



Western Norway
University of
Applied Sciences

BACHELOR'S THESIS

A scalable order tracking system built on serverless cloud technology.

Et skalerbart ordresystem bygget på serverless skyteknologi.

**Toril Sunde Apelthun, Ingrid Elisabeth Hjelle and
Synnøve Sørensen**

Computing and Information Technology

Department of Computer Science, Electrical Engineering and Mathematical Sciences

Faculty of Engineering and Science

Supervisor: Sven-Olai Høyland

Submission Date: 4 June 2021

I confirm that the work is self-prepared and that references/source references to all sources used in the work are provided, cf. Regulation relating to academic studies and examinations at the Western Norway University of Applied Sciences (HVL), § 12-1.

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: A scalable order tracking system built on serverless cloud technology	Dato: 4. juni 2021
Forfattere: Synnøve Sørensen, Ingrid Elisabeth Hjelle, Toril Sunde Apelthun	Antall sider u/vedlegg: 38
	Antall sider vedlegg: 3
Studieretning: Dataingeniør/informasjonsteknologi	Antall disketter/CD-er: Ingen
Kontaktperson ved studieretning: Sven-Olai Høyland	Gradering: Ingen
Merknader: Lenke til kodereservoar: https://github.com/HVL-12-2021/bachelor	

Oppdragsgiver: Lunsj på Hjul	Oppdragsgivers referanse: Ingen
Oppdragsgivers kontaktperson: Anne Lill Stene	Telefon: +47 971 19 400

<p>Sammendrag: Anne Lill Stene tilbyr gjennom Lunsj på Hjul en abonnementsløsning hvor man kan få levert bærekraftig mat ved hjelp av en elektrisk sykkel. Som abonnent av Lunsj på Hjul har man blant annet mulighet til å kansellere leveringer, bytte leveringsdag og pause abonnement. Dette er oppgaver Stene administrerer manuelt.</p> <p>I dette prosjektet presenteres planleggingen og utviklingen av en webapplikasjon som vil kunne automatisere noen av Stenes manuelle oppgaver. Webapplikasjonen er laget skalerbar slik at bedrifter med samme behov også kan benytte applikasjonen. Et system med flere leverandører vil gjøre det mulig for hver kunde å ha flere abonnementer hos forskjellige leverandører, med én konto og brukerprofil.</p> <p>Abstract: With Lunsj på Hjul, Anne Lill Stene offers a subscription system where you can have sustainable food delivered by an electrical bicycle. As a subscriber to Lunsj på Hjul, you have the opportunity to cancel deliveries, change delivery days and pause the subscription. These are tasks Stene are managing manually.</p> <p>This project presents the planning and development of a web application which makes it possible to automate some of Stene's manual tasks. The web application has been made scalable, which makes it possible to add more vendors to the system should the need arise. A multi-vendor system will enable each customer to have several subscriptions with different vendors, while only needing one account and user profile.</p>

Stikkord:

Webapplikasjon	TypeScript	NoSQL
Fullstack utvikling	Serverless arkitektur	Fakturering

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitenskap

Postadresse: Postboks 7030, 5020 BERGEN

Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00

Fax 55 58 77 90

E-post: post@hvl.no

Hjemmeside: <http://www.hvl.no>



PREFACE

This report documents the bachelor project of computer science students Toril Apelthun, Ingrid Hjelle and Synnøve Sørensen at Western Norway University of Applied Sciences (HVL).

The project consisted of developing an order tracking system for Anne Lill Stene and her company Lunsj på Hjul (Lunch on Wheels). We would like to thank Anne Lill Stene for a fun and exciting challenge, and for valuable feedback throughout the project.

We would also like to thank our supervisor at HVL, Sven-Olai Høyland, for advice and guidance along the way and for the evaluation of this report.

We are also grateful to Steinar Søreide for offering us technical assistance during the development phase of the project.



TABLE OF CONTENT

1	INTRODUCTION.....	1
1.1	MOTIVATION AND GOAL.....	1
1.2	LIMITATIONS.....	1
1.3	RESOURCES	1
1.4	ORGANIZATION OF THE REPORT	2
2	PROJECT DESCRIPTION	3
2.1	PROJECT OWNER	3
2.2	PREVIOUS WORK.....	3
2.3	INITIAL REQUIREMENTS SPECIFICATION	3
2.4	INITIAL SOLUTION IDEA.....	4
3	PROJECT DESIGN	6
3.1	POSSIBLE APPROACHES	6
3.1.1	<i>Alternative approach 1 Mobile Application.....</i>	<i>6</i>
3.1.2	<i>Alternative approach 2 Web Application</i>	<i>6</i>
3.1.3	<i>Discussion of alternative approaches.....</i>	<i>7</i>
3.2	SELECTION OF TOOLS AND PROGRAMMING LANGUAGES.....	7
3.3	PROJECT DEVELOPMENT METHOD	10
3.3.1	<i>Development method.....</i>	<i>10</i>
3.3.2	<i>Project Plan</i>	<i>10</i>
3.3.3	<i>Risk management.....</i>	<i>11</i>
3.4	EVALUATION METHODS.....	11
4	DETAILED DESIGN.....	13
4.1	SERVER ARCHITECTURE.....	13
4.1.1	<i>Serverless.....</i>	<i>13</i>
4.1.2	<i>AWS Lambda</i>	<i>13</i>
4.1.3	<i>Handlers</i>	<i>14</i>
4.1.4	<i>CORS.....</i>	<i>15</i>
4.2	DATABASE DESIGN	16
4.2.1	<i>NoSQL databases</i>	<i>16</i>
4.2.2	<i>Data modelling.....</i>	<i>16</i>
4.2.3	<i>Queries</i>	<i>18</i>



4.3	REST API	19
4.4	CLIENT.....	20
4.4.1	<i>Client-side Frameworks</i>	20
4.4.1.1	Making components.....	21
4.4.1.2	Lifecycle.....	21
4.4.1.3	Rendering	22
4.4.2	<i>Vue in this project</i>	22
4.4.2.1	Styling of components.....	22
4.5	USER MANAGEMENT AND SECURITY.....	22
5	EVALUATIONS	24
5.1	EVALUATION METHODS.....	24
5.1.1	<i>Unit testing</i>	24
5.1.2	<i>Integration testing</i>	24
5.1.3	<i>Usability testing</i>	25
5.2	EVALUATION RESULTS	26
5.2.1	<i>Unit and integration testing</i>	26
5.2.2	<i>Usability testing</i>	26
6	RESULTS	28
7	DISCUSSION	32
8	CONCLUSIONS AND FURTHER WORK	33
9	BIBLIOGRAPHY	34
10	APPENDICES	38
10.1	GANTT CHART	38
10.2	RISK LIST	39
10.3	VUE LIFECYCLE HOOKS.....	40

LIST OF FIGURES

Figure 1 An overview of the technology stack used for the backend of the web application. 9

Figure 2 API request 14

Figure 3 Handler for user profile 15

Figure 4 Database entities and relations..... 17

Figure 5 Dialog box for login, registration and resetting of password..... 23

Figure 6 Integration tests running in the terminal of Visual Studio Code..... 25

Figure 7 The calendar showing a customer's deliveries..... 28

Figure 8 Profile view for logged in customer. 29

Figure 9 The payment view for the logged in vendor. 30

Figure 10 The registration of new payments. 31

LIST OF TABLES

Table 1 Core functionality 3

Table 2 Database access patterns 18

Table 3 REST API 20

1 INTRODUCTION

1.1 Motivation and goal

Lunsj på Hjul (Lunch on Wheels) is a small company with approximately 35 customers. The company is owned and run by Anne Lill Stene, and her aim is to make sustainable, local, and healthy food from scratch. The food is made from organic surplus goods and local delicacies, and Stene delivers the meals by electrical bike three days a week.

Customers have the possibility to order subscriptions where they pay for the deliveries in advance. Stene has to keep an overview of the subscriptions and how many deliveries each customer has to their credit. It is possible for customers to cancel or postpone deliveries by writing it into a third-party online form, sending an SMS or an e-mail. This solution makes it difficult to get an overview of the deliveries for the different days, and when it is time to send invoices to the customers.

The problem space for this bachelor project is to solve the challenges of the company by creating a web application that handles all the tasks that today must be done manually.

The overall goal is to develop a web application making the client's workflow easier and more efficient. This means that the client can relate to one system instead of several different systems and reduce the time spent on administration and invoicing.

The main goal is to make a web application with core functionality. The core functionality includes keeping track of how many meals each customer has received or cancelled and making it possible to invoice the customers correctly each period. In addition, customer information such as delivery address and email must be stored. It must be possible to register that a customer has paid for the next period, and which weekdays they want deliveries.

There are several subsidiary goals that are not defined as a core functionality and will therefore be implemented if they are feasible technically and there is enough time. One of these goals includes to link the web application to the accounting program Fiken, making it possible to automatically invoice customers. Another is SMS notifications to alert the client if a customer requests a change of a delivery. The core functionality and the subsidiary goals will be discussed in Chapter 2.3

1.2 Limitations

Because of the short duration of a bachelor project, time is a major limitation. A considerable amount of time must be set aside to learn the technologies we are unfamiliar with in the initial phase of the project. Learning new technologies and techniques may also lead to some trial and error at the beginning of the development.

1.3 Resources

For the project to succeed, it is important to understand the client's workflow. Her needs and requirements must be clarified at an early stage. We will work closely with the client so that she will have an influence on the final product.

As the selected technology stack is modern and popular, there are numerous online resources and courses available to us. We will consult with our supervisor during the project, to ensure that our progression is in accordance with the plan.

The new web application will be integrated with an existing website for the company, so that all functionality is in one place. The plan is to develop a web application using serverless architecture for the backend. By using a Serverless infrastructure provider, the security, maintenance, scalability, and redundancy will be taken care of by the service provider.

Different methods will be implemented to test the code during development. This includes unit testing of functions and classes, API testing of interfaces and integrations in the application. To ensure good user experience, user tests will be performed by a selection of the company's customers.

The tools and technologies will be described in detail in Chapter 3.2.

1.4 Organization of the report

The first three chapters describe the pre-project phase. This includes an introduction, a project description, and a chapter on project design. In the introduction chapter, the context, motivation, and goal, as well as limitation and resources are defined. Initial requirement specifications and the solution idea is presented in the chapter on project description. Project design includes discussing different approaches and justifying the selected approach. This chapter also describes selected tools, how the results will be evaluated and presents a project plan and risk management.

The other part of the report is about the implementation phase of the project, starting with a detailed description of the application design. Chapter 5 presents the chosen evaluation methods, and the evaluation results. Chapter 6 and 7 presents the results and discusses the consequences of the selected approach, as well as how the choices have influenced the results. A summary of the project goals and the conclusion can be read in Chapter 8. The last two chapters contains the references and appendices.

2 PROJECT DESCRIPTION

2.1 Project owner

The students will be the project owners, and the web application will be available for the client, Stene, for an unlimited period.

The project is important for the client because of the need to improve the handling of subscriptions and invoicing. By improving the current solution, the company will be able to free up time that can be spent on other things.

2.2 Previous work

The client has a website for her company (Stene, n.d.). The website includes information about the food, areas available for delivery, and a web form the customer can use to register a subscription. The customer information must be manually handled by the company after submission, and invoice details must be collected after the registration is done. The plan is to link the new web application to the existing website, making it invisible to the customers that it consists of two different web applications.

2.3 Initial requirements specification

The initial requirements were set at the first meeting with the client. Clarifications, adjustments, and new requirements were made consecutively as the project progressed.

In the initial phase it was decided that the web application should contain the core functionality listed in Table 1.

Functionality for the customers	Functionality for the company
Registration and login.	Overview of how many and which deliveries that are to be prepared each day.
Delete subscription and data.	
Changing the day of delivery.	Overview of coming invoices.
Date for next invoice.	Overview existing customers and subscriptions.
Postponing deliveries or pausing the subscription for a period.	Accepting/rejecting new subscription requests.

Table 1 Core functionality

Registration and login

The customers shall be able to register and log into the application as first-time users. After logging in, they can choose a subscription and provide all necessary details such as delivery address and billing information.

Deletion of subscriptions and data

The customers shall be able to stop an ongoing subscription and delete all data about themselves.

Cancelling deliveries and pausing the subscription

Customers shall be able to cancel coming deliveries, one by one. There will also be an option for pausing the subscription for a selected period.

Changing the day of delivery

Customers shall be able to send a request for changing the day of delivery, e.g., having Wednesday's lunch delivered on Thursday. This will have to be done a certain time in advance and must be confirmed by the company.

Invoice

The customers shall be able to view the next invoice and information about how many meals that have been delivered and/or cancelled the current period. The web application will contain a calendar displaying the dates for the next deliveries and deviant delivery dates, e.g., because of holidays.

It has been decided that the web application should have an administrator account for the company. When logged into this account the client should be able to get an overview of how many deliveries that are to be prepared each delivery day, and to whom the meals are to be delivered. It should also contain an overview of the subscriptions and the corresponding invoices for each period. If possible, there should be an option for automatic invoicing. This can be solved with an interaction between the web application and Fiken accounting software.

Additional functionality

Additional functionality that will be implemented if there is enough time includes:

- SMS notifications for customers when the meal has been delivered to a pick-up point.
- Email and/or SMS notifications for vendor when new customers make subscriptions.
- Email and/or SMS notifications for vendor when customers request a change in delivery time.
- Linking the web application to the accounting program Fiken, making it possible to automatically invoice customers.

2.4 Initial solution idea

The initial solution idea is a responsive web application which should work on computers and mobile devices. The application will include functionality for users and admins to log in, handle their personal

information and subscription details. It will be possible for both admins and users to cancel or make changes to deliveries. All relevant information about the users, the subscriptions and deliveries will be stored in a database.

The serverless technology, together with a NoSQL database, have been chosen to make the web application scalable, as this is a design we are eager to learn more about. It will be possible to add more vendors to the system should the need arise. A multi-vendor system will enable each customer to have several subscriptions with different vendors, while only needing one account and user profile.

3 PROJECT DESIGN

3.1 Possible approaches

The client's initial request was a web application or a mobile application. The requirements for functionality were the same for both alternatives. One of the client's main requirements was an application it is easy to use and access, for both her and her customers. The alternative approaches will be outlined in the following chapters.

3.1.1 Alternative approach 1 Mobile Application

As one of the requirements from the client was an application that she and her customers would have easy access to, a mobile application was discussed. When developing applications for mobile phones, native, hybrid and cross-platform development must be considered, as there are different mobile operating systems available with their own development techniques.

In native development, separate applications are developed for the different mobile operating systems, such as iOS and Android, which gives the developers certain advantages. Some of these advantages consists of developers having access to native hardware functions for the mobile operating systems. Native applications are written in the operating system's native languages, meaning it is made for one specific operating system. Two different applications must therefore be developed from separate codebases (Bavosa, 2020).

A different approach is to develop one single mobile application using cross-platform development. This makes it possible to develop for different platforms using one single codebase with some modifications for each operating system. One of the benefits of cross-platform development compared to native is the need of maintaining and making changes to one single application (Bavosa, 2020).

A third option is to build a hybrid application, which is a blend of native and web development and are written using web technologies such as JavaScript, HTML and CSS. The difference between a web application and a hybrid one is that the hybrid is not run in the user's web browser, but as a native application (Griffith, n.d.).

However, all these approaches require the users to download and install an application on their mobile device and makes it not available on computers.

3.1.2 Alternative approach 2 Web Application

An alternative to a mobile application is a web application. The biggest difference is that a web application runs on a web server, while a mobile application run locally on the mobile operating system. This means that the user does not need to install a program to use the application, and storage on the device is not an issue.

One major advantage is that you can create one application and it can be accessed from most devices, like computers, tablets, and smart phones. Independent of operating system, all users can access the same version of the application, making it easy to maintain and correct potential bugs. Another advantage is

the cost, when developing one application that works on most devices from one single code base, both time and money is saved (Gibb, 2016).

One thing to consider is that there are several different devices in different sizes, with different screen resolutions, layouts etc. To make a web application optimal for all devices, responsive web design is a solution. Responsive web design means that the application will adapt to different devices, screen resolutions and size of the browser window (Emorphis Technologies, 2019).

When considering a web application, it is also important to examine the disadvantages, the most important one being security. If the web security is not handled correctly, data breach is a risk which may lead to serious consequences (Fox, 2015).

To use a web application the user must have internet access which can be a disadvantage. With today's technology like 4G, 5G and high-speed Wi-Fi this is usually not a problem. However, it does happen that the internet provider has problems leaving the customers with slow connection or no connection for some time (Brombach, 2021). Nevertheless, internet providers do not often have problems, and if the security is handled correctly, web applications have more advantages than disadvantages.

3.1.3 Discussion of alternative approaches

A web application was the selected approach. This was based on the several different reasons, the main one being that we wanted to implement the new web application with the already existing web page for the company. We also consider our current knowledge to be greater for building web application compared to building native or hybrid mobile applications. This will make a web application quicker and easier to build, and the result will probably be more robust.

Other reasons for choosing to build a web application includes that a web application does not need to be downloaded or installed, it functions in a browser. It is easier to maintain a web application as it has one codebase, in opposition to mobile applications which need updates for each different platform. Finally, a web application does not require app store approval, and can be launched quickly.

By using responsive web design, the web application will function properly on devices with different screen sizes and input methods.

3.2 Selection of tools and programming languages

In this chapter the tools, programming languages and technologies selected for the project is described in detail. These technologies were chosen by the students because they represent a way of developing modern and scalable web applications that can run in the cloud with very inexpensive infrastructure.

IDE and programming language

TypeScript is a programming language developed by Microsoft. It is a strict superset of JavaScript and adds optional static typing, classes, and interfaces to the language (Bright, 2012). This provides better code structure and enables object-oriented programming techniques. It also provides better development time tooling such as auto-completion.

TypeScript is used for all the code in the project, including code for the frontend and for the backend of the web application. This is made possible by using Node.js, which is a runtime system for server and network applications. Node.js runs JavaScript code using Google's V8 engine so that JavaScript programs can run on servers outside of web browsers (nodejs, n.d.).

Visual Studio Code is the IDE (integrated development environment) used in the project. It is made by Microsoft for Windows, Linux and macOS. Its features include support for debugging, syntax highlighting, code completion, and embedded Git integration. It can be used for a variety of programming languages, like Java, JavaScript, and TypeScript. With additional add-ins, it is also possible to run and debug Node.js applications in Visual Studio Code (Visual Studio Code, n.d.).

Vue and Vuetify

Vue is an open-source JavaScript library for building interactive web applications, created by Evan You.

The Vue.js architecture is based on declarative rendering and component composition. The core library is focused on the view layer only. Vue.js lets you extend HTML with attributes called directives. Essentially, a directive is a special token in the markup telling the library to do something to a DOM element (Vue.js, n.d.).

Vuetify is a UI component library built on top of Vue.js. It is developed according to the Material Design specification and provides a range of ready-to-use widgets such as tree-views, tables, and an easy-to-use layout system (Vuetify, n.d.).

Combining these two libraries makes it possible to develop a user-friendly user interface with a modern feel and look.

Serverless and AWS Lambda

Serverless computing is an execution model where a cloud provider is responsible for executing code by dynamically allocating the resources. The code is typically run inside stateless containers that can be triggered by a variety of events including http requests, database events and file uploads. The code is sent to the cloud provider for execution in the form of a function, and serverless can therefore be referred to as “Functions as a Service”, or FaaS for short (Conally, 2020).

AWS Lambda is Amazon Web Service’s FaaS system. It is a compute service making it possible to run code without provisioning or managing servers. The code can be run in response to events such as HTTP requests or changes to a database. The code is only run when triggered and scales automatically. AWS Lambda runs the code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging (AWS, n.d.).

The entire web application’s backend is developed with serverless architecture. All the functions for communication with the database are implemented using AWS Lambda functions.

DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service that supports key-value and document data structures. DynamoDB allows developers to purchase a service based on throughput, rather than storage. With the right settings, the database will scale automatically. DynamoDB will spread the data and traffic over a number of servers using solid-state drives, allowing predictable performance (AWS, n.d.). NoSQL databases makes it possible to store different entities and their attributes in the same table. For this project, a single table is used for all the information that needs to be stored in the database.

An overview of how the different tools and technologies are used to create the backend of the web application can be seen in Figure 1.

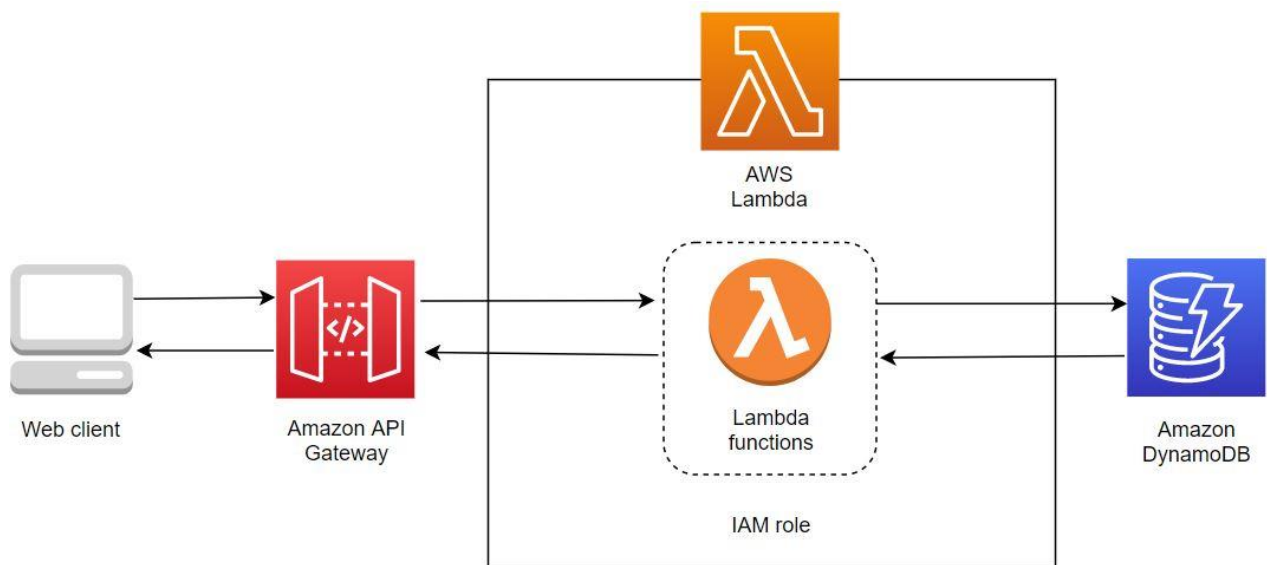


Figure 1 An overview of the technology stack used for the backend of the web application.

AWS S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and archives. AWS S3 will be used to host the web application (AWS, n.d.).

Git/GitHub

Git is a free and open-source distributed source code version control system (GIT, n.d.). Git allows multiple local branches that can be entirely independent of each other. Whenever a new feature is to be implemented or a bug fix is needed, a new branch can be made. The new branch will contain the code from the main branch at the time the branch was made. When the additional code is ready and tested, the branch can be merged into the main branch. To merge the new branch into the main code base, a pull request is made. This lets the team discuss and review the changes made before merging them into the base branch. This feature makes it harder for unstable code to get merged into the main code base and enables the group to work separately on different tasks without having to worry about making conflicting code.

GitHub is a cloud-based hosting service for managing Git repositories (GitHub, n.d.). It offers the distributed version control and source code management functionality of Git, and many other features. One of the features used in the project is GitHub's project board. The project board is used to organize, delegate, and prioritize the development tasks, and is made up of tasks, pull requests, and notes that are categorized as cards. These are sorted in columns to separate planned work, ongoing work, and finished work from each other.

Mocha and Chai

Mocha.js is an open-source JavaScript test framework that runs on Node.js and in the browser. It is designed for testing both synchronous and asynchronous code. Mocha.js provides functions that execute in a specific order and logs the results in the terminal window (Mocha, n.d.). It is commonly used with Chai, an assertion library that performs equality checks and compare actual results with expected results (Chai Assertion Library, n.d.). Mocha and Chai will be used for the unit and integration tests throughout the project.

3.3 Project development method

3.3.1 Development method

Kanban, which is an agile methodology, is chosen as the development method. Visualization of workflow, limitation of work in progress, and an iterative structure are the main principles of Kanban.

A project board is used to visualize the workflow. The board is divided into three or more columns. Each column represents a stage of the process, with the columns at the left and right sides of the board representing the start and end of the process (Weissman, 2018). Our project board includes the columns "To do", "In progress", "In review" and "Done". Tasks are represented by cards that can be assigned to the different team members, and each member move their card to the next column to the right as they complete each process stage.

This helps identifying problems, if multiple cards start piling up in one column, it indicates that there is a bottleneck in this area. After identifying the problem, the team members can collaborate on a solution. Solving problems as they arise improves team efficiency and productivity (Weissman, 2018).

Kanban does not have fixed length sprints. Instead, a continuous workflow is used, and updates are released whenever they are ready, without a regular schedule or predetermined due dates. There are no specific roles, it is a collective responsibility to collaborate on and deliver the tasks on the board (Rehkopf, u.d.).

The change philosophy of Kanban differs from other agile methods. A Kanban workflow can change at any time, while in other approaches, like Scrum, changes should not be made during a sprint (Rehkopf, u.d.).

3.3.2 Project Plan

The project plan describes the planned progress in a project, which will have a great impact on whether or not the project will succeed. For planning and scheduling events and milestones a Gantt chart is a useful

tool. This chart shows planned activities displayed against time. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration, and end date of the activity (Gantt.com, 2021). The Gantt chart for this project is enclosed in Appendix 10.1.

3.3.3 Risk management

Risk management is the practice of identifying, evaluating, and preventing risks to a project that have the potential to impact the desired goals. To measure the identified risks, the probability and consequence of each risk is determined. A scale from one to five is used, where one represents a probability of “not very likely to happen” and a consequence of “insignificant”. Five represents a probability that is “very likely to happen” and a consequence that is “disastrous” (Skjølvik & Voldsund, 2016, pp. 342-344).

A risk factor is then calculated by multiplying the probability with the consequence, the lowest risk factor is one and the highest risk factor is 25. The risk matrix for this project is enclosed in Appendix 10.2.

Of the identified risks, the two risks with the highest risk factor are the time limitation and potential absence of group members due to Covid-19 or other reasons. The time limitation has a risk factor of 12 and the consequence is that there is not enough time to finish the core functionality. The risk factor of potential absence is calculated to be nine. If members should get ill, there might be more to do for the rest of the group members for a period. This might also lead to the project not being finished within the set time frame.

In order to reduce the risk connected with the time limitation, a project plan is developed. The project plan includes start dates, duration, and end dates for all activities in the project. To reduce the risk of getting ill, the team members will take the necessary precautions and follow the national covid restrictions.

3.4 Evaluation methods

To ensure that the application works both technically and as the customer expects, the following evaluation methods will be performed:

- Unit testing of functions and classes.
- API testing of interfaces and integrations within the application.
- User tests to ensure that the usability of the application is satisfactory.

Unit testing

Unit tests will be used to verify that the model code in the application is working in accordance with expectations. An example of model code that will be tested using unit tests is a class that calculates which days that shall be marked with deliveries in the user’s calendar. These methods consider how many deliveries the customer has paid for in a new period, which weekdays they have ordered deliveries, and which date those weekdays corresponds to. Unit testing will be performed with the Mocha framework together with the Chai library.

API testing / Integration testing

Interfaces and integrations in the application will also need to be tested. Integration testing is a level of software testing where individual components are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units (Software Testing Fundamentals, n.d.). Integration testing of the classes containing the methods for inserting, extracting, and deleting instances in the database will be performed using the Mocha framework and Chai library.

Usability testing

Usability testing is the practice of testing how easy a design is to use for a group of representative users (Interaction Design Foundation, 2021). When the core functionality of the web application is implemented, three to five of the client's customers will be asked to perform user testing. The customers will be given tasks like signing up for an account and registering a subscription. We will observe them performing the tasks they are given and interview them about the overall experience afterwards. This will give valuable information about the usability of the application, and about the expectations from the end users. The initial plan is to perform these tests in the beginning of May.

4 DETAILED DESIGN

The web application is designed to support more than one vendor using the system. If more companies want to use the application in the future, it will be easy to add new vendors, users, and their subscription to the system.

The main difference when developing an application supporting several vendors instead of only one, is found in the design of the database system. The data structure and the queries need to support a system containing more than one vendor, and more than one subscription for each user.

The server-side functionality and the REST API also need to support this design for the client side to be able to fetch the right vendors and all the subscriptions a user might have with different vendors.

4.1 Server architecture

4.1.1 Serverless

Serverless most often refers to serverless applications. Serverless applications are ones that do not require the development team to provision or manage any servers. By building the application on a serverless platform, the platform manages these responsibilities.

For a service or platform to be considered serverless, it should provide the following capabilities:

- No server management: The developer does not have to provision or maintain any servers.
- Flexible scaling: The application should be able to scale automatically or by adjusting its capacity through toggling the units of consumption (for example, throughput or memory) rather than units of individual servers.
- High availability: Serverless applications have built-in availability and fault tolerance.
- No idle capacity: There is no charge when the code is not running.

The AWS Cloud provides many different services that can be components of a serverless application. This project has included capabilities for:

- Computing using AWS Lambda
- APIs using Amazon API Gateway
- Storage and web hosting using Amazon Simple Storage Service (Amazon S3)
- Database functionality using Amazon DynamoDB
- User management and authentication using AWS Cognito

(Amazon Documentation, n.d.)

4.1.2 AWS Lambda

AWS Lambda is a serverless compute offering based on functions. It provides the cloud logic layer for our application. These functions can be triggered by a variety of events. When there are multiple, simultaneous events to respond to, AWS Lambda will run multiple copies of the function in parallel. This makes Lambda functions scale with the size of the workload and reduce the likelihood of an idle server. Architectures using these serverless functions are thus designed to reduce wasted capacity.

Each Lambda function contains the code to be executed, the configuration that defines how the code is executed and, optionally, one or more event sources that detect events and invoke the function as they occur. In our web application the event source is the API Gateway, which will invoke the right Lambda function anytime an API method created with API Gateway receives an HTTPS request (Amazon Documentation, n.d.).

Amazon API Gateway is a service that makes it possible for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the “front door” for applications to access data, business logic, or functionality from the backend services (Amazon, n.d.).

4.1.3 Handlers

When a Lambda function is invoked, code execution begins at what is called the handler. The handler is an entry point function that is specified when creating a Lambda function. Once the handler is successfully invoked inside the Lambda function, the function will execute the desired code (AWS Documentation, n.d.).

When the Lambda function is invoked, one of the parameters provided to the handler function is an event object. As our events are created by the API Gateway, they will contain details related to the HTTPS request that was made by the API client, like path, query string and request body (Amazon Documentation, n.d.).

To illustrate Lambda functions and handlers it is useful to look at an example from our project: the client wishes to add a new user profile to the database. The API client makes a HTTPS request and includes the user profile object in the request body. The request URL decides which Lambda function that will be triggered. The HTTPS request can be seen in Figure 2 below.

```
56     async putUserprofile(userprofile: interfaces.Userprofile): Promise<interfaces.Userprofile> {
57         await this.ensureFreshToken();
58         const addedUser = await this.apiAxios.put(urlPrefix + "/userprofile", userprofile);
59         return addedUser.data;
60     }
```

Figure 2 API request

When the request reaches the API Gateway, the Lambda function and its handler will be invoked. The handler in this example can be seen in the Figure 3.

The handler function (at line no 8) calls a function based on the type of HTTPS request that is received. In our example this is a PUT request, and the incoming event is passed to the function putUserprofile on line 38. This function retrieves the id for the logged in user and parses the event body from JSON format. The method for adding a user profile or changing an existing user profile in the database is then called from the handler. This method is also part of the Lambda function all though it is located in another file with all the other methods for database communication. The database methods used in the handler is imported at line 5.

```

1  import 'source-map-support/register'
2  import middy from 'middy';
3  import cors from '@middy/http-cors';
4  import { APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
5  import { deleteUserprofileInDb, getUserprofileFromDb, putUserprofileInDb } from './dbUtils';
6  import { getUserInfoFromEvent } from './auth/getUserFromJwt';
7
8  async function handler(event: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
9      if (event.httpMethod == "GET") {
10         return getUserprofile(event);
11     }
12     if (event.httpMethod == "PUT") {
13         return putUserprofile(event);
14     }
15     if (event.httpMethod == "DELETE") {
16         return deleteUserprofile(event);
17     }
18 }
19 export const mainHandler = middy(handler).use(cors());
20
21 > async function getUserprofile(event: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> { ...
36 }
37
38 async function putUserprofile(event: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
39     let userId = getUserInfoFromEvent(event);
40     let body = JSON.parse(event.body);
41     let userprofile = await putUserprofileInDb(body, userId);
42
43     return {
44         statusCode: 200,
45         body: JSON.stringify(userprofile)
46     };
47 }

```

Figure 3 Handler for user profile

4.1.4 CORS

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism allowing a server to indicate any other origins, like a domain, scheme, or a port than its own from which a browser should permit loading of resources.

CORS also relies on a mechanism called “pre-flight” for HTTP request methods that can cause side-effects on server data. In this pre-flight the browser checks that the server will permit the request by sending headers indicating the HTTP method and headers that will be used in the actual request. Upon approval from the server, the browser will send the actual request. Servers can also inform clients whether credentials such as HTTP Authentication should be sent with requests. By using CORS browsers can mitigate the risks of cross-origin HTTP requests (MDN Web Docs, n.d.).

In the code seen in Figure 3, a middy module is imported at line 2. This is a middleware library making it easy to set the HTTP CORS headers necessary for making cross-origin requests to the response object.

4.2 Database design

4.2.1 NoSQL databases

AWS DynamoDB was chosen as the database system to be used in the project. DynamoDB is a NoSQL database. These database systems differ from the traditional relational database management systems (RDBMS) in several ways. In a SQL database data is represented in the form of tables which consist of a number of rows of data. In comparison, NoSQL databases like DynamoDB, are document based, key-value pairs, graph databases. Where SQL databases have predefined schema, NoSQL databases have dynamic schema for unstructured data.

SQL databases are vertically scalable whereas the NoSQL databases are horizontally and vertically scalable. This means that the SQL databases are scaled by increasing the power of the hardware, while NoSQL databases are scaled by increasing the number of servers in the resource pool (Höck, n.d.).

NoSQL databases also differ from SQL databases when it comes to queries. In a SQL database querying can be done with much flexibility, but the queries will often not scale well in high-traffic situations. In a NoSQL database, data can be queried efficiently in a limited number of ways, outside of which queries can be expensive and slow.

These differences will affect how the two different database systems are designed. In SQL databases normalization is important while query optimization generally will not affect the design. In NoSQL databases, the schema is designed specifically to make the most common and important queries as fast and as inexpensive as possible.

The entities must be implemented with regards to the specific query patterns that the system will use. This can be achieved by using key-value pairs. Other major design principles include keeping related data together, use sort order, and to distribute queries so that traffic is evenly distributed across partitions (AWS Documentation, n.d.).

4.2.2 Data modelling

Our web application needs to store data about the vendor and their customers and subscriptions. The database system is designed so that all records are stored in the same table in DynamoDB. The main reason for using a single table in DynamoDB is to retrieve multiple, heterogenous item types using a single request, and this is one of the reasons this database is fast and efficient (Alex DeBrie, 2020).

As SQL databases store data in different tables, they require the concept of joins. Joins make it possible to combine records from two or more tables, but they require scanning large portions of multiple tables and comparing different values, to return a result set. This is an expensive operation, and it can result in linear time complexity if the data set is large (Alex DeBrie, 2020).

NoSQL databases avoids this expensive operation by pre-joining the data into item collections. This is achieved by modelling each data structure with a composite primary key combined by a partition key (pk) and a sort key (sk). A query operation can then be used to read multiple items with the same partition key. Items with the same partition key will be ordered by the sort key value. It is therefore important to structure the sort keys so that it gathers related information together in one place. This makes it possible

to retrieve groups of related items using range queries with operators such as “begins with”, “between”, “less than” and “larger than” (AWS Documentation, n.d.).

The partition key determines the physical storage internal to DynamoDB, in which the item will be stored. This means that all items with the same partition key will be stored together. The primary key elements should therefore be structured to avoid one heavily requested partition that will slow down the overall performance. An example of a good partition key value in a system with many users, is a user id. An example of a bad partition key is a status code in a system with only a few different status codes (AWS Documentation, n.d.).

In all queries the partition key and value must be specified as an equality condition. Only the sort key condition can use comparison operators like begins with, between, less than or greater than.

Some applications might need to perform many kinds of queries, using a variety of different attributes as query criteria and sometimes the composite primary key is not enough to retrieve the right item types. To support these requirements, it is possible to create one or more global secondary indexes and make queries against these indexes instead of the primary key. Secondary indexes contain a selection of attributes from the base table, but they are organized by a primary key that is different from that on the base table. Secondary indexes will be further explained with an example in Chapter 4.1.3 (AWS Documentation, n.d.).

The data stored for the web application have been structured into four different database entities. Figure 4 describes the different entities, or data structures, and the relation between them. Each vendor in the system offers subscriptions to several users, and each user can have subscriptions to several vendors’ services. Each subscription will have several deliveries.

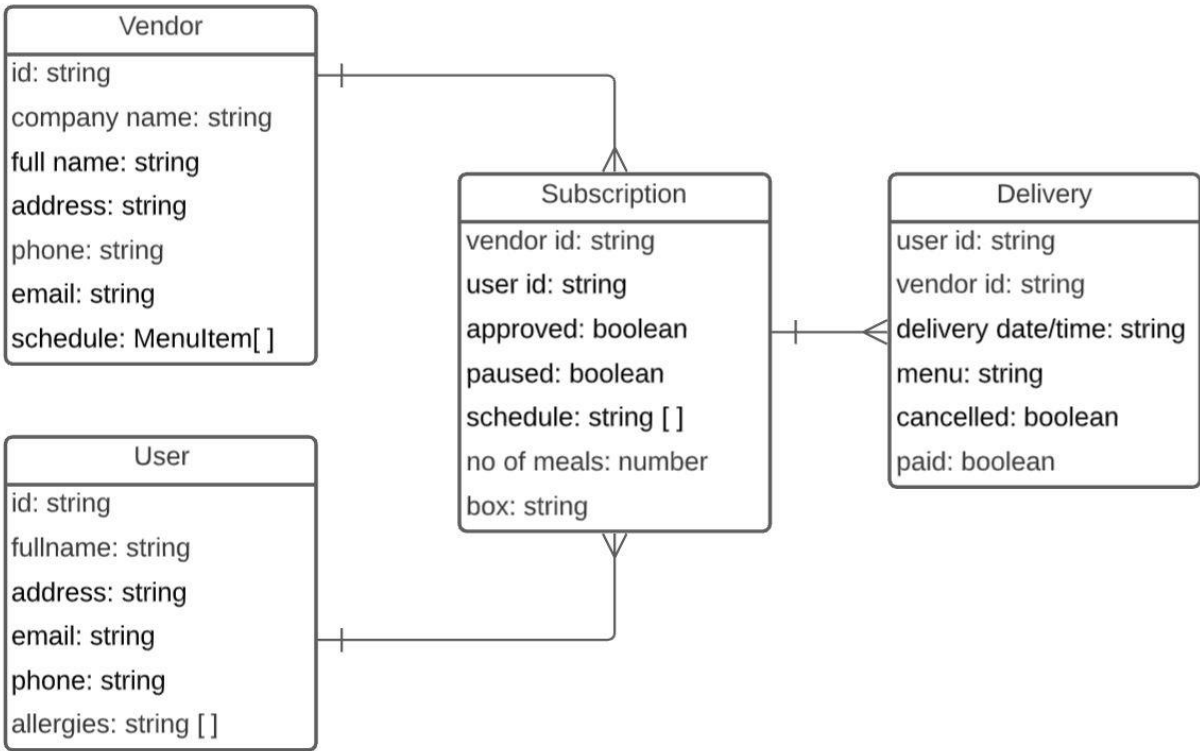


Figure 4 Database entities and relations

4.2.3 Queries

When designing a NoSQL database, it is necessary to start with identifying and defining the access patterns the system needs to support. The access patterns will determine which keys that will be needed for the different entities to make efficient queries.

The main access patterns needed for our web application are shown in Table 2. In this table, *pk* refers to partition key, and *sk* to sort key. Each row in the table represents one or more access patterns for a specific item type, or entity. For example, the second row describes the access patterns for retrieving, editing, or deleting a given subscription in the database. In these queries the subscription will be identified with the partition and sort keys. The partition key for a subscription is the vendor id for the vendor the user subscribes to. The sort key is the user id of the user the subscription belongs to. This design will result in all subscriptions for the same vendor to be partitioned at the same resource.

Access pattern	Key condition
Get, edit, or delete a given vendor	pk: "v#vendorId", sk: "v#vendorId"
Add, edit, or delete a given subscription	pk: v#vendorId, sk: u#userId
Get, edit, or delete a given user profile	pk: "u#userId", sk: "u#userId"
Get all vendors	GSI pk: "vendor", GSI sk begins with "v#"
Get all subscriptions for a given vendor	pk: "v#vendorId", sk begins with "u#"
Get alle subscriptions for a given user	GSI pk: "u#userId", GSI sk begins with "v#"
Get all deliveries for a given customer in given time frame	GSI pk: "u#userId", GSI sk between start date and end date
Add, edit, or delete a delivery for given customer, given vendor, and given time	pk: "v#vendorId", sk begins with "d#time#u#userId"
Get all deliveries for a given vendor in given time frame	pk: "v#vendorId", sk between "d#start" and "d#end"
Add new deliveries to a user with specified vendor	pk: "v#vendorId", sk: "d#time#u#userId", GSI pk: "u#userId", GSI sk: "time"
Find latest delivery for a user	GSI pk: "u#userId#v#vendorId", GSI sk: *
Cancel several deliveries by a vendor	pk: "v#vendorId", sk between "d#startTime" and "d#endTime"

Table 2 Database access patterns

The prefixes used with the keys are included to make it easier to provide a second condition for the sort key using comparison operators. A query for all of a given vendor's subscribers will retrieve all items where the partition key is the vendor id that corresponds to the specific vendor and where the sort keys begin with "u#". This is possible when using the operator "begins with" for the prefix of the sort key. The

prefixes are determined when designing the database. The following prefixes and keys have been used for the different entities:

- Vendor: "v#vendorId" for partition key and sort key
- User: "u#userId" for partition key and sort key
- Subscription: "v#vendorId" for partition key and "u#userId" for sort key
- Delivery: "v#vendorId" for partition key and "d#deliverytime#u#userId" for sort key

Three of these entities also have one or more global secondary indexes (GSI) to support queries that cannot be performed with the composite primary key alone.

- Vendor: "vendor" for GSI partition key and "v#vendorId" for GSI sort key.
- Subscription: "u#userId" for GSI partition key and "v#vendorId" for GSI sort key.
- Delivery: "u#userId#v#vendorId" for GSI1 pk and "deliverytime" for GSI1 sk, "u#userId" for GSI2 pk and "deliverytime" for GSI2 sk.

One example of a query using the global secondary indexes is retrieving all the vendors in the database. In this query the partition key is "vendor", and the sort keys starts with "v#". Another example is retrieving all the subscriptions a user has with different vendors. In this query the partition key is the user id, and the sort key starts with "v#".

4.3 REST API

A REST API is an application programming interface that conforms to specific architectural constraints, like stateless communication and cacheable data. A REST API can be accessed through a number of communication protocols. In this project the API is called over HTTPS, which is the most common protocol to access the REST API (John Au-Yeung, 2020).

Our REST API accepts request payload and sends responses with JSON. JSON is short for JavaScript Object Notation. It is a syntax for storing and exchanging data as text, and it is written with JavaScript object notation. When exchanging data between a browser and a server, the data can only be text. Any JavaScript object can be converted into JSON and sent to the server. Likewise, any JSON received from the server can be converted into JavaScript objects. This allows working with data as JavaScript objects with no complicated parsing and translations (JSON introduction, n.d.).

JavaScript has built-in methods to encode and decode JSON and different server-side technologies have libraries that can decode JSON. Since both client- and server-side of this application is written in TypeScript, a superset of JavaScript, encoding and decoding of JSON is available with built-in methods.

The resources include user profile, vendor, subscription, and delivery. Some of these resources corresponds to the entities in the database, and some of them combines attributes/properties from the different entities. For example, the subscription object returned when accessing /v/subscription includes information retrieved from both the user profile object and from the corresponding subscription object for a specific user in the database. The REST API designed for this project is summarized in Table 3.

Path	Parameters	Method	Return type
/userprofile		GET/PUT/DELETE	Userprofile
/vendor		GET/PUT/DELETE	Vendor
/vendors		GET	Vendor[]
/v/subscriptions		GET	Subscription[]
/u/subscriptions		GET	Subscription[]
/v/subscription	UserId	GET/PUT/DELETE	Subscription
/v/subscription	UserId, approval	PATCH	
/u/subscription	VendorId	GET/PUT/DELETE	Subscription
/u/subscription	VendorId, action	POST	Subscription
/v/deliveries	Start time, end time	GET	Delivery[]
/v/deliveries	UserId, number of new deliveries, start date	POST	Delivery[]
/v/deliveries	Delivery[]	PUT	Delivery[]
/u/deliveries	Start time, end time	GET	Delivery[]
/delivery	UserId, time	GET/PUT/DELETE	Delivery
/unpaidDeliveries	UserId, afterDate?, month	GET	Number
/cancelDeliveries	Delivery[]	POST	

Table 3 REST API

The methods used include GET, POST, PUT, PATCH and DELETE.

- GET retrieves resources
- POST submits new data
- PUT updates existing data
- PATCH applies partial modifications to existing data
- DELETE removes data

The “v” in the path symbolises that this is a path that someone with the vendor role will use. The “u” is for the user role. Paths that do not have a “v” or a “u” is common for both roles, but there might be restrictions implemented on the server, for example, a user will not be able to put or delete a delivery.

4.4 Client

4.4.1 Client-side Frameworks

Before the introduction of JavaScript in 1996, the internet was composed of static documents. With the interactivity that JavaScript added, the web became a place to do things, not only read things.

As JavaScript’s popularity steadily increased, developers who worked with JavaScript wrote tools to solve the problems they faced. These tools were packaged into reusable libraries, so they could be shared with other developers. This shared ecosystem of libraries helped shape the growth of the web.

Modern JavaScript frameworks has made it easier to build dynamic and interactive applications. A framework can be defined as a library offering opinions about how software gets built. These opinions allow for predictability and homogeneity in an application and allows software to scale to an enormous size and still be maintainable.

Every time an application's state changes, there arises a need to update the user interface to match the changes made. Building HTML elements and rendering them in the browser at the appropriate time takes a considerable amount of code and working directly with the DOM requires an understanding of how the DOM works in detail.

JavaScript frameworks were created to provide a better developer experience. The frameworks do not add new powers or properties to JavaScript, but they allow developers to write code that describes how the UI should look. The framework will then make the changes happen in the DOM "behind the scenes".

In addition to making DOM manipulation easier, frameworks offer other advantages as well. Examples include tools for testing, and a linting tool. The latter is an analytical tool used to flag programming errors and ensure that the code is stylistically consistent. Most frameworks also support to abstract the user interfaces into components; blocks of code that can be re-used, are maintainable, and that can communicate with one another. These components can be stored in a single file, called Single File Components (SFC), making it practical to change components should the need arise.

Finally, frameworks offer client-side routing; the possibility to let users navigate the different views in a single page application (MDN Web Docs, 2021).

4.4.1.1 Making components

Most frameworks have their own component model. Vue, which is used in this project, writes components with a templating syntax that lightly extends HTML.

Each component needs to describe the external properties they may need, the internal state the component should manage, and the events a user can trigger on the component's markup. External data that a component requires to render is called a property or a prop. The state of a component will persist as long as the component is in use. Like props, state can be used to affect how a component is rendered.

To be able to respond to the users of the application, the components need a way to respond to browser events. Each framework provides their own syntax for listening to browser events. An example is listening for the mouse click event, in Vue this event is called "onClick".

A component-based UI architecture allows for components to be used inside other components, and components can utilize and depend on other components. This will sometimes lead to child components needing data the parent component has. These props can be passed from parent to child, and the child can emit events to the parent.

To avoid having to pass props through many layers when nesting components in a large web application, frameworks provide a functionality known as dependency injection. Data needed can be directly passed to the components that need it, without passing it through intervening levels (MDN Web Docs, 2021).

4.4.1.2 Lifecycle

Web frameworks typically allow developers to perform certain actions at different stages of the component's lifecycles. These stages include when the component mounts, meaning the time it is appended to the DOM, when it renders, when it unmounts, and at many phases in between these. The Vue lifecycle hooks is shown as a diagram in Appendix 10.3 (MDN Web Docs, 2021).

4.4.1.3 Rendering

Frameworks track the current rendered version of the browser's DOM and makes decisions about how the DOM should change as components in your application re-render. Vue utilizes a virtual DOM model when rendering. A virtual DOM is a concept where the browser's DOM is stored in JavaScript memory. The application will update this copy of the DOM and compare it to the DOM that is actually rendered in the browser. The application compares the differences between the updated virtual DOM and the currently rendered DOM and uses those differences to apply updates to the real DOM (MDN Web Docs, 2021).

4.4.2 Vue in this project

Vue is a progressive framework (Vuejs, n.d.). In this context, progressive means that the developers can use parts of the Vue framework and progressively add more parts as they wish. It is possible to use Vue to enhance existing HTML on a single page with a script tag, and it is also possible to use Vue to write entire Single Page Applications (SPAs).

In this project we are using single file components. Components are reusable Vue instances with a name. They make it possible to design an application with logical components and re-use them several places within the application. A single file component (SFC) is a component that is contained in their own file (Horn, 2019).

The Vue components have been written using a special HTML template syntax, which also allows plain JavaScript functions. We are also using *Vue Class Component* in every SFCs. This is a library that makes it possible to make Vue components in class-style syntax. By defining the component in class-style, it is possible to utilize ECMAScript language features such as class inheritance (Vue Class Component, n.d.).

4.4.2.1 Styling of components

The different frameworks offer different ways to define styles for the components. When developing our web application, we used Vuetify, a complete UI framework built on top of Vue.js. It is the most complete user interface component library for Vue applications that follows the Google Material Design specifications (Wanyoike, 2019).

Material Design was created by Google in 2014, and it was inspired by paper-based design styles. Google's aim was to ensure that users would have a consistent user experience regardless of how users were accessing their products. The Material Design specification includes guidelines for typography, grids, space, scale, colour, and imagery (Chapman, n.d.).

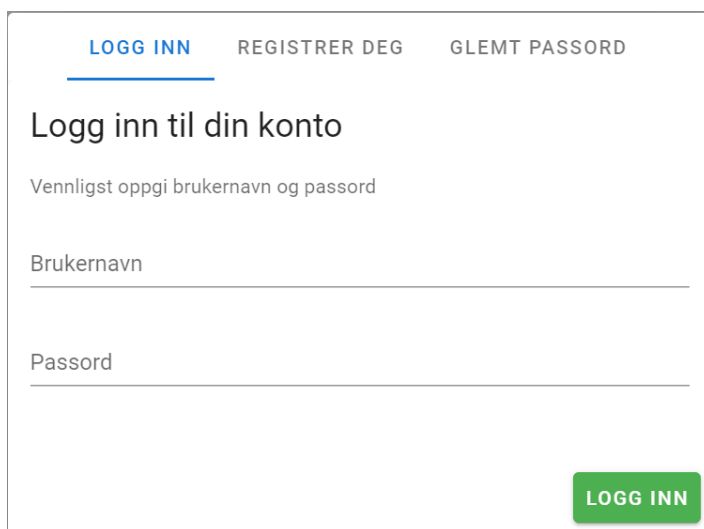
4.5 User management and security

To offer the users a secure way to sign into our web application, AWS Cognito was chosen to provide authentication, authorization, and user management. Amazon Cognito has two main components, user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for the application users. Identity pools are used to grant the users access to other AWS services (AWS Documentation, n.d.).

For our project it was sufficient to set up a single user pool to provide secure user registration and login through Amazon Cognito. It is planned to also let users sign in through social identity providers like Google and Facebook. Whether the users sign in directly or through a third party, all members of the user pool have a directory profile that can be accessed through a Software Development Kit (SDK).

In addition to providing sign-up and sign-in services, the user pool includes security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification. We decided to include multi-factor authentication when registering for an account at our web application, and when requesting a new password. Users will be asked to enter a code sent to their email when performing these actions.

The dialog box for login, registration and resetting a forgotten password is shown in Figure 5.



The image shows a web form for logging in. At the top, there are three navigation links: 'LOGG INN' (highlighted with a blue underline), 'REGISTRER DEG', and 'GLEMT PASSORD'. Below these is the heading 'Logg inn til din konto'. A sub-heading reads 'Vennligst oppgi brukernavn og passord'. There are two input fields: 'Brukernavn' and 'Passord'. A green button labeled 'LOGG INN' is positioned at the bottom right of the form.

Figure 5 Dialog box for login, registration and resetting of password.

After a successful authentication, Amazon Cognito issues JSON web tokens (JWT) that the web client will include as an authorization token in each HTTPS request. All the API endpoints are protected by the AWS API Gateway. This is done by using an authorizer function which will validate the JWT token before triggering the underlying Lambda function. The desired code will then be run, and the expected result is returned as a JSON document in the body of the HTTP response. This ensures that only authenticated users can access the API endpoints as the Lambda functions can determine which user is triggering the function. This is possible since the JWT token includes the user ID as part of its payload which is cryptographically signed.

5 EVALUATIONS

5.1 Evaluation methods

The three testing methods described in Chapter 3.4 on evaluation methods, have all been performed. This includes unit, integration, and user testing. The Mocha test framework and the assertion library Chai were used for performing unit and integration tests. The user tests have been performed in collaboration with the client and three of her customers.

5.1.1 Unit testing

Because it is considered a best practice, datetimes was stored as UTC in the database and backend code (Moesif, 2018). Since browsers use local time, it is necessary to convert datetimes between UTC and the user's local time whenever communication between the frontend and backend is required.

Functions for time conversion and other calculations involving datetimes have therefore been implemented as part of the development. This includes calculating how many deliveries a customer shall pay for each month, and dates for the deliveries. The dates are calculated based on the weekday the customer has chosen for deliveries, and which month the dates are to be calculated for. To make sure that these important calculations were correct, unit tests were made for the calculations involving time and dates.

5.1.2 Integration testing

Because the server-side code was distributed between several handler files and a separate file for all the database methods, integration tests were performed to ensure that the functions worked together correctly. All the methods for database communication were tested.

Integration tests were also used to ensure that the REST API worked as expected. One test class was made for each HTTP endpoint, and all the endpoints were tested. The tests were run in the terminal of Visual Studio Code, and a set of tests can be seen in Figure 6. Both passing and failing tests can be observed in the figure.

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Date and time tests
  ✓ From date to WeekTime
  ✓ Find next weekTimes
  ✓ Find next weekTimes
  ✓ Count deliveries in a month
  ✓ Find next delivery dates
  ✓ Millisecond test

Delivery tests
  ✓ Put, get and delete a delivery (2974ms)

Last delivery
  ✓ finds the latest delivery for a user (687ms)

Subscription test
  ✓ Putting, getting and deleting a subscription (969ms)

User profile test
  ✓ Putting, getting and deleting a userprofile (559ms)

Test of user subscriptions
  ✓ Getting list of subscriptions with users info from DB (2216ms)

Vendor profile test
  1) Putting, getting and deleting a vendor

12 passing (10s)
1 failing

1) Vendor profile test
   Putting, getting and deleting a vendor:

   AssertionError: expected '2' to equal '1'
   + expected - actual

   -2
   +1

   at Context.<anonymous> (src\tests\dbTests\vendorTest.spec.ts:53:51)
   at processTicksAndRejections (node:internal/process/task_queues:94:5)
```

Figure 6 Integration tests running in the terminal of Visual Studio Code.

5.1.3 Usability testing

During the project there has been frequent meetings with the client to show her the progression in development. These meetings have also been an opportunity for receiving feedback on implemented features. This ensured that the web application was developed in accordance with the client's needs and expectations.

To ensure that the web application has good usability, it is important to get feedback from the end users. User testing will therefore be performed with three of the client's customers.

The usability testing will consist of specific tasks. The customers will be asked to:

- Create a new user profile
- Order a subscription
- Change their password
- Cancel a delivery
- Pause subscription

The client will be asked to:

- Log in to her account.
- Approve new subscriptions based on the customer's address.
- Register customer payments that have been received for the coming period.
- Check the list of deliveries for the current day and sort the list using the drag and drop option.

5.2 Evaluation results

5.2.1 Unit and integration testing

The unit and integration tests were implemented in parallel with writing the code the tests were made for. This made it possible to check that all the code worked as expected, both when writing the functions for the first time, and after each change made throughout the project.

For time and date calculations the unit tests proved to be especially useful, as calculating dates was one of the more challenging parts of the development.

Separating the different tests by which methods and HTTP endpoints they were meant for, made it easy to see which methods passed and which failed.

5.2.2 Usability testing

Jacob Nielsen concludes "The best results come from testing no more than 5 users and running as many small tests as you can afford" (Nielsen, 2000). In this project only one usability test was planned, as the web application is quite small, and the client has been giving frequent feedback. However, if the time aspect of this project were longer, a second user test would be considered testing potential feedback given during the first round.

In the initial project plan usability testing was to be performed in the beginning of May. However, it was decided to postpone these tests until the end of May. The reason was an aim to implement as much of the functionality as possible before usability testing. This way, the users would not be distracted by missing functionality. The negative aspect of postponing the tests was less time left to improve the web application based on the feedback.

Stene chose three of her customers to perform usability tests, the tests were then done over video. Before starting, a scenario was given to the testers as well as a list of the tasks described in Chapter 5.1.3. The testers were asked to perform the tasks without any interaction with the interviewers before finishing. The last part of the test was used to interview the users about the user experience, and it was possible for them to give concrete feedback.

One of the observations that were made during the tests was that the customers wrote their home address when signing up for a subscription. The meals are usually delivered to the customers workplaces, and this should be solved by adding a field for delivery address. One of the testers also requested the ability to change the delivery address, due to her switching between working from home and from the office.

After getting approved and paying the invoice, the customers will see a calendar when they log in. In the calendar deliveries are presented as events marked in different colours. The deliveries the customers are subscribing to are green, and possible delivery days offered by the vendor are yellow. Information about the different colours is given to the user when an event is clicked. This led to some confusion and the feedback was to include more general information on the calendar view. There were also requests for more information on the invoice view. A screen capture from the calendar view can be seen in Figure 7.

The tests also revealed some software bugs. If the customer pauses and un-pauses their subscription, deliveries that was cancelled earlier will not be cancelled after the pause.

One tester commented that the web application was served over http instead https, causing the browser warning the user about privacy and security concerns. This is a temporary problem as we did not configure the secure connection yet.

All in all, the testers agreed that they liked the user interface and that it was easy to navigate and to solve the given tasks. The observation of the testers confirmed this. One of the testers navigated through the application and performed the tasks without any hesitation, while the two other users needed some time to figure out where the different information was located.

When asking the testers if there was anything they would change, two of the testers requested an option to un-cancel a delivery. However, due to Stene's need for predictability, it was decided earlier in the project that the users will have to contact Stene to un-cancel the delivery. Information about this is shown to the user when they click on an event in the calendar. The calendar view was mentioned as the thing the users were most satisfied with.

On an open question asking if the testers had any more feedback they wanted to add, one of the testers commented that she did not associate the colour scheme with that of the vendor (Stene uses a yellow bicycle for delivering meals). She also suggested to add some photos to make the web application prettier. However, web design is not a part of this project, and due to time limitations, this is not something that can be prioritized. One of the other testers commented that they appreciated the clean and simplistic look of the web application.

After performing these user tests, it became very clear why Jacob Nielsen considers just one single usability test to be insightful (Nielsen, 2000) as these three tests proved to be very informative. We are therefore glad we set aside some time for usability testing.

There was not enough time to let Stene test the system. However, we are planning to let her test, and teach her everything she needs to know about the system before delivering the final product.

6 RESULTS

During this project, an order system in the form of a web application has been developed. A user can register an account using their email address and a chosen password, and log in using this information. Functionality for resetting the password if forgotten has been implemented as well. When the user is signing up for the first time or resetting their password, they will be asked to enter a code sent to their email.

When a user logs in for the first time, they are presented with a registration form where they must provide information such as an address, telephone number and any allergies. They must also sign up for a subscription to Lunsj på Hjul and choose which day(s) and how many meals they want delivered. After registering, the user must be approved by Stene. User information is stored in a database.

After being approved by Stene, the customer must pay for a month of deliveries in advance. When payment is registered, a logged-in user will be able to see their upcoming deliveries. The deliveries are presented as events in a calendar. If the user clicks on one of the deliveries, they get the option to cancel that delivery. The calendar also shows all the deliveries Lunsj på Hjul offers on different days. These deliveries are in another colour than the deliveries the customer has paid for in their subscription. This was requested by Stene because she wanted to make it visible which days she offers deliveries, also after a customer has signed up for deliveries on specific days. It is planned to let the customer order extra deliveries by clicking these deliveries in the calendar. A calendar showing a logged in customer's deliveries can be seen in Figure 7.

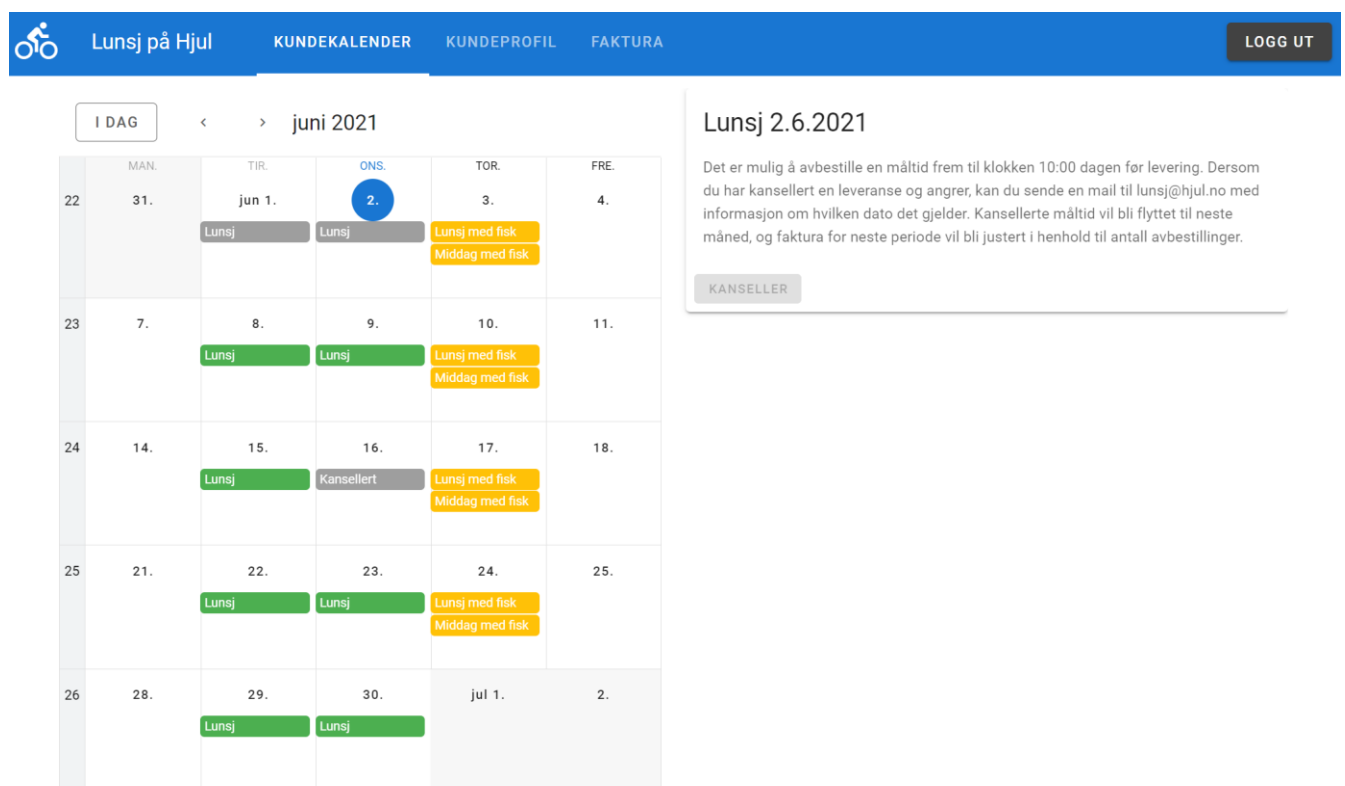


Figure 7 The calendar showing a customer's deliveries.

At the top of the page there is a menu bar where the user can navigate to different views. They will be able to see their profile with registered information, and the date for their next invoice. In the profile view shown in Figure 8, the user will be able to edit information about themselves and their subscription. There is also an option to pause their subscription. If the user changes information like an address, Stene will be notified by email. This last part is yet to be implemented.

Min profil

Navn Ole Olsen

Adresse Smauet 110 5000 Sentrum

Telefonnummer 12345678

Epost synnescool+user1@gmail.com

Mine allergier melk

Antall porsjoner 1

Valgt boks Engangsboks

Leveringsdag Tirsdag - Lunsj
Onsdag - Lunsj

[PAUSE ABONNEMENT](#) [ENDRE PROFIL](#)

Figure 8 Profile view for logged in customer.

On the invoice page the user will be able to see the date of the last delivery in the period they have paid for, as well as the date for when the next payment is due.

Code has been implemented to identify Stene as a vendor, thus giving her access to different content than her customers. When logging in, she is shown a calendar, which shows her delivery days and the number of deliveries for each day. When clicking on one of the events in the calendar, she is presented with a list of customers who have ordered deliveries that day. The list shows each customer's address, allergies and if they have cancelled the delivery. The list contains a search field, and it can be sorted by each of the values in the headers (for instance names and addresses). It is also possible to make a custom order by dragging and dropping the rows. It is possible for Stene to cancel all deliveries on a certain date in this view.

The menu bar lets Stene navigate to different views. Her profile view shows the information stored about her such as an address, company name and her delivery schedule. In the payment view she can choose a customer from a list and see information about when they paid the last invoice and how many meals they have paid for in the current period. Stene will be able to register new payments for each customer. The web application will calculate how many delivery days there is in the next period, based on which weekdays the customer subscribes to. This number and the start date for the next period can be changed by Stene, in case a customer is late with their payment or for other reasons. The payment view can be seen in Figure 9, and the registration for new payments can be seen in Figure 10. The last view will show all Stene's customers, separated into three different lists; those with an active subscription, those who have paused their subscription and customers that not yet have be approved. In this list she can approve new customers, one by one.

The screenshot displays the payment view for a logged-in vendor. The interface features a blue navigation bar at the top with the following elements: a bicycle icon, the text "Lunsj på Hjul", and menu items "ADMINKALENDER", "FIRMAPROFIL", "BETALINGER", "KUNDELISTE", and a "LOGG UT" button. The main content area is divided into two panels. The left panel, titled "Kundeliste", contains a list of customer names: Synne Synne, Synne Hansen, Olga Karidotter, Fjldjof Larsen, Penny Pobel, Solveig Vår, Stig Hugo, Jan Jankovich (highlighted), Ingrid Pingrid, Toril Apelthun, Jarle Parle, Katy Sekken, and Ole Olsen. The right panel, titled "Valgt kunde", shows details for the selected customer, Jan Jankovich. The details include: Navn (Jan Jankovich), Adresse (Lykkegata 8 5002 Øvreåsen), Telefon (56789012), Epost (synnescool+user5@gmail.com), Siste betalte levering (27.5.2021), and Ubetalte måltid i 2021-07 (9). Below the details are two buttons: "FORRIGE" and "NESTE". At the bottom of the right panel is a blue button labeled "REGISTRER BETALING".

Figure 9 The payment view for the logged in vendor.

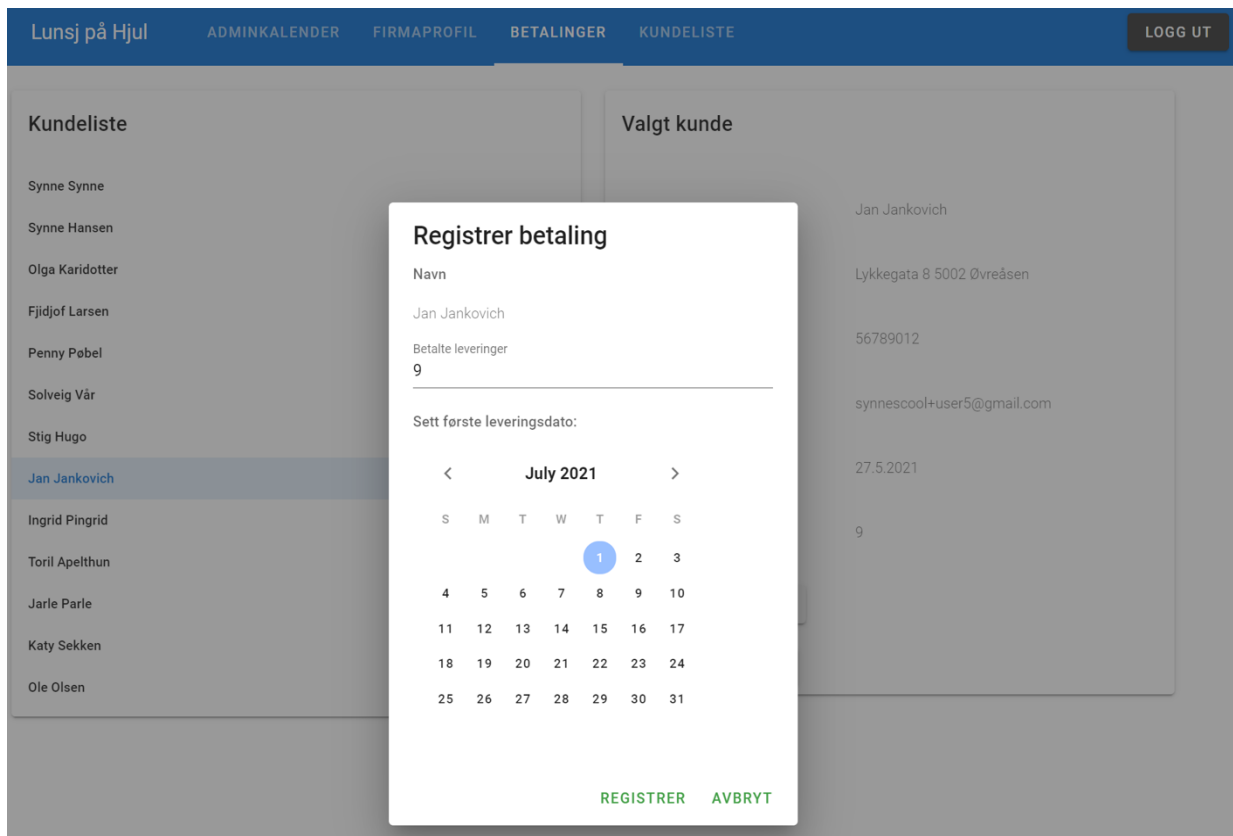


Figure 10 The registration of new payments.

7 DISCUSSION

As planned, quite some time was dedicated to learning the chosen technologies at the beginning of the project. Getting to know several new technologies in one project proved to be a time-consuming task. Learning to use Vue was one of the more challenging exercises as we had no prior experience with client frameworks. More time could have been spent on other tasks if we had some experience with frameworks before starting the project.

The choice of making the app scalable proved to be a good exercise when designing the database system and making the REST API. The workload was very similar to the workload for making a system supporting a single vendor only. Thus, no extra time had to be spent to make a scalable server architecture.

Developing a client-side that lets the customers choose between different vendors would have required some extra work. Since the web application will be used for a single company at this time, we decided to hide the fact that the system supports multiple vendors from the end users. This saved us some work and it also provided a better user experience, as there is only vendor to choose at this time. Should there be a need in the future, the client-side can easily be adapted to support more vendors and customers.

There are some features that were brought to our attention at a (very) late time of the project. This includes a need for the system to support users that buy deliveries on an ad hoc basis, without signing up for a subscription, and the possibility for the customer to change their delivery address based on where they are that day. This was not described by the client in the initial task, and it was not mentioned as a need in the initial planning meetings where a list of features and specification was made. These features will be added to the list of missing functionalities. We plan on implementing some of the missing functionality during the next couple of weeks. A list of the work that will be prioritized will be made together with the client. There is also a possibility for further work during the summer, but this will have to be planned together with the client.

In the beginning of the project, the implantation of the accounting program Fiken was discussed as an additional feature. This was originally suggested by the developers, as Stene mentioned she uses the program for accounting. By integrating with Fiken the system would have supported automatic generation of invoices. However, due to time limitations this possibility was not thoroughly researched, and it was therefore decided not to implement this functionality. When discussing whether to implement this or not, usability and the time saved for Stene was taken into consideration. It was concluded that Stene would not save much time having the option to send invoices automatically as she sometimes must adjust invoices manually. The developed web application will provide Stene with the information needed to generate invoices for each customer in Fiken.

8 CONCLUSIONS AND FURTHER WORK

This project has provided us with a steep learning curve and a lot of valuable experience as future software programmers and web developers. The client meeting and the user tests gave us valuable insights into software development. Topics covered throughout our studies have proven important in the process of developing this web application.

In this bachelor project the problem space was to solve the client's challenges of having to manually manage her customers subscriptions by creating a web application. The goal was defined as a list of core functionalities that the web applications should implement.

This core functionality included the following functionality for the customers:

- Registration and login.
- Deleting a subscription and the registered information.
- Changing the day of delivery.
- Date for next invoice.
- Cancelling deliveries and pausing the subscription for a period.

The core functionality included the following functionality for the client:

- Overview of how many and which deliveries that are to be prepared each day.
- Overview of coming invoices.
- Overview over customers and subscriptions.
- The possibility to accept or reject subscription requests based on delivery address.

Subsidiary goals included:

- Integration with Fiken accounting software for automatic invoicing.
- Email notifications for the vendor every time a new customer registers and need approval.
- Email notifications for customers when the vendor has approved a change of a delivery date.
- SMS notifications for customers when the lunch is delivered at the registered address.

At this time, none of the subsidiary goals have been implemented. The plan is to implement email and SMS notification before the client starts using the web application. Integration with Fiken has not been implemented, the reason for this was discussed in Chapter 7.

The initial solution idea was to develop a responsive web application which could be used on most devices, not depending on the size of the device or which operating systems being used. This is a desired feature as the client will use her phone while delivering lunches. The plan is to implement a responsive web design before the client starts using the web application. This will make the web application more user friendly, both for the client and for her customers.

The serverless technology has made our web application scalable, and it will be easy to add more vendors to the system should the need arise. A multi-vendor system will enable each customer to have several subscriptions with different vendors, while only needing one account and user profile.

9 BIBLIOGRAPHY

- Alex DeBrie.* (2020, February). Retrieved May 18, 2021, from The What, Why, and When of Single-Table Design with DynamoDB: <https://www.alexdebrie.com/posts/dynamodb-single-table>
- Alex DeBrie.* (2020, January 6). Retrieved May 18, 2021, from SQL, NoSQL, and Scale: How DynamoDB scales where relational databases don't: <https://www.alexdebrie.com/posts/dynamodb-no-bad-queries/#sql-joins-have-bad-time-complexity>
- Amazon.* (n.d.). Retrieved May 19, 2021, from AWS Whitepapers: <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/aws-lambdathe-basics.html>
- Amazon.* (n.d.). Retrieved May 19, 2021, from API Gateway: <https://aws.amazon.com/api-gateway/>
- Amazon.* (n.d.). Retrieved May 19, 2021, from API Gateway: <https://aws.amazon.com/api-gateway/>
- Amazon Documentation.* (n.d.). Retrieved May 19, 2021, from AWS Whitepapers: <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/introduction.html>
- Amazon Documentation.* (n.d.). Retrieved May 19, 2021, from AWS Whitepapers: <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/aws-lambdathe-basics.html>
- Amazon Documentation.* (n.d.). Retrieved May 19, 2021, from AWS Whitepapers: <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/aws-lambdathe-basics.html>
- Amazon Documentation.* (n.d.). Retrieved May 19, 2021, from AWS Whitepapers: <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/aws-lambdathe-basics.html>
- Amazon Documentation.* (n.d.). Retrieved May 19, 2021, from AWS Whitepapers: <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/the-event-object.html>
- AWS.* (n.d.). Retrieved April 12, 2021, from AWS: <https://aws.amazon.com/lambda/faqs/>
- AWS.* (n.d.). Retrieved April 12, 2021, from AWS: <https://aws.amazon.com/dynamodb/>
- AWS.* (n.d.). Retrieved April 12, 2021, from AWS: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>
- AWS.* (n.d.). Retrieved May 19, 2021, from <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/introduction.html>
- AWS Documentation.* (n.d.). Retrieved May 19, 2021, from AWS Whitepapers: <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/the-handler.html>
- AWS Documentation.* (n.d.). Retrieved May 18, 2021, from Amazon DynamoDB: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-general-nosql-design.html>

AWS Documentation. (n.d.). Retrieved May 18, 2021, from Amazon DynamoDB Developer Guide: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-sort-keys.html>

AWS Documentation. (n.d.). Retrieved May 18, 2021, from Amazon DynamoDB Developer Guide: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html>

AWS Documentation. (n.d.). Retrieved May 18, 2021, from Amazon DynamoDB Developer Guide: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html>

AWS Documentation. (n.d.). Retrieved May 18, 2021, from Amazon Cognito: https://docs.amazonaws.cn/en_us/cognito/latest/developerguide/what-is-amazon-cognito.html

AWS Documentation. (n.d.). Retrieved May 18, 2021, from Amazon Cognito User Pools: https://docs.amazonaws.cn/en_us/cognito/latest/developerguide/cognito-user-identity-pools.html

Bavosa, A. (2020). *Medium*. Retrieved April 10, 2021, from <https://medium.com/swlh/native-vs-non-native-mobile-apps-whats-the-difference-b3a641e06f52>

Bright, P. (2012, 10 03). *Microsoft TypeScript: the JavaScript we need, or a solution looking for a problem?* Retrieved from ars Technica: <https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/>

Brombach, H. (2021, February 5). *digi.no*. Retrieved April 2021, from <https://www.digi.no/artikler/telenor-rammet-av-flere-tekniske-problemer/506474>

Chai Assertion Library. (n.d.). Retrieved April 12, 2021, from Chai Assertion Library: <https://www.chaijs.com/>

Chapman, C. (n.d.). *Designers*. Retrieved May 19, 2021, from Why Use Material Design? Weighing the Pros and Cons: <https://www.toptal.com/designers/ui/why-use-material-design>

Conally, D. (2020, 07 20). *Liquid Web*. Retrieved April 12, 2021, from Liquid Web: <https://www.liquidweb.com/kb/what-is-serverless-a-beginners-guide/>

Emorphis Technologies. (2019, November 20). *medium*. Retrieved April 12, 2021, from <https://medium.com/emorphis-technologies/progressive-web-apps-vs-responsive-web-apps-how-they-are-different-from-each-other-f0cd3640747d>

Fox, L. (2015, August 14). *objective*. Retrieved April 11, 2021, from <https://objectiveit.com/blog/the-advantages-and-disadvantages-of-web-apps/>

Gantt.com. (2021). *Gantt.com*. Retrieved April 09, 2021, from <https://www.gantt.com/>

Gibb, R. (2016, May 31). *Stackpath*. Retrieved April 11, 2021, from <https://blog.stackpath.com/web-application/>

GIT. (n.d.). Retrieved April 12, 2021, from GIT: <https://git-scm.com/>

GitHub. (n.d.). Retrieved April 12, 2021, from GitHub: www.github.com

Griffith, C. (n.d.). *Ionic*. Retrieved April 11, 2021, from <https://ionic.io/resources/articles/what-is-hybrid-app-development>

- Höck, D. (n.d.). *LinkedIn*. Retrieved May 18, 2021, from <https://www.linkedin.com/pulse/sql-vs-nosql-whats-difference-david-h%C3%B6ck/>
- Horn, T. (2019, April 11). *Travis Horn*. Retrieved May 19, 2021, from Getting Started with Vue Single File Components: <https://travishorn.com/getting-started-with-vue-single-file-components-f29765a771a3>
- Interaction Design Foundation*. (2021, April 12th). Retrieved April 14, 2021, from Interaction Design Foundation: <https://www.interaction-design.org/literature/topics/usability-testing>
- John Au-Yeung, R. D. (2020, March 2). *The Overflow*. Retrieved May 23, 2021, from The Overflow: <https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/>
- JSON introduction*. (n.d.). Retrieved May 23, 2021, from W3 Schools: https://www.w3schools.com/js/js_json_intro.asp
- MDN Web Docs*. (n.d.). Retrieved May 19, 2021, from Cross-Origin Resource Sharing (CORS): <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
- MDN Web Docs*. (2021, April 9). Retrieved May 19, 2021, from Introduction to client-side frameworks: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction
- MDN Web Docs*. (2021, April 27). Retrieved May 19, 2021, from Framework main features: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Main_features
- Mocha*. (n.d.). Retrieved April 12, 2021, from Mocha: <https://mochajs.org/>
- Moesif*. (2018, November 19). Retrieved May 20, 2021, from Based on thousands of APIs, what is the best approaches and format for handling timezone, timestamps, and datetime in APIs and Apps: <https://www.moesif.com/blog/technical/timestamp/manage-datetime-timestamp-timezones-in-api/>
- Nielsen, J. (2000). *nngroup*. Retrieved May 19, 2021, from <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>
- nodejs*. (n.d.). Retrieved April 12, 2021, from nodejs: <https://nodejs.dev/learn/the-v8-javascript-engine>
- Rehkopf, M. (n.d.). *atlassian.com*. Retrieved April 10, 2021, from <https://www.atlassian.com/agile/kanban/kanban-vs-scrum>
- Roberts, M. (2018, May 22). *martinfowler.com*. Retrieved March 26, 2021, from <https://martinfowler.com/articles/serverless.html>
- Skjølvsvik, T., & Voldsund, K. H. (2016). *Forretningsforståelse* (1st edition ed.). Oslo, Norway: Cappelen Damm Akademisk.
- Software Testing Fundamentals*. (n.d.). Retrieved April 11, 2021, from Software Testing Fundamentals: <https://softwaretestingfundamentals.com/integration-testing/>
- Stene, A. L. (n.d.). *Lunsj på Hjul*. Retrieved from Stene Matglede : <https://www.stenematglede.com/lunsjpaahjul>

Visual Studio Code. (n.d.). Retrieved April 12, 2021, from Visual Studio Code:
<https://code.visualstudio.com/docs/supporting/faq>

Vue Class Component. (n.d.). Retrieved May 19, 2021, from <https://class-component.vuejs.org/>

Vue.js. (n.d.). Retrieved April 12, 2021, from Vue.js: <https://vuejs.org/v2/guide/>

Vuejs. (n.d.). Retrieved May 19, 2021, from Guide: <https://vuejs.org/v2/guide/>

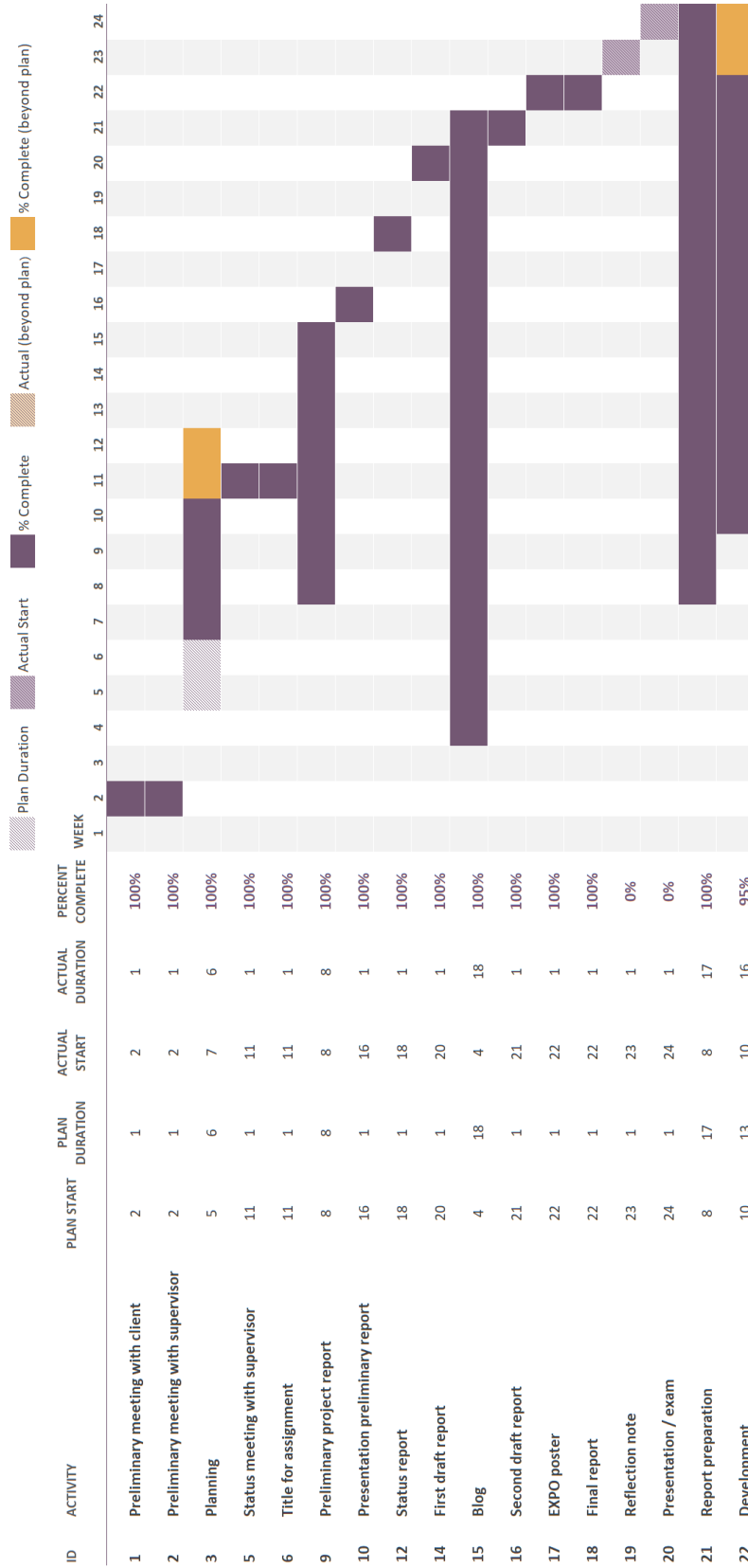
Vuetify. (n.d.). Retrieved April 12, 2021, from Vuetify: <https://vuetifyjs.com/en/introduction/why-vuetify/>

Wanyoike, M. (2019, June 26). *Sitepoint*. Retrieved May 19, 2021, from How To Get Started with Vuetify:
<https://www.sitepoint.com/get-started-vuetify/>

Weissman, S. (2018, December 5). *codingsans.com*. Retrieved April 10, 2021, from
<https://codingsans.com/blog/kanban-in-software-development>

10 APPENDICES

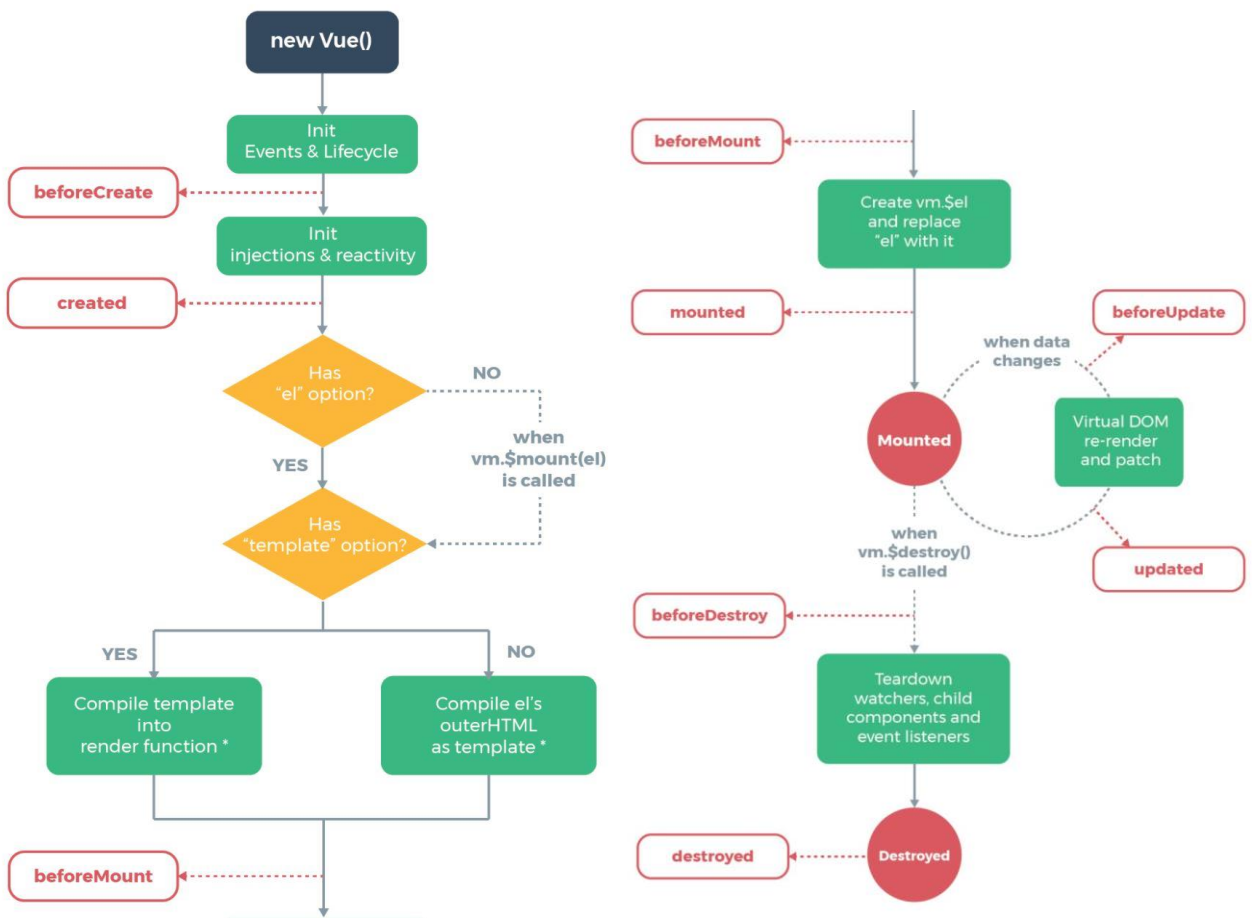
10.1 GANTT Chart



10.2 Risk List

Risk	Consequence	P	C	RF	Measures
Lacking knowledge about new technologies	Takes time to get started on the project	1	3	3	Group members reads up on different technologies, and helps the rest of the group if necessary
Uneven workload	Someone has to do more than others, may have an impact on if we have time to finish the project in time	2	2	4	Frequent meetings to make sure everyone gets the same amount of tasks. Everyone needs to do the assignments they get.
Illness/Absence/Covid-19	More work for the other group members for a period of time. Might be problematic if one of the members have knowledge the others do not have	3	3	9	Follow reccomended covid measures, knowledge transfer
Time	Do not have time to complete the assignment, or do not have time to implement all desired functions	4	3	12	Follow the plan and iterations
Problems with work tools	Can not work with the project	1	3	3	All group members have newer computers, fix the possibility to borrow/rent computers if necessary
The customer can not use the application/ misunderstood the customers needs	The application is not satisfying the customers needs	1	5	5	Frequent customer meetings, make sure to finish core functionality early, testing
The application is not user friendly	Application will not be used	2	4	8	Frequent customer meetings, let users test the application as soon as possible
The development gets prioritized over writing the report	The report will not be finished in time	2	3	6	Work regulary with the report, stick to deadlines

10.3 Vue lifecycle hooks



Source: <https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram>.