature with GraphQL implementations is that most frameworks provide an IDE like solution for exploring and browsing GraphQL API's. The two most common are Graph*i*QL and Apollo GraphQL Playground.

Apollo

Apollo is a framework for GraphQL that delivers implementations for multiple programming languages and for both client and server.

The Apollo Server provides a default endpoint to work with the data. The server can connect to REST-endpoints, micro services and databases. This is extremely powerful and will enable the implementer to access the most common data sources there is. After writing the type definitions and queries on the server these gets exposed and documented for you. This makes hooking up to the GraphQL endpoint very easy and structured.

Apollo Client and Apollo Link State Management allows for both managing remote data through GraphQL and managing local data at the same time (i.e. caching). The client is a JavaScript UI which makes it easy to create interactive and flexible apps.

3.4 Project development method

3.4.1 Development method

For this project, the development team have chosen to combine agile and lean software development methodologies, mainly by using two different frameworks; Kanban and XP (extreme programming). The agile and lean development methods are great strategies by themselves but work even better together.

Some of the key aspects and advantages of the agile development method are continuous delivery, high emphasis on cooperation and continually adopting to changing circumstances. This will help the development team to work iteratively and stay on track, both in the given time frame and the stated requirements (Agile-methodology, 2018).

The lean development method emphasizes productivity and elimination of waste, respect for people and continuous improvement (Lean-Pathways-inc, 2011).

In lean development, waste is defined as anything that's not adding value to the project in the eyes of the customer. Respect is about acknowledging others work and giving them objectives based on their skills. Also, pair programming, focusses on constructive critique, rather than meaningless gibberish like pointing out someone's fault without bringing a better solution to the table.

The Kanban framework in combination with XP, utilizes the features of both lean and agile development that we deem the most important to the project.

The project team will work iteratively with new features together in pair programming, overviewing each other's code while it is written to ensure high quality and rotate the keyboard between coders frequently to ensure that everybody is on the same page. This is useful as the team can't always gather together to write code and when this happens, there is some agreement on what is to be done and how.

To familiarize with the content and context of banking, the team can utilize the principles of Domain Driven Design. This allows for better division of tasks in which each one can be handled at the same time. If this is used right it will allow for the team to work with different components on both the client and server side, and not interfere with each other's code.

3.4.2 Project Plan

The primary objective is to implement a solution that reduces the number of requests made to the server directly from the client, while at the same time having clear and precise APIs. This will consequently be given top priority. To be able to implement this the team needs to have enough knowledge to understand how the systems are currently working today.
The second objective, highly dependent on the first, is to implement the APIs in a client that can used to represent the use of the APIs. This objective is not something being prioritized before the first objective is finished, or at least almost complete.

To be able to reach the goals and objectives the team always needs to cooperate throughout the project. In order to secure good quality, the team is to agree upon work schedules that will fit the whole team, and deadlines need to be placed on tasks to ensure progress.
Organizing the tasks in a tool will greatly help with keeping track of everything consequently, the team chose Trello as their task management tool. Another important aspect of planning how long time a task will take is to always estimate more than what initially is thought. By using this technique, the team aim at having all tasks completed prior to the due date.

3.4.3 Risk management

There are multiple potential risks connected to this project. After identifying the risks, the team had to figure out how likely they were to occur, and what the significance would be. This is called qualitive risk analysis and is most focused on the likelihood of something happening and what the outcome would be.

Poor time management: It's important to have a schedule where all the tasks are laid out. As the team does not have a lot of previous knowledge of the work ahead, there must be made compensations in the form of adjusting the dates set on the tasks. If the time would get mismanaged the team would most likely fall behind on the schedule and as a result of that develop a poorly performing solution. To combat this the team is really focused on communication, with more communication there is a good foundation on which upon adjustments can be made along the way.

Product becomes unusable: This is the main concern for the project owner and is usually a result of poor communications between the project team and the owner. To combat this the team is focused on keeping close contact with the project owner to ensure that the focus is right, and the deliverables fits in with their vision.

Unmanageable workload: If the workload becomes unbearable/unmanageable the team intends to handle this by using the timetable and good communication. The team has also planned the tasks ahead of time to hopefully be more able to tackle any situation that may arise.

3.5 Evaluation method

The first and easiest way to evaluate the project is to compare the product with the functional and non-functional requirements listed in sections 3.1.3 and 3.1.4. This will create a baseline evaluation which will very clearly outline any nonconformities.

Secondly, the team intends to evaluate the applications performance by using Google Lighthouse. This is a webpage auditing tool that gives insights into a page's performance, SEO, usability and accessibility. It generates a report on the overall performance, accessibility, best practices and progressive web applications. Improving the app performance goes together with improving the visitor experience. Since the project is about making a PWA mobile application performance, best practices and PWA support has been the focus on how to evaluate the application by metrics.

To make sure the project is evaluated and approved by the project owner, the project team should have regular meetings with the project owner. This is to make sure that they can give feedback as the project and solution develops. Stacc Insight, the project owner, have access to all the source code, and can therefore at any point see what the project team is working on. The team welcomes praise and constructive criticism. This will make sure that the project team has understood what they ought to deliver and what the project owner is expecting.

At the end of the project it is natural that the project teams walk through their solution with the project owner, Stacc Insight. With this the project team should also present their evaluation and findings of the final product. Stacc Insight stands free to evaluate the solution on their own premises.

4 DETAILED DESIGN

This section describes the overview of the different parts in the project design implementation. The infrastructure overview in section 4.1 describes how the different services in the solution is connected at a high level. The Kubernetes overview in section 4.2 shows the complexity on where the client and server are running in Kubernetes, how TLS and SSL are managed, and how the network traffic flows in the service mesh to reach the client and server.

Section 4.3 and 4.4 describes how the client and server communicates over GraphQL, how the server communicates with RPM over REST, how different caching and persistence techniques are used to optimize data fetching and storing application state in the client, and how service worker is being used to provide offline support and caching off the app shell.

Section 4.5 gives an overview of the continuous delivery pipeline using the GitOps pattern, whereas section 4.6 describes the application flow from an end-user perspective.

4.1 Infrastructure overview

The overall infrastructure consists of a Kubernetes cluster (Azure Kubernetes Service) running in Azure that contains the server and client applications running in Docker containers. The server application communicates with RPM through the *visningsrom.stacc.com* endpoint.

When a client makes a request to our application the request needs to go through an Azure load balancer. The load balancer will redirect the traffic into the virtual network to the Kubernetes cluster. The load balancer will take care of both incoming and outbound traffic and balance traffic across nodes in the Kubernetes cluster. Figure 4.1 shows the infrastructure overview.

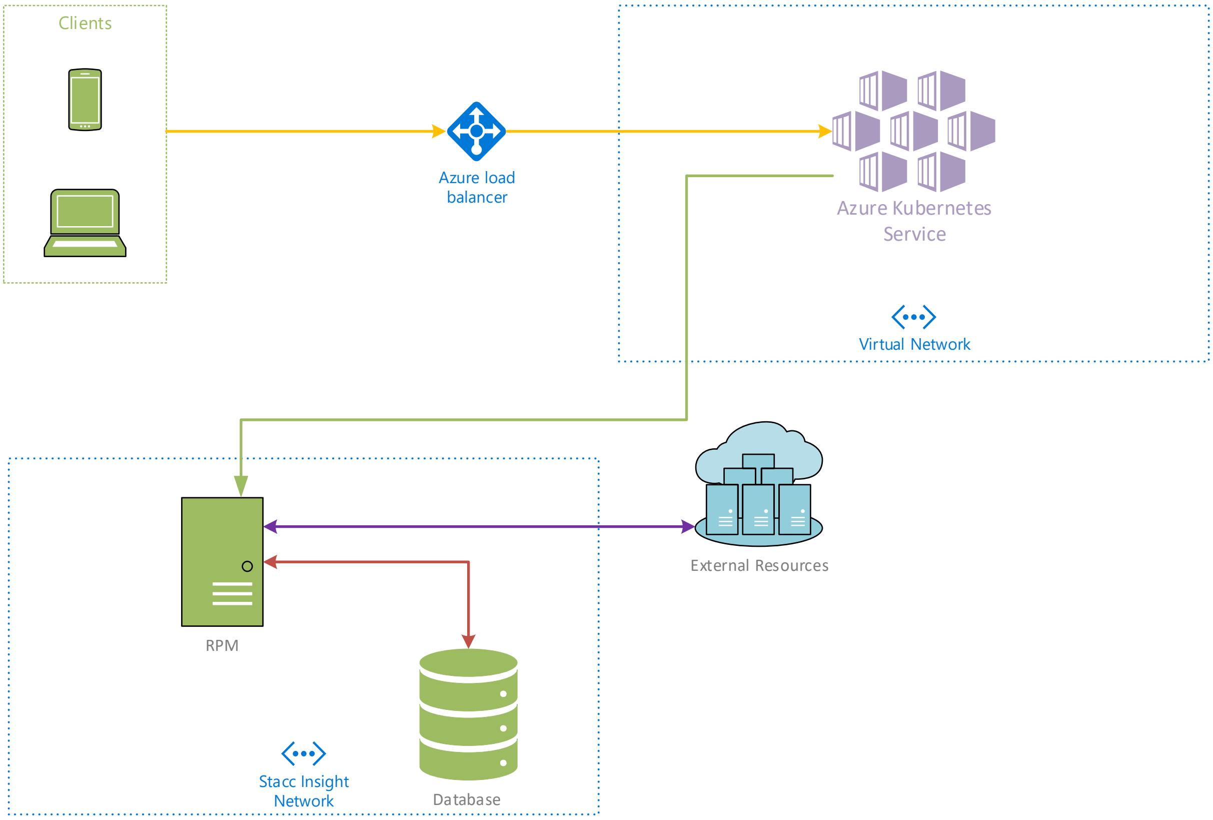


Figure 4.1 Infrastructure Overview

The RPM application is in Stacc Insight's own network where it's running on a VM and using a MySQL database. The beta environment is accessible at the *visningsrom.stacc.com* endpoint, making it easy for our application to connect. This is a temporary solution until the whole RPM application is running in Azure Kubernetes Service.

4.2 Kubernetes overview

Figure 4.2 shows how the Kubernetes cluster works.

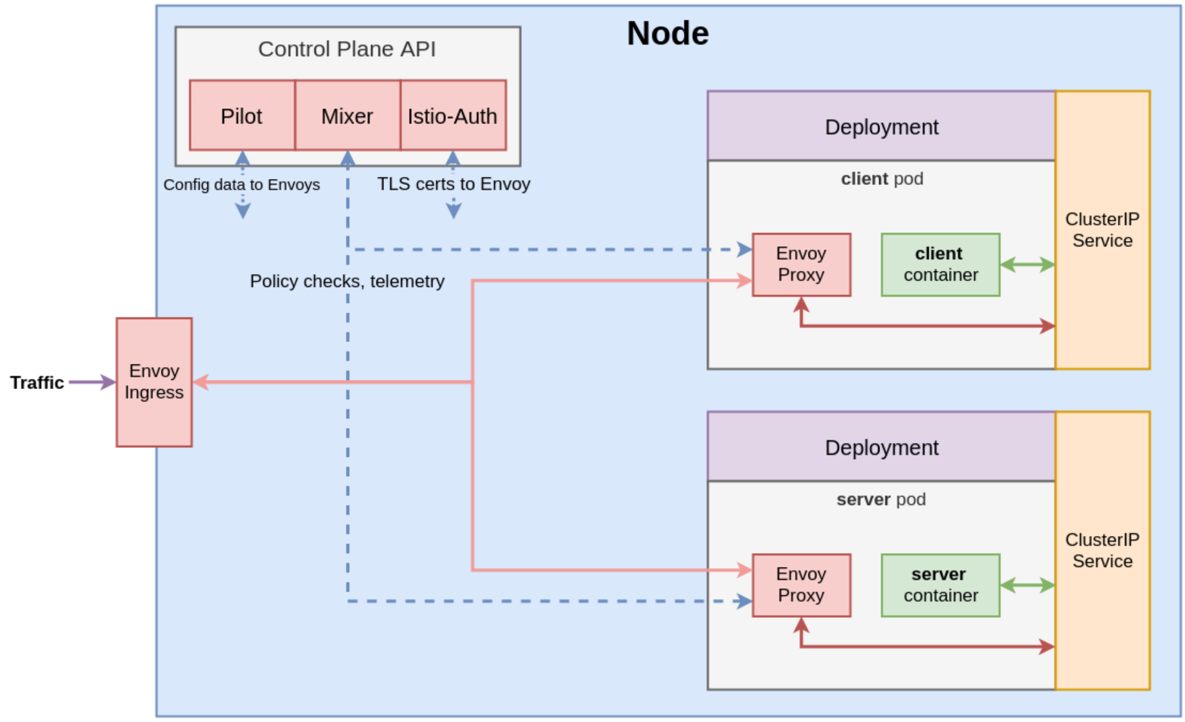


Figure 4.2 Kubernetes Overview

Inside the Kubernetes cluster there are several things happening. All traffic, both inbound and outbound, is controlled through Istio. Istio is a service mesh that delivers self-discovery of services, load balancing, failure recovery, metrics, and monitoring. It also provides A/B testing, canary rollouts, rate limiting, access control, and end-to-end authentication. This project is using Istio to control the traffic coming into the cluster and redirect to the right service and provide end to end encryption with mTLS (Multiplexed Transport Layer Security) and HTTP/2. The Azure Load Balancer (from 4.1) connects to the Istio Ingress (an Envoy Edge Proxy) and forwards the traffic based on rules from the Istio Mixer in the Control Plane. The Istio Ingress also terminates a TLS certificate issued from Let's Encrypt, so that communication from a client to the application is secured and encrypted and reachable from a domain.

The server application is running inside a Pod in a Docker container based on NodeJS. The server is written in TypeScript, and the code needs to be compiled to JavaScript to be able to run on NodeJS.

TS-Node is a tool that solves this quite smoothly. TS-Node is a TypeScript executer and REPL (Read-Eval-Print-Loop) for NodeJS, that makes it possible to run TypeScript directly on runtime, instead of compiling the server code before it is run or in a pipeline. The server is both running the GraphQL Server and GraphQL Playground.

The client application is also written in TypeScript, and it is using VueJS. When VueJS is built the result is a purely static application consisting of static HTML, CSS and JavaScript files bundled with webpack. In order to serve a VueJS application we only need a static file server. For this project, the NGINX web server was chosen due to the team's previous experience with this technology. To summarize, the client application runs inside a separate Pod in a Docker container, running a NGINX web server that serves the static application files.

4.3 Client/Server communication and local state management

The server is running an Apollo Server, which is an implementation of a GraphQL server. For the client to be able to communicate with the GraphQL server it needs a GraphQL client. The client uses the Apollo Client library. All communication between the client and server uses GraphQL, as shown in figure 4.3

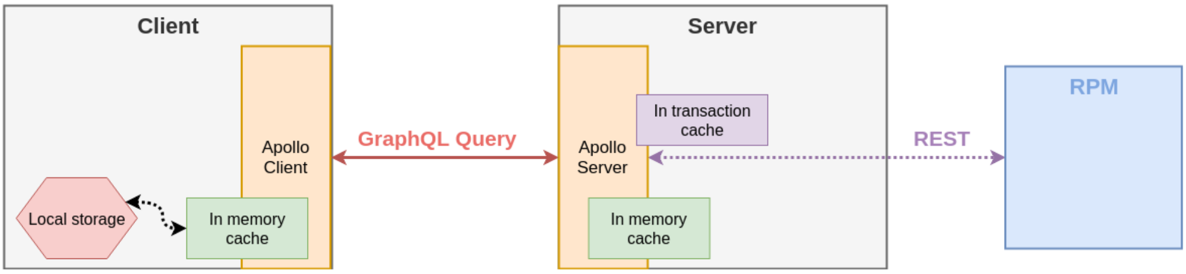


Figure 4.3 Client/Server communication

On the client all GraphQL queries are cached by the Apollo InMemoryCache. The in-memory cache is fast and stores all data in a normalized format by creating a unique identifier for each object and stores those objects in a flattened data structure. Apollo persistCache has been used in order to persist application and user data for offline support. Apollo persistCache will persist every write to the cache to the localStorage in the browser asynchronously. The localStorage is an HTML5 implementation and provides at least 5MB of storage. Data is stored in a key/value-based data structure and the information is never transferred to the server, as opposed to cookies. LocalStorage is chosen because it supports most modern web browsers on both mobile and desktop devices.

The client also uses Apollo Local State Management, which makes it possible to store the state of the application and use client only GraphQL queries and mutations to manage the state. This obviates the need for a separate store like Vuex and makes Apollo cache the single source of truth for all data in the client application. The Apollo Client can handle both local and remote data in the same query, making it a universal interface on how to handle both state and data in the application. Figure 4.4 shows how Vue uses the local state management and the local storage.

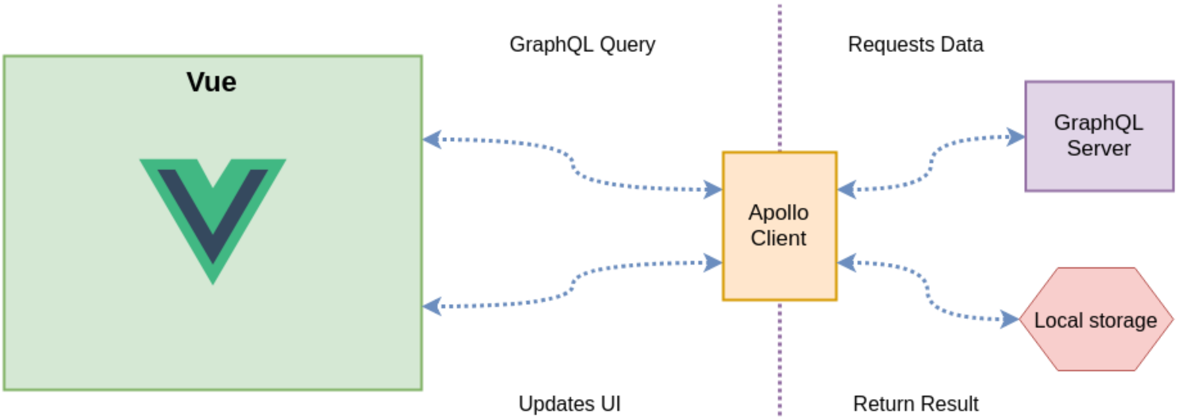


Figure 4.4 Local State Management

The server is running the Apollo Server which also uses Apollo InMemoryCache for caching whole queries. The RPM application uses REST for exposing its resources, whereas the server uses Apollo REST Data Source for fetching data from the RPM endpoint. Apollo REST Data Source has built-in support for caching, deduplication and error handling. These features make it possible to do in-transaction caching of REST calls to avoid fetching the same data multiple times in one transaction, which often happen with REST.

4.4 Service Worker and cache API

A service worker is a relatively new piece of technology that has been rolled out to the more modern web-browsers over the last couple of years. Together with PWA, the service worker enables previously native only functionalities like push notifications and offline support. This means that there is now a method to write a cross platform application (Android, IOS, Web), that gives a native app feeling if the device supports it - and if not - renders the webpage as it normally would (Copes, n.d.). Figure 4.4 shows the workflow that the service worker is following.

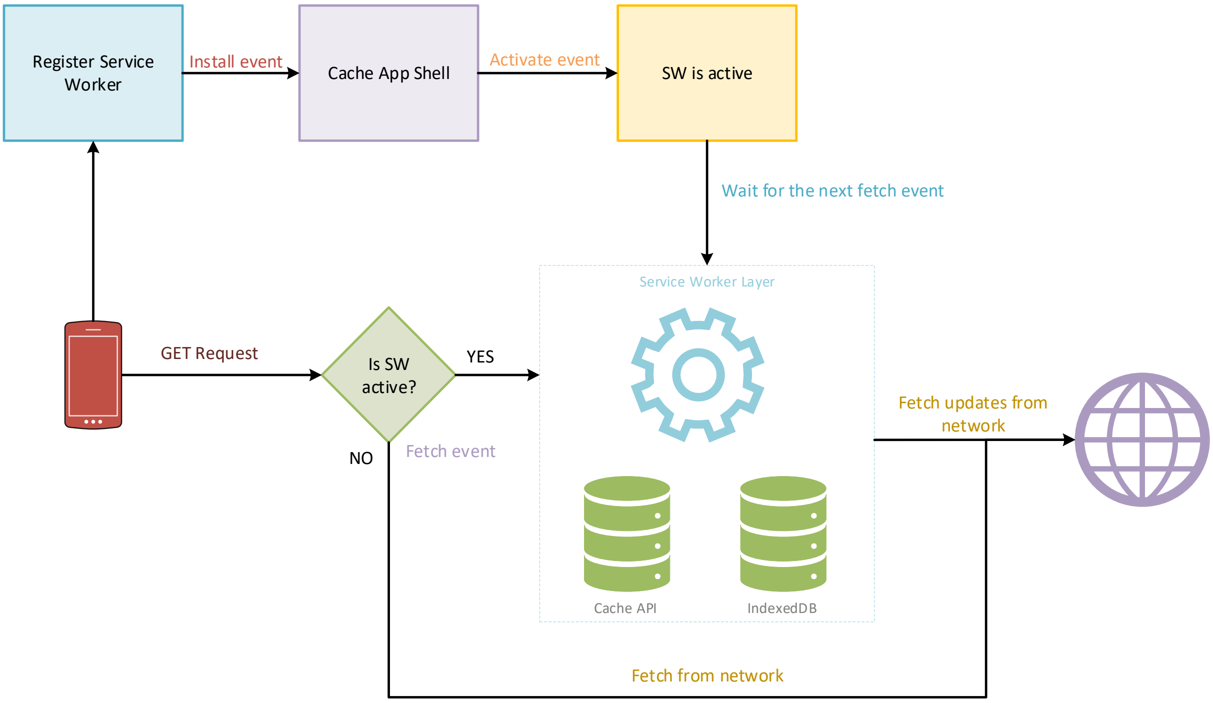


Figure 4.5 Overview of Service Worker

In the “My Pension” (Min Pensjon) application, the service worker is caching the app-shell (all static data, like html, JavaScript and CSS) by utilizing the standardized cache API. This makes the application load almost instantly and offer offline support on the app-shell.

There are many different approaches to configure the service worker to handle cache. “First cash then network” or “cache and update” is an example and the strategy used in this project. This means that whenever the app is launched, it will load the content from cache, as well as sending a fetch request to the server and load new content into the cache. The new content will be rendered from cache on the next app-launch (service-worker.js, n.d.).

4.5 Continuous Delivery GitOps Pipeline

The project's pipeline follows a GitOps pattern (operations through Git). There are several components in the pipeline like Bitbucket Repository, Bitbucket Pipelines, Azure Container Registry, Weaveworks Flux, Helm and Kubernetes. Figure 4.6 shows the flow of the pipeline.

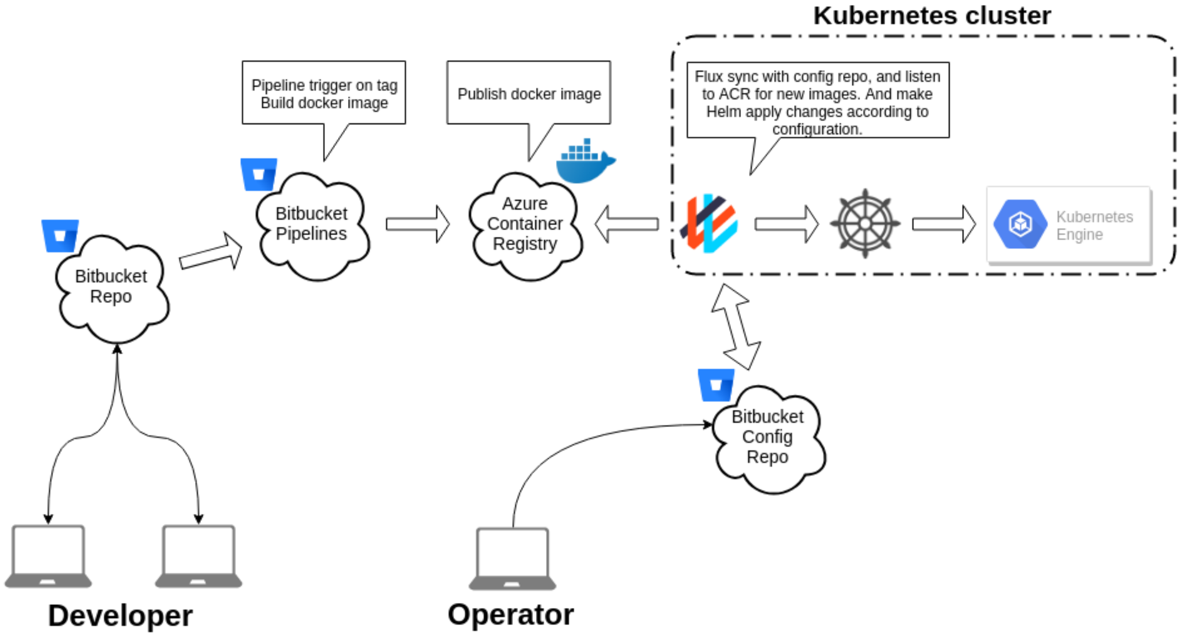


Figure 4.6 Continuous Delivery Pipeline

There are two main flows in this pipeline, the developer's pipeline and the operator's pipeline. First, the developer commits a change to the code repository. This triggers a Bitbucket Pipelines build. The build results in a Docker image that is pushed to Azure Container Registry. Inside the Kubernetes cluster, a Flux operator listens for updates in Azure Container Registry and according to the configurations in the config Bitbucket Repository a new deployment will be deployed.

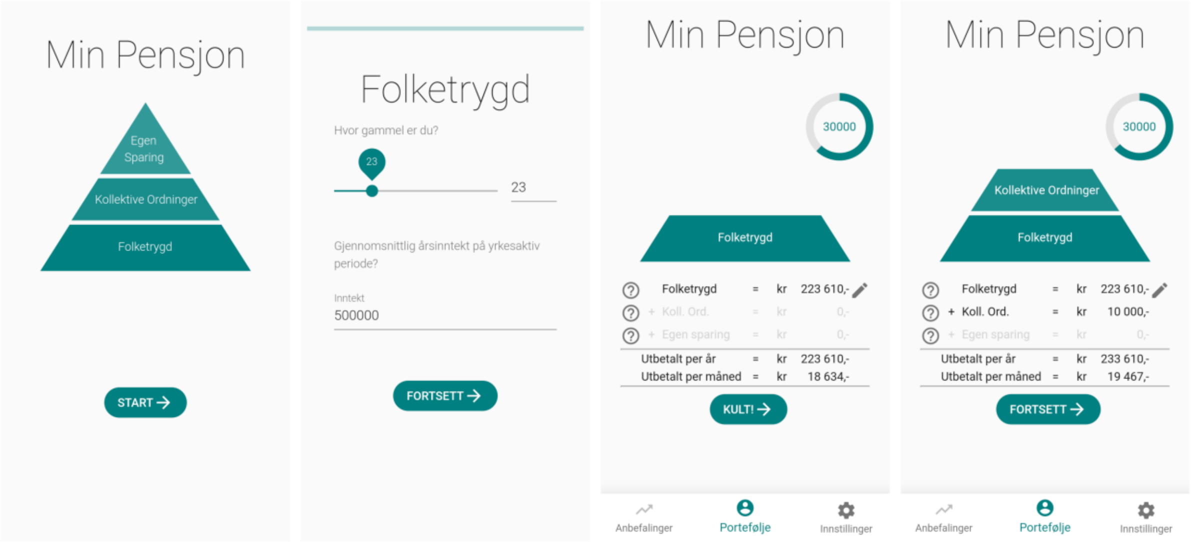
The other flow is the operator. As Kubernetes follow a declarative infrastructure model, the operator defines the whole Kubernetes cluster, with all the manifests, in a configuration Bitbucket Repository. This repository contains the single source of truth of the desired state of the cluster. All changes done by the Flux operator will also be committed back to the config repository. Consequently, whether an operator or the Flux operator updates the cluster state, everything will be tracked in Git.

This makes it easy to define the project's two applications, the client and the server, and how they should be deployed in the cluster. The deployment manifest includes which ports should be available, what amount of memory and CPU and how traffic should be routed to the applications making them accessible externally from a web browser.

4.6 Application flow

By using the principals of Domain Driven Design, the project team could use event storming as a fast design technique to focus on the business process rather than on low level implementation details such as data properties. With this setting in mind, the project team could discuss around events in the application and define boundaries for different context. By focusing on the events such as “calculate national insurance” and “calculate individual savings for pension”, the views and commands needed to achieve the event and the bounded context was easy to find and, in most cases, they naturally appeared in the discussion.

For instance, to fulfill the “calculate national insurance” event, a survey needs to be populated with the age and the average yearly salary for the active working years. By also asking for the desired monthly payout, combined with the output from calculation national insurance, the input for calculating the individual saving goal is reached.



Hvor gammel er du? 23

Gjennomsnittlig årsinntekt på yrkesaktiv periode?

Inntekt
500000

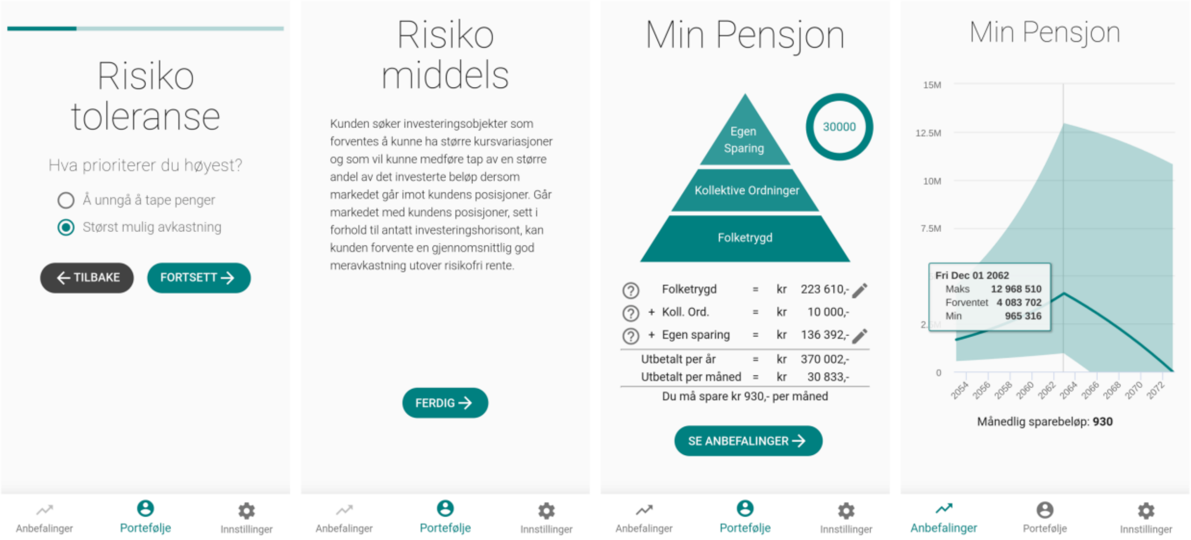
?	Folketrygd	=	kr	223 610,-
?	+ Koll. Ord.	=	kr	0,-
?	+ Egen sparing	=	kr	0,-
	Utbetalt per år	=	kr	223 610,-
	Utbetalt per måned	=	kr	18 634,-

?	Folketrygd	=	kr	223 610,-
?	+ Koll. Ord.	=	kr	10 000,-
?	+ Egen sparing	=	kr	0,-
	Utbetalt per år	=	kr	233 610,-
	Utbetalt per måned	=	kr	19 467,-

Figure 4.7 Screenshots of first part of the flow

The application flow is well defined, and the user is following a path through the whole application. The user needs to go through two surveys to get to the final recommendation for the own savings. After the user has gone through each step they can always go back and edit their answers. And since everything is cached on the user’s device they will only need to go through the whole flow once. When the user returns later the answers and settings are saved making it fast and easy to experiment with different inputs.

The first survey contains inputs for calculating the national insurance and setting the goal of monthly payout that will be the input to the individual savings goal.



Risiko toleranse

Hva prioriterer du høyest?

○ Å unngå å tape penger

● Størst mulig avkastning

Risiko middels

Kunden søker investeringsobjekter som forventes å kunne ha større kursvariasjoner og som vil kunne medføre tap av en større andel av det investerte beløp dersom markedet går imot kundens posisjoner. Går markedet med kundens posisjoner, sett i forhold til antatt investeringshorisont, kan kunden forvente en gjennomsnittlig god meravkastning utover risikofri rente.

Min Pensjon

	Folketrygd	=	kr	223 610,-
+	Koll. Ord.	=	kr	10 000,-
+	Egen sparing	=	kr	136 392,-
	Utbetalt per år	=	kr	370 002,-
	Utbetalt per måned	=	kr	30 833,-

Du må spare kr 930,- per måned

Min Pensjon

Fri Dec 01 2062

Maks	12 968 510
Forventet	4 083 702
Min	965 316

Månedlig sparebeløp: **930**

Figure 4.8 Screenshots of the second part of the flow

The second survey is for calculating the risk tolerance for the user before giving recommendations. The questions and answers in the survey contains weights that is used in a formula to calculate a score between 0-100. This score will be matched against the limits of the different risk groups provided by the recommendation framework in RPM. The selected risk group will be included for calculating the individual saving's goal.

5 EVALUATION

This section describes the different evaluation methods used to evaluate the project result, the methods that are used to ensure project result was obtained with high quality, as well as the results from theses evaluations. The evaluation methods will include functional requirements, non-functional requirements, evaluation by project owner, functional testing and evaluation by different metrics.

5.1 Evaluation methods

In order to properly evaluate the developed application, we need to look at several different ways of doing this. This section presents the evaluation methods we have made use of.

5.1.1 Functional requirements

Taking the functional requirements into account, the team made implementation decisions throughout the project that deviated slightly from the original requirements. Copied from section 3.1.3.

A user should be able to create a savings goal first time the user visits the application.
To create a savings goal in the context of actually creating a plan that was disbanded in favor of displaying a possible return on savings. This was done for several reasons, where the most important was for the application to be able to be stateless.

The user must provide two out of three key figures: initial capital, monthly savings or monthly payout.
In order to keep the scope of the application within the timeframe of a bachelor assignment this requirement is not fully met. The user now must provide three inputs; estimated yearly salary, current age and desired monthly payout.

The application sets various values as standard, including retirement age (67 years), payout period (10 years), risk profile set to medium (unless they have performed a risk tolerance analysis) and a selection of founds.
The application sets default retirement age to 67 years, payout period to 10 years and the user must complete a risk tolerance analysis to get a risk profile (low, medium or high).

The user will then be redirected and shown key figures and expected savings requirements. If the user is already registered and has created a savings goal, the user will be redirected to an overview of the user portfolio.
The user does not need to register and there is no check to see if there is an existing savings goal on the user object, however if the user has already performed all the steps, the data will persist in the cache and is available once the user opens the application another time. There is a check in place that checks whether all the required data exists, and the user will not have to enter this a second time.

In the portfolio section of the application the user should be able to view key figures, historical numbers, overview of savings goal and returns.
The application shows key figures and concentrates on showing the end user how much the user needs to save in order to reach their goal. This was done in order to keep the information concise and more intuitive.

5.1.2 Non-functional requirements

The non-functional requirements defined in 3.1.4 has been the overall goal to reach for this project.

The project shall deliver an API to interact with data located on the Stacc Insight beta environment. The API will be a wrapper around the existing API using the endpoint visningsrom.stacc.com.
The GraphQL server is setup to fetch data from the API exposed at *visningsrom.stacc.com,* this is the main source of data for the client. The client uses the GraphQL to fetch data from the GraphQL through its API.

One of the goals is to reduce the number of API requests between the client and the server. This can be solved in many ways by utilizing caching techniques on both the server and the client and combine multiple request to bigger ones to reduce the RTT on unstable networks.
By using *InMemoryCache* enables the client to cache data from certain queries and store the information to be used in the future. This will be discussed further in section 5.1.5.

It should be easy and intuitive to fill out the initial values when the user first starts the application. The information required for the user to use the application for the first time must be kept to a minimum. And the information the user provides should be easy to change and modify.
The user is required to enter 3 key values; estimated yearly salary, age and desired monthly payout. In addition to this they would need to perform a risk tolerance analysis which consists of 5 questions.

The graph provided for the user, should display key figures be easy to read and understand its content. The graph should display worst case, best case and predicted case for the saving goal.
In the end the user is provided with a graph showing the worst case, best case and predicted case for the savings plan. The user will also be displayed how much they'll need to save each month in order to reach that goal.

5.1.3 Evaluation by project owner

At an early stage in the project, it was our goal to make a savings calculator that was easily accessible for the end user. However, after several meetings with the project owner on how to scope the project correctly, the project eventually shifted focus over to pension savings. After this, the project owner again was able to help scope the project as some key modules in RPM were made stateless, meaning that the project team would not have to do mutations on the data inside RPM. Instead, we could use this as a data source and an engine for running heavy calculations.

Developers at Stacc Insight was used frequently throughout the project as resources, and they provided useful feedback on points that were unknown and unsolvable by us. These meetings were valuable to us as the feedback given helped to avoid common pit falls and bugs, saving us for many hours of debugging.

By using the front-end developers/UX designers we received invaluable feedback on *their* "lessons learned" on how a user interacts with content in an application. E.g. they suggested that we should show the user which step he's at in going through the application, which resulted in the triangle filling up as the user progressed through the flow.

5.1.4 Functional testing

Functional testing involves testing that the function or method created has the desired output with the given input. The team verified this as a part of writing the code that made up this solution. As writing tests for this is very complicated and would require quite some time, we as developers, decided instead on verifying with each other after implementing features that they had the desired result.

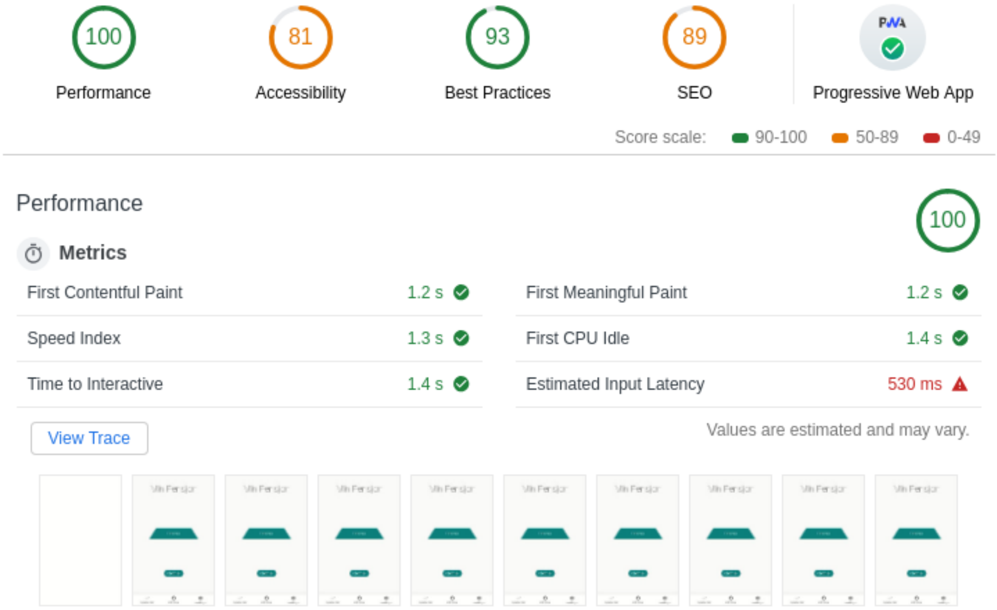
We started early with functional testing of the application to ensure that the code worked as intended. It was discussed early whether to write unit tests or end to end testing for the application. This was, however, not feasible in the timeframe of this project.

The ideal solution would include systematically and properly documented testing, where the tests should revolve around the user stories. The user stories could be broken down into steps where the tests would quickly identify any step not working properly.

5.1.5 Evaluation by metrics

Lighthouse is an open source webpage benchmarking tool from Google Chrome team. Lighthouse audit's the web page and ranks the page out of 100. A score above 80 is considered *Good* whereas anything less than that calls for further optimizations.

In figure 5.1 the overall score by each category is shown together with the summary of the app performance. The lighthouse report was running with simulated 3G network with 1500kb/s download speed, 550ms latency and 4x slowdown of the computers CPU to simulate a mobile device.



100 Performance

81 Accessibility

93 Best Practices

89 SEO

PWA ✓ Progressive Web App

Score scale: 🟩 90-100 🟧 50-89 🟥 0-49

Performance 100

Metrics

First Contentful Paint	1.2 s ✓	First Meaningful Paint	1.2 s ✓
Speed Index	1.3 s ✓	First CPU Idle	1.4 s ✓
Time to Interactive	1.4 s ✓	Estimated Input Latency	530 ms ▲

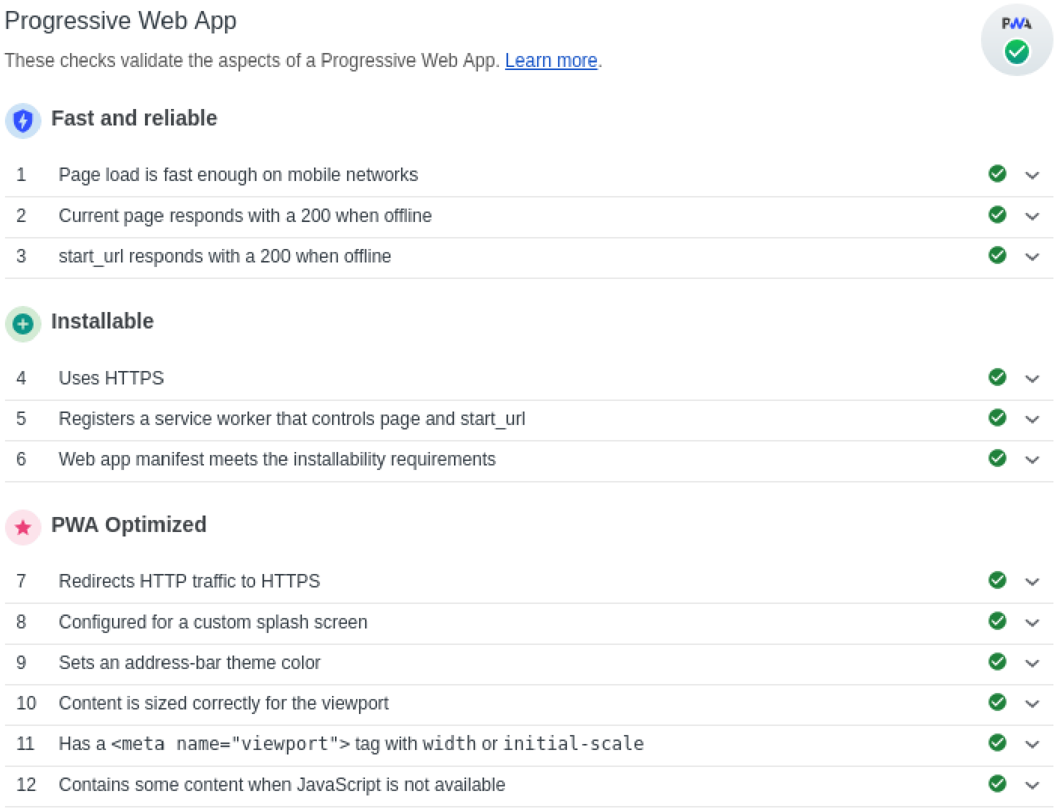
View Trace

Values are estimated and may vary.

Figure 5.1 Lighthouse Audit overall test with simulated Fast 3G and 4x CPU slowdown

The app performance test covers how fast the application loads, how long it takes for the DOM to render and when the user can interact with the application. As the client is written in VueJS, it's a SPA (Single Page Application), this is often a huge problem with loading times because the app comes bundled with all the JavaScript needed to run the whole app. This works fine when the app is loaded because you have access to everything the app needs to run, but the initial load will often be very slow because of the huge bundle size of the JavaScript. This was solved by using Webpack 4 dynamic imports, and by splitting up the JavaScript in smaller bundles and load them later when needed. Our chosen approach has the drawback of increased network requests, that is a disadvantage on devices with low network bandwidth or unstable connection. However, the approach will improve the initial load that is very important when it comes to user experience.

The PWA section of the lighthouse report includes a checklist to help developers create the best PWA experiences and what the Google Chrome team think takes to be a baseline PWA. The lighthouse report is tight coupled to the performance part of the report such as loading time om mobile network and HTTPS support. It also has other aspects such as the page needs to return status code 200 when the page is offline. This is because a PWA application should always be accessible offline. In our application this is taken care of by the service worker that caches the necessary files and serves them even if no network is available. Figure 5.2 shows an example of how such a report can look.



Progressive Web App

These checks validate the aspects of a Progressive Web App. Learn more.

Fast and reliable

1 Page load is fast enough on mobile networks ✓
2 Current page responds with a 200 when offline ✓
3 start_url responds with a 200 when offline ✓

Installable

4 Uses HTTPS ✓
5 Registers a service worker that controls page and start_url ✓
6 Web app manifest meets the installability requirements ✓

PWA Optimized

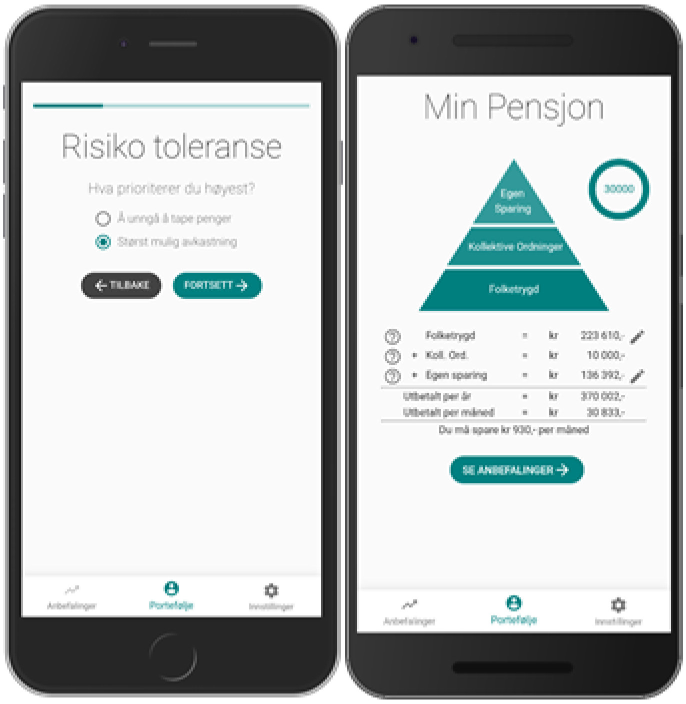
7 Redirects HTTP traffic to HTTPS ✓
8 Configured for a custom splash screen ✓
9 Sets an address-bar theme color ✓
10 Content is sized correctly for the viewport ✓
11 Has a `<meta name="viewport">` tag with `width` or `initial-scale` ✓
12 Contains some content when JavaScript is not available ✓

Figure 5.2 PWA score with simulated Fast 3G and 4x CPU slowdown

Another pure PWA optimizations is that traffic always should be redirected from HTTP to HTTPS. This is taken care of by the gateway running in the Kubernetes cluster. The PWA should be installable by registering a service worker that controls the network flow in the page, and providing a web manifest with a start URL, theme color, custom splash screen, and have an initial scale on the viewport for mobile device support.

5.2 Evaluation results

The current state of the product allows for further development and scalability. The application is built in such a way that it is possible to rewrite large parts of it to suit any adjustment the project owner wants to make in the future. Both the functional and non-functional requirements were implemented, and the we consider these to be fulfilled. Hence, the product is ready for user testing, features can be tested and evaluated by the user group. Figure 5.3 shows screenshots taken from both the iOS (to the left) and the Android (to the right) platform shows examples of what the application looks like in different stages of the application flow.



Risiko toleranse

Hva prioriterer du høyest?

○ Å unngå å tape penger
● Størst mulig avkastning

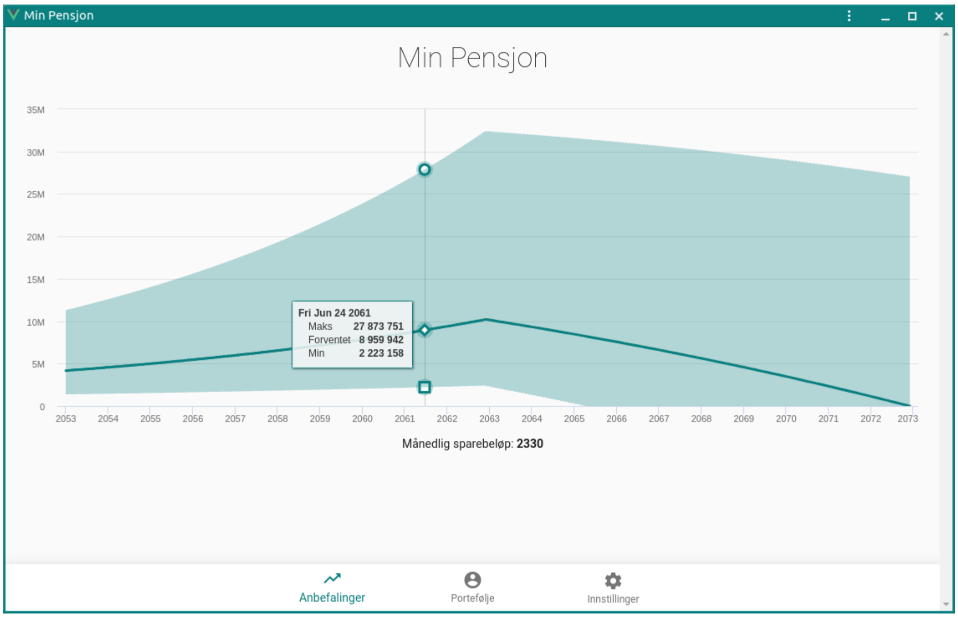
Min Pensjon

ⓘ	Folketrygd	=	kr	223 610,-
ⓘ	• Koll. Ord.	=	kr	10 000,-
ⓘ	• Egen sparing	=	kr	136 392,-
	Utbetalt per år	=	kr	370 002,-
	Utbetalt per måned	=	kr	30 833,-

Du må spare kr 930,- per måned

Figure 5.3 Screenshot of the application on different mobile devices

The application will work on all platforms that support PWAs. As this is essentially a website it will also work in a normal web browser on a desktop client. The effect of this is an incredible versatile solution, accessible on the platform preferred by the user. Figure 5.4 shows how the application runs on a desktop machine.



Min Pensjon

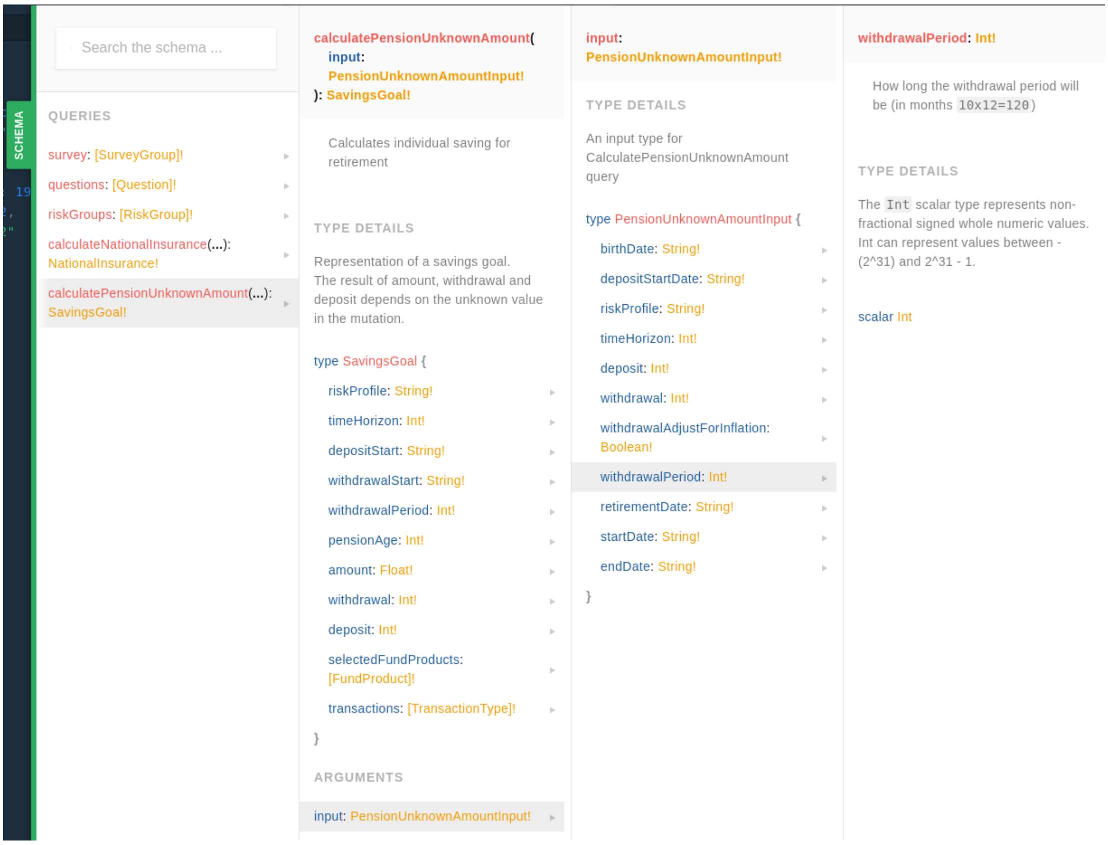
Fri Jun 24 2061
Maks 27 873 751
Forventet 8 959 942
Min 2 223 158

Månedlig sparebeløp: **2330**

Figure 5.4 Screenshot of the application running as PWA on desktop

From a programmer's perspective the application does what it is supposed to do, and the metrics shows good performance of the application and the servers on the backend. The testing done by the project team on the functionality of the application is satisfactory, which is critical in order to have a functional product.

As stated in one of the requirements the project team should deliver an API to work with the endpoints that Stacc has exposed. This was accomplished by implementing GraphQL as a wrapper around everything. One result of this is the extensive API documentation that GraphQL presents. This makes it very easy for other developers to see what is available and how the API is working today.



QUERIES

survey: [SurveyGroup]!

questions: [Question]!

riskGroups: [RiskGroup]!

calculateNationalInsurance(...): NationalInsurance!

calculatePensionUnknownAmount(...): SavingsGoal!

```
calculatePensionUnknownAmount(
  input: PensionUnknownAmountInput!
): SavingsGoal!
```

Calculates individual saving for retirement

TYPE DETAILS

Representation of a savings goal. The result of amount, withdrawal and deposit depends on the unknown value in the mutation.

```
type SavingsGoal {
  riskProfile: String!
  timeHorizon: Int!
  depositStart: String!
  withdrawalStart: String!
  withdrawalPeriod: Int!
  pensionAge: Int!
  amount: Float!
  withdrawal: Int!
  deposit: Int!
  selectedFundProducts: [FundProduct]!
  transactions: [TransactionType]!
}
```

ARGUMENTS

input: PensionUnknownAmountInput!

input: PensionUnknownAmountInput!

TYPE DETAILS

An input type for CalculatePensionUnknownAmount query

```
type PensionUnknownAmountInput {
  birthDate: String!
  depositStartDate: String!
  riskProfile: String!
  timeHorizon: Int!
  deposit: Int!
  withdrawal: Int!
  withdrawalAdjustForInflation: Boolean!
  withdrawalPeriod: Int!
  retirementDate: String!
  startDate: String!
  endDate: String!
}
```

withdrawalPeriod: Int!

How long the withdrawal period will be (in months `10x12=120`)

TYPE DETAILS

The `Int` scalar type represents non-fractional signed whole numeric values. Int can represent values between -(2^31) and 2^31 - 1.

scalar Int

Figure 5.5 Example of development situation with GraphQL. All documentation is available for the developer, and the final query can be used directly in the code

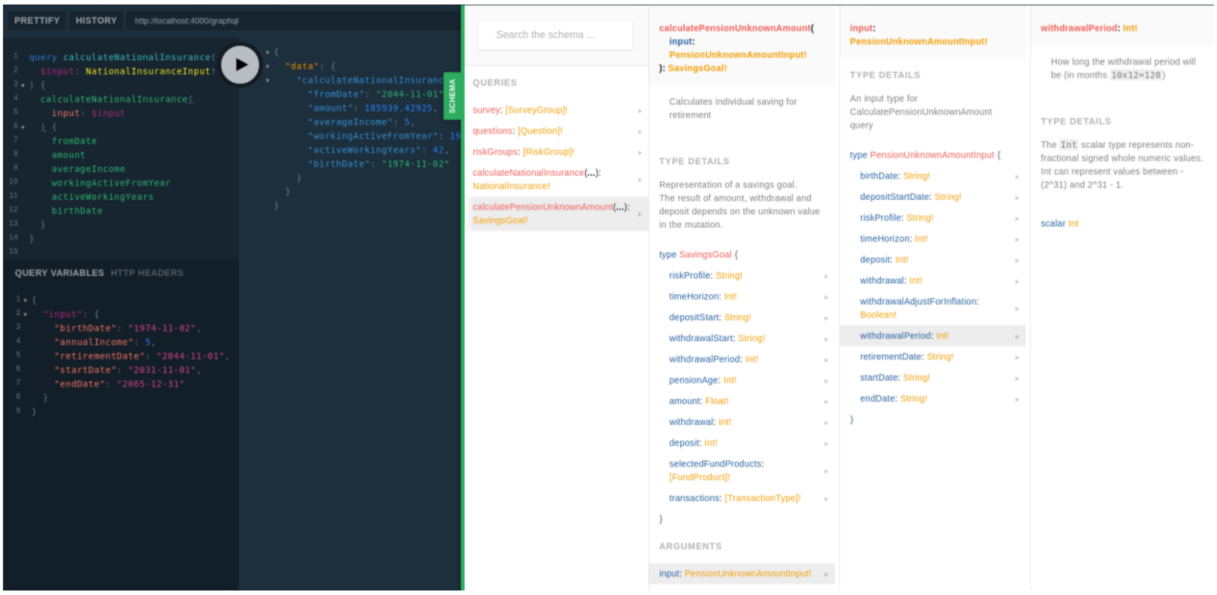
6 DISCUSSION

This section delves on the choices made with regards to the chosen technologies, particularly their strengths and constraints. Finally, the chapter ends with a discussion on the team's collaboration and the roles each team member had.

6.1 Technical solution

RPM is running inside Stacc Insight's network and the project is running in Kubernetes in Azure Cloud. To make the two communicate, the visningsrom.stacc.com endpoint in the beta environment was used. This is a temporary workaround as in the future the whole RPM solution will run in Kubernetes, so this is a temporary workaround.

The choice of using GraphQL instead of traditional REST for querying for data between a client and server has been a great success. The developer experience by using Apollo Playground to explore and test the API before implementing it in the client has increased the development speed.



```
query calculateNationalInsurance(
  $input: NationalInsuranceInput!
) {
  calculateNationalInsurance(
    input: $input
  ) {
    fromDate
    amount
    averageIncome
    workingActiveFromYear
    activeWorkingYears
    birthDate
  }
}
```

```
{
  "data": {
    "calculateNationalInsuranc
      "fromDate": "2044-11-01"
      "amount": 185939.42925,
      "averageIncome": 5,
      "workingActiveFromYear": 19
      "activeWorkingYears": 42,
      "birthDate": "1974-11-02"
    }
  }
}
```

```
{
  "input": {
    "birthDate": "1974-11-02",
    "annualIncome": 5,
    "retirementDate": "2044-11-01",
    "startDate": "2031-11-01",
    "endDate": "2065-12-31"
  }
}
```

survey: [SurveyGroup]!
questions: [Question]!
riskGroups: [RiskGroup]!
calculateNationalInsurance(...): NationalInsurance!
calculatePensionUnknownAmount(...): SavingsGoal!

calculatePensionUnknownAmount(
  input: PensionUnknownAmountInput!
): SavingsGoal!

Calculates individual saving for retirement

Representation of a savings goal. The result of amount, withdrawal and deposit depends on the unknown value in the mutation.

```
type SavingsGoal {
  riskProfile: String!
  timeHorizon: Int!
  depositStart: String!
  withdrawalStart: String!
  withdrawalPeriod: Int!
  pensionAge: Int!
  amount: Float!
  withdrawal: Int!
  deposit: Int!
  selectedFundProducts: [FundProduct]!
  transactions: [TransactionType]!
}
```

input: PensionUnknownAmountInput!

input: PensionUnknownAmountInput!

An input type for CalculatePensionUnknownAmount query

```
type PensionUnknownAmountInput {
  birthDate: String!
  depositStartDate: String!
  riskProfile: String!
  timeHorizon: Int!
  deposit: Int!
  withdrawal: Int!
  withdrawalAdjustForInflation: Boolean!
  withdrawalPeriod: Int!
  retirementDate: String!
  startDate: String!
  endDate: String!
}
```

withdrawalPeriod: Int!

How long the withdrawal period will be (in months `10x12=120`)

The `Int` scalar type represents non-fractional signed whole numeric values. Int can represent values between -(2^31) and 2^31 - 1.

scalar Int

Figure 6.1 Example of development situation with GraphQL. All documentation is available for the developer, and the final query can be used directly in the code

All the documentation needed is provided right in the Playground editor. Apollo Playground will always show the latest of what the GraphQL server provides. Auto-completion is also provided both in the Apollo Playground, and by plugins in Visual Studio Code for writing queries. Typescript provide the rest of the type completion for the datatypes by a tool for generating the types and queries directly from the GraphQL server. When a developer has created a query in Apollo Playground it can be copied directly into the code base and used.

In order to make it easier to use GraphQL in Vue, a library called Vue-Apollo was used. This is an Apollo Client wrapper to use GraphQL inside Vue components effortlessly.

6.1.1 Local state management

Apollo Client v2.5 provides a library called Apollo Link State that makes it possible to manage the application's local state with local GraphQL queries. By default, Apollo Client also holds the state of the remote queries to be able to utilize caching.

In the initial project setup for the client, Vuex store was chosen as it is the traditional way of managing state in a Vue application. When the Apollo Client stores the state of the remote queries, and Vuex was used to store the result from Apollo in Vuex Store, there was a problem by managing two storages. All remote queries were stored in both storages resulting in duplicate unnecessary caching. A choice was made to remove the Vuex Store and manage the local application state in Apollo Link State. This led the project team to use GraphQL queries for both local and remote data, making GraphQL the universal interface of communication in the entire application.

Vue-Apollo creates some boiler plate code inside the Vue components that was earlier in separate files with the Vuex implementation.

Apollo Client Developer Tools a Google Chrome extension also made it possible to query local state in the client.

6.1.2 Bleeding edge

Several libraries used in this project both on the server and the client side is bleeding edge with a lifespan from only weeks to a couple of months. This leads to little documentation, few examples, little to no help on forums like Stack Overflow with error codes. On the overall project state, this has led to increased developing speed because the project team needed to familiarize themselves with the libraries and technologies. Some of the libraries also had bugs that forced the project team to find new solutions and workarounds to make it work. Luckily some of the critical bugs where solved in the project period, making it possible to refactor.

6.1.3 TypeScript

TypeScript was used both on the server and the client side. By making use of types and interfaces the developer experience was amazing. All the normal type errors in JavaScript disappeared, and Visual Studio Code provided excellent support for TypeScript by leveraging intelli-sense, auto completing and refactoring possibilities. By using tools like json2ts and graphql-code-gen, support for TypeScript types for all queries, REST responses and arguments was added. The TypeScript compiler also detects any errors in the code, making it safer to have a continuous delivery pipeline and be more confident that code will not break once it reaches the end-user.

TypeScript also introduces classes and annotations. This makes it very similar to other language like C++, Java and Kotlin the project team was most familiar with.

6.1.4 MVVM

Vue uses the Model-View-ViewModel (MVVM) pattern. MVVM architecture is built around three elements: Models, Views and VMs. Models in VueJS are simple constructs witch hold data used by the VM and the app. The models have no logic apart from data validation logic and they don’t access services to retrieve or save data. Views are active and render the data contained by the VM. They are defined by “templates” which are either custom DOM tags or plain HTLM. The VM’s hold all the business login required to manipulate the data used by the app. By leveraging these principals, the server code and client code could be totally separate. The MVVM pattern in Vue makes it very easy to manipulate and update data in the application and changes made to the application state will be reflected on all views and subcomponents. Figure 6.2 visualises how the MVVM concept works in VueJS.

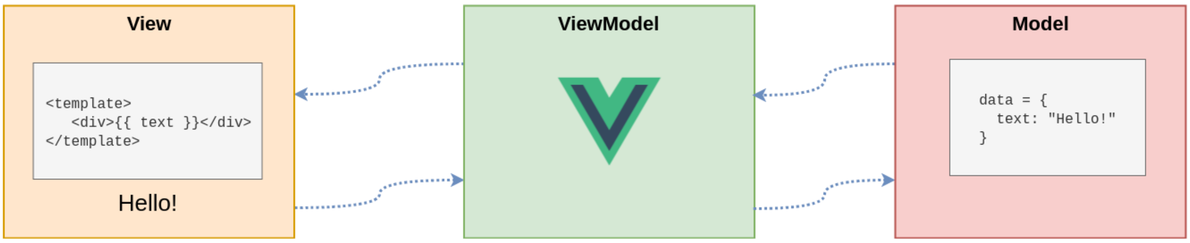


Figure 6.2 MVVM pattern in VueJS

The use of Vue in the project has made it possible for us to easily create components with dummy data without thinking on how the data should be gattered, and rather thinking about how to present the data in the component. This has made it possible to develop the client and the server independently, making system highly flexible.

By using state management referenced in 6.1.1 there is only one place to handle data. Without this state management, passing data between components would always go via a parent component and down to a child component causing a pattern that is very difficult to maintain.

6.2 Roles and collaboration

At the beginning of this, we agreed on that there would be no clear role definition put on anyone, and there was never a project leader appointed. This resulted in our project team working as equals. In the first weeks the team spent a lot of time together learning the different technologies and discussing on how to set things up. In theory we would have been more effective with a clearer division of tasks, but all the project members wanted to join in on everything and learn as much as possible.

As the structure of the setup became clearer, the team decided on a split in tasks in order to work more efficiently, only getting together a couple of times a day to discuss what had been done and if anyone were facing any challenges. These tasks were split using the principles of Domain Driven Design, using the functional requirements defined in section 3.1.3 as the basis. By using this we were always focused on small areas of the application, and not interfering with each other's code. This proved to be a great success, and we made a lot of progression in the main part of the project.

There is a principle called *The Pareto Principle* that can be interpreted in the context of programming like this: “20% of the input creates 80% of the result” (Pareto-Principle, 2007).
In hindsight, the team observed that this principle applied to the project, where large portions of the application was build using very little time. The rest of the time was spent on working out issues together. This did not take a lot of time, but 3 people working together on one problem will quickly increase the number of hours spent on the application.

7 CONCLUSIONS AND FURTHER WORK

The result meets all the requirements for what was defined as the MVP (Minimum viable product). The initial goals were, however, to also meet all the requirements in the *Should have* and *Could have* categories. Given more time, this wouldn't be much of a problem to implement, even without restructuring the existing code.

As with the old saying, "There are several ways to get to Rome", there have been proposed many different solutions over the course of this project. All the feasible ideas have been discussed in plenary; that is not to say that the correct choice was made every time, but necessary reach a subsidiary goal, e.g. mobile friendliness.

Throughout the project, there has been a lot of focus on communication and discussion. Every time a problem arose, it was soon resolved through communication and discussion. Either by weighing pros and cons, or discussing relevance to the project, and most importantly – helping others understand one's points in a respectful manner and wanting to understand theirs.

Over the course of the project, the project team gradually got better at communicating. This led to better solutions, faster. By discussing different frameworks and technologies, every participant received insight into the why's, the when's and the how's of implementing them. This has led to a better general understanding of developing in general, but more specifically, of developing progressive and regular web apps.

From our point of view, the prototype we have developed is a viable and functional product and it got this way through hard work and determination. We have used some of the hottest and *fresh out of the box* frameworks and technologies currently available to achieve what we have done. Most of what is going on behind the scene of our prototype won't be visible from the user interface, consequently so it may be a bit hard to elaborate on how things work in layman's terms.

As mentioned earlier, the project scope and definition has shifted since the beginning. This has also affected its usefulness to the project owner. The initial project would have been highly relevant for Stacc insight, but ultimately it was changed to something that was more relatable to what we have been working with at HVL. From the start the code was structured to be easily maintainable, rewritable and expandable, providing a great foundation for further work. The technology used for the application will be highly relevant to the future of Stacc. Early in the project we were asked by Stacc to arrange a short presentation on the subject and a speakers' corner for their employees to discuss some of the benefits and constraints, hopefully they will find it inspiring.

7.1 Future Work

If the work was continued, we would recommend implementing everything in the MoSCoW analysis in section 3.1.1. This would ensure a well-functioning and feature rich application with great potential for both Stacc Insight and their customers. We would also recommend Stacc Insight to migrate all their functions inside RPM to the cloud instead of an on-premise closed environment they operate with today. This application may not have the greatest value for Stacc Insight at first sight, but the ideas and technologies used would make it possible for Stacc Insight to expand the portfolio of services they offer. E.g. by creating stateless applications using already robust infrastructure.

7.2 The Risk list

R1+R3+R4 – Initially, we intended to add a “fake” stateless support by adding and deleting dummy person-objects. This would have made the application seem stateless based on the number of inputs but would also have made it slower than desired. The best solution was to alter the source-code of RPM and add stateless support to the features needed by the new solution. This would, however, have been very difficult for us to do. Luckily, one of the engineers at Stacc Insight, who already was very familiar with the framework, offered to help and everything worked out.

R2 – The stateless problem extended to R2 as well. However, the severity of the consequences was much smaller.

R5 – Lack of knowledge has been a frequent issue during the development. This risk was accounted for early in the process by setting aside time to gain the appropriate knowledge required to complete the tasks. As a result, the continuous delivery was barely affected by this risk.

8 LITERATURE/REFRENCES

Agile-methodology, 2018. *Agile methodology.* [Internett]
Available at: https://project-management.com/10-key-principles-of-agile-software-development/
[Funnet 21 02 2019].

Ansible, 2019. *Ansible.* [Internett]
Available at: https://github.com/ansible/ansible
[Funnet 16 2 2019].

Apollo, 2019. *Apollo Playground.* [Internett]
Available at: https://www.apollographql.com/docs/apollo-server/features/graphql-playground
[Funnet 27 04 2019].

Azure, u.d. *Microsoft Azure: Cloud Computing Platform & Services.* [Internett]
Available at: https://azure.microsoft.com/
[Funnet 16 2 2019].

Bitbucket, u.d. *Bitbucket Pipelines.* [Internett]
Available at: https://confluence.atlassian.com/bitbucket/get-started-with-bitbucket-pipelines-792298921.html
[Funnet 5 Mars 2019].

Chang, L., 2017. *Digital Trends.* [Internett]
Available at: https://www.digitaltrends.com/mobile/americans-download-app/
[Funnet 6 Mars 2019].

Copes, F., n.d. *Aligator.io.* [Online]
Available at: https://alligator.io/js/service-workers/
[Accessed 24 04 2019].

Dascalescu, D., 2018. *Why "Progressive Web Apps vs. native" is the wrong question to ask.* [Internett]
Available at: https://medium.com/dev-channel/why-progressive-web-apps-vs-native-is-the-wrong-question-to-ask-fb8555addcbb
[Funnet 6 Mars 2019].

Docker, u.d. *Docker.* [Internett]
Available at: https://www.docker.com/
[Funnet 17 2 2019].

Docker, u.d. *Docker.* [Internett]
Available at: https://www.docker.com/
[Funnet 16 2 2019].

Docker, u.d. *What is a Container?.* [Internett]
Available at: https://www.docker.com/resources/what-container
[Funnet 5 Mars 2019].

ESMA, u.d. *European Securities and Markets Authority.* [Internett]
Available at: http://en.wikipedia.org/wiki/European_Securities_and_Markets_Authority
[Funnet 18 2 2019].

Hagelund, A. & Grødem, A. S., 2017. Build Your Own Pension: Framing Pension Reform and Choice in Newspapers.. *Journal of Aging & Social Policy*, 29 Mars, pp. 218-34.

IntelliJ, u.d. *IntelliJ IDEA.* [Internett]
Available at: https://www.jetbrains.com/idea/download/
[Funnet 16 2 2019].

Jenkins, u.d. *Jenkins.* [Internett]
Available at: http://jenkins-ci.org/content/jenkins
[Funnet 16 2 2019].

KLP, 2018. *Dette bør du vite om pensjon som ung.* [Internett]
Available at: https://www.klp.no/om-klp/dette-b-r-du-vite-om-pensjon-som-ung-1.40258
[Funnet 18 03 2019].

Kubernetes, u.d. *kubernetes/kubernetes.* [Internett]
Available at: http://github.com/kubernetes/kubernetes/
[Funnet 16 2 2019].

Lean-Pathways-inc, 2011. *http://www.leansystems.org.* [Internett]
Available at: http://www.leansystems.org/images/Lean_Pathways_Lean_Manifesto.pdf
[Funnet 21 02 2019].

MiFID, u.d. *MiFID II.* [Internett]
Available at: https://www.fca.org.uk/markets/mifid-ii
[Funnet 18 02 2019].

NFR, u.d. *Non-functional requirements.* [Internett]
Available at: https://en.wikipedia.org/wiki/Non-functional_requirement
[Funnet 5 Mars 2019].

OpenAPI, u.d. *OpenAPI Specification.* [Internett]
Available at: http://en.wikipedia.org/wiki/OpenAPI_Specification
[Funnet 16 2 2019].

Pareto-Principle, 2007. *Understanding the Pareto Principle (The 80/20 Rule).* [Internett]
Available at: https://betterexplained.com/articles/understanding-the-pareto-principle-the-8020-rule/
[Funnet 27 04 2019].

service-worker.js, n.d. [Online]
Available at: https://serviceworke.rs/strategy-cache-and-update.html
[Accessed 24 04 2019].

VSCode, u.d. *Visual Studio Code.* [Internett]
Available at: https://code.visualstudio.com/
[Funnet 16 2 2019].

9 APPENDIX

9.1 Risk list

Risk nr	Description	Probability **(S)** Can occur	Consequence **(C)** Causes	Product **(S x C)**	Fallback	Consequences of fallback
R1	Fetch "risk analysis" and "risk survey" stateless from RPM	5	3	15	Not stateless	Slower application
R2	Be able to calculate national insurance with RPM stateless	2	3	6	Calculate in client	Not as wide support
R3	Be able to calculate individual savings and get recommendations stateless from RPM	5	5	25	Not stateless	Slower application
R4	Make the application not dependent on a customer object in RPM	4	2	8	Not stateless	Slower application
R5	Lack of knowledge	4	2	8	Simplify the project	Not the desired application

Figure 9.1 Risk list

R1 To be able to deliver recommendations a survey must be done to map the end-user knowledge about risk tolerance.

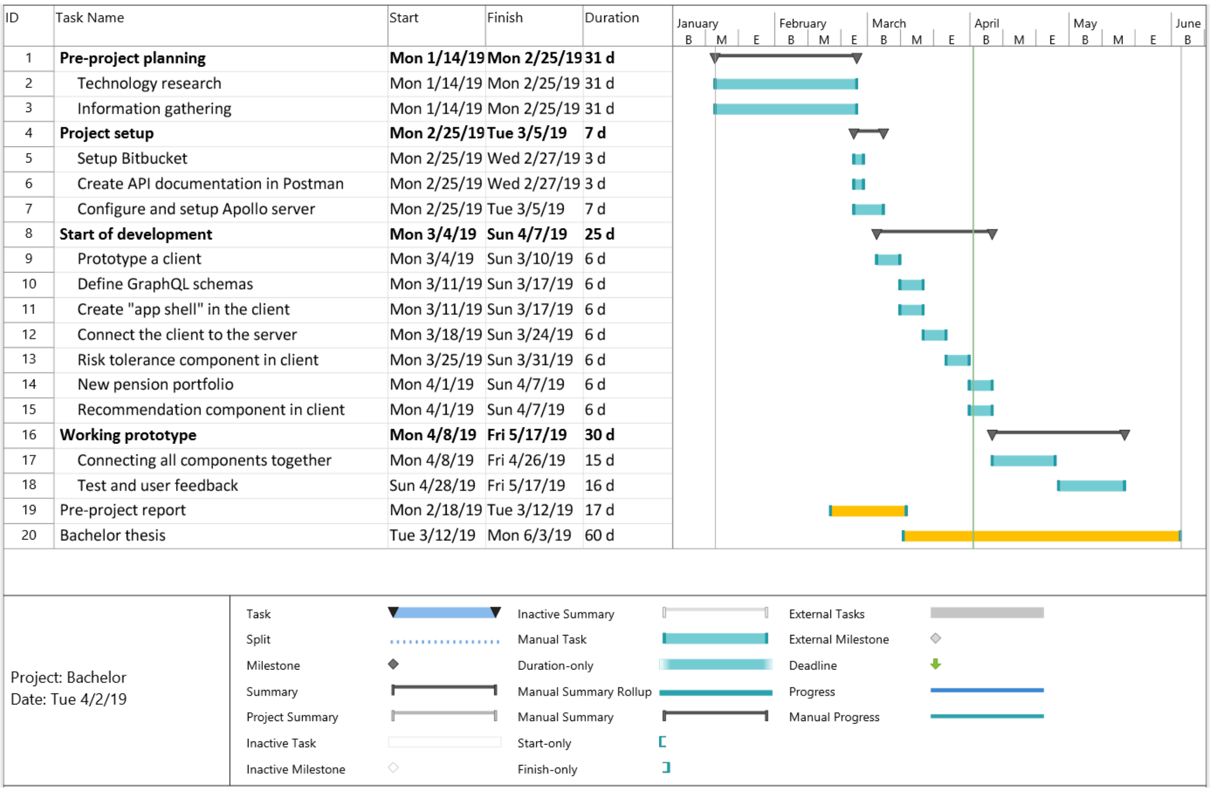
R2 RPM is built around a financial advisor and to be able to calculate national insurance stateless without a customer object will decrease the complexity in the application and provide a much faster application.

R3 Same as **R2** but to be able to calculate individual savings and get recommendation **R1** must also be achieved. Without a risk group a recommendation cannot be done.

R4 If it's possible the application should not be dependent on a customer object in RPM. The domain model around a financial advisor in RPM does not fit the domain of the application. Also, by making the application not depend on anything it will work as a stateless calculator, there will be no user to authenticate and to hold information about. The local state off the client application itself can be persisted as cache.

R5 The risk of lack of knowledge could slow down the project and not make the project team be able to deliver in time or at all. Some fallback to the risk would be to simplify the project or increase the knowledge to reach the desired level.

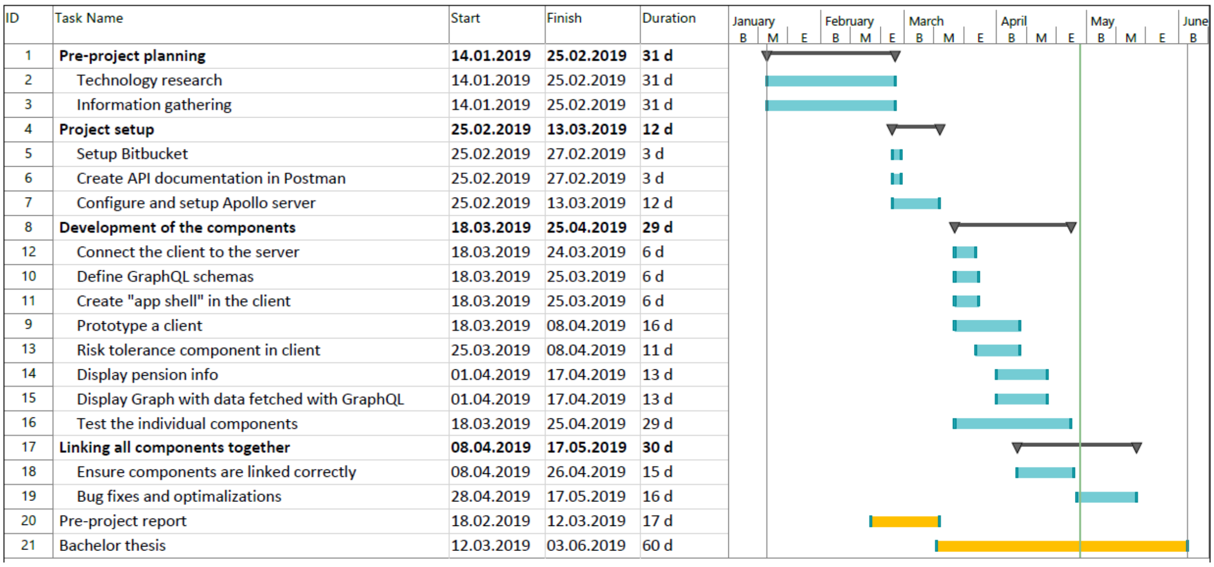
9.2 GANTT diagram



ID	Task Name	Start	Finish	Duration
1	**Pre-project planning**	**Mon 1/14/19**	**Mon 2/25/19**	**31 d**
2	Technology research	Mon 1/14/19	Mon 2/25/19	31 d
3	Information gathering	Mon 1/14/19	Mon 2/25/19	31 d
4	**Project setup**	**Mon 2/25/19**	**Tue 3/5/19**	**7 d**
5	Setup Bitbucket	Mon 2/25/19	Wed 2/27/19	3 d
6	Create API documentation in Postman	Mon 2/25/19	Wed 2/27/19	3 d
7	Configure and setup Apollo server	Mon 2/25/19	Tue 3/5/19	7 d
8	**Start of development**	**Mon 3/4/19**	**Sun 4/7/19**	**25 d**
9	Prototype a client	Mon 3/4/19	Sun 3/10/19	6 d
10	Define GraphQL schemas	Mon 3/11/19	Sun 3/17/19	6 d
11	Create "app shell" in the client	Mon 3/11/19	Sun 3/17/19	6 d
12	Connect the client to the server	Mon 3/18/19	Sun 3/24/19	6 d
13	Risk tolerance component in client	Mon 3/25/19	Sun 3/31/19	6 d
14	New pension portfolio	Mon 4/1/19	Sun 4/7/19	6 d
15	Recommendation component in client	Mon 4/1/19	Sun 4/7/19	6 d
16	**Working prototype**	**Mon 4/8/19**	**Fri 5/17/19**	**30 d**
17	Connecting all components together	Mon 4/8/19	Fri 4/26/19	15 d
18	Test and user feedback	Sun 4/28/19	Fri 5/17/19	16 d
19	Pre-project report	Mon 2/18/19	Tue 3/12/19	17 d
20	Bachelor thesis	Tue 3/12/19	Mon 6/3/19	60 d

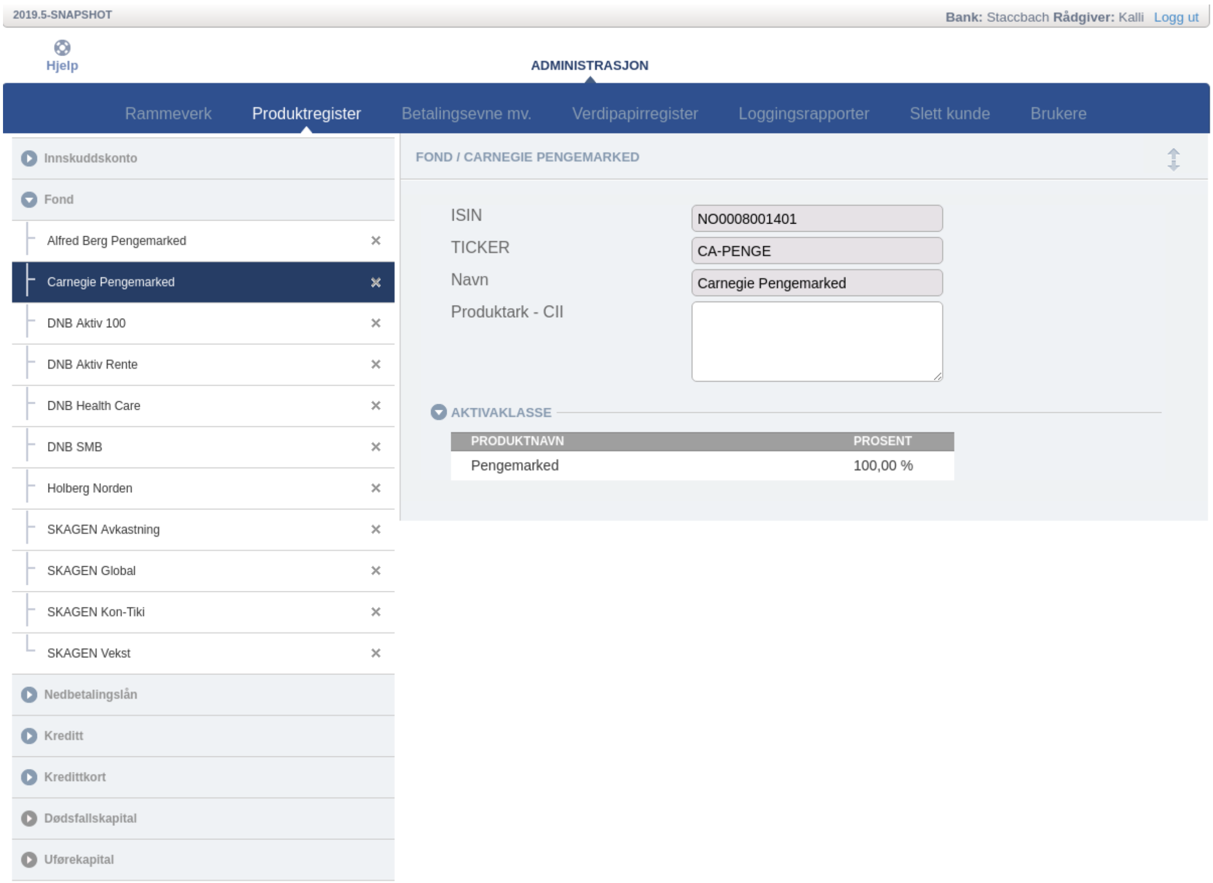
Project: Bachelor
Date: Tue 4/2/19

Figure 9.2 Gantt diagram



ID	Task Name	Start	Finish	Duration
1	**Pre-project planning**	**14.01.2019**	**25.02.2019**	**31 d**
2	Technology research	14.01.2019	25.02.2019	31 d
3	Information gathering	14.01.2019	25.02.2019	31 d
4	**Project setup**	**25.02.2019**	**13.03.2019**	**12 d**
5	Setup Bitbucket	25.02.2019	27.02.2019	3 d
6	Create API documentation in Postman	25.02.2019	27.02.2019	3 d
7	Configure and setup Apollo server	25.02.2019	13.03.2019	12 d
8	**Development of the components**	**18.03.2019**	**25.04.2019**	**29 d**
12	Connect the client to the server	18.03.2019	24.03.2019	6 d
10	Define GraphQL schemas	18.03.2019	25.03.2019	6 d
11	Create "app shell" in the client	18.03.2019	25.03.2019	6 d
9	Prototype a client	18.03.2019	08.04.2019	16 d
13	Risk tolerance component in client	25.03.2019	08.04.2019	11 d
14	Display pension info	01.04.2019	17.04.2019	13 d
15	Display Graph with data fetched with GraphQL	01.04.2019	17.04.2019	13 d
16	Test the individual components	18.03.2019	25.04.2019	29 d
17	**Linking all components together**	**08.04.2019**	**17.05.2019**	**30 d**
18	Ensure components are linked correctly	08.04.2019	26.04.2019	15 d
19	Bug fixes and optimalizations	28.04.2019	17.05.2019	16 d
20	Pre-project report	18.02.2019	12.03.2019	17 d
21	Bachelor thesis	12.03.2019	03.06.2019	60 d

9.3 RPM



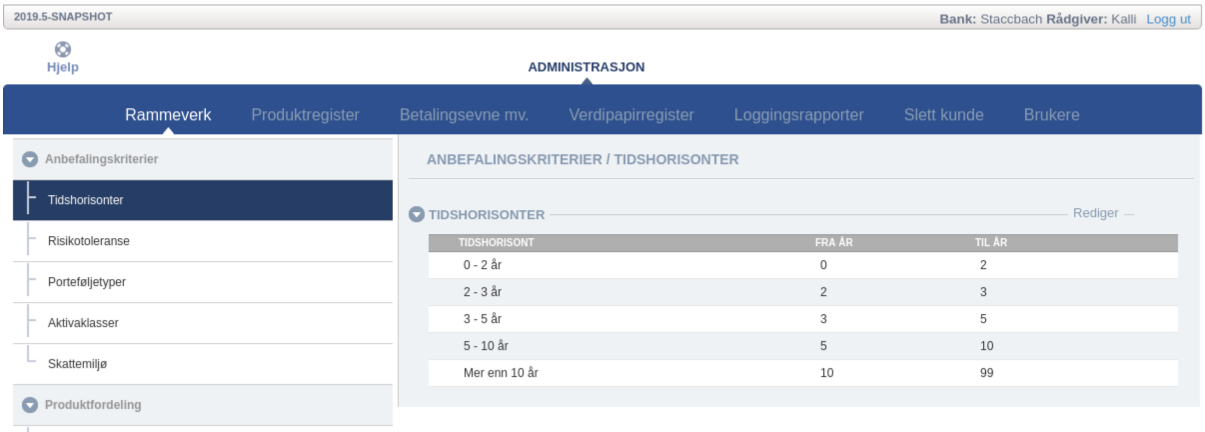
FOND / CARNEGIE PENGEMARKED

ISIN	NO0008001401
TICKER	CA-PENGE
Navn	Carnegie Pengemarked
Produktark - CII	

AKTIVAKLASSE

PRODUKTNAVN	PROSENT
Pengemarked	100,00 %

Figure 9.3 Recommendation Framework list of funds

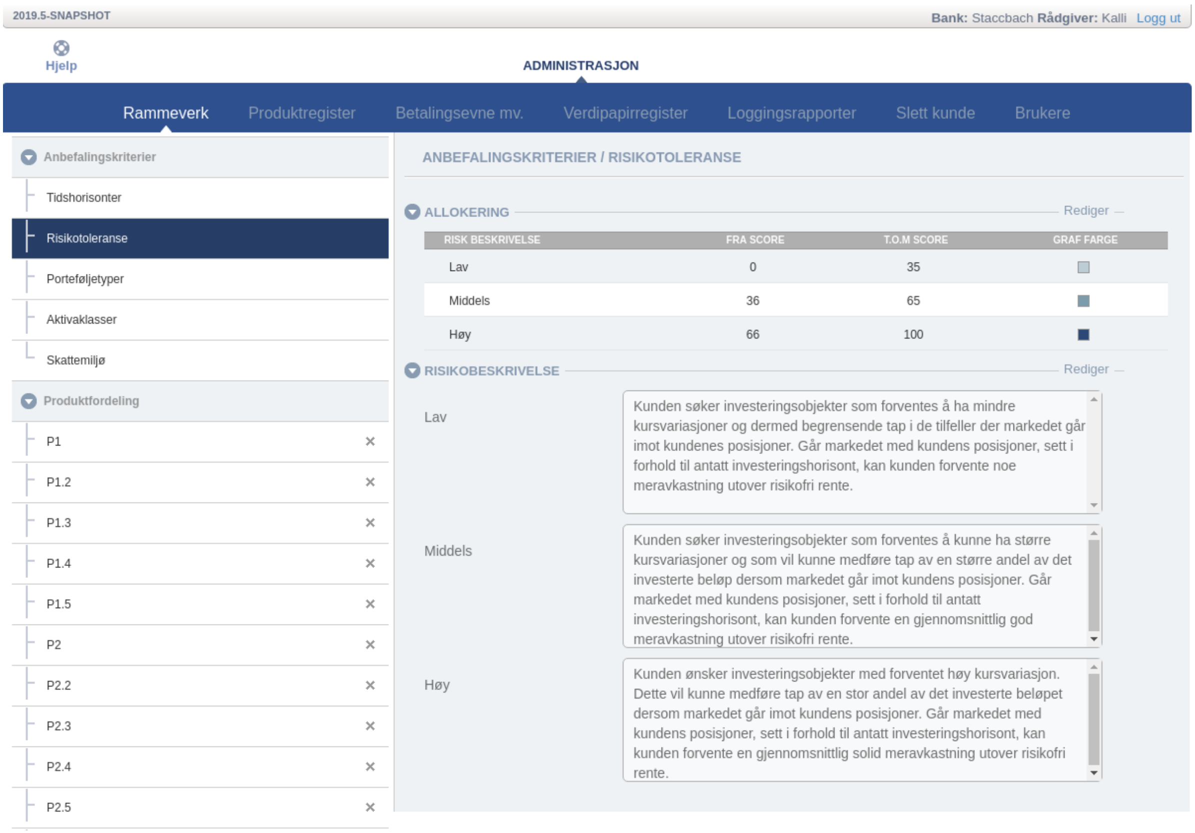


ANBEFALINGSKRITERIER / TIDSHORISONTER

TIDSHORISONTER

TIDSHORISONT	FRA ÅR	TIL ÅR
0 - 2 år	0	2
2 - 3 år	2	3
3 - 5 år	3	5
5 - 10 år	5	10
Mer enn 10 år	10	99

Figure 9.4 Recommendation Framework time horizons



ANBEFALINGSKRITERIER / RISIKOTOLERANSE

ALLOKERING

RISK BESKRIVELSE	FRA SCORE	T.O.M SCORE	GRAF FARGE
Lav	0	35	■
Middels	36	65	■
Høy	66	100	■

RISIKOBESKRIVELSE

Lav

Kunden søker investeringsobjekter som forventes å ha mindre kursvariasjoner og dermed begrensende tap i de tilfeller der markedet går imot kundenes posisjoner. Går markedet med kundens posisjoner, sett i forhold til antatt investeringshorisont, kan kunden forvente noe meravkastning utover risikofri rente.

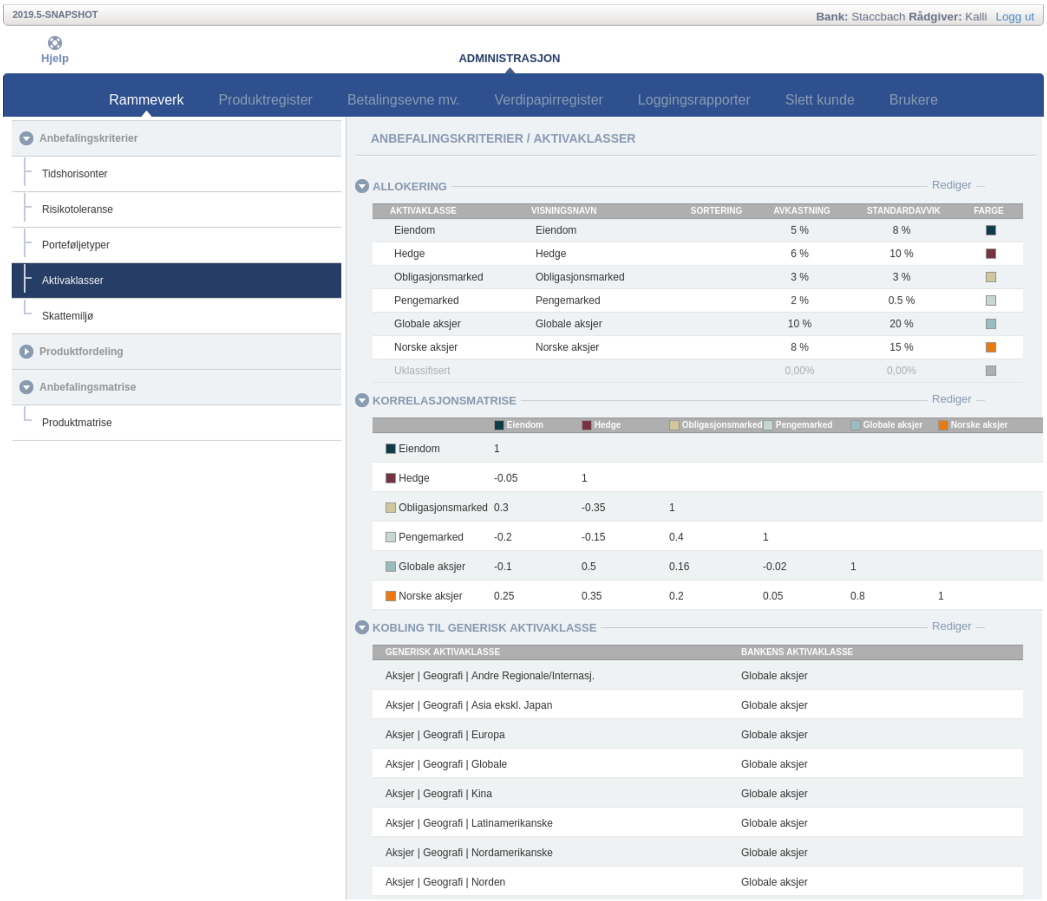
Middels

Kunden søker investeringsobjekter som forventes å kunne ha større kursvariasjoner og som vil kunne medføre tap av en større andel av det investerte beløp dersom markedet går imot kundens posisjoner. Går markedet med kundens posisjoner, sett i forhold til antatt investeringshorisont, kan kunden forvente en gjennomsnittlig god meravkastning utover risikofri rente.

Høy

Kunden ønsker investeringsobjekter med forventet høy kursvariasjon. Dette vil kunne medføre tap av en stor andel av det investerte beløpet dersom markedet går imot kundens posisjoner. Går markedet med kundens posisjoner, sett i forhold til antatt investeringshorisont, kan kunden forvente en gjennomsnittlig solid meravkastning utover risikofri rente.

Figure 9.5 Recommendation Framework risk tolerances



ANBEFALINGSKRITERIER / AKTIVAKLASSER

ALLOKERING

AKTIVAKLASSE	VISNINGSNAVN	SORTERING	AVKASTNING	STANDARDAVVIK	FARGE
Eiendom	Eiendom		5 %	8 %	■
Hedge	Hedge		6 %	10 %	■
Obligasjonsmarked	Obligasjonsmarked		3 %	3 %	■
Pengemarked	Pengemarked		2 %	0.5 %	■
Globale aksjer	Globale aksjer		10 %	20 %	■
Norske aksjer	Norske aksjer		8 %	15 %	■
Uklassifisert			0,00%	0,00%	■

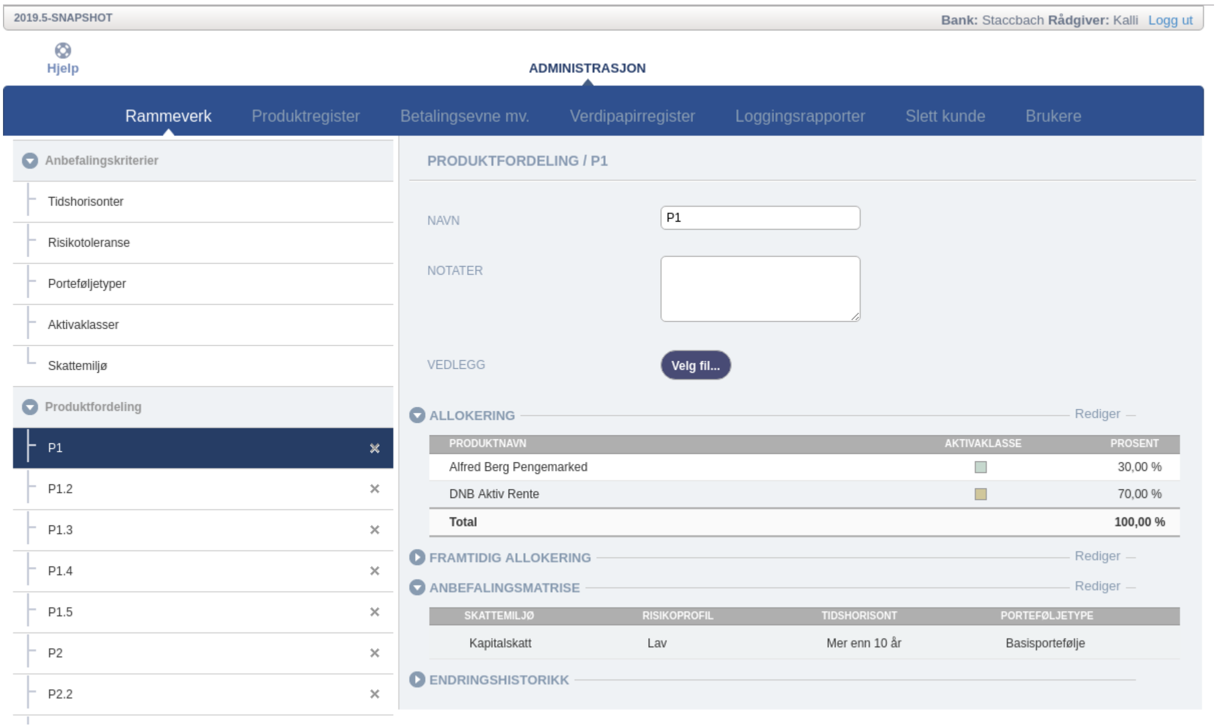
KORRELASJONSMATRISE

	■ Eiendom	■ Hedge	■ Obligasjonsmarked	■ Pengemarked	■ Globale aksjer	■ Norske aksjer
■ Eiendom	1					
■ Hedge	-0.05	1				
■ Obligasjonsmarked	0.3	-0.35	1			
■ Pengemarked	-0.2	-0.15	0.4	1		
■ Globale aksjer	-0.1	0.5	0.16	-0.02	1	
■ Norske aksjer	0.25	0.35	0.2	0.05	0.8	1

KOBLING TIL GENERISK AKTIVAKLASSE

GENERISK AKTIVAKLASSE	BANKENS AKTIVAKLASSE
Aksjer | Geografi | Andre Regionale/Internasj.	Globale aksjer
Aksjer | Geografi | Asia ekskl. Japan	Globale aksjer
Aksjer | Geografi | Europa	Globale aksjer
Aksjer | Geografi | Globale	Globale aksjer
Aksjer | Geografi | Kina	Globale aksjer
Aksjer | Geografi | Latinamerikanske	Globale aksjer
Aksjer | Geografi | Nordamerikanske	Globale aksjer
Aksjer | Geografi | Norden	Globale aksjer

Figure 9.6 Recommendation Framework allocations and correlation matrix



PRODUKTFORDELING / P1

NAVN: P1

NOTATER

VEDLEGG

ALLOKERING

PRODUKTNAVN	AKTIVAKLASSE	PROSENT
Alfred Berg Pengemarked	■	30,00 %
DNB Aktiv Rente	■	70,00 %
Total		100,00 %

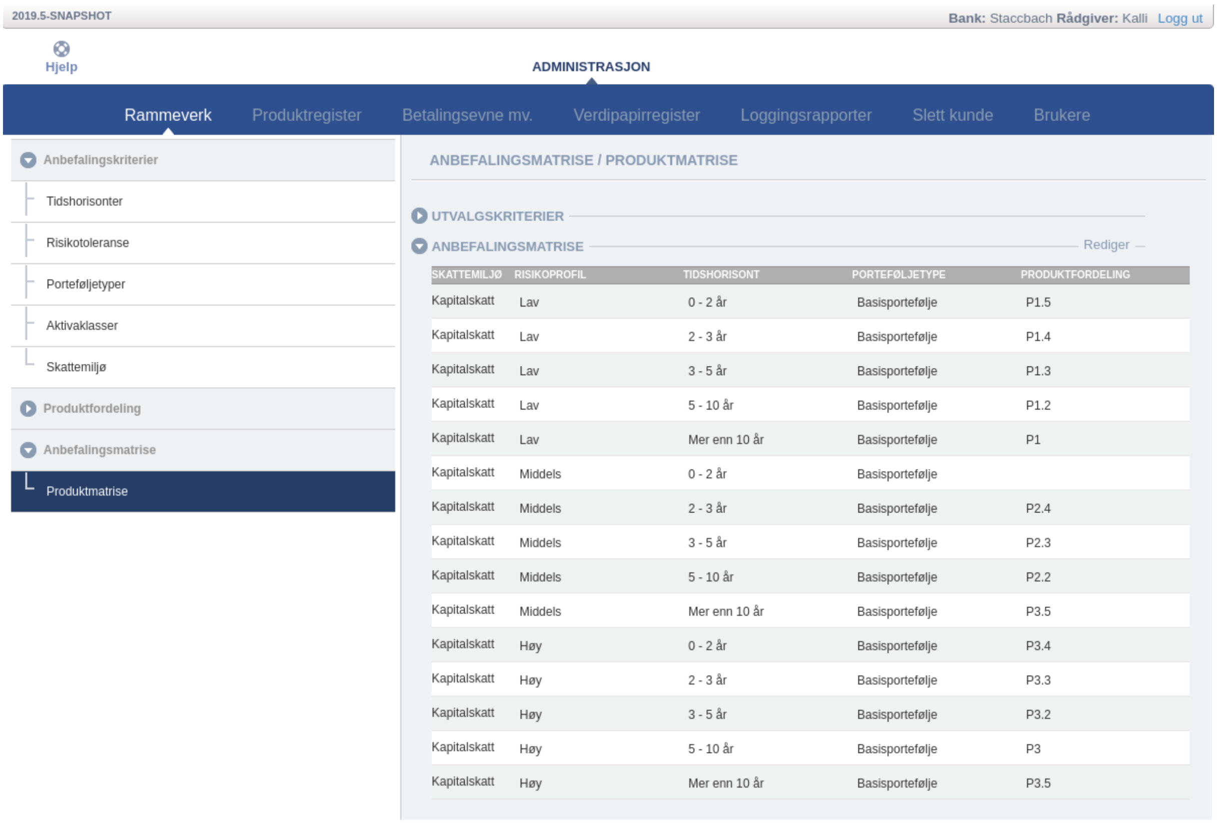
FRAMTIDIG ALLOKERING

ANBEFALINGSMATRISE

SKATTEMILJØ	RISIKOPROFIL	TIDSHORISONT	PORTEFØLJETYPE
Kapitalskatt	Lav	Mer enn 10 år	Basisportefølje

ENDRINGSHISTORIKK

Figure 9.7 Recommendation Framework product allocation of funds



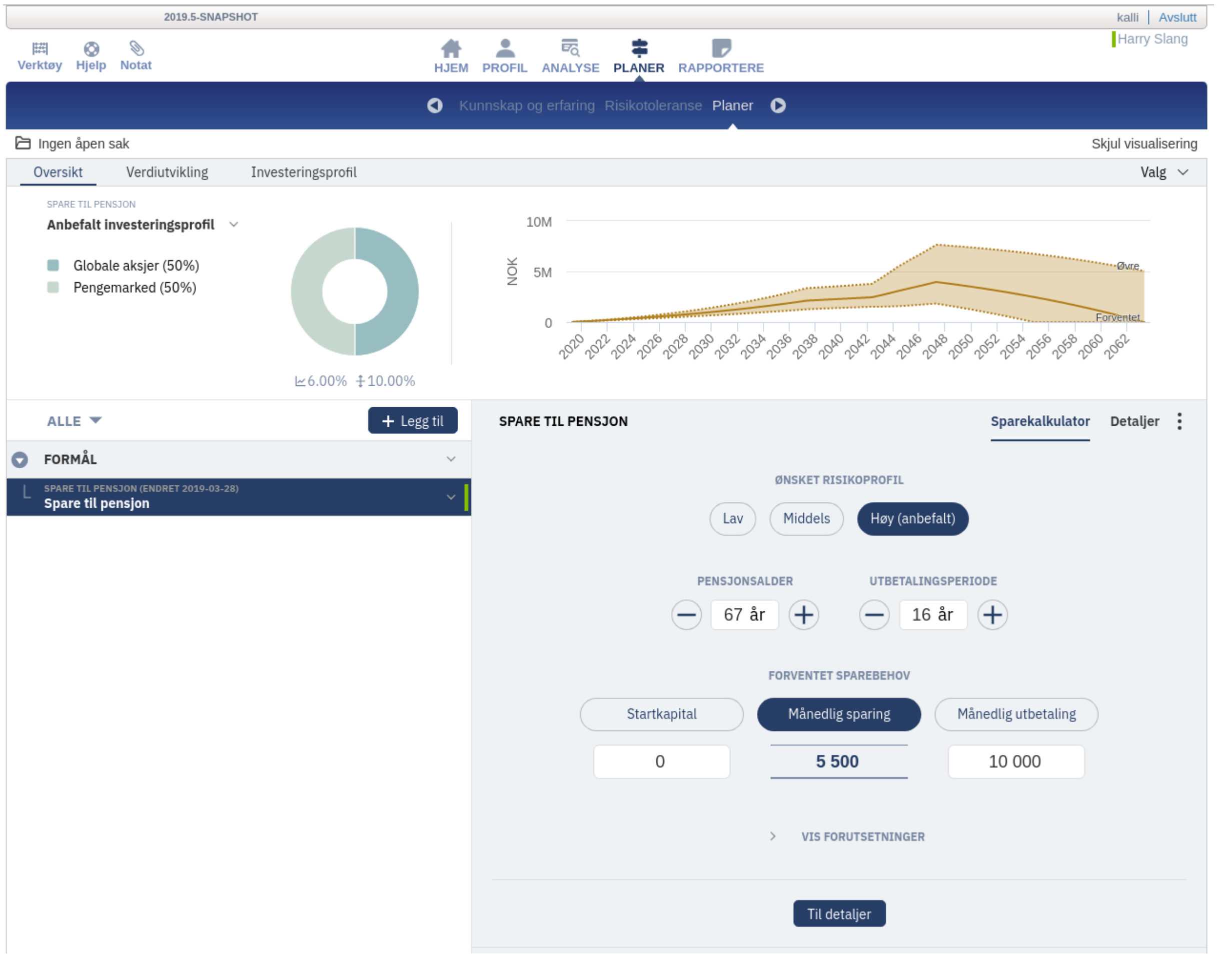
ANBEFALINGSMATRISE / PRODUKTMATRISE

UTVALGSKRITERIER

ANBEFALINGSMATRISE

SKATTEMILJØ	RISIKOPROFIL	TIDSHORISONT	PORTEFØLJETYPE	PRODUKTFORDELING
Kapitalskatt	Lav	0 - 2 år	Basisportefølje	P1.5
Kapitalskatt	Lav	2 - 3 år	Basisportefølje	P1.4
Kapitalskatt	Lav	3 - 5 år	Basisportefølje	P1.3
Kapitalskatt	Lav	5 - 10 år	Basisportefølje	P1.2
Kapitalskatt	Lav	Mer enn 10 år	Basisportefølje	P1
Kapitalskatt	Middels	0 - 2 år	Basisportefølje	
Kapitalskatt	Middels	2 - 3 år	Basisportefølje	P2.4
Kapitalskatt	Middels	3 - 5 år	Basisportefølje	P2.3
Kapitalskatt	Middels	5 - 10 år	Basisportefølje	P2.2
Kapitalskatt	Middels	Mer enn 10 år	Basisportefølje	P3.5
Kapitalskatt	Høy	0 - 2 år	Basisportefølje	P3.4
Kapitalskatt	Høy	2 - 3 år	Basisportefølje	P3.3
Kapitalskatt	Høy	3 - 5 år	Basisportefølje	P3.2
Kapitalskatt	Høy	5 - 10 år	Basisportefølje	P3
Kapitalskatt	Høy	Mer enn 10 år	Basisportefølje	P3.5

Figure 9.7 Recommendation Framework product matrix



Anbefalt investeringsprofil

- 🟦 Globale aksjer (50%)
- 🟩 Pengemarked (50%)

6.00% 10.00%

ØNSKET RISIKOPROFIL: Høy (anbefalt)

PENSJONSALDER	UTBETALINGSPERIODE
67 år	16 år

FORVENTET SPAREBEHOV

Startkapital	Månedlig sparing	Månedlig utbetaling
0	5 500	10 000

Figure 9.8 Savings Goal calculator