NTNU

INFORMATICS PROJECT II - IT2901
PROPERTY DATABASE
KOMMUNAL RAPPORT

Bachelor project report

Group 9:
Lasse Drevland, Caroline Odden, Kasper Rynning-Tønnesen, Tor August Hødnebø, Kathrine Løfqvist, Adrian Hundseth

May 28, 2016

Abstract

This report concerns a project conducted by six bachelor students at the Department of Computer and Information Science, in the course IT2901 Informatics project II, at the Norwegian University of Science and Technology. The project was purposed by, and developed for Kommunal Rapport, a business to business newspaper.

The goal of the project was to create a web service, giving customers and journalists the ability to investigate property-transactions conducted by municipalities in Norway. The report documents the complete process from prestudy to delivery, as well as technical details about the solution. The customer provided a dataset containing the transaction details. Around which the team built a single-page web application implemented in AngularJS, with a PHP back-end connected to a MySQL database. It features graphical visualization of transactions, several methods of discovering transactions as well as flagging of suspicious transactions.

The solution was influenced by similar applications, and will serve as a powerful investigative tool to journalists and customers of Kommunal Rapport. We would like to thank our supervisor Sofia Papavlasopoulou for her valuable feedback, as well as our customer Kommunal Rapport for the interesting project and great partnership.

Trondheim, 30th of May 2016

Lasse Drevland

Adrian Hundseth

Tor Hødnebø

Kathrine Løfqvist

Caroline Odden

Kasper Rynning-Tønnesen

Contents

1 Introduction

This document is the report for the project in the course IT2901 - Informatics project II. The goal of the project was to create a web application that makes it possible for both customers and reporters to detect anomalies in real estate transactions in Norway involving municipalities. This introduction presents the course, the team and the customer. It briefly explains the problem that was to be solved, the milestones that were planned, the goals of the project and the solution. A user manual (Appendix D) and an installation guide (Appendix E) is provided in the appendix.

1.1 The course

The learning goal of the course IT2901 - Informatics project II is to give the students practical experience in system development by performing a systematization project with a customer [14]. At the start of the semester, each customer presents themselves and their product idea. The students are divided into groups and each group is assigned a supervisor. The groups make a prioritized list of the presented products, and are assigned to the product that suits them best. During the semester each group works closely with the customer to meet all their product requirements.

1.2 The team

The group consisted of six students on their third year of a Bachelor's degree in Informatics, at the Norwegian University of Science and Technology. All group members have experience with system design and programming from previous courses and projects. More specifically, each member has experience with Java, Python, MySQL, Javascript, HTML and CSS. Each member has been part of every step of the development process, but was given a specific area of responsibility as listed below.

- **Tor August Hødnebø** has some additional experience with LaTeX, which is the document preparation system that were used to write the report, as well as general report writing. He was given responsibility for the documentation, such as report tasks and deliveries.
- **Caroline Odden** has had several leader positions in the past, namely leader of the welcome committee at Online and deputy chairman of the programme union in Informatics. She was the obvious choice when deciding upon a leader for the project. The leader specific tasks were attending leader meetings, and having supervisory control over the tasks.
- **Lasse Drevland** has extensive knowledge of and experience with databases and the programming language SQL. He was therefore chosen to be in charge of the database. This included keeping track of the database tasks and keeping contact with the data supplier, Ambita.
- **Adrian Hundseth** has extensive experience with Javascript, AngularJS, JQuery and PHP. He was chosen to be responsible for the back-end and the testing, and was tasked with keeping track of the back-end assignments.
- **Kathrine Løfqvist** has been a member of the business committee in Online for three years and is experienced in communication with companies. She was chosen to be the communication administrator. Her tasks consisted of writing and sending mails to the project contacts.
- **Kasper Rynning-Tønnesen** has extensive knowledge and experience with website design, web development, and the languages Javascript, jQuery, PHP, CSS3. He was the chosen to be responsible for the web-solution of the project. The web responsible was in charge of the server and front-end task distribution.

1.3 Customer

The customer, Kommunal Rapport, a business to business newspaper focused on covering news related to municipalities in Norway. Kommunal Rapport was founded in 1987, and is located in Oslo. They are organized as a corporation, owned by Kommunens Sentralforbund. They have two editorial versions of their product, each with different content. The paper version is focused on giving an overview and insight into cases important to the municipalities, the digital version offers current news every day.

In the recent years, Kommunal Rapport published "Kommunebarometeret" [19], a measure put together based on different indicators from the municipalities. It consists of four tables that display the placement of the municipalities in terms of schools, elderly care, economy, and health. They published "Leverandør-databasen" [21] as well, which is a compilation of providers that deliver services to the municipalities in Norway. Kommunal Rapport recommended the latter one as inspiration for the property database.

1.4 Problem to be solved

Kommunal Rapport, hereinafter referred to the customer, purchased a dataset containing every property transaction that every municipality in Norway has ever been part of. The municipality's involvement can be as either the one buying or selling, and going back as far as the mid 1800s. The dataset was structured in a file with comma-separated values, which was difficult to utilize efficiently for their journalists. The dataset in its original form was 34 columns wide, and 1.8 million rows long.

The customer wanted their journalists and paying users to be able to search for persons, municipalities or organizations in the dataset. The search results had to be presented in an intuitive way, allowing the user to easily find transactions they were searching for. Another requirement was an interactive map as an optional way of specifying a search for a county or municipality. The customer wanted a solution where their journalists and paying customers could easily find anomalies in transactions between municipalities and other participants. Another problem to be solved was to integrate the already existing log-in solution from Kommunal Rapoort, that are developed by Ramsalt. This allowed the users to have one account across Kommunal Rapports´ web applications.

1.5 The goal

The dataset involved a large quantity of data, making visualization an important part of the system, as well as an effective way of sorting the data. The goal was to structure the data and make it more accessible, and present it through a web-based interface where their journalists and subscribers could access and utilize the data efficiently in their research.

The customer wanted three different ways of searching the database:

1. Through plain text search, where a customer could search for a person, organization, county or municipality.
2. Select a county or municipality using a map.
3. Select a specific municipality using a dropdown selection.

1.6 Milestones

The project was divided into five parts. The Gantt chart (Figure 5) shows the overview of the different parts of the project. The task "Technical Design" encompasses development of the database, back-end, front-end function and design. There were five major development milestones, as shown in table 1. The deadline for the requirements was planned to be at the end of week 3. The following milestone was the database, with a deadline to be functional at the end of week 8. The front-end design was supposed to be done in week 14, with the front-end and back-end function in progress. The front-end function was planned to be finalized in week 17, with testing still in progress until week 18.

Table 1: Milestones

Parts	Week
Requirements	3
Database	8
Front-end design	14
Front-end function	17
Testing	18

1.7 The solution

The solution consisted of a functional database paired with a view. The database contained a dataset purchased by the customer, which was processed and inserted into a MySQL database. The web application, referred to as the "property database", contains a search field where the user can type in the name of a person, organization, county or municipality. These different types are collectively referred to as "agents". The user will then be presented with a table generated by the result of the query. If the user selects a row in the table, another table will replace the existing view and presents the transactions specific to the selected agent. Transactions with anomalies are flagged with color coded numbers. If a user selects a property, the transaction history for a given property will appear. The new site includes two tabs, one with a timeline for the transaction history, and the other tab contains a table with the results. The transaction history is shown in a chart at the top of the page, to give the user a brief overview.

Instead of using the text field for searching, the user can select an agent type from the first dropdown menu, the second menu specifies county and the third municipality, but only if county has been specified. The result table will appear as if you typed in the specific municipality in the text field and selected it from the first result table. Alternatively, the map can be navigated to choose a specific county, and a larger and more detailed map of the county will appear. In the map of the county the user can choose a municipality, and the result table will appear.

1.8 Report outline

This section describes the content of each section in the report.

- **Section 1** is the introduction. This section will introduce the project, stakeholders, and give an overview of what the report presents.
- **Section 2** is about the prestudy for the project. Sections such as alternate solutions, the different process models and the research that was done prior to starting the project will be discussed.
- **Section 3** is about the project management and project planning. Here the collaboration, what methods were used, and the risk management is explained.
- **Section 4** describes the functional and non-functional requirements for the project. The use-cases will be presented, both textually and with diagrams.
- **Section 5** is about the development environment. The different tools that were used will be discussed; the programming languages, testing tools, the frameworks, and collaboration tools. Lastly, the deployment how the project was deployed on a server.
- **Section 6** is about the design and implementation. The product architecture and the different diagrams will be presented. This section will give a more detailed explanation of the system.
- **Section 7** is about the testing. In this section the test methods are presented. This section explains which methods were used, and the results of the test implementations.
- **Section 8** is about the evaluation. The evaluation is a review of the project development and implementation.
- **Appendix** contains additional information about the project.

2 Prestudy

This section presents the alternate solutions and the different process models that were considered for the project. It evaluates other products that accomplish similar goals to what the customer was requesting. The different development methods were weighed and measured, and the decision is presented in the conclusion of this section.

2.1 Alternate Solutions

There exists similar solutions that provide some of the same functionality as the property database, but they are all intended for consumers. Kommunal Rapport is a newspaper directed towards Business To Business, and the database created in this project is intended for both professionals and normal users. Other solutions had removed transactions related to the municipalities. Compared to other solutions, the property database was based on transactions where the municipalities were in focus.

The alternate solutions that were discussed were created by Adressa.no [3], Ditt Næringsliv [13], Budstikka.no [12], Forbrukerrådet [20], and Osloby [23]. The two solutions discussed in the following sections were the ones that inspired the project the most.

2.1.1 Osloby

Osloby had developed a solution for looking up property values in Oslo and Akershus. The solution enables users to find an estimated price for a given property, and lists its transaction history. The solution includes a feature where the users can see the most recent purchases nearby. The search requires the property address to be known.

Osloby's solution excelled at usability, and contained several components requested by the customer. Their solutions to search and transaction history were especially good and were used as an inspiration for the property database. The biggest drawback was that their search only retrieved exact matches, and required the address to be located in Oslo or Akershus. The main difference between the property database and Osloby's solution was the scope of the data.

2.1.2 Forbrukerrådet

Forbrukerrådet has conducted annual evaluations of municipalities in Norway since 2005. Their web application shows seven of these test with rankings based on the quality of different sectors. Each municipality is given a score between 1-100, and is shown in a list and on a map of Norway. The user is able to navigate to a given municipality by map, scrolling through a list or by searching.

Forburkerrådet's solution included some functionality requested by the customer. The property database drew inspiration from their use of map as a navigation tool, and the range of different charts and tables used to present the data. The problem was the lack of usability and explanation of the presented data. Forbrukerrådet's solution displayed too much information at once, so it was decided that limiting the amount of data shown to the user would be beneficial.

2.1.3 Conclusion

The alternate solutions not discussed in the previous section were in general lacking in user-friendliness and did a bad job of conveying their data. The web pages that were presented were too focused on individual properties, contrary to the goals of this project. Most of the alternate solutions were centered around a search-field, which was incorporated into the property database. This enhances usability as the majority of people are familiar with this layout.

The evaluation of the alternate solutions was used when setting the design guidelines, and creating the requirements for the project. Due to the inadequate user-friendliness found in some of the alternate solutions, the team decided to focus on the usability and presentation of data. Additionally, the placement and usage of the search bar and the map featured in the Osloby's and Forbrukerrådet's solutions, influenced the final design.

2.2 Development method

During the planning phase the team had to decide which development process would be most beneficial for the project. The choice was between plan-driven development methods and agile development methods. A plan-driven process is a process where all activities are planned in advance, and all progress is measured relative to that plan. In agile development the planning and execution is done incrementally and takes an iterative approach to software development. The agile development is supported by frequent and small releases during the development period. An agile method makes it possible to respond quickly to change in requirements without adding excessive amounts of extra work. This section presents two agile development methods, Extreme Programming in section 2.2.1 and Scrum in section 2.2.2, and one plan driven development method called Waterfall in section 2.2.3.

2.2.1 Extreme Programming (XP)

Extreme Programming is an agile development method. In extreme programming the developers work in pairs, which is called pair programming, and develop unit tests before the main code. All tests must be successfully executed before new code is integrated into the system. The extreme programming method keeps the design simple and the releases frequent.

When using extreme programming the customer is intimately involved in the process. The customer is a member of the development team and is responsible for bringing the system requirements to the team. The requirements are expressed as scenarios, called user stories, which are implemented directly as a series of tasks [27]. The figure 1 illustrates the extreme programming process.

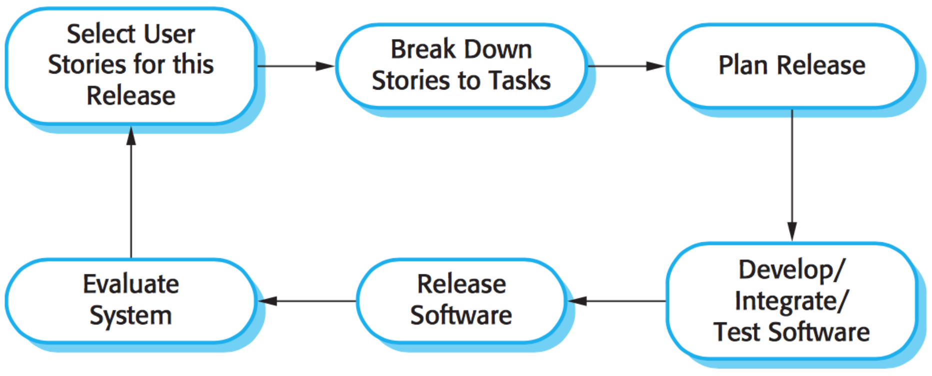

Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

Figure 1: Extreme Programming [27]

2.2.2 Scrum

Scrum is an agile development method, and figure 2 shows the scrum process. The first step is the planning stage, followed by a series of sprint cycles, and then project closure. When using Scrum, the most central phase is the sprint cycles. The most important information to this process is the following:

1. A sprint normally lasts between two weeks and a mont.

2. At the beginning of each sprint the product backlog is planned. The product backlog is the list of work to be done on the project, and the distribution of hours can be calculated using Planning Poker 3.1.7 This is reviewed in the assessment phase of the sprint, and the priorities and risks are assigned to each task. The customer is closely involved in the process, and can introduce new requirements and tasks.

3. In the selection phase the team works with the customer to select the functionality to be developed in the development phase

4. After the team has established the functionality to be implemented, they start the development. Every day during this phase, a short daily meeting is held, where the whole team is present. The purpose of this meeting is to review progress and, if necessary, re-prioritize work. During this stage the Scrum-master is the communication channel, allowing the development team to work undisturbed.

5. When the sprint is finished, the work is presented to the stakeholders. Then the next sprint cycle begins [27].

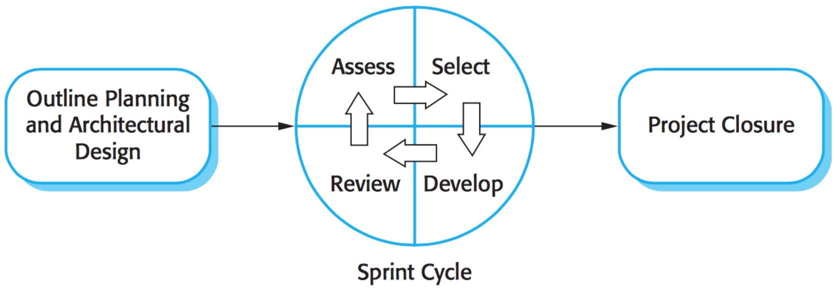


Figure 2: Scrum process [27]

2.2.3 Waterfall

The waterfall method is a plan-driven development method. In this model you must plan and schedule all process activities in advance. Figure 3 shows how the waterfall method advances from one phase to the next. The next phase should not be started until the current phase has been completed.

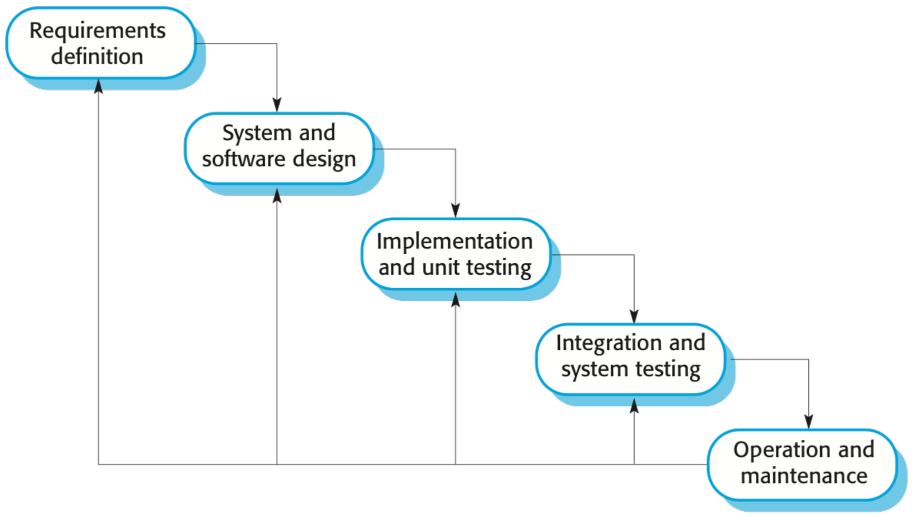

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 3: Waterfall model [27]

The waterfall model is described by:

1. **Requirements analysis and definition** is the stage where system constraints and goals are established by consultation with the users of the system. After that the requirements and goals are elaborated upon and functions as the system specification.

2. **System software and design** is the stage that allocates the requirements by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions/components and how they interact with each other.

3. **Implementation and unit testing** is the stage that realizes the software design as a set of programs. The unit tests involves verifying that each unit meets its requirements.

4. **Integration and system testing** is the stage that integrates all the individual program units and tests the complete system. This is to be sure that all the requirements have been met. After testing, the system is delivered to the customer.

5. **Operation and maintenance** is the stage where the system is installed and put into practical use. This phase involves correcting any errors that may have occurred, improving the implementation of the system units and enhancing the system as new requirements are discovered [27].

2.3 Decision of development method

The agile development strategy Scrum was chosen for the project. The ability to quickly adjust to the customer's needs was an important part of this decision. The team discarded the plan-driven development method, and chose an incremental delivery strategy. An incremental delivery strategy revolves around delivering software continuously, resulting in more frequent feedback from the customer. When using Scrum, the software could be delivered after every sprint. An agile development method is a very suitable method when the system is not excessive and is developed by a small co-located team. The customer wanted to be a part of the development process, giving feedback after every iteration, and making sure the team stuck to the customer's vision and to contribute new ideas to be implemented. The customer's involvement gave the group the opportunity to ask specific questions about the product at any point during the development.

Another agile development method is extreme programming (Section 2.2.1). Relative to Scrum, extreme programming is a test-driven development method where unit tests are written before the team starts the development. Writing the tests at the start is beneficial and assures functioning code. XP and Scrum are similar in many ways, and both of the development methods have iterations lasting between one to four weeks. In extreme programming the team must start with the highest prioritized requirement from the customer, but in Scrum the team can choose which tasks to start working with. This was one of the reasons why it was chosen to use Scrum. Scrum does not allow more requirements to be added during a sprint, the customer has to wait until the next sprint. In XP the requirements can change during an iteration. The customer was not able to be a part of the team, in the way that XP requires, and therefore Scrum was a better choice than XP.

2.4 Decision of frameworks

Due to large amount of data to search and present, decision of framework was an essential part of the project. This decision revolved around front-end frameworks (Section 2.4.1) such as ReactJS [26] and AngularJS [5], and was an important decision due to the responsive design.

2.4.1 ReactJS and AngularJS

React is commonly used as the View in a Model-View-Controller type of program. It was developed by Facebook and Instagram, and is on the rise in popularity. React was considered for most of the beginning of the project as the View part, but this was discarded due to the lack of documentation and tutorials on the internet. This lead to the final choice of AngularJS, which had a lot more tutorials and documentation. AngularJS is meant to be used as both the View and Controller in a program. See section 6.1 for explanation of View and Controller.

3 Project Management

This section presents how the project was managed, how the team was organized and the planning of the project with Work Breakdown Structure and Gantt chart. The estimated development time of each sprint will be shown in the section 3.1.3

3.1 Development method

In section 2.3 there was decided to go with the development method Scrum. The project structure and team organization was assigned after Scrum guidelines.

3.1.1 Project Guidelines

It was decided it would be beneficial to set the duration of each sprint to two weeks. This gave time to make significant progress, and meet with the customer and the supervisor every sprint. Working iterative made it possible for the customer to continuously give feedback on the product, as the developers updated the website with features almost every week. A summary of all iterations is shown in section 3.1.3 and a more specific description of each iteration is in the appendix B. In every iteration there was held a stand-up meeting every morning before the team started to work. The stand-up meeting was held so that every team member was updated on each others work at all times, and knew what to work with that day. Conducting the meetings standing up contributed to effective meetings. After every sprint a meeting with the customer was held, and during the sprint there was held a meeting with the supervisor.

3.1.2 Team Organization

In Scrum the team consists of the product owner, a Scrum master and the development team.

Product owner
The product owner represents the customer, and is the customer's voice. The customer for this project was Kommunal Rapport and the product owner was Henning Aarseth.

Scrum master
The Scrum master is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog. The Scrum master also communicates with the customer and other stakeholders. During a sprint, the scrum master also protects the team from external distractions [27].

Development team
The development team consists of members with different development background. It is important that the team has all the competence needed to finish the project, because it is the development team that does all the development, including design, and testing.

Table 2 shows all the team members area of responsibility, which are explained more in detail in section 1.2. In this project, Caroline Odden was the Scrum master and managed all the tasks that came with this role. The development team consisted of the whole group.

It is important to emphasize that all the members have been part of every step of the development, the roles mentioned in table 2 only regards the individual responsibilities.

Table 2: Roles

Named	Role
Caroline Odden	Leader and Scrum master
Lasse Drevland	Database developer
Kasper Rynning-Tønnesen	Front-end developer
Adrian Hundseth	Developer, testing responsible
Kathrine Løfqvist	Communication responsible
Tor Hødnebø	Documentation responsible

3.1.3 Iteration overview

Table 3 shows how the project was planned towards the deadline. The first phase of the project was the startup, with a duration of five days. This phase was not its own iteration, just a phase to get the project started. There were seven planned iterations for this project, each lasting for two weeks, and had ten working days. For more details about each iteration see appendix B. The finalizing period was from eight to ten days, and the remaining work on the different parts was to be finished in these iterations. The final project had to be delivered at the deadline of 30.05.2016.

Table 3: Iteration overview

Period	**Range**		
Startup	25.01-31.01.2016	5 days	120 hours
Iteration 1	01.02-14.02.2016	10 days	240 hours
Iteration 2	15.02-28.02.2016	10 days	240 hours
Iteration 3	29.02-13.03.2016	10 days	240 hours
Iteration 4	14.03-27.03.2016	10 days	60 hours
Iteration 5	28.03-10.04.2016	10 days	220 hours
Iteration 6	11.04-24.04.2016	10 days	240 hours
Iteration 7	25.04-08.05.2016	10 days	240 hours
Project finalization	09.05-22.02.2016	10 days	240 hours
Report finalization	23.05-30.05.2016	8 days	192 hours

3.1.4 Burndown Chart

A burn down chart is an important tool in agile software development. It shows the remaining work and time in a given iteration. It is both useful for seeing how the work is progressing during the iteration and for seeing the efficiency and quality of the estimations at the end of it.

In this project the actual charts were created at the end of each iteration with the help of the data continuously added to the backlog as tasks were being completed.

The burn down charts for each iteration can be found in appendix B.

3.1.5 Meetings

Customer meetings
In this project there was decided to have sprints lasting two weeks, and there was scheduled a meeting with the customer at the end of each sprint. The meeting was held over Google Hangout and would normally last one hour. The meeting agenda was focused on the requirements that had been meet since the last meeting, feedback from the customer and discussing the plan for the next sprint. There was one customer meeting held in Trondheim with the customer. Meeting minutes were written, see appendix A.3.1.

Supervisor meetings
Monday every two weeks there was a meeting scheduled with the group supervisor. The meetings were held in "IT-bygget" at 10:00, and would normally last between 30 to 45 minutes. The required preparation from the group was weekly status reports, see appendix A.1. The meeting agenda was then focused on questions from the group, which the supervisor had received and prepared before the meeting. During the meetings the supervisor also gave feedback on the project report.

3.1.6 Time plan

The team had weekly working hours scheduled for every iteration.

Table 4: Timeschedule

Day	Time
Monday	08:15 - 12:00
Tuesday	08:15 - 12:00
Thursday	08:15 - 10:00
Friday	12:15 - 16:00

The working hours always started with a stand-up meeting, explained in section 3.1.1. After that all team members started working with assigned tasks. Wednesday there was not scheduled a weekly meeting, due to other courses the team members had to attend. If there was an important discussion, questions regarding the whole team or something else important, the team could meet at school to discuss this outside scheduled hours.

The goal was that each team member would work for 20 hours a week, and since the weekly meetings only covered 14 hours, there was expected that every team member would work 6 hours when they had time.

3.1.7 Estimation

Planning poker is one way to estimate the workload for each task in a sprint. Each member is given a stack of card with a given number on them. When deciding the estimation of a task, everyone picks the card with the appropriate number of hours, and the group members revealed their cards at the same time. If the difference between the estimates was too big, the group discussed the results, so that each member had the ability to be heard. After the discussion, this was repeated until the group had come to a solution.

One of the advantages with planning poker is that none of the group members are affected by other proposed estimates. Each group member have to give their own estimate and explain the group why they have chosen this. Planning poker can also be related to as a game, which can make the process more fun. On the other hand, planning poker can take more time than just a normal discussion. In this project the team decided to use planning poker to estimate each task the best way possible.

3.2 Project planning

In this project it was decided to structure the plan based on the Scrum method. As seen in the Gantt chart in section 3.2.3 the project was intended to have five phases. This was based on back-end to the database and its query to get the transactions, front-end search function and design, visual elements like charts and maps, and the report itself. How the project structure was planned to be performed during this spring is shown in Work Breakdown Structure (WBS) in section 3.2.1. Tasks like daily meetings and sprint planning was not included in the Gantt or WBS chart. This also applied to meetings with the supervisor and the product owner.

3.2.1 Work Breakdown Structure

In the Work Breakdown Structure (WBS) shown in figure 4, the packages in the project are shown and how they were divided. It was to give an indication about the overall parts the project encompasses. The WBS-diagram was split into five parts that were split into additional tasks connected with the superior task. The Management branch shows the parts this project was based on, such as the setup of the sprints, status reports, risk assessment, and meeting with the supervisor and customer. These tasks were important so the project process should go well. The planning branch specified the requirements, development method, the team organization and the milestones. It was important that these tasks were done well, allowing the project to have a steady foundation Technical design were the parts of the system that needed to be developed. The design and implementation of the back-end with the database, and the front-end regarding design and function, were important aspects. The fourth branch were the testing branch, this were included in the project to evaluate the system and have control of the development. Unit testing, integration testing, system testing with usability- and compatibility testing, and acceptance testing were tests that were planned and performed during this project. The Finalize-branch in the WBS was planned to include finalization of the product and the report, and deliver the product to the customer.

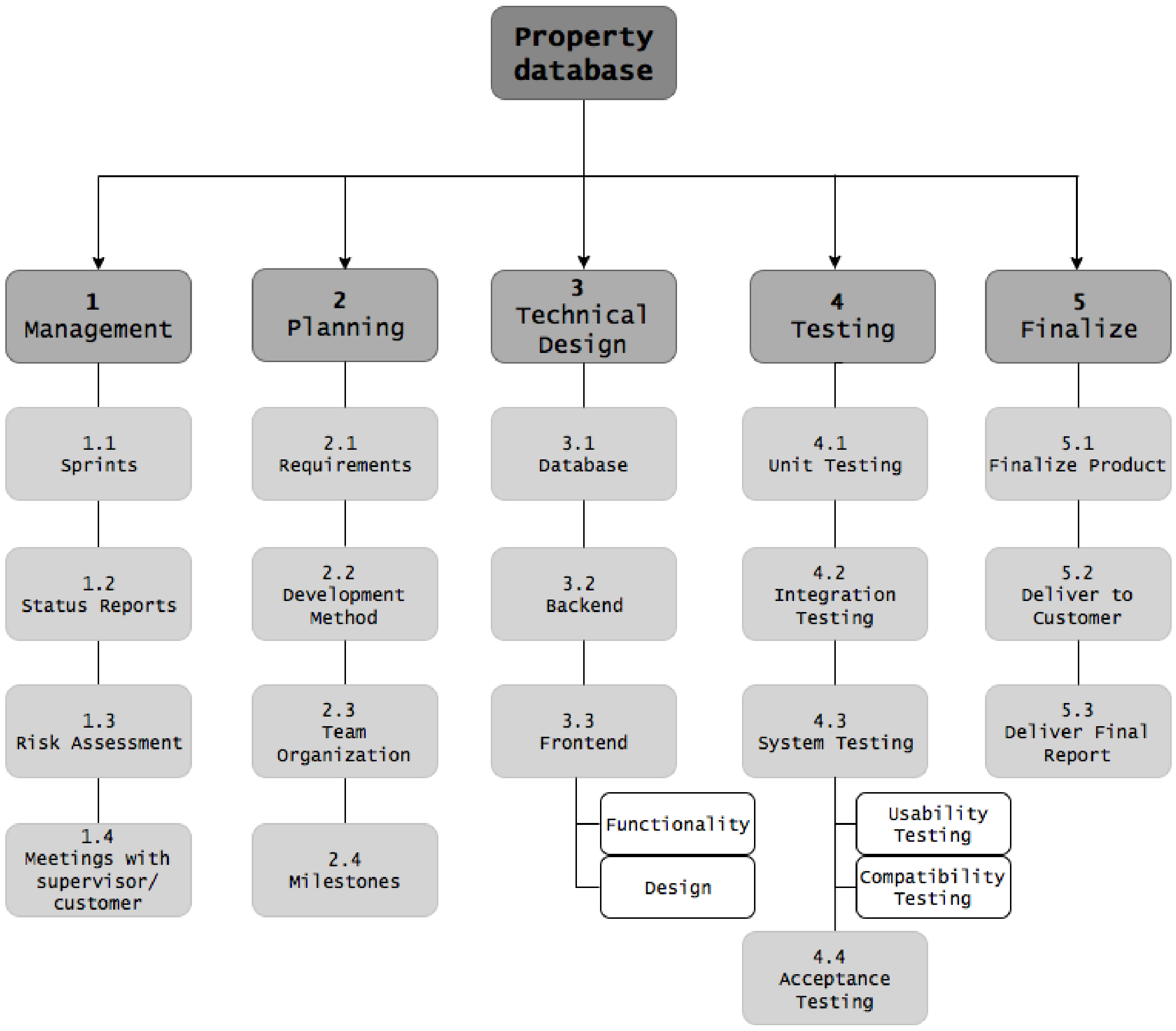


Figure 4: WBS Diagram

3.2.2 Work Breakdown Structure Description

This table supplements the Gantt chart Figure 5 and is a description of the work breakdown structure shown in Figure 4.

Table 5: Work Breakdown Structure Description

ID	Section	Description
1	**Management**	Continuous planning and macro-management of the project.
1.1	Sprints	Planning of the bi-weekly development cycles.
1.2	Status reports	Reports on the progress, made every other week.
1.3	Risk assessment	Continuous assessment of risks and adding potential risks as the project delves into new territories.
1.4	Meetings with supervisor/customer	meetings with the supervisor every other week, where the progress and report was discussed. Sporadic meetings with the customer in order to update them and make sure the project was on the right track.
2	**Planning**	The planning prior to starting the development.
2.1	Requirements	Functional and non-functional requirements were established together with the customer.
2.2	Development Method	Research and discussion about which development method and tools to use.
2.3	Team Organization	Determining which role each member should have and their individual responsibilities.
2.4	Milestones	Planning of milestones for the project.
3	**Technical Design**	All development of the product, design and implementation.
3.1	Database	Design and deployment of the database.
3.2	Back-end	Design and deployment of the back-end functions.
3.3	Front-end	Design and deployment of the front-end function and GUI.
4	**Testing**	The testing phase of the project.
4.1	Unit Testing	Development and execution of unit tests.
4.2	Integration Testing	Development and execution of integration tests.
4.3	System Testing	Execution of system tests.
4.4	Acceptance Testing	Execution of acceptance tests.
5	**Finalize**	The final stretch of the project.
5.1	Finalize Product	Final touches on the product and finalization of the documentation.
5.2	Deliver to Customer	Final implementation to the customer's server.
5.3	Deliver Final Report	Completion of the report and delivery.

3.2.3 Gantt chart

The Gantt chart below illustrates the project plan and all the major tasks that were had defined, and is based on the Work Breakdown Structure. As mentioned above there were five parts, and as seen in figure 5 those are split into several additional tasks. It was decided to use a Gantt chart to be able to keep track of the progress and the general expectations for the given sprint. The diagram below shows when the different tasks were supposed to start, and when they were expected to be finished. The time estimated gives an evaluation about how much effort before each task is finished.

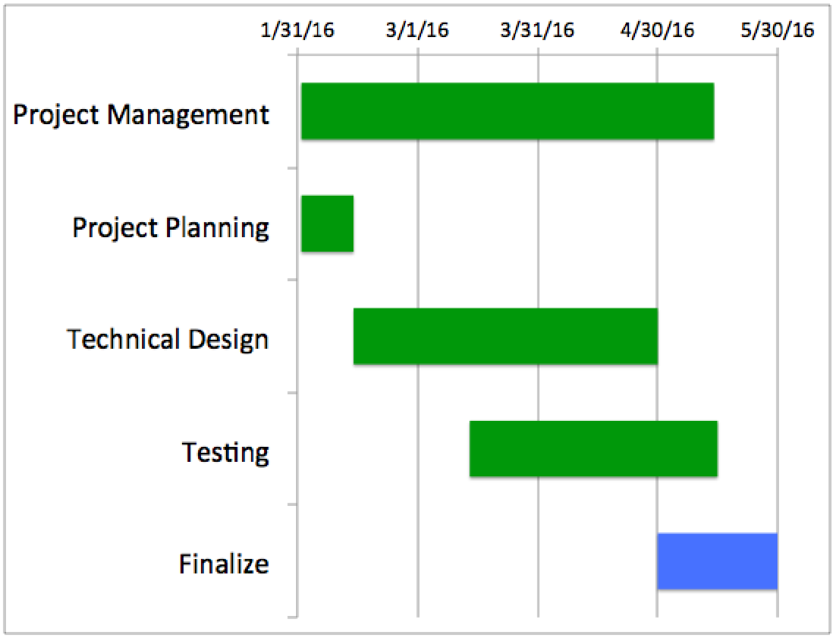


Figure 5: Gantt chart

3.3 Risk Analysis

The risk analysis in table 6 consists of a collection of potential risks for this project. Each of the risks defined below has an associated *likelihood* and *impact* value. The higher the number, the higher the *likelihood* or *impact*. *Importance* is the multiplied value of *likelihood* and *impact*. There were described the preventive actions that had to be done, the potential remedial actions the group should do if one of the scenarios occurred. Identifying these risks is important so that the group can try to avoid or minimize its effects on the project.

The risk analysis describes some general risks that are common in such projects. Typically risks are for instance sickness or the group missing a deadline. It also includes some more specific scenarios regarding the project, like server downtime, and back-end error.

Table 6: Risk Analysis

Description	Likelihood (1-9)	Impact (1-9)	Importance	Preventive action	Remedial action
Poor prioritization of tasks	8	7	56	The group needs to spend enough time planning to establish which tasks are the most important	Re-prioritizing is required, the team needs to take a new look at the backlog
Group gets stuck with a problem	5	8	40	The team needs to be realistic and choose solutions within the reach.	Ask the supervisor for help. Re-evaluate how important each function is, are there other ways of implementing them?
Unclear organization/ responsibilities	6	6	36	Communication is very important. Decide early on which tools to use, and find which is optimal for the group. Make sure everyone is involved in reporting.	Establish new rules for reporting and follow up on them.
Uneven workload	6	6	36	The team should make sure the workload in even in each sprint-meeting, good communication is key.Everyone should track their work-hours in the sheet.	Equalize for the next sprint, maybe lower the workload for the person(s) who was(/were) overloaded.

Table 6: Risk Analysis

Description	Likelihood (1-9)	Impact (1-9)	Importance	Preventive action	Remedial action
Missing a deadline	7	5	35	The team should plan the sprints properly, with regards to the members capacity and knowledge.	Increase the workload for the next sprint so it can be catched up, re-evaluate time-estimates and optionally cut out some functionality.
Someone gets sick	6	4	24	Dress well, eat healthy, sleep enough, don't stress too much.	It is preferable to stay at home one extra day, rather than infect the rest of the group. Get better as soon as possible. Redistribute workload if necessary.
Accidental FTP upload of broken version	3	9	27	Regular backups, automatic deployment software/only one person uploads	Revert to previous backup
Server breaks down	3	8	24	Continue maintenance of the server and ensure the access for the members	Check for the cause and revert to last working configuration
Sudden change in customer demands	3	7	21	Frequent communication with customer, keep up with scrum aspects that allows change of requirements	Reschedule workload, reprioritize tasks.
Data gets lost	2	9	18	Data should be backed up on several places, locally/github/-dropbox.	Data should be reconstructed as quickly as possible, based on memory and diagrams so the team can continue ASAP.
Database crashes	2	9	18	Check for errors and have a secure connection to	Have a backup of the database, and a local server so the development can continue
Back-end error	2	9	18	Extensive tests, run before each deploy	Backup and revert to last working build

Table 6: Risk Analysis

Description	Likelihood (1-9)	Impact (1-9)	Importance	Preventive action	Remedial action
A member takes leave	5	3	15	Everyone is obligated to report to the group if they're going away, preferably before the relevant sprint.	Finish work beforehand, optionally do more on a later occasion.
Cloud server goes down	1	9	9	Fall back solutions, different backups can be accessed if necessary	Fall back to alternative server, eventually continue on local sever.
Get hacked by unauthorized persons	1	9	9	Encrypt the server access and the database, have a continuously secure the data.	Backup and shut down server
Sudden change in customer demands	3	7	21	Frequent communication with customer, keep up with scrum aspects that allows change of requirements	Reschedule workload, reprioritize tasks.

4 Requirements

This section contains the requirements that were agreed upon with the customer. Some of the requirements were defined from the start, like a map to navigate from, access control and transaction view. Later on some requirements were discussed and concluded when the project was already in progress, like security, and the ease use of the application. Some of the requirements were also specified more than the original product description. The requirements were split into two groups, functional, and non-functional requirements, as shown in section 4.1 and 4.2. The functional requirements are shown together with use cases in section 4.3 to get an easier view of the application.

4.1 Functional requirements

Functional requirements are tasks the system is supposed to be able to perform. As mentioned, the requirements are shown together with use cases in section 4.3, where it takes the perspective of a user. The functional requirements for this project encompasses mostly how the system shall behave. To get a overview of the different functional requirements, they are shown in table 7

FR01 Non-logged

If a user/customer of Kommunal Rapport is not logged in, the system shall redirect them to the existing log-in page. The log-in system is for the already customers of Kommunal Rapport.

FR02 Access Control

The system shall use the existing access control solution that Kommunal Rapport uses towards their user, to differentiate between paying and no-paying customers. An user should have access to the application if they have not paid for the service.

FR03 Payment

The user shall be able to pay for the service for this application, this is applied if they are not already a subscriber of Kommunal Rapport and only want access to this application.

FR04 Search

The system shall be able to find information about every transaction related to a person, organization, property or municipality.

FR05 View Transaction

The user shall be able to view transactions by diagrams or graphs to show information about transaction and history of a property, or on a map of Norway with the counties and municipalities.

FR06 Flagging

The user shall be able to easily identify irregular transactions if the change of price is remarkable or suspicious.

FR07 Map Navigation

The user shall be able to navigate a map of all the counties and municipalities in Norway in an intuitive way.

FR08 County Location

The user shall be able to locate a county and get information about sales/purchases of different properties in transactions for a specified county.

Table 7: Functional requirements

ID	Used in use case:
FR01 Non-logged	1
FR02 Access Control	1
FR03 Payment	1
FR04 Search	2
FR05 View Transaction	2
FR06 Flagging	4
FR07 Map Navigation	5
FR08 County Location	5

4.2 Non-functional requirements

Non-functional-requirements are requirements that specifies criteria used to judge the system, and its functions, rather than specific behaviours. Requirements like security towards unwanted guests, the users accessibility to the application at all times, stability towards downtime of server or database are important aspects of a web application. There were added some requirements that were not listed by the customer as mentioned above. In this project the non functional requirements were mostly based on access control, reliability and availability for the web application.

NFR01 Ease of Use
The system shall be easy to use, it is important to the customer that the content of the system is intuitively displayed and easy to understand with only basic background knowledge. The system shall have a short response time, to provide a fluid and fast user experience.

NFR02 Access Control
The system shall be compatible with most modern web-browsers, the user shall be free to use the system with the choice of browser he or she desires.

NFR03 Security
The system shall be secure and encrypted in a manner that secures customer log-in information, and restricts unauthorized access to the database. This includes securing the connection with HTTPS, and hindering SQL injections. The system shall have documentation on how to setup the system, edit data, and add new data and troubleshoot.

NFR05 Modularity
The system shall not be too dependent of other factors, in case one (or more) of the modules break, the system will still be able to function.

NFR06 Restricted Access
The system shall limit access to the website to only logged in users, the usage of the site shall be restricted to the customer and paying subscribers of the customer.

NFR07 Data Retrieval
The database shall retrieve all information relevant to a query, erroneous or missing data from retrievals can cause false suspicion and shall be avoided at any cost.

NFR08 Sufficient Backup
The database shall have a backup. In case of a fatal crash, there must be a backup of the database and its contents such that recovery can be swift.

4.3 Use Cases

An use case shows an operation that the user can do using the system. The use cases were created for the functional requirements, and in some of the cases the one or more requirements were merged to get an overall impression of the functions. Each use case is described first by using a brief textual description, then by an associated diagram, and lastly in a more detailed and structured manner. This is in order to explain the operation as thoroughly as possible. There are five main use cases in this project, and they are shown in the following sections.

4.3.1 Use Case 1: Access

This use case is a combination of FR01, FR02, FR03 as shown in table 7. It describes how the user should be able to log in to the system if they are not logged in, and possible failed end conditions. As described in section 4.1, the log-in solution is based on already existing log-in system that Kommunal Rapport uses toward their users. The function of this use case was developed during the 6th and 7th sprint.

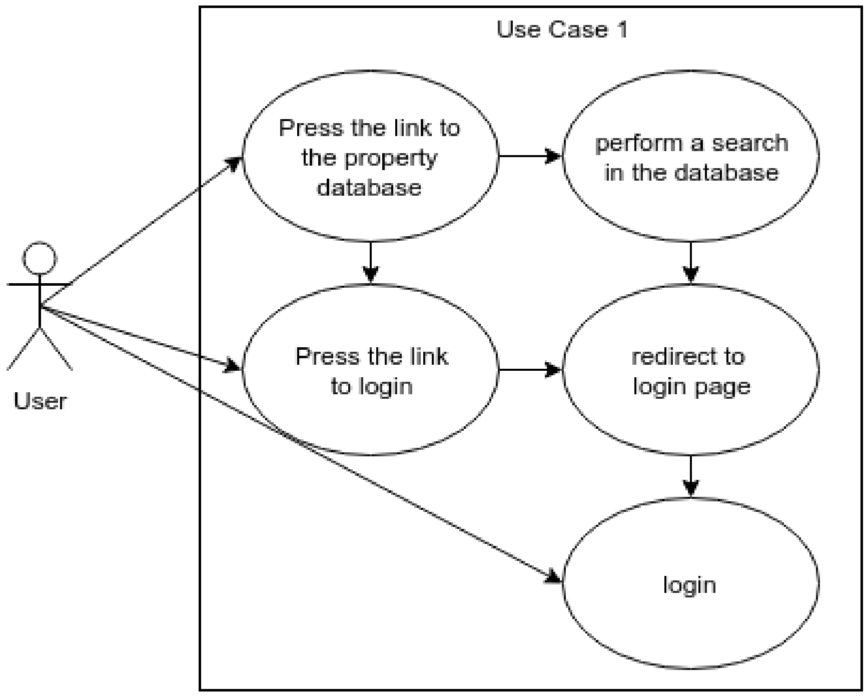

Use Case 1
Press the link to the property database
perform a search in the database
Press the link to login
redirect to login page
login
User

Figure 6: Use Case 1 - Access

Use case 1	**Access**
Goal:	A user wishes to log-in and need to be directed to the existing login page by the system.
Primary Actor:	User and system
Stakeholders and Interests:	User
Precondition:	The user have an existing user, or registers a user

Success End Condition:	User get successfully logged in.
Failed End Condition:	User does not get successfully logged-in
Trigger:	User presses the link to go to the property transaction database

Main Success Scenario:
Step 1: User presses the link to the property transaction database
Step 2: The user searches the database
Step 3: The user needs to sign-in
Step 4: The user can now log-in
Step 5: The user can successfully use the system

Extensions:
Step 2a: The system fails to redirect the user, and a error message is sent.
Step 4a: The system fails to log-in the user

Alternate Flow:
Step 2a: User presses the login-link

4.3.2 Use Case 2: Transactions

This use case is a combination of FR04, FR05, FR06, FR07, shown in table 7. It describes how the user should be able to search for a specific transaction for a given person, company, county or municipality in the system and possible suspicious transactions. The use case also shows possible failed end conditions if necessary. The function of this use case was developed between the 2nd and 5th sprint.

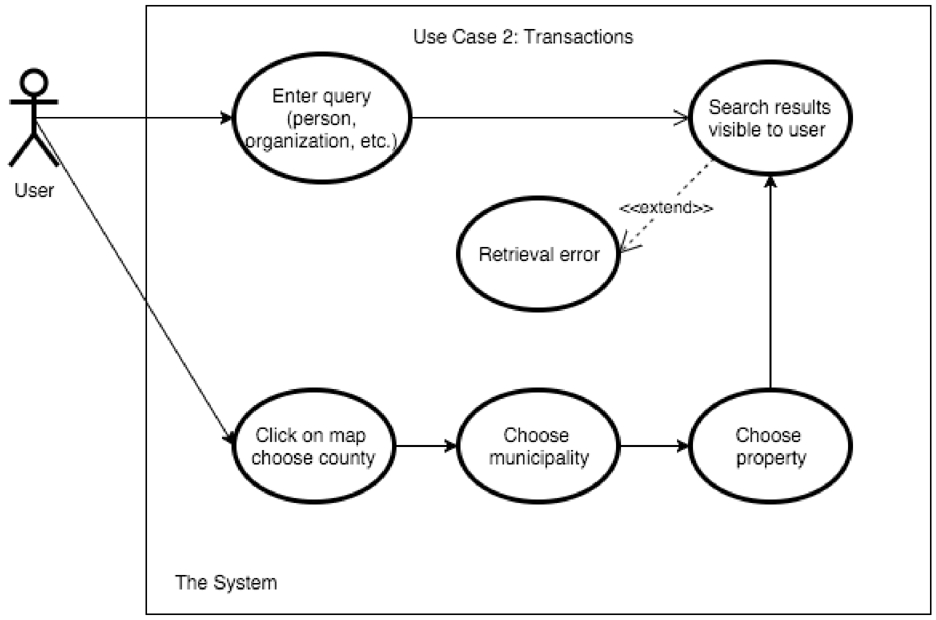


Figure 7: Use Case 2 - Transactions

Use case 2	**Transaction**
Goal:	A user wishes to see all transactions.
Primary Actor:	User
Stakeholders and Interests:	User
Precondition:	The user need to be logged in, and have a valid subscription.
Success End Condition:	The user can successfully see all transactions.
Failed End Condition:	User can not see all transactions.
Trigger:	The user enter a search query in the search field or click on the map
Main Success Scenario:	

Step 1: The user enter a search query in the search field or click on the map
Step 2: The search result is visible to the user
Step 3: The user is now able to see all transactions related to the query or click on the map

Extensions:
Step 2a: The search result is not visible to the user, or only a small part of the search result is visible.
Step 3a: The user can not see all transaction related to the query.

4.3.3 Use Case 3: Diagram transactions

This use case is based on FR08 as described in table 7. It describes how the user should be able to see the search result in form of a diagram or charts, to easier get an overview of the transaction history for a property. It also shows possible failed end conditions. The function of this use case was implemented during the 2nd and 3rd sprint.

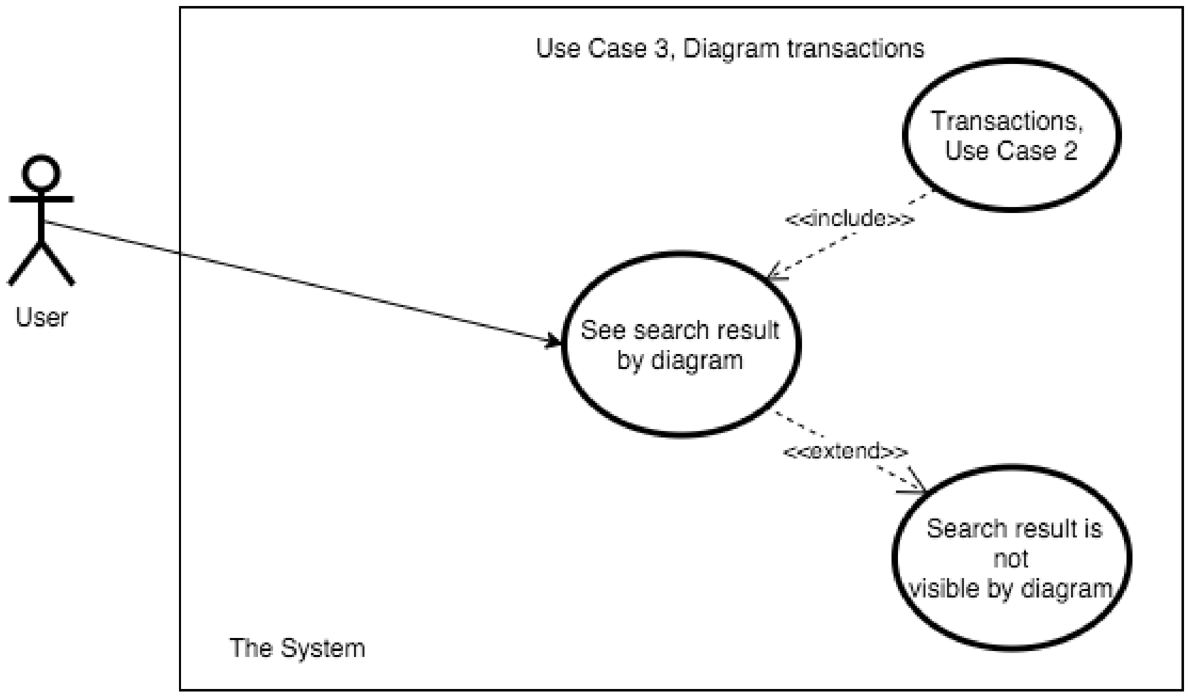

Use Case 3, Diagram transactions
Transactions, Use Case 2
<<include>>
User
See search result by diagram
<<extend>>
Search result is not visible by diagram
The System

Figure 8: Use Case 3 - Diagram transactions

Use case 3	**Diagram transactions**
Goal:	A user wishes to see all transactions represented in diagrams and correlating locations on a map.
Primary Actor:	User
Stakeholders and Interests:	User

Precondition:	The user need to be logged in, and have a valid subscription.
Success End Condition:	The user can successfully see all transactions graphically represented in a diagram, or the locations of the transactions on a map.
Failed End Condition:	User can not see all transactions.
Trigger:	The user enters a search query in the search field or click on the map

Main Success Scenario:

Step 1:	The user enter a search query in the search field or click on the map
Step 2:	The search result is visible to the user in form of a diagram or on a map
Step 3:	The user is now able to see all transactions related to the query or click on the map

Extensions:

Step 2a:	The search result is not graphically represented to the user in a diagram
Step 2b:	The search result does not return any/all locations on the map

4.3.4 Use Case 4: Irregular transactions

This use case is based on FR06 as seen in table 7. It describes how the user should be able to see all irregular transaction regarding a property based on a given search query, and possible failed end conditions of this action. The funciton of this use case was implemented in the 5th sprint.

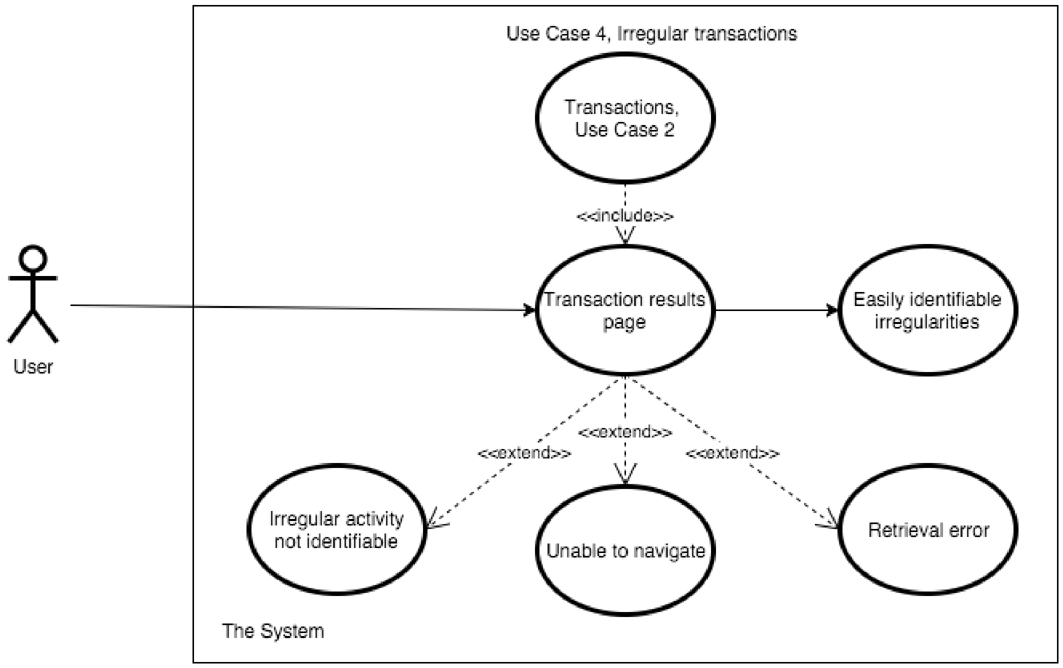


Figure 9: Use Case 4 - Irregular transactions

Use case 4	**Irregular transactions**
Goal:	A user wishes to easily identify irregular transactions
Primary Actor:	User
Stakeholders and Interests:	User
Precondition:	The user need to be logged in, and have a valid subscription.
Success End Condition:	The user can easily see all the irregular transactions.
Failed End Condition:	The user can not see all the irregular transactions; The irregular transactions are not easily identifiable.
Trigger:	The user enters a search query in the search field or has specified its search by browsing the map
Main Success Scenario:	

Step 1:	The user enters a search query in the search field or clicks on the map
Step 2:	The search result is visible to the user, and the irregular transactions are clearly identified
Step 3:	The user is now able to see all transactions related to the query or click on the map

Extensions:

Step 2a:	The search result is not visible to the user; Only a small part of the search result is visible;
Step 2b:	Only a small part of the search result is visible
Step 2c:	The result is visible, but the irregular transactions are not identifiable.
Step 3a:	The user can not see any/all the transactions related to the query

4.3.5 Use Case 5: Navigate

This use case is a combination of FR07 and FR08 referring to Table 7. It describes how the user should be able to navigate by using the map of Norway with counties and municipalities, and possible failed end conditions for this interaction. The function of this use case was developed in the period between the 3rd and the 6th sprint.

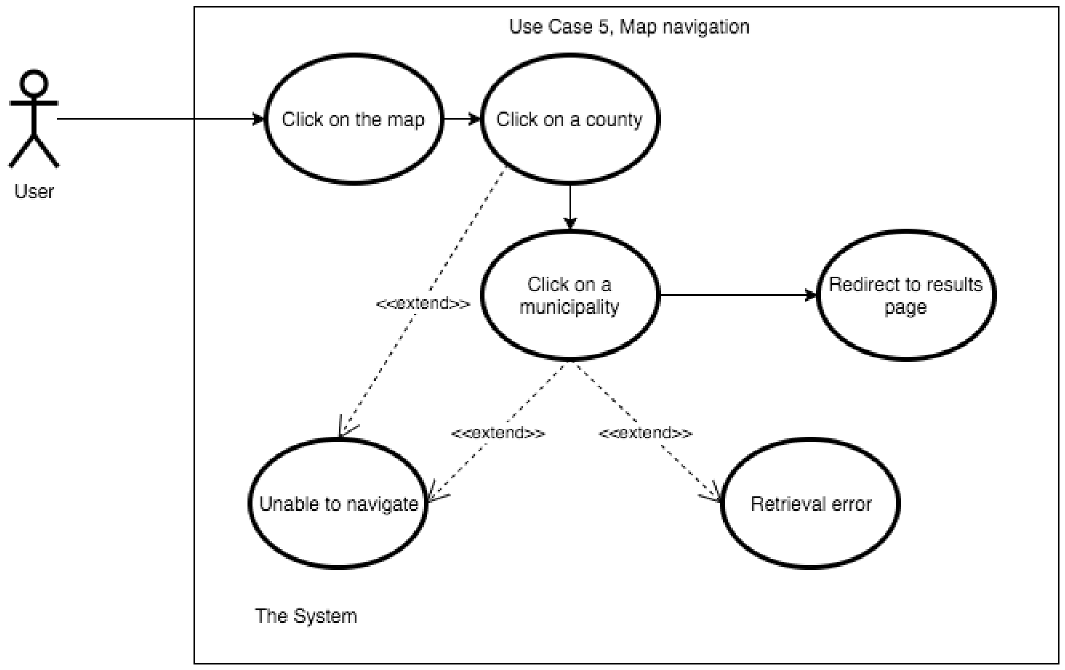


Figure 10: Use Case 5 - Navigate

Use case 4	**Map navigation**
Goal:	A user wishes to browse by using the map and navigating to a county and subsequently municipality and receive all transactions within that municipality
Primary Actor:	User
Stakeholders and Interests:	User
Precondition:	The user need to be logged in, and have a valid subscription.
Success End Condition:	The user is able to navigate to a county and subsequently a municipality by using the map and receives all the transactions connected to that municipality.
Failed End Condition:	The user can not use the map to navigate to a municipality; The user does not receive (all) the transactions related to that municipality.
Trigger:	The user selects a county by clicking on the map

Main Success Scenario:

Step 1:	The user clicks on the map to select a county
Step 2:	The user clicks on the map of the county to select a municipality
Step 3:	The user is now able to see all transactions related to the municipality or click on the map

Extensions:

Step 2a:	The user is not able to navigate to a county; The user is not able to navigate to a municipality; The search result is not visible to the user;
Step 3a:	The user can not see all the transactions related to the query

5 Development environment

This section will describe the different tools that were used during the project. It describes the languages and software choices, collaboration tools, testing methods, how version control was done and the documentation tools that were utilized. Lastly, deployment will be discussed to describe how the project was maintained and updated throughout the development.

5.1 Development Frameworks

The subsequent frameworks were utilized in the finished product described in this report.

5.1.1 MySQL

MySQL is a popular relational database management system, chosen for this project due to its proven capabilities of handling medium large to large datasets. It is widely documented and has an extensive userbase, allowing for easy integration with most tools and frameworks. Being well documented and popular simplifies the delivery and maintenance part of the project, as the customers engineers would likely be familiar with it and could quickly continue development.

In addition, the team had extensive previous knowledge of this system and could commence development on the database as early as sprint 1 (Appendix B.2). Other solutions such as Elasticsearch, a modern equivalent prized for its speed and comprehensive list of features, was considered early on. But the smaller userbase and the limited documentation secured MySQL as the datastorage framework of choice for this project.

5.1.2 OAuth 2.0

OAuth is a common open standard for access control and authorization, currently in use on many major web-pages and deemed to be very secure. This project had to conform to the customer's system as it would be integrated with their front page and previous web applications. OAuth's primary functionality is allowing a third party access to protected resources on behalf of the resource owner. In practice this allows the customer's users to log into the new application with their existing credentials.

5.1.3 PHP

PHP is a general purpose scripting language designed for web development, mostly used for server-side interfaces to databases. PHP is considered the de facto standard for connecting databases to web applications. It provides useful features such as prepared statements that were essential in securing the database connection from intrusion [24].

5.1.4 AngularJS

AngularJS is an open-source application framework extending Javascript to address issues commonly encountered when developing single page applications, such as the one developed for this project. Having the site be a single page application was critical to the user experience as to provide the speed and responsiveness modern users have grown accustomed to. It is maintained by Google making it dependable and well documented as opposed to its main counterpart React, as mentioned in section 2.4.1. React is built to solve many of the same complications as AngularJS, and was first selected as the front-end framework for the project. However early development stagnated due to limited documentation on database integration and general development, and was thus scrapped in sprint 1 (Appendix B.2) favor of AngularJS.

5.1.5 Latex

Latex is a document preparation system commonly used for scientific documents in most fields, including computer science. It allows for precise control of the document layout and supports complex symbols most other text processors struggle with. It was used in this project to create the report.

5.2 Development Tools

These tools were selected for writing the application and providing a graphical user interface to the database.

5.2.1 Sequel Pro and MySQL Workbench

Sequel Pro and MySQL Workbench are visual database design tools, that integrate SQL development, administration and database design into a single development environment. The difference being that Sequel Pro is compatible with OSX while MySQL Workbench is compatible with Windows and Linux. Which tools that waere used to design the database were not important as the end result would be the same anyhow, thus it was the matter of each team member to select the database design tool they were comfortable with.

5.2.2 Sublime

Sublime is a cross-platform code editor, supporting a wide range of programming languages. Most team-members chose to use Sublime as it could be extended to support all frameworks used in the project, and provide syntax-highlighting and shortcuts to aid development. Various other text-editors were used sporadically but Sublime was maintained as the core editor.

5.3 Collaboration Tools

The following tools were used to facilitate communication and cooperation within the group, in order to create the report and develop the application simultaneously .

5.3.1 GitHub

GitHub is a free and open source distributed version control system, designed to handle distributed development on projects of any size. GitHub was selected as the team had access to student accounts and could take advantage of premium features normally limited to paying customers. This quickly excluded all alternatives, as anything that came close required a fee.

GitHub was integrated with Slack to provide the team with constant updates on what was being developed between meetings. A key feature of version control systems such as GitHub is the ability to differentiate experimental and stable versions of the system while in development. The team took advantage of this by providing access to a an always stable and well tested version of the system to the customer, in order for him to give continous feedback and follow the development.

5.3.2 Trello

Trello is a virtual taskboard, that lets users create collaborative rooms, and enables them to pin "posts" on the taskboard. This enabled the group to easily distinguish between backlog tasks, tasks in progress, finished tasks, and urgent tasks.

5.3.3 Google Drive and Docs

Google Drive was used to keep track of the tasks, store diagrams and document the programming. Trello worked as a help for documentation, where it was used to see what tasks were progress, and see the backlog.

5.3.4 Google Hangouts

Google Hangouts is a web based videoconference solution, it allows for several people to take part in a shared conference. Google Hangouts was used in place of physical customer meetings as the customer being based out of Oslo, limited the amount of in-person meetings that could be done.

5.3.5 Slack

Slack is a web based collaboration tool used by many prominent teams for project communication, its ability to divide the chat into topic specific rooms encouraged all team members to follow development discussion within topics that they currently were not involved with. This kept non relevant discussion to a minimum, and provided a log of important decisions.

5.3.6 Facebook chat

Facebook Chat was chosen as the organizational platform as all members were active users, and would actively check for updates. Most discussion and planning of extraordinary meetings took place here, as well as urgent group announcements.

5.3.7 Share Latex

Share Latex is a tool for collaborating on Latex documents simultaneously, across devices. Again the team had access to student accounts which unlocked premium features not available in similar free solutions. Thus Share Latex remained the only viable solution.

5.4 Testing Tools

The testing tools that were used in this project were used for evaluating the front-end and the back-end in the product.

5.4.1 PHPUnit

PHPUnit was used for the back-end testing, which is a programmer-oriented testing framework for PHP. It is for unit testing frameworks, and is an instance of the xUnit architecture [25].

5.4.2 Jasmine

Jasmine was used to test the front-end code. It is a framework for behavior-driven testing of JavaScript code. Jasmine does not depend on any other JavaScript frameworks. It does not require a DOM and it has a clean and obvious syntax allowing easily development of tests [18].

5.5 Deployment

Kommunal Rapport did not have a server available, therefore the group was free to set the limitations to what kind of server they would get. Deploying a server on one of the member's computer was considered. However, this idea was discarded as it would have been difficult to meet the required server uptime.

DigitalOcean was selected as the server hosting service. DigitalOcean offers simple cloud solutions that are meant for developers. They give you full access to a virtual server with just the bare essentials installed. This marked the beginning of the development cycle. With an offshore server, the group members were able to manipulate and update the dataset without having to rely on another group member's computer's uptime. Having an offsite server with constant uptime enabled the group to utilize GitHub's Webhooks services [15]. Webhooks allows you to build integrations that subscribe to specified events on GitHub. The integration that was utilized in this project was one that automatically updates the website when new code is submitted to GitHub. This made it possible for the customer to gain insight into the incremental development and progress of the product. Two versions of the website were kept throughout the development process, a "test", and a "master"-website. These were based on the two main branches of the GitHub. A branch is essentially a way of keeping several revisions of the code for simultaneous development. The intention of this separation was to have a more polished website for the customer to review, and to be able to do tests with minimal risk.

Figure 11 shows the sequence diagram of how the deployment to the server works. The whole sequence begins with the user "pushing to a repository". This means the user is uploading local changes to the shared folder on GitHub. Then GitHub sends a notification to the integrated Webhooks service, which then fetches both revisions of code, master- and test-branch, and updates the master- and test-server with the newly uploaded code.

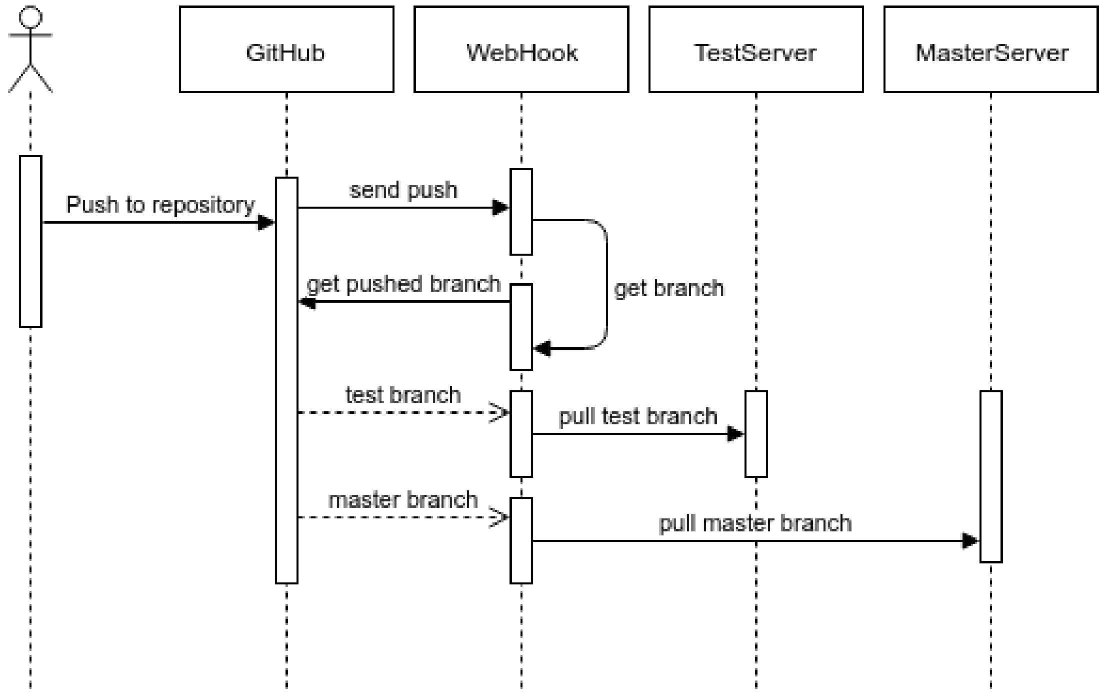


Figure 11: Sequence Diagram of Deployment

As a security measure to prevent unwanted listeners on the server during development, Apache2's built in authentication function was used. This prompts the user with a log in when visiting the website. Some challenges were met with regards to PHP, but this explained in detail in "back-end challenges", section 8.1.3.
As seen in pull.php (Listing 1) and sequence diagram of deployment (Figure 11), two versions of the product were deployed. The master branch was located at bachelor.kasperrt.no, viewable for the customer and anyone with a password, while the test-branch was located at bachbeta.kasperrt.no, and was only accessible to the group members. This solution was setup in order to aid development, and make it easier to follow the progress of the project without having to switch branches.

6 Design and Implementation

This section describes the architecture and flow of the the product. It explains more in depth how the database was configured and implemented. The front-end and back-end will also be explained more in depth, with the back-end description focused on how the queries were accomplished through PHP, and how the login OAuth2.0 system works, while the front-end section will revolve around its design and implementation.

6.1 Architecture

The architecture of a system is an important part of helping identify the different components that are going to be a part of the system. One of the purposes of the architecture model is to show the different components and how they interact. These architecture models makes it easier for the developers to gain insight into how the system is arranged.

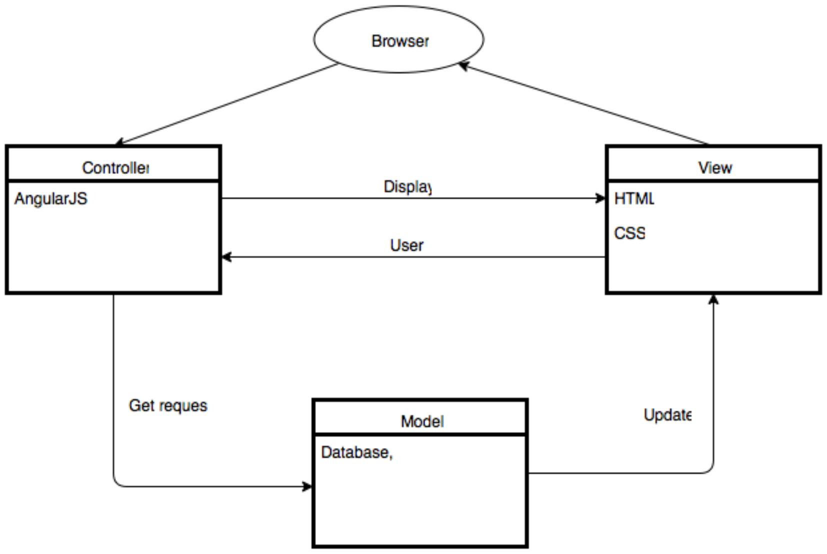


Figure 12: Model View Controller

The system is structured after the model-view-controller pattern (Figure 12), a layered architecture where interaction and presentation is separated from the system data. It divides the system into three components that interact with each other: The model component manages the system data and related operations. The view component is the pages of the application that represents the data to the user. The controller component manages the user interactions and passes the them to the two other components. Such user interaction include pressing a button or typing in an input field to perform a search. [27]

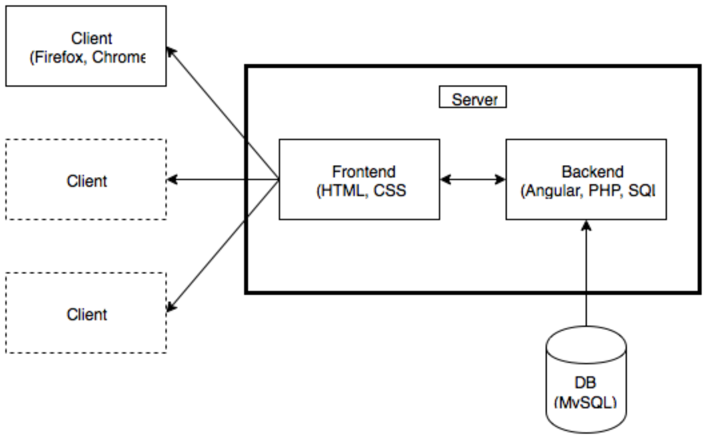


Figure 13: Architecture

The physical deployment of the system consists of a database the user can communicate with using search queries. Figure 13 shows how the server encompasses both front-end and back-end functionality. The client connects directly to the server, and indirectly to the database.

6.1.1 Activity Diagram

An activity diagram is used to describe how activities are dependent on each other. The diagram explains whether activities can be conducted at the same time or if one activity needs to be finished before another [27].

Figure 14 shows how the user must first decide between making a search in the search field or click on the map to perform the search. If the user decides to click on the map, the user needs to choose county, municipality and finally select a property. All of these activities need to be finished before the next one can start, and can not be performed at the same time. If the user decides to use the search field, then the user can do a search by person, address or by municipality. These searches can be done at the same time. Both decisions end in the same results; showing the user the selected transactions for the given property.

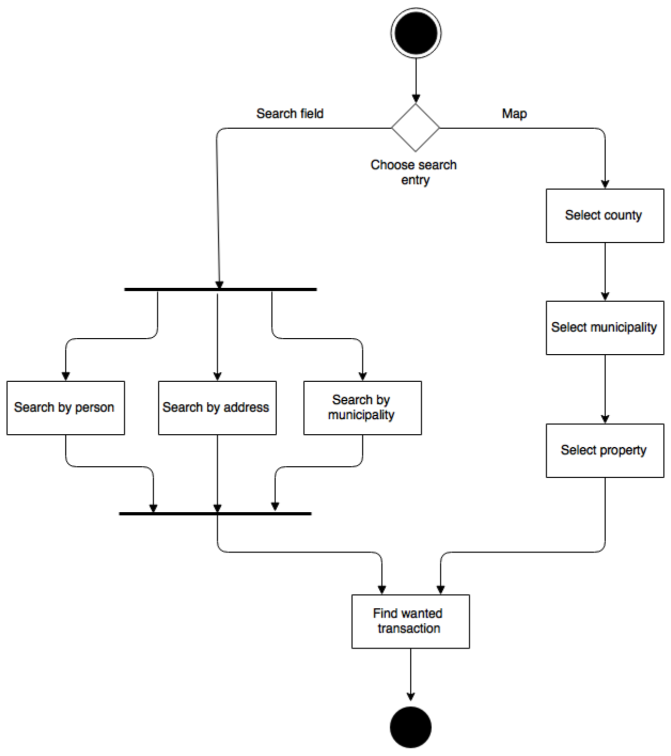


Figure 14: Activity Diagram

6.1.2 Sequence Diagram

A sequence diagram is an interaction-diagram which shows how the different components in the system cooperate, and in what order the processes occur. The diagram is straightforward depicting the different objects in the system, and what their functions are [27].

In figure 15, one can see the flow of the program, and how the different components talk to each other. The whole sequence of components begin with the user logging in to the system. If this fails, the user will return to the front page, and has to log-in if the user intends to perform any searches. The log-in is handled by Kommunal Rapport's log in server.

When a user is logged in, the user will be presented with the search-view, where the user will be able to search for people, addresses, companies and municipalities. The user can filter the search to highlight counties or municipalities in Norway. From the view, the search-controller extracts the data the user is searching for, and proceeds to query the back-end for the appropriate data.

The back-end queries the database accordingly, and fetches the requested data from the database. This is passed back to the controller which is then used to populate graphs or tables in the view. There is only one query executed between the controller and the back-end, as only one of the parameters is sent with the request to specify what type of query it is. From the back-end to the database, there are three different queries, which are view-specific, meaning they belong to their own respective views.

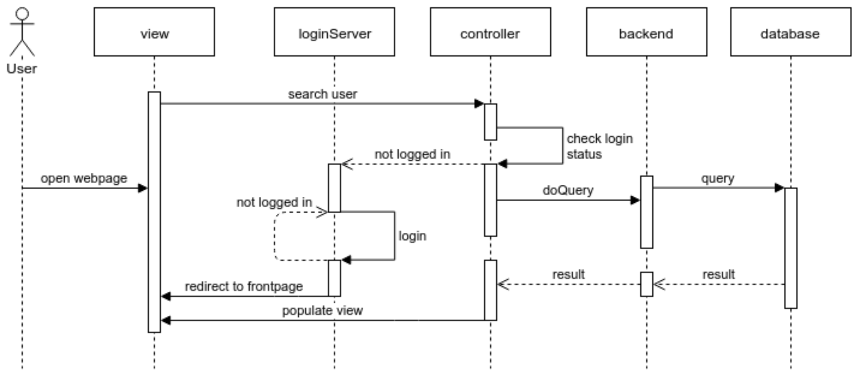
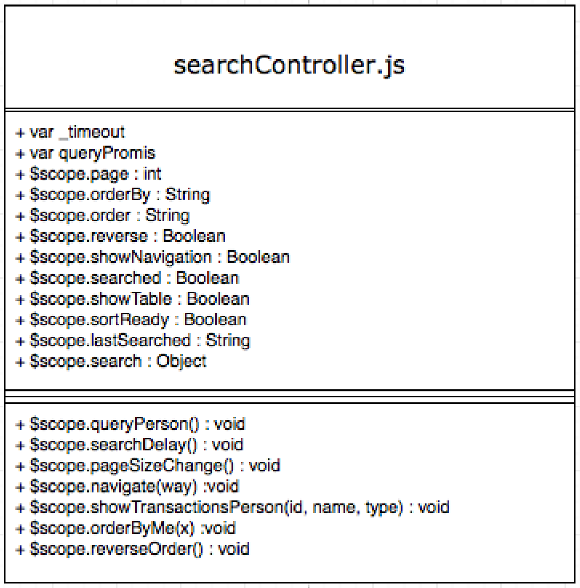


Figure 15: Sequence Diagram

6.1.3 Class Diagram

A class diagram shows the object classes in a system and how they interact with each other [27]. The complete class diagram is in the appendix G.1.2 and G.1.1. Because AngularJS has been utilized to such an extent in this project it makes more sense to represent a class diagram based around that. This means that instead of using classes, the diagram shows the modules and controllers and how they are connected. Controllers do not function as classes, they can for example not be instantiated, but include variables and functions that are called upon from other parts of the code, mostly through HTML. A module's main function is to connect the controllers under its scope to their respective page and to handle navigation.

Figure 16 shows an example controller from the front-end. The filename matches the controller name, and its functions and variables are shown as methods and attributes would be in a normal class.



```
searchController.js

+ var _timeout
+ var queryPromis
+ $scope.page : int
+ $scope.orderBy : String
+ $scope.order : String
+ $scope.reverse : Boolean
+ $scope.showNavigation : Boolean
+ $scope.searched : Boolean
+ $scope.showTable : Boolean
+ $scope.sortReady : Boolean
+ $scope.lastSearched : String
+ $scope.search : Object

+ $scope.queryPerson() : void
+ $scope.searchDelay() : void
+ $scope.pageSizeChange() : void
+ $scope.navigate(way) :void
+ $scope.showTransactionsPerson(id, name, type) : void
+ $scope.orderByMe(x) :void
+ $scope.reverseOrder() : void
```

Figure 16: searchController.js

6.2 Database

The database was a big part of the project, since the group initially received a csv-file which contained 1.8 million rows and 34 columns. The structure of the database and how the data was implemented had a major role in how usable the web application would be. Effective search and retrieval of specific data was especially important, and had to be taken into consideration when creating the database and performing the queries. The database was mainly developed in the first sprint 1 (AppendixB.2), but smaller incremental changes were made throughout the development phase.

6.2.1 Entity-Relationship Diagram

Figure 17 shows the Entity Relation diagram (ER-Diagram) for the database. It shows how the data was structured and the connections created within the table. The main table is called "Omsetninger" (Table 33), it connects directly or indirectly to all other tables. This is where all the transactions are stored. From a transaction one can get more information about the property and the persons, municipalities, counties and companies that have been involved with the given property.

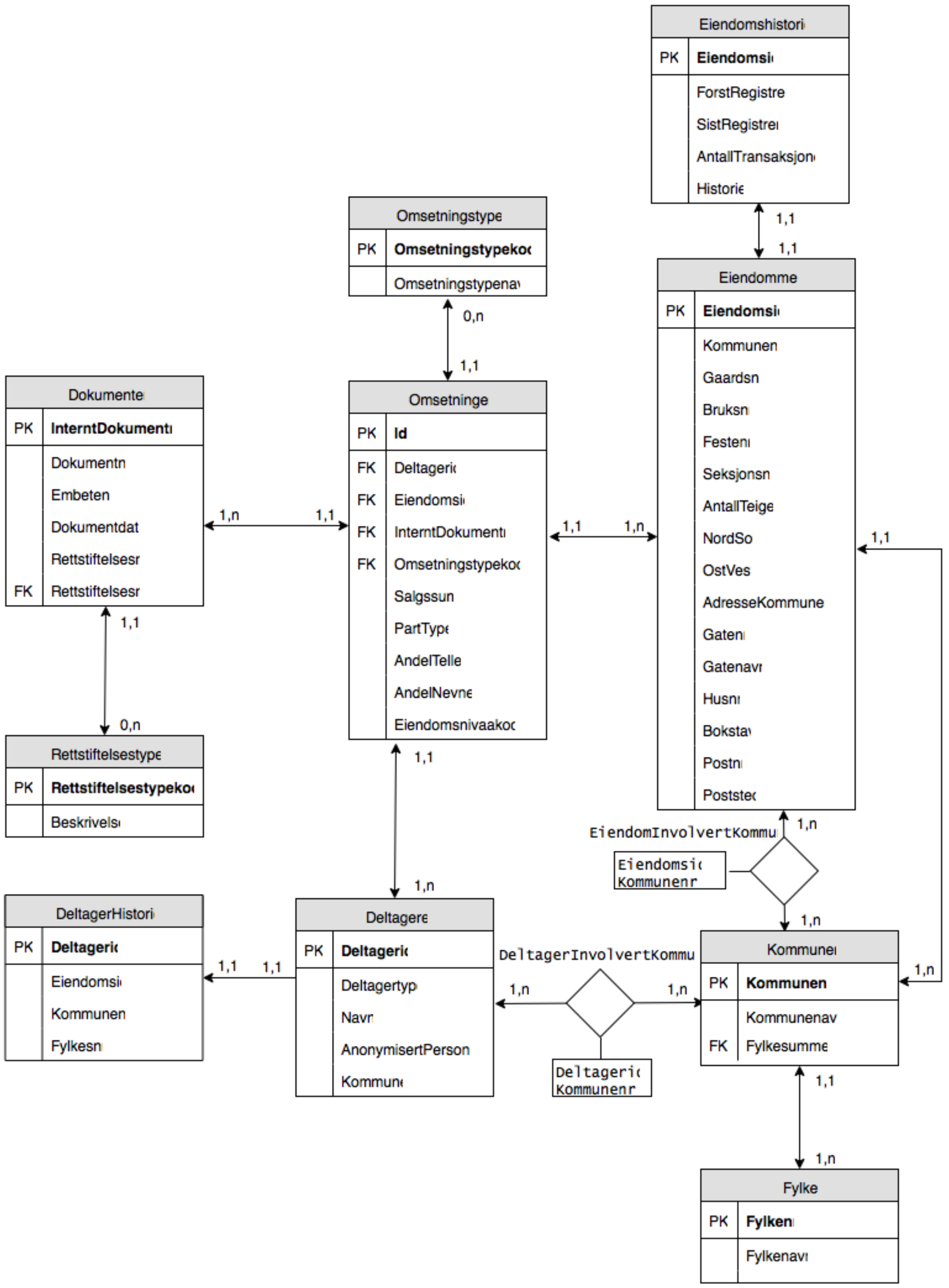


Figure 17: ER Diagram

6.2.2 Design

The design-goal of the database was to eliminate redundancy found in the csv-file and maximize query and retrieval speed. Eliminating redundancy is essential as it makes the database difficult to maintain and increases query time considerably. Additionally it makes the database unnecessarily large. Columns with data related to properties, documents and participants in transactions were often repeated for new transactions and were therefore separated into content specific tables where they were assigned a unique ID.

The “Omsetninger” table (Table 33) is the backbone of the database and contains much of the core information displayed as well as references to related data. These connections are critical to the system and are used when searching to find transactions related to a property, agents who have owned properties, and properties municipalities at some point have owned.

6.2.3 Implementation

The source data was delivered in a comma separated values format, and imported into the database using the “Import from CSV” feature of Sequel Pro. This resulted in a singular table with similar structure to the original file, one table with 34 columns and 1.8 million rows.

Next, the table was divided into separate tables following the first draft of the ER-diagram, which was constructed in the design phase. This was carried out using MySQL queries and the database management tools found in Sequel Pro. The new tables were then assigned indexes on primary keys and all columns needed in searching the tables. Some of the website’s features required complex historical data, such as ownership over time and price points. Query experiments early on in the development phase revealed that generating this information while searching would slow the system down too much. To solve this the data was precompiled into new columns, which could then be joined with the original search without adding too much delay.

6.3 Front-end

The Front-end is, in software engineering, defined as the interface between the user and the data access layer. In this project, this corresponds to the web-page with all the associated files. It was developed using a combination of AngularJS, HTML5 and CSS3. A very basic front-end with HTML and CSS3 was implemented in sprint 2 (Appendix B.3) and sprint 3 (Appendix B.4) to have a foundation to develop upon. The finalization of the front-end design was done in sprint 7 (Appendix B.8). Visualization of the transactions was a big part of the task. The JavaScript framework was in place to make the web application reactive, allowing the data from the transactions in the database to be put into result tables. Changing between views and changing content on a specific page was an important task which was made possible using AngularJS. AngularJS was also used for some general functions like sorting tables, and hiding different parts from the view. HTML and CSS were used for adding HTML-elements and styling of the web application.

6.3.1 Design

During the planning phase the team developed the basic guidelines for the front-end of the project. It needed to be user-friendly, responsive and be able to convey the complex data of the original dataset in a manner that anyone could easily understand. To accomplish this it was decided that a single page application structure would suit the project best, as it would result in good performance, and reduce overall loading time. Instead of having to download the entire page with resources every time something changed, it could download only the elements that changed.

To target user friendliness and to provide quick access to the data, the front-page would feature a search-bar with a corresponding results table. Additionally a map of the counties in Norway would be added to enable the user direct access to municipalities and counties. To avoid overwhelming the user with results, filtering methods were to be added to the search bar. The effect of these features would be that the user could access the data in multiple ways. To further reduce the amount of data the user encounters, the web-page needed to be split into two more pages, each narrowing the search, to ultimately display the transaction history of a single property.

Another measure to reduce the complexity, was to add graphical representations of the transaction history on the property page. Two different solutions were planned, a graph of the sale's price over time with the data presented below it, and a time-line highlighting change of ownership.

As well as the structural plan, some design guidelines were set. The web-page would follow modern design conventions, such as keeping the site minimalistic and avoiding clutter. It was also important to the customer that the site would follow the same theme as their main page. This included a menu allowing the user to quickly switch between the customers different services.

Figure 18 shows the mockup that was created during the initial planning phase of the project.

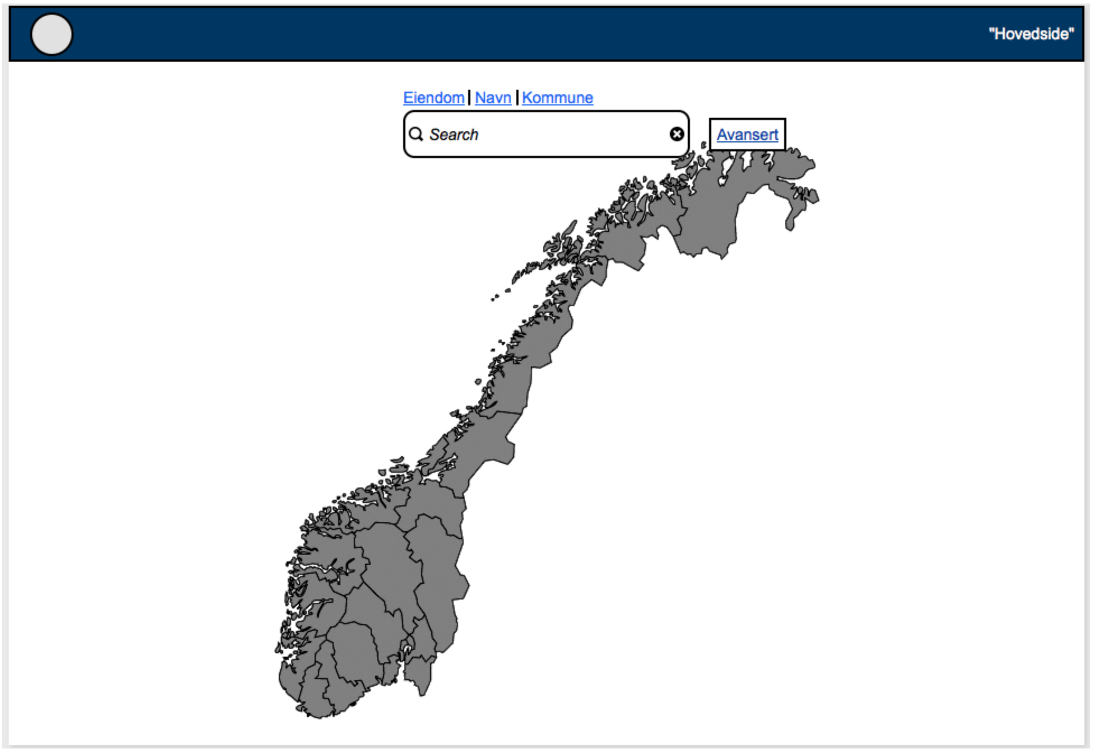


Figure 18: Front-page Mockup

6.3.2 Implementation

Some of the controllers and views needed directives, this was in order to extend the HTML file with more attributes and functions. A directive is a marker on a HTML element that tells the compiler to attach specified behaviour to that element, for example via event listeners [6].

The template view contains the header and footer which is implemented on all views, **index.html** contains these elements. The first view the user encounters is `search.html` which contains the search field and the dropdown menus allowing the user to choose different kinds of search. It also features the map of the counties. When a user searches, `searchTableRow.html` provides the result page with data from the database. `searchController.js` relay user queries to the back-end, which in turn is connected to the database. The controller decides what elements to display on the site, and fetches more results when the user navigates the result pages. In sprint 3 (Appendix B.4), a basic filter for the table headers was developed. This allowed the user to sort results, for example; sorting properties by most recent municipality involvement. To let the user choose more specifically what they want to see, advanced search was developed in sprint 5 (Appendix B.6) with dropdowns to filter results based on, type and what county or municipality had been involved. In sprint 6 (Appendix B.7), a dropdown with the option to search for addresses was added after getting an additional dataset from the customer containing addresses for most of the properties.

When a user selects a person, company, county or municipality, the user will be directed to `transactions.html`. This view shows the user which transactions are related to a specific type. The data for this view is provided by `transactionTableRow.html`. The controller belonging to this file is `transactionPersonController.js`, which decides which query needs to be sent to the database to get the appropriate data. The result table for the transactions was developed in sprint 3 (Appendix B.4), sprint 5 (Appendix B.6) and sprint 6 (Appendix B.7).

Figure 19 shows the front-page of the final product. It displays the different approaches a user can take to find what they are looking for. The rest of the views are presented in appendix G.2.

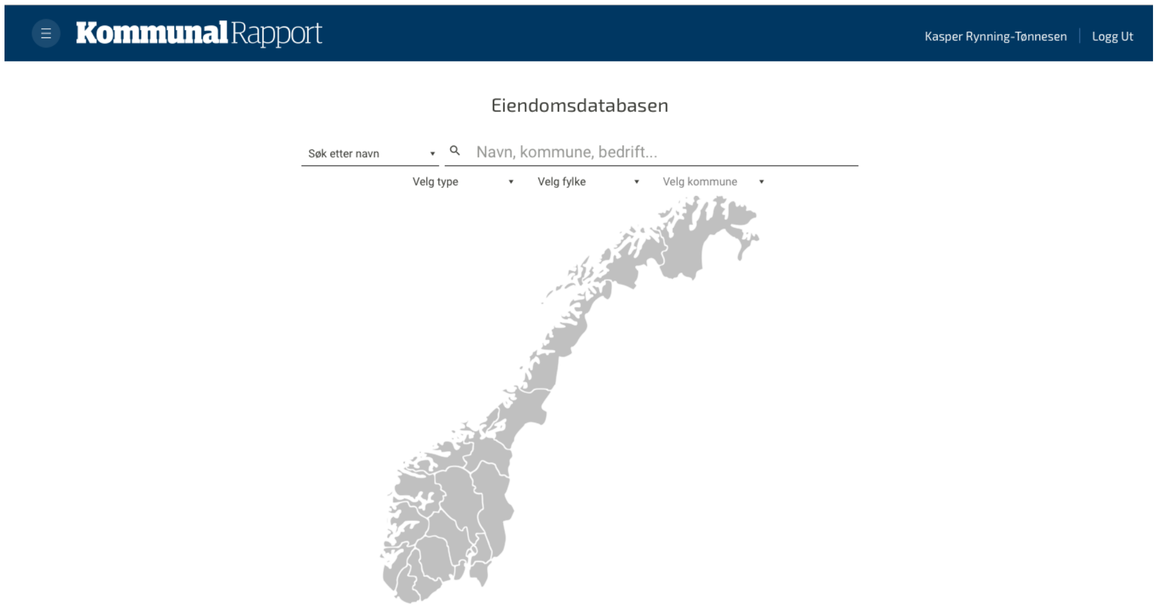


Figure 19: Front-page Screenshot

When a user selects a property, `transactionsDetailed.html` will be displayed. This page contains two tabs, one with a graph and a table containing the results and the other containing the results visualized with a timeline. `transactionsDetailedController.js` contains the logic for changing between the two tabs. The timeline view is contained in `transactionPropertyTimeline.html` and its corresponding controller in `transactionsPropertyTimelineController.js`. The controller receives the data specific to this view. The timeline view was developed in sprint 7 (Appendix B.8) to get an additional visualization of the transaction history. The tab with the graph and the able is contained in `transactionProperty.html` and displays the transaction history in a table. `transactionPropertyTableRow.html` fetches the data specific to each transaction history. The controller bound to this view is `transactionPropertyController.js`. The tables were developed along with the transaction tables in sprint 3 (Appendix B.4), sprint 5 (Appendix B.6) and sprint 6 (Appendix B.7).

The log-in API in the back-end was called from each controller to check whether a user was logged in or not. This was developed and designed in sprint 6 (Appendix B.7) and sprint 7 (Appendix B.8). The controllers check if there exists a cookie with the same name as the user, and if no match is found, the user will be redirected to the log-in page at Kommunal Rapport's main page.

Figure 20 shows the web-application's sitemap. All the pages are built upon index.html, which encompasses about.html, header.html and footer.html. The figure shows how the pages are connected, and what controllers are associated with each page.

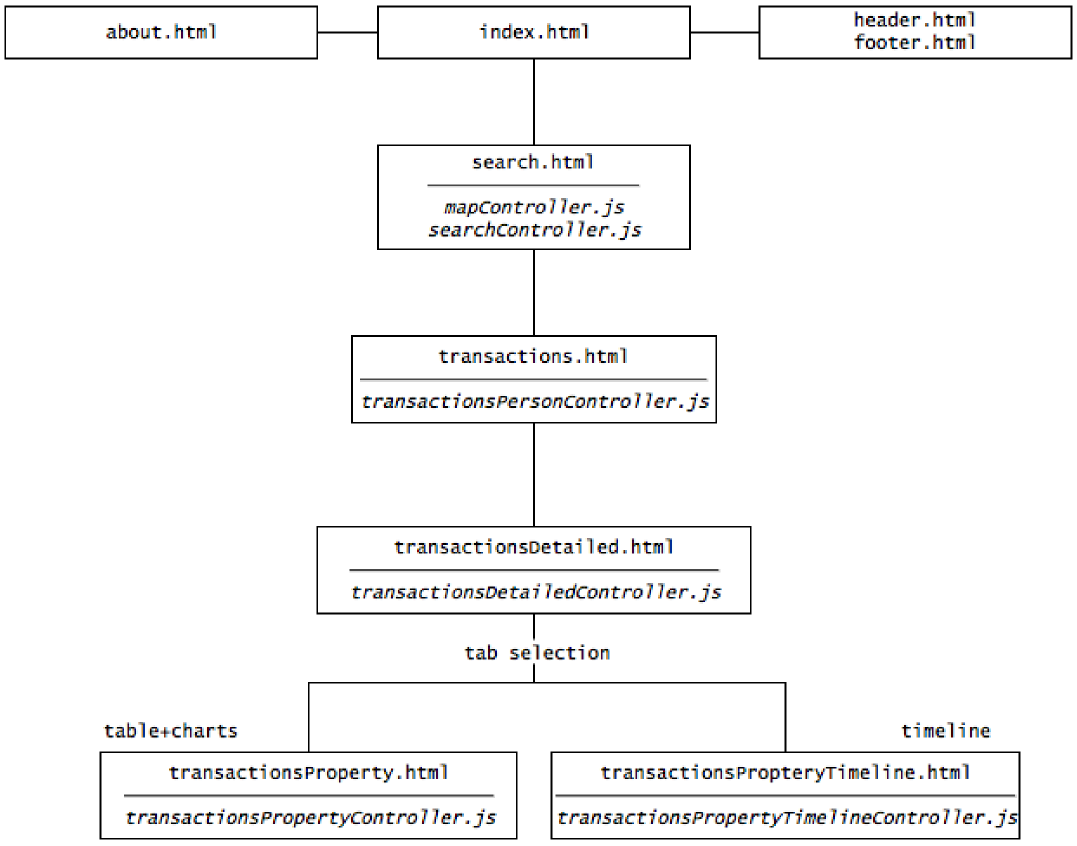

about.html
index.html
header.html
footer.html
search.html
mapController.js
searchController.js
transactions.html
transactionsPersonController.js
transactionsDetailed.html
transactionsDetailedController.js
tab selection
table+charts
timeline
transactionsProperty.html
transactionsPropertyController.js
transactionsPropteryTimeline.html
transactionsPropertyTimelineController.js

Figure 20: Sitemap

6.4 Back-end

A back-end serves as the middle layer between the user interface and the underlaying data. In this case the back-end provides communication between the web-application and the database, managing authentication and providing transaction-data. It was implemented in PHP, running on an Apache2 server. This was the most fitting approach, as the back-end would only be used for authentication against Kommunal Rapport's log-in server and database queries. Since the queries are a "one-time" event that does not require any further processing, a static back-end only running GET request would be sufficient. This proved beneficial when the project was integrated with the existing authentication solution, provided by Ramsalt, Kommunal Rapport's host of log-in and servers. They had a ready-to-go framework that only needed to be integrated. The back-end was developed between sprint 2 (Appendix B.3) and sprint 7 (Appendix B.8).

The back-end is made up of 5 PHP files; `ask.php`, `config.php`, `login.php`, `logout.php`, and `query.php`. `ask.php` receives GET requests from the front-end, and creates a query using `query.php`. It then returns the database-results to the front-end. `config.php` contains the database configuration, and handles the connection, while `login.php` and `logout.php` handles log-in, and log-out from the server. This will be explained more in depth in the following subsections.

6.4.1 Design

The back-end was designed with the intention of keeping inter-component-communication to a minimum, while still keeping the system secure and requiring authentication to access resources. This was ensured by having one script handling all data requests, and checking whether the user is properly authenticated. The requested elements are then fetched from a query-class, which again checks whether the user is authenticated. The query class then proceeds to fetch data from the database, and returns it to the request-handler. The request-handler relays the data to the front-end.

The log-in was planned with the intention of utilizing Kommunal Rapports own authentication service, in order to redirect users to their log-in pages, and return the needed profile information to the system. Any user would still be able to log-in to the system, even without having the correct subscription connected to their account. If such an account was signed in, the system was to alert the user that they did not have the correct subscription, or that they did not have a subscription at all. The log-out was to log the user out of the current system, and Kommunal Rapport's authentication services. This would prevent any confusion, and decrease threats present when leaving a computer unattended.

6.4.2 Implementation

The first scripts created on the back-end, were `config.php` and `ask.php`, as these were the main components in the system. Development of these scripts were initiated and almost completed in sprint 2 (Appendix B.3). They were continuously designed and refined throughout the development process. The development regarding `query.php` was started during sprint 3 (Appendix B.4), and finished in sprint 5 (Appendix B.6). Due to lack of communication with Kommunal Rapport's server-host Ramsalt, the log-in function, `login.php`, and `logout.php` was not implemented until sprint 6 (Appendix B.7). Fortunately, these were small and simple tasks, and were completely finished during sprint 7 (Appendix B.8).

ask.php is one of the main components in the back-end, and was one of the first things to be finished on the back-end. It was prioritized early on, and worked on from sprint 2 (Appendix B.3), and throughout the project. `ask.php` handles every request received from the client-side, ensuring that the user is logged in and has the correct authorization to view the requested resources. At each request, the script sets it's header to say it will return JSON-formatted text and it sets the CORS (Cross Origin Resource Sharing) for the file. CORS settings declare whether the system can utilize resources from other domains. During development, the CORS setting was set to `"*"`, allowing connections from any site on the world wide web. After deployment to Kommunal Rapport's servers, this was changed to fit the domain the server was running on. This was in order to prevent the script from being ran outside of the desired domain.

In `ask.php` there is a test to check if the user is logged in, and that the user has a subscription-ID connected with the session variable. Which subscription-ID the user had connected with their account on Kommunal Rapport's pages, is tested to ensure that only users with the correct subscription are allowed to search. Anyone is still able to fetch the counties and municipalities without having a valid subscription. This was done by excluding the queries involving counties and municipalities from the subscription-ID check. If this test fails, the script returns the value of `login_required` to the client, and redirects the user to the error-page. If the script sees that you have the wrong type of subscription, it returns an array with the value of `wrong_subscription`, and the front-end acts accordingly.

query.php handles every query, which makes this one of the most central scripts of the project. It was refined throughout the project, as more functionality on the front-end demanded corresponding functionality on the back-end. It incorporates a function called `authenticate`, which is called in almost every function to double check that the user has the correct permissions. The functions responsible for retrieving municipalities and counties do not require authentication because these functions only populate the map with names. At first, this was kept in a large JavaScript object, but was deemed unnecessary and converted to several queries in `query.php` in sprint 6 (Appendix B.7). This was to save space and avoid clutter in the script.

When a function is called from `ask.php`, the parameters gets bound to a PDO-prepared statement. Use of PDO, known as PHP Data Objects, restricts execution of SQL-queries to prevent attempts at SQL-injection or injection of harmful code. The statement is executed, and all the rows are fetched with `fetchAll(PDO::FETCH_ASSOC)`. This array of data is inserted into a JSON-encoded array, and returned to `ask.php`.

config.php is the configuration file for the database. This class fetches the login-credentials from a file called bachelor.ini, and uses these values to establish a connection to the database, which is then used in all queries.

login.php is the authentication script. For this script, Chris Hemmings' [8] library "oauth2-drupal" had to be included. This script sets the session variables being used to check if the user is logged in, oauth2state, a cookie for the users name, and a subscription-ID. The subscription-ID and username are fetched from the user profile on Ramsalt's server, using a HTTP-request with the authentication-token received after logging in. This is done with two separate queries, since they are stored in separate locations. OAuth 2.0 is elaborated in the next section 6.4.3.

At the top of `login.php`, `ask.php` and `logout.php` (Listing 2), the lifetime of a cookie is set, and the HTTPOnly-flag is turned on for the session-cookie. This prevents alterations to the cookie by cross-site-scripting, or front-end cookie changes. The name-cookie gets the same lifetime as the session-cookie, to ensure that they expire at the same time and preventing them from existing without one another. After the session is initiated, the session-cookie is updated to last another 5 days from the current time. The name-cookie is updated accordingly.

logout.php has a similar structure to `login.php`, with some slight alterations. As this script handles log-out events, it only authenticates through the OAuth2 server to check whether the user is logged in, and to get an authorization-code. The authorization-code is utilized when sending a request to Ramsalt's OAuth-server. The script proceeds to unset all session variables, and deletes cookies assigned by the system. The user is finally redirected to the front-page of the website.

6.4.3 Authentication

Authentication is done through Ramsalt's OAuth2 server, using Chris Hemmings' library, "oauth2-drupal". Chris Hemmings' is used after instructions from Ramsalt on what library to use. A request for authentication is created (Listing 3) with the following parameters; "clientId", "clientSecret", "redirectUri" and "baseUrl". The "clientId" specifies the ID of the current project, and is used to identify who issued the request. It is verified against the unique "clientSecret"-parameter on Ramsalt's server to verify that the authentication request is genuine. The "redirectUri" is the uri the OAuth2 server should redirect to after the user is authenticated. "BaseUrl" references the server the OAuth2 is located at. If any of these parameters change from what is specified by Ramsalt, the system will fail to authenticate users through the OAuth2 server.

Before the user is redirected to the authentication-page, the user's profile is set as the scope (Listing 4), which specifies what information is to be accessed. The script generates an authorization-url, and an authentication state is stored for later reference. The user is finally redirected to the authentication-page.

The user is directed back to `login.php` after being authenticated, values set in the authentication process are tested against the previously stored authentication state. If this fails, the script will remove assigned session variables and cookies, and redirect the user to the front-page. Else, the script sends two http-request to Ramsalt's server, requesting information about the users subscription-ID, and the users real-name. The real-name is stored in a session-based cookie, and the user is redirected to the search-page.

7 Testing

This chapter contains the testing strategy and the testing results for various testing methods employed to ensure the functionality of the product and that the product met the requirements. The tests include software tests on the code itself, and more high-level tests like usability and acceptance testing.

7.1 Testing Strategy

This section details the strategies used when testing the product. Unit testing was performed mostly in the early stages of development, followed by integration testing. After all units had been successfully integrated, system testing was performed. Usability testing and compatibility testing was then performed to ensure user friendliness and cross browser compatibility. At the final stages of development, acceptance testing was performed together with the customer in order to determine whether the final product met their requirements.

7.1.1 Unit Testing

Unit testing consists of making small functions for testing each unit of the code, a unit being the smallest testable part of an application. Each test typically tests a single function or module in the application, and should ideally work independently of each other. Rigorous unit testing is an important and useful part of software development, not only for ensuring that the code runs as it should, but also to ensure that changes to the code does not break itself or another part of the code. This is why continuous maintenance of the tests and writing of new tests is important [4][27].

Because an agile software development strategy was utilized, unit testing became a very important part of the development process. In order to quickly adapt to changes in the project requirements, it was important to be able to easily change the code and know it would function properly. The early and continuous writing of tests ensured that the development could stay agile without unnecessary amounts of extra work.

Working with a database would normally require a lot of tests to ensure safety and proper data input. Considering that the program never performs any changes to the database, the test development accordingly became more focused on the front-end of the application. This was to make sure that the data was handled correctly after being retrieved from the back-end. The front-end was written in AngularJS, and as such, it was decided to use Jasmine, a widely popular testing framework for JavaScript.

Listing 5 shows an example of how unit tests are written in Jasmine. When used with Angular, modules, functions and controllers have to be injected into the test to run.

7.1.2 Integration Testing

After the unit testing was completed, units are combined into components and integrated into the application. The process was to ensure the functionality and reliability is preserved on integration is called integration testing. The purpose of these tests is among other things to discover potential problems in the interaction between units, and is therefore largely dependent on extensive unit testing of each unit, so as to make the testing as efficient and quick as possible [16][27].

Because of the limited amount of time available for the development, testing had to be as efficient and quick as possible, while still being reliable and useful. Because this was a web development project, the interactions largely consisted of the connection between the front-end and the back-end, and between the back-end and the database, as opposed to several integrations within each part. Because the application is mostly dependent on the database and presenting the information in it, it was decided to use the bottom-up approach to integration testing. This method suggests that the integration of back-end and the database is tested first, and then the front-end to the back-end is tested. This ensured that the connection to the database was a part of the testing throughout the development process.

7.1.3 System Testing

After integrating all the parts of the system, system testing is performed. System testing is the process of testing specific functionality in order to make sure it complies with the projects' requirements and use cases [27]. The system tests are shown in tables 13, 14, 15, 16, and 17.

Table 13: System Test 01

Test ID	ST01
Test Name	Log in
Test Description	Log in using a valid Kommunal Rapport user account
Precondition	FR01 - User is logged out
Test Steps	• Click the log in link • Enter log in credentials
Success Condition	The logged in user is displayed in the top right corner

Table 14: System Test 02

Test ID	ST02
Test Name	Search
Test Description	Perform a search
Precondition	FR02 - User is logged in
Test Steps	• Specify what kind of participant the search is related to • Enter a search term
Success Condition	Search results for the search term given the search specification are shown

Table 15: System Test 03

Test ID	ST03
Test Name	View transaction
Test Description	Perform a search and view a transaction
Precondition	FR02 and FR04 - User is logged in and able to search
Test Steps	• Perform a search • Select a participant • Select a transaction
Success Condition	The appropriate transaction page is shown

Table 16: System Test 04

Test ID	ST04
Test Name	Flagging
Test Description	Identify flagging of transactions when searching
Precondition	FR02 and FR04 - User is logged in and able to search
Test Steps	• Perform a search • Select a participant
Success Condition	A list of transactions is shown and the appropriate flags are shown next to them

Table 17: System Test 05

Test ID	ST05
Test Name	Map navigation
Test Description	Perform a search using only the map
Precondition	FR02 - User is logged in
Test Steps	• Click on a county • Click on a municipality
Success Condition	The search result for the specific municipality is shown

7.1.4 Usability Testing

Usability testing allows a user to give feedback on how they use the product and how it is perceived. This is invaluable to the development process, because it provides insight that is easily missed as a developer. The people that were used for usability testing could be random people with no prior knowledge of the software, someone who is a part of the target user base, or an 'expert' with knowledge of how usability testing works and what to look for when using the software [28][27].

From the beginning of the development process, the customer put a big emphasis on usability. They wanted the website to be intuitive and self-explanatory, and to cause as little confusion for the user as possible. As a developer, it is very easy to lose sight of usability because of the high level of familiarity with the software and data. This makes usability testing all the more important, and the reason why the person testing should be someone who has not been part of developing the software.

The user base for the website consisted mostly of journalists and people who already know what to look for in the presented data, so the customer was asked to provide feedback from their journalists and employees on how user friendly and straight-forward the website was perceived, and suggestions for improvement. The testing itself followed no specific instructions, and they were not given any specific actions to perform or results to obtain. This meant that the contents of the test was largely decided by the person performing the testing.

7.1.5 Compatibility Testing

Compatibility testing, or in the case of web development, user experience testing, is testing to ensure that the software is compatible with the computing environment [9]. Because what was being developed was a website meant that browser compatibility would have to be tested. The functionality and the user's visual experience would have to be the same for all major browsers, both the latest installments and older versions.

This type of testing, as opposed to usability testing, can be done by someone involved in development, and there are even online tools available that automatically show the user how a website displays across different browsers. It is then up to the developer to either make the website work correctly for unsupported browsers, or in the case that this is not possible, tell the user that the browser they are using is not compatible [10][27].

In this sense, there was a limitation in respect to the frameworks and languages used while developing. The AngularJS framework is compatible with a wide range of browsers, new and old, but does, for instance, not support earlier versions of Internet Explorer than 9 [17]. This is not something that can be controlled, nor is it possible to adapt to without rewriting the website with a different framework. This leaves no other option than to let the user know that the website might not function correctly, or at all, in their browser.

7.1.6 Acceptance Testing

The acceptance test is one of the final stages of testing. The purpose of this testing phase is to ensure, for the customer's sake and the developer, that the developed product is what the customer wanted, and that it meets the requirements. For this purpose, test cases were developed, covering as many parts of the code as possible. They describe scenarios that the user will find themselves in when using the software, from start to finish. The testing itself has to be performed by someone intimately familiar with the software or the requirements put forward by the customer [1][2][27].

In this case, the requirements were very loosely defined, and there was to a large degree freedom to implement them as found reasonable and practical. This made acceptance testing all the more important because it was required to know if the way the requirements were interpreted and implemented were acceptable to the customer. The requirements mostly specified certain use cases, and the acceptance testing needed to demonstrate all of these very extensively in order for the customer to see what had been developed, and whether or not it was satisfactory.

7.2 Test Results

This section contains all the results from the various testing performed over the duration of the project. Some of the results are only described in this section, with the actual result data shown in the appendix.

7.2.1 Unit Testing

In total, 42 unit tests were written and run. At the time of delivery, all tests passed (Appendix C). Making sure that this was the case throughout development, ensured that there were no unexpected errors in the code as a result of new features being added.

7.2.2 Integration Testing

Because there was no testing framework utilized for integration testing, the testing was done manually by ensuring the components worked as expected after integration. All components integrated successfully throughout the project, and no further issues were detected at the time of delivery.

7.2.3 System Testing

The results of the system testing are shown in table 18.

Table 18: System Testing Results

Test ID	**Result**	**Comment**
ST01	PASSED	The website only restricts access to the database, not the actual website itself
ST02	PASSED	If no results are found, the result table is not shown
ST03	PASSED	It is possible that a participant contains no transactions
ST04	PASSED	If a transaction contains no prices, the flag is not shown
ST05	PASSED	

7.2.4 Usability Testing

The feedback from the customer was given verbatim by email, and provided a more general direction in terms of feedback than a more structured test would. Most of the feedback pertained to the way data was displayed to the user and how intuitive it was in doing so. There were also some comments on the design elements and the way the website looked. Some of the data was trimmed down, and visual cues and tooltips were added to make it more user friendly.

It also became apparent that displaying the data intuitively to someone who was not familiar with neither the data nor the website itself was even more challenging than anticipated. It was unclear whether the nature of the data made it inherently hard to provide an intuitive way of presenting it, or if there was further room for improvement. More extensive usability testing might have given more insight into this, and provided more opportunities to better understand what makes the data intuitive.

7.2.5 Compatibility Testing

By going through each use case across most modern browsers, it was possible to find any inconsistencies in how the website displayed and behaved, and readjust it accordingly. Mobile compatibility was also a big focus, and through testing different screen sizes, it was possible to make the website responsive enough to support most modern phones. The results of the compatibility testing is shown in table 19.

Table 19: Browser compatibility results

Browser	**Version**	**Result**	**Comment**
Chrome	50	PASSED	The main browser used for development
Firefox	46	PASSED	
Microsoft Edge	13	PASSED	Slightly different dropdown style
Internet Explorer	11	PASSED	Different dropdown style, map considerably smaller, tooltip persists on map. All functionality retained
Opera	37	PASSED	Slightly different dropdown style
Safari	9	PASSED	

7.2.6 Acceptance Testing

Throughout the development period, frequent meetings were held with the customer. One of the topics often discussed was how the customer felt about the current state of the product, and what they felt could be improved upon or added. With the initial requirements so loosely defined, this kind of informal and procedural acceptance testing was very helpful in determining the direction the development was heading, and what features required the most focus. The customer gave positive feedback throughout development, as well as expressing what parts could better suit the requirements or be otherwise improved upon.

The final acceptance test itself was performed informally through a meeting with the customer at the end of the development period. All the requirements and initial goals expressed by the customer were discussed, as well as suitable features for further development. The customer was ultimately satisfied with the current state of the product, and felt that the requirements had been met.

8 Evaluation

This chapter addresses challenges encountered during the project, evaluation of aspects such as customer and group interaction and a section concluding on the entire project.

8.1 Challenges

Challenges that occurred during the project and applied to the parts of the product, such as back-end, front-end and database.

8.1.1 Front-end

The most difficult challenge encountered during the development of this project was making the application intuitive. Being built upon specialized, and detailed technical data, the application needed to convey the essence in a manner where anyone using the application could make sense of it. This proved to be challenging as a lot of the content was unfamiliar to the development team, and even some was unfamiliar to the customer. The problem was tackled by abstracting it to three layers, and visualizing it through a graph and a timeline.

Further, a lot of significant information was missing from the original dataset. Documenting price when registering a transaction was not required, thus many of the transactions are missing essential data, leaving holes in the history. Many properties detailed in the dataset does not have an address, and is only referenced by a hierarchy of numbers. These complications further limited the ways to make the application intuitive, as missing price removes continuity and missing an address makes properties difficult to relate to. It meant that the team could not rely on these fields, and had to handle the case where the data was not present.

8.1.2 Database

After receiving the dataset, the team struggled for some time with the task of converting the dataset from its original csv format to a more useful MySQL format. The importation into the database failed repeatedly because one of the 1.8 million rows was corrupted. This issue was solved by removing the corrupted row from the original file.

The domain specific and technical data caused some challenges in the database as well. Firstly, identifying what columns could be used as unique identifiers proved difficult. For example, finding what makes a property unique required a combination of 5 columns. Finding a similarly unique identifier for documents, agents and transactions required a lot of trial and error, and careful inspection to ensure that no data would be lost. Secondly, the dataset contained several columns of redundant data, where names and properties were repeated for several rows. Some columns contained misspelled agent names and erroneous agent-types, all of which needed to be manually fixed.

New requirements were added throughout the project, demanding more complex information extraction. The customer wanted the system to be able to highlight properties with radical price variations, requiring the database to match and compile price history of every property in a given query. This proved to be too demanding for the database, and would slow the system down considerably. The solution to this problem was to pre-compile the necessary fields, but in turn this restricted the filtering capabilities, and made it impossible to fulfill a later requirement from the customer.

8.1.3 Back-end

The most significant challenges concerning the back-end was handling log-out and securing the data. Early iterations of the log-out-system would only log the user out of the system, but not the customer's network. After some time, it was discovered that the cause of this was an error in Ramsalt's OAuth2 Server, preventing the system from logging the user out. This resulted in keeping the code for logging the user out of their system, for any future updates they might have.

The system was to be deployed on a web-server, where it would be subject to attacks, requiring the system to be kept secure. This was ensured by the back-end, and was a continuous challenge during development. The system had to be exclusively accessible by subscribing members, which required the system to enforce access control. Subscription-IDs and saved credentials provided by Kommunal Rapport could not be saved on the client-side, as this would expose critical information and make the user vulnerable. Even though one can prohibit alterations of a cookie by enabling HTTP-only, it is still open to attacks, and is a security threat. Another security threat that would occur if the authentication was handled by the client-side, was that the client information would be exposed to the public. This resulted in keeping the authentication on the back-end, and storing the users real-names as a cookie-value.

8.2 Requirements

Most of the requirements were met

8.3 Customer interaction

Meetings with the customer were planned at the end of each sprint, as mentioned in section 3.1.5. The progress and problems encountered were presented and discussed, and further development was agreed upon. In these meetings the customer provided feedback on the application, which the group would take into consideration when planning the next sprint. The interaction with the customer took place over e-mail and Google Hangouts. One meeting was held in person with the customer. This meeting was very productive and beneficial to the project. The interaction with the customer did not suffer from the lack of co-location, but having more in-person meetings could have been helpful in regards to communication efficiency.

The customer provided access to help from external parties if needed, such as Ramsalt for details about the server-deployment and access control, and Ambita for data information.

8.4 Group interaction

Regular group meetings were held four times per week, including meetings with the supervisor and the customer. The weekly work hours were set to 20 hours, of which meetings counted for 14, leaving 6 hours to spend working individually. Each sprint expected a total of 240 working hours, some sprints were shorter due to holidays and the planned excursion to Japan. The group communicated otherwise through Slack and Facebook-chat. All group appointments were listed in a shared Google calendar all members subscribed to.

The meetings were efficient, and the project benefited from the regular meetings by ensuring that every member was up to date on the current progress. It helped the group complete their work on time, and ensured good morale. Sporadically, members were late for meetings, or would not show at all. To combat this, a small fine in the form of bringing something for the group to eat was introduced. This was enforced sparingly, and did not reduce the problem. The group used GitHub for code-collaboration. This was mostly a great benefit to the project. Occasionally, major rewrites made incorporating simultaneously developed code very difficult. This could have been avoided by better communication regarding major changes, and limiting work while these were being introduced.

8.5 Further development

At completion, most of the customer's requirements had been met. The most prominent requirement that was not fulfilled, and would be a useful addition, was the ability to pay for the service without being a subscriber at Kommmunal Rapport. This would potentially expand the user base considerably, as the users would be able to pay specifically for the service without having to commit to a subscription. Another unfulfilled requirement was the implementation of a map overview of the properties. This would increase user-friendliness by making the user able to identify properties by location.

There are several features that could be added to further increase the intuitiveness of the application. Adding more detail and specific views to the pages listing transactions conducted by municipalities, would emphasize the desired focus on municipalities. In order to give the user an overview and more ways to browse data, a rankings page could be made, highlighting different aspects such as; the most substantial transactions, the users who have been involved in the most transactions and the transactions with the most radical price development.

A feature that was purposed late in the development process, but not added, was displaying which political party was in charge of the municipality at the time of a transaction. This would have given the user a political aspect of the data, and could be further expanded upon to enable deeper analysis. Finally, the customer plan to publish the service annually with updated transaction data. Being able to automatically import the csv-file into the current database structure 17 could drastically simplify this process and enable more frequent updates.

8.6 Lessons learned

Several lessons were learned throughout this project regarding customer communication, project management and project planning. For instance, customer involvement is a crucial part of agile software development. Though the team practiced this, it is difficult to substitute in-person meetings, and it is apparent that this would have enabled earlier user-testing and better requirement specification.

Having regular scheduled meetings proved beneficial and impacted the project positively, in comparison to earlier projects, where meetings have been more sporadic. The meetings could have been more structured, and more productive, possibly by starting with a stand-up meeting. The team learned that having a light punishment for skipping meetings was not sufficient to make ensure members attend.

8.7 Conclusion

The csv-file acquired by Kommunal Rapport was successfully inserted into a database. An intuitive and detailed interface was developed for journalists and customers of Kommunal Rapport, to examine property transactions involving municipalities. The property database fulfilled most of the customer's requirements, and the most important goals were met. Regular meetings and interaction with both the customer and the supervisor proved useful, and led to a finalized product that was delivered to Kommunal Rapport.

List of Figures

List of Tables

Listings

References

[1] *Acceptance Testing*. Accessed: 2016-03-15. URL: http://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm.

[2] *Acceptance Testing*. Accessed: 2016-03-15. URL: http://istqbexamcertification.com/what-is-acceptance-testing/.

[3] *Adressa eiendomsbasen*. Accessed: 2016-02-10. URL: http://www.adressa.no/forbruker/eiendomsbasen/.

[4] Agile Alliance. *Unit Testing*. Accessed: 2016-03-15. URL: http://guide.agilealliance.org/guide/unittest.html.

[5] *AngularJS*. Accessed: 2016-05-28. URL: https://angularjs.org/.

[6] AngularJS. *AngularJS Directives*. Accessed: 2016-05-10. URL: https://docs.angularjs.org/guide/directive.

[7] Apache. *Apache2 installation*. URL: https://httpd.apache.org/docs/current/install.html.

[8] *Chris Hemmings - oauth2-drupal GitHub page*. Accessed: 2016-03-04. URL: https://github.com/chrishemmings/oauth2-digitalocean.

[9] *Compatibility Testing*. Accessed: 2016-03-15. URL: http://www.tutorialspoint.com/software_testing_dictionary/compatibility_testing.htm.

[10] *Compatibility Testing*. Accessed: 2016-03-15. URL: http://istqbexamcertification.com/what-is-compatibility-testing-in-software/.

[11] Composer. *Install Composer*. Accessed: 2016-05-15. URL: https://getcomposer.org/download/.

[12] *Dette selges boligene for der du bor*. Accessed: 2016-02-10. URL: http://www.budstikka.no/okonomi-og-naringsliv/nyheter/okonomi-og-bolig/dette-selges-boligene-for-der-du-bor/s/5-55-38972.

[13] *Ditt Næringsliv Boligbasen*. Accessed: 2016-02-10. URL: http://www.dn.no/boligpriser/.

[14] *Emne - Informatikk prosjektarbeid II, NTNU*. Accessed: 2016-04-19. URL: https://www.ntnu.no/studier/emner/IT2901#tab=omEmnet.

[15] GitHub. *GitHub Webhooks*. Accessed: 2016-05-15. URL: https://developer.github.com/webhooks/.

[16] *Integration Testing*. Accessed: 2016-03-15. URL: https://msdn.microsoft.com/en-us/library/aa292128%5C%28v=vs.71%5C%29.aspx?f=255&MSPPError=-2147217396.

[17] *Internet Explorer Compatibility*. Accessed: 2016-03-24. URL: https://docs.angularjs.org/guide/ie.

[18] *Jasmine*. Accessed: 2016-02-10. URL: http://jasmine.github.io/edge/introduction.html.

[19] *Kommunebbarometeret*. Accessed: 2016-04-19. URL: http://kommunal-rapport.no/content/om-kommunal-rapport.

[20] *Kommunetesten*. Accessed: 2016-02-10. URL: http://www.forbrukerradet.no/tips-og-r%C3%A5d/kommunetesten/.

[21] *Leverandørdatabasen*. Accessed: 2016-04-19. URL: http://data.kommunal-rapport.no/.

[22] MySQL. *MySQL installation*. URL: http://dev.mysql.com/doc/refman/5.7/en/installing.html.

[23] *Osloby Boligverdi*. Accessed: 2016-02-10. URL: http://boligverdi.osloby.no.

[24] *PHP: Hypertext*. Accessed: 2016-05-28. URL: http://php.net/.

[25] *PHPUnit*. Accessed: 2016-02-10. URL: https://phpunit.de/.

[26] *ReactJS*. Accessed: 2016-05-28. URL: http://reactjs.net/.

[27] Sommerville, Ian. *Software Engineering*. 9th. Pearson Education, Inc., 2011.

[28] *Usability Testing*. Accessed: 2016-03-15. URL: http://www.usability.gov/how-to-and-tools/methods/usability-testing.html.

A Projects Documents

This section contains the important documents regarding the project. Documents such as the status report, activity plan and the meeting minutes will be presented.

A.1 Status Report

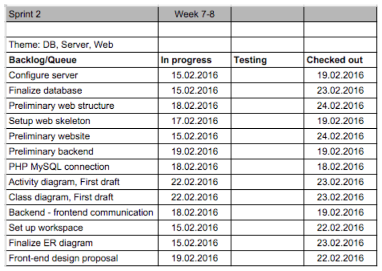
The status report for sprint 2, that took place in week 7 and 8, is included in this section. The report were delivered to the supervisor after every sprint.

A.1.1 Introduction

We started the second sprint monday in week 7. The duration of the sprint was two weeks, from monday 15.02.2016 to monday 26.02.2016. We have completed one skype meeting with our customer on friday 19.02.2016. All group members have participated in the work so far.

A.1.2 Progress summary

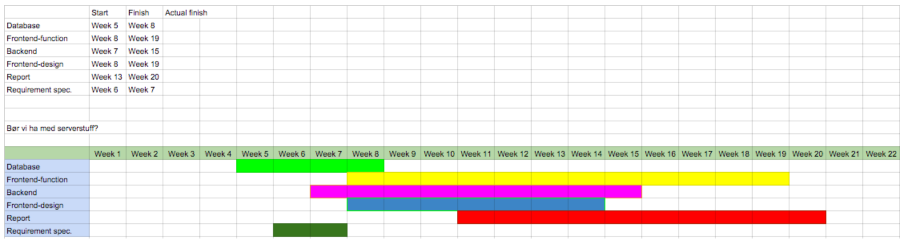
Figure 21 is our activity plan for the second sprint. We have completed the sprint, and completed all the tasks we had in the backlog for this sprint. We finished the tasks which we had left over from the previous sprint early in sprint. We underestimated how much time each task would require, so we had to expand the backlog (Figure 24).



Sprint 2	Week 7-8		
Theme: DB, Server, Web			
Backlog/Queue	**In progress**	**Testing**	**Checked out**
Configure server	15.02.2016		19.02.2016
Finalize database	15.02.2016		23.02.2016
Preliminary web structure	18.02.2016		24.02.2016
Setup web skeleton	17.02.2016		19.02.2016
Preliminary website	15.02.2016		24.02.2016
Preliminary backend	19.02.2016		19.02.2016
PHP MySQL connection	18.02.2016		18.02.2016
Activity diagram, First draft	22.02.2016		23.02.2016
Class diagram, First draft	22.02.2016		23.02.2016
Backend - frontend communication	18.02.2016		19.02.2016
Set up workspace	15.02.2016		22.02.2016
Finalize ER diagram	15.02.2016		23.02.2016
Front-end design proposal	19.02.2016		22.02.2016

Figure 21: Backlog Sprint 2

The activity plan in figure 21 is based on the milestone plan in figure 22. All the tasks in our activity plan are steps that must be performed in order to complete the milestones.



	Start	Finish	Actual finish
Database	Week 5	Week 8	
Frontend-function	Week 8	Week 19	
Backend	Week 7	Week 15	
Frontend-design	Week 8	Week 19	
Report	Week 13	Week 20	
Requirement spec.	Week 6	Week 7	

Bør vi ha med serverstuff?

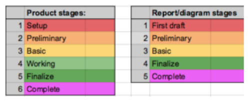
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20	Week 21	Week 22
Database					🟩	🟩	🟩	🟩														
Frontend-function								🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨			
Backend							🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪							
Frontend-design								🟦	🟦	🟦	🟦	🟦	🟦	🟦								
Report											🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
Requirement spec.						🟩	🟩															

Figure 22: Milestone Plan

A.1.3 Open / closed problems

We have experienced few problems this sprint, so this means that there has not been that much delay according to our plan. One problem we have met was deciding which javascript graphing framework to use. We wanted a framework that was easy to use and not to much time consuming. We also had some challenges setting up our own workspace, we spent a lot more time doing this than what we had expected.

In this sprint we found ourselves spending a lot less time on each task than we previously anticipated. We estimated spending roughly 220 hours on 12 tasks, but everyone worked very efficiently, so we ended up starting many other tasks. We had also not defined the extent of each status term, f.ex: setup and preliminary, and therefore finished tasks very quickly compared to what we had estimated. Now we have defined the different stages of progress for each task.

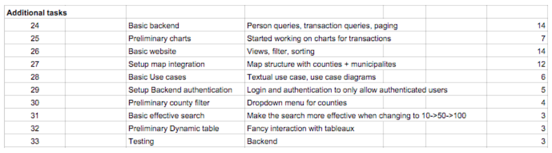


	Product stages:
1	Setup
2	Preliminary
3	Basic
4	Working
5	Finalize
6	Complete

	Report/diagram stages
1	First draft
2	Preliminary
3	Basic
4	Finalize
5	Complete

Figure 23: Task Stages

In conclusion we feel that we are well on our way with the product part of the project as well as keeping up to date with most of our diagrams. We have added the additional tasks we did during this sprint in the image below:



Additional tasks			
24	Basic backend	Person queries, transaction queries, paging	14
25	Preliminary charts	Started working on charts for transactions	7
26	Basic website	Views, filter, sorting	14
27	Setup map integration	Map structure with counties + municipalites	12
28	Basic Use cases	Textual use case, use case diagrams	6
29	Setup Backend authentication	Login and authentication to only allow authenticated users	5
30	Preliminary county filter	Dropdown menu for counties	4
31	Basic effective search	Make the search more effective when changing to 10->50->100	3
32	Preliminary Dynamic table	Fancy interaction with tableaux	3
33	Testing	Backend	3

Figure 24: Additional Tasks

A.1.4 Planned work for next period

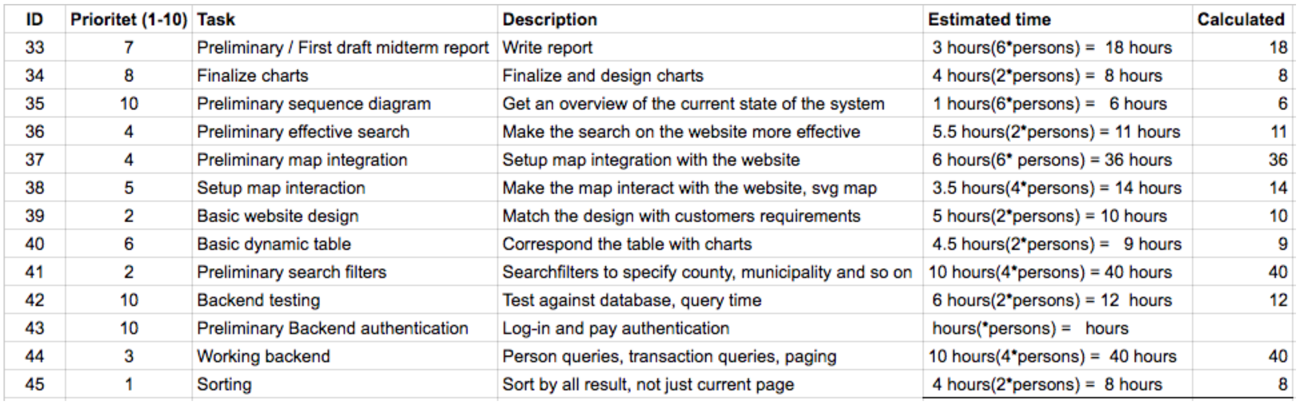
Here is our backlog for the third sprint, week 9 and 10. We are mostly focusing on expanding the search functionality, displaying search results in a proper and intuitive manner, implementing charts, map functionality and design improvements.

A.1.5 Updated risks analysis

Table 6 shows our risk analysis. We have experienced some sickness within the group, but no remedial action has been required yet.

A.2 Activity Plan

The following is an example of the activity plan for sprint 3. It follows the same template for all the status reports and documented the actual time teach task required.



ID	Prioritet (1-10)	Task	Description	Estimated time	Calculated
33	7	Preliminary / First draft midterm report	Write report	3 hours(6*persons) = 18 hours	18
34	8	Finalize charts	Finalize and design charts	4 hours(2*persons) = 8 hours	8
35	10	Preliminary sequence diagram	Get an overview of the current state of the system	1 hours(6*persons) = 6 hours	6
36	4	Preliminary effective search	Make the search on the website more effective	5.5 hours(2*persons) = 11 hours	11
37	4	Preliminary map integration	Setup map integration with the website	6 hours(6* persons) = 36 hours	36
38	5	Setup map interaction	Make the map interact with the website, svg map	3.5 hours(4*persons) = 14 hours	14
39	2	Basic website design	Match the design with customers requirements	5 hours(2*persons) = 10 hours	10
40	6	Basic dynamic table	Correspond the table with charts	4.5 hours(2*persons) = 9 hours	9
41	2	Preliminary search filters	Searchfilters to specify county, municipality and so on	10 hours(4*persons) = 40 hours	40
42	10	Backend testing	Test against database, query time	6 hours(2*persons) = 12 hours	12
43	10	Preliminary Backend authentication	Log-in and pay authentication	hours(*persons) = hours	
44	3	Working backend	Person queries, transaction queries, paging	10 hours(4*persons) = 40 hours	40
45	1	Sorting	Sort by all result, not just current page	4 hours(2*persons) = 8 hours	8

Figure 25: Activity Plan

A.3 Example of meeting minutes

The meeting minutes with the customer is presented in this section. The meeting was in Trondheim and the development team and the customer were gathered to discuss the process and the product.

A.3.1 Meeting with customer

Present: Tor, Kasper, Kathrine, Lasse, Caroline, Adrian and Henning Location of meeting: E204, Gløshagen Date and time: 09.03.2016, 17:00

A.3.2 Status from the team

The development process is going as planned, and there is now possible to perform a search on transactions at the website. The chart is also implemented.

A.3.3 Customer test/feedback and discussion

Henning tests the website and gives feedback:
When hovering the map, he wants a feature that shows which county he i hovering over. He wants an advanced option that makes it possible to do a search after transaction in both county and municipality. In the search result table, he wants the description at the top row to follow when scrolling the page. He wants the result table to be more intuitive, to more easily understand the transactions. Make the description row in the result table more intuitive, by changing name. What does "løpe" mean? Is this a row that needs to show? Make it possible to see if a property has some irregularities.
He also tells ut that the most important for a journalist is the change in price, buyer and seller of the property.
We also discuss the possibility to get data with addresses.

A.3.4 Summary

Check what "løpe" means Make the website more intuitive and user friendly. Add features the customer is missing Henning: Check if there is possible to get a dataset with addresses

B Iterations

The following section shows the iterations in this project. The total estimated hours each sprint is 240 hours, as seen in estimated time for each sprint it does not fulfill exactly so many hours. The reason for this is that the sprints do not include the daily meetings, customer meeting, nor the meetings with the supervisor. Give or take some hours, the leftover time in each sprint were used on daily meetings with 8-10 hours, 1-2 hours were planned meetings with the customer, and 1 hour was a planned meeting with the supervisor.

B.1 Startup

In the preliminary stages of this project, the focus was on getting a grasp of what the customer wanted with the product. There were quickly established contact with the customer and got the information required in order to make the plans. The rest of the time was mostly spent planning the time schedule, milestone timeline and discussions regarding development tools.

B.2 Sprint 1

The first sprint was in week 5 and 6, where the basic tasks were planned. The main focus were setup of different elements in the project and to get a overview over what had to be done.

B.2.1 Goals

This sprint was focused on the requirements, both functional and non-functional. In this iteration it was also started planning the database by modelling an ER diagram, setup and configuration of the database. The time estimation was done by using planning poker, and the actual time shows how many hours actually used. Total hours were calculated based on given hours per person. All tasks have a priority from 1 to 10, where 1 is indicating top priority and were put first in the queue over tasks that had to be done. Table 20 shows the tasks that was assigned to this sprint.

Table 20: Sprint 1 Backlog

ID	Priority (1-10)	Task	Time estimated	Actual time
1	1	Planning database	48 hours	80 hours
2	1	ER diagrams, First draft	15 hours	12 hours
3	2	Database setup	16 hours	18 hours
4	2	Database insert	30 hours	16,5 hours
5	4	Report writing	24 hours	28 hours
6	1	Finalize very preliminary report	12 hours	10,5 hours
7	6	Use-case diagrams, outsource (first draft)	15 hours	0 hours
8	5	Functional requirements	12 hours	18 hours
9	5	Non-functional requirements	6 hours	8 hours
10	7	Activity diagram, First draft	8 hours	0 hours
11	4	Class diagram, First draft	8 hours	0 hours

B.2.2 Burndown chart

The burndown shown in Figure 26 shows how many days there were planned to use on each task and how many days we actually used.

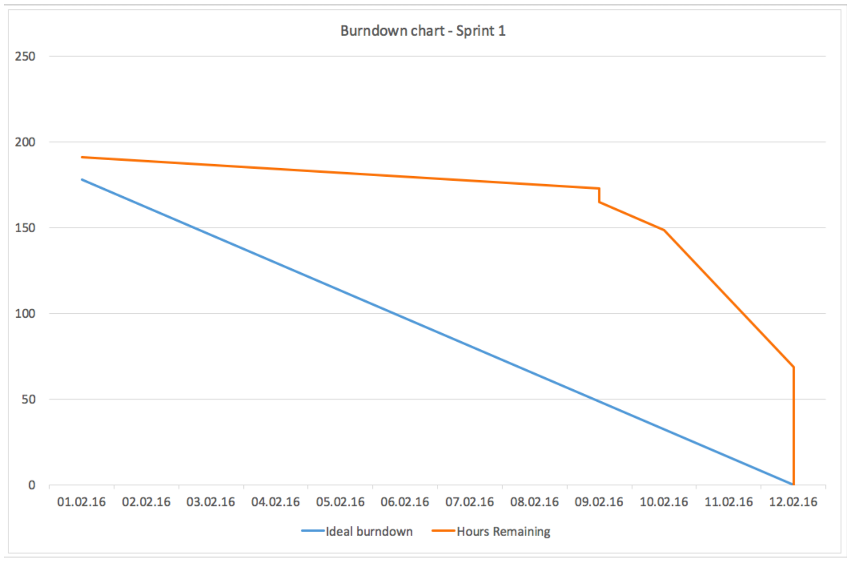


Figure 26: Burndown chart sprint 1

B.2.3 Retrospect and result

All tasks assigned to this sprint were not finished. The team did not manage to create class diagram or activity diagram. The focus was on the database and the report. However, the two tasks that was not finished were taken into the planning of the next sprint.

There was used a lot more time than estimated on:

- Writing the report
- Establishing the requirements
- Planning of web (angular vs react) and database (discussion on setup)

Overall it was spent pretty much exactly the total time estimated in the beginning, but some tasks required a lot more work than expected. The total estimated time was 194 hours, and the actual time used was 191 hours.

B.3 Sprint 2

Sprint 2 took place in week 7 and 8, the main focus in this iteration were on configuration of the server and the database. The setup of the workspace for each team member was important.

B.3.1 Goals

This sprint was mainly focused on finalizing the database, getting the server up and running and making a working web skeleton so that the customer could test the early version of the system. The following tasks of descending priority is shown in 21

Table 21: Sprint 2 Backlog

ID	Prioritet (1-10)	Task	Time estimated	Actual time
12	1	Configure server	6 hours	8 hours
13	1	Finalize database	40 hours	30 hours
14	3	Preliminary web structure	36 hours	4 hours
15	4	Setup web skeleton	36 hours	20 hours
16	7	Preliminary website	40 hours	22 hours
17	3	Preliminary backend	8 hours	8 hours
18	2	PHP MySQL connection	2 hours	3 hours
19	8	Activity diagram, First draft	8 hours	4 hours
20	8	Class diagram, First draft	8 hours	8 hours
21	7	Backend - frontend communication	12 hours	16 hours
22	5	Setup workspace	12 hours	18 hours
23	2	Finalize ER diagram	18 hours	18 hours

B.3.2 Burndown chart

As shown in Figure 27, the burndown chart for sprint 2 is shown.

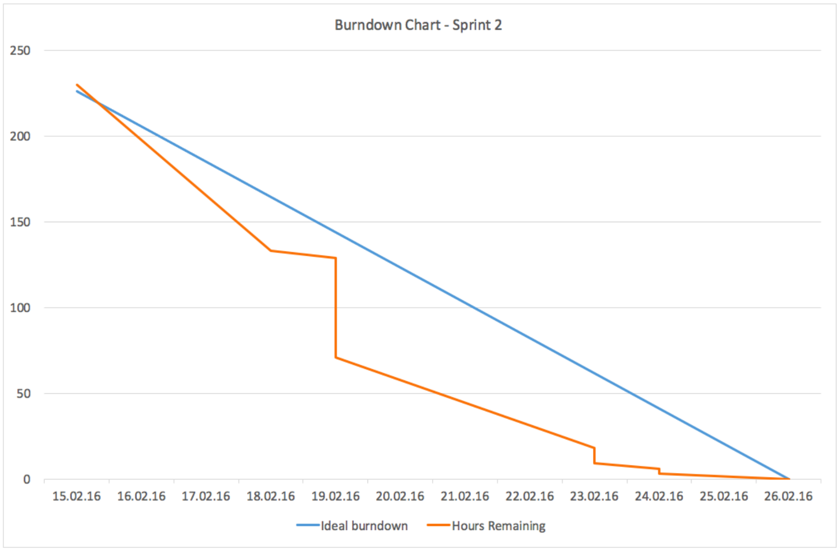


Figure 27: Burndown chart sprint 2

B.3.3 Retrospect and result

There were only a few problems that were experienced in this sprint, that meant that there was no delay according to the plan. One problem was deciding which JavaScript graphing framework to use. It was desired to use a framework that was easy to use and not too time consuming. There were also some challenges setting up the workspace, and required more time than calculated.

In this sprint there were spent less time on each task than previously anticipated. There were estimated spending roughly 220 hours on 12 tasks, but everyone worked very efficiently, and ended up starting many other tasks as shown in Table 22. There were not defined the extent of each status term, f.ex: setup and preliminary, and therefore finished tasks more quickly compared to what were estimated. The defined stages of progress were set and followed.

Table 22: Sprint 2 additional tasks

ID	Priority (1-10)	Task	Time estimated	Actual time
24	5	Basic back-end	0 hours	14 hours
25	4	Preliminary charts	0 hours	7 hours
26	3	Basic website	0 hours	14 hours
27	4	Setup map integration	0 hours	12 hours
28	6	Basic Use cases	0 hours	6 hours
29	2	Setup Backend authentication	0 hours	5 hours
30	5	Preliminary county filter	0 hours	4 hours
31	7	Basic effective search	0 hours	3 hours
32	7	Preliminary Dynamic table	0 hours	3 hours
33	7	Testing	0 hours	3 hours

B.4 Sprint 3

The third sprint were set in week 9 and 10, the main focus were basic front-end and working back-end.

B.4.1 Goals

During this sprint there were mainly work towards finalizing the charts, (which represents how the value of a property evolves over time), making and integrating the front-page map, and general design improvements. The following tasks is shown in Table 23.

Table 23: Sprint 3 Backlog

ID	Prioritet (1-10)	Task	Time estimated	Actual time
34	7	Preliminary / First draft midterm report	18 hours	24 hours
35	8	Finalize charts	8 hours	8 hours
36	10	Preliminary sequence diagram	6 hours	5 hours
37	4	Preliminary effective search	11 hours	2 hours
38	4	Preliminary map integration	36 hours	23 hours
39	5	Setup map interaction	14 hours	10 hours
40	2	Basic website design	10 hours	11 hours
41	6	Basic dynamic table	9 hours	11 hours
42	2	Preliminary search filters	40 hours	23 hours
43	10	Backend testing	12 hours	7 hours
44	10	Preliminary Backend authentication	0 hours	0 hours
45	3	Working backend	40 hours	27 hours
46	1	Sorting	8 hours	7 hours

B.4.2 Burndown chart

The burndown chart for sprint 3 is shown in Figure 28.

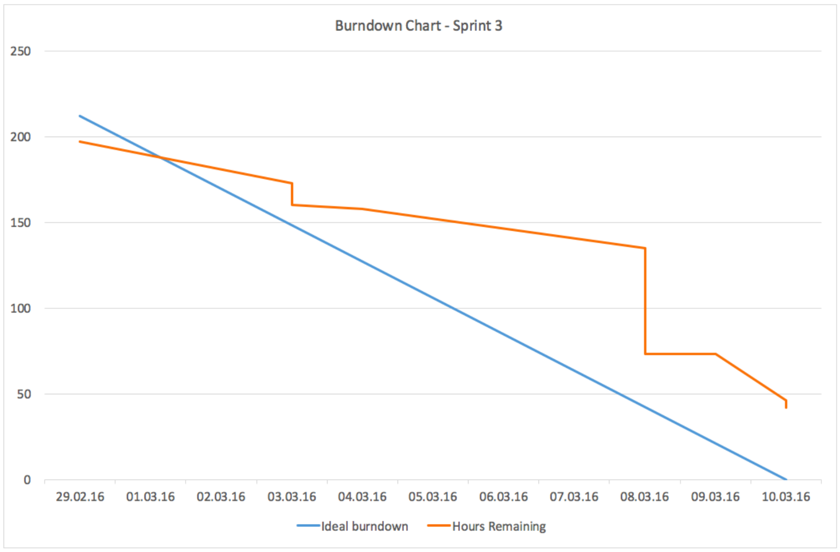


Figure 28: Burndown chart sprint 3

B.4.3 Retrospect and result

One problem that were experienced were not being able to finish all the tasks in this sprint. Half the team left for Japan on the last Friday. This loss of labor was not taken into consideration when planning this sprint. Nevertheless, this were not seen as a big issue, as there were some left-over tasks in one of the coming sprints as shown in Table 24.

Table 24: Sprint 3 additional tasks

ID	Prioritet (1-10)	Task	Time estimated	Actual time
47	7	Basic back-end	18 hours	24 hours
48	8	Preliminary charts	8 hours	8 hours
49	10	Basic website	6 hours	5 hours
50	4	Setup map integration	11 hours	2 hours
51	4	Basic Use cases	36 hours	23 hours
52	5	Setup Back-end authentication	14 hours	10 hours
53	2	Preliminary county filter	10 hours	11 hours
54	6	Basic effective search	9 hours	11 hours
55	2	Preliminary Dynamic table	40 hours	23 hours

B.5 Sprint 4

Sprint 4 were in week 11 and 12, and the main focus was on the report.

B.5.1 Goals

During this sprint the goal were to organize, work on, and finalize the midterm report.

Table 25: Sprint 4 Backlog

ID	Prioritet (1-10)	Task	Time estimated	Actual time
56	1	Organize midterm report	15 hours	20 hours
57	1	Work on midterm report	15 hours	22 hours
58	1	Finalize midterm report	9 hours	11 hours

B.5.2 Retrospect and result

The midterm organized and finalized, and were delivered before deadline. There were total 39 hours estimated on these tasks, but the actual time spent on this sprint were 53 hours. There were some reduction in the estimated time because of the exhibition to Japan and the Easter holidays. The burndown chart was not generated for this sprint due to the reduced time and the limited tasks that were worked on.

B.6 Sprint 5

Sprint 5 were in week 13 and 14, and flagging of the transactions and map function were important tasks.
.

B.6.1 Goals

During this sprint the focus was the search results and report. The main focus was to make the website more user friendly, such as the result table and advanced search. Another important goal was to implement the feature "flagging". The table 26 shows the tasks planned for this sprint.

Table 26: Sprint 5 Backlog

ID	Prioritet (1-10)	Task	Time estimated	Actual time
59	7	Setup Flagging	12 hours	14 hours
60	2	Working result table	20 hours	20 hours
61	4	Basic Map	30 hours	17 hours
62	3	Scrolling function	8 hours	12 hours
63	1	Advanced Search	30 hours	40 hours
64	5	Database fixes	10 hours	7 hours
65	2	Report	90 hours	100 hours

B.6.2 Burndown chart

Figure 29 shows the burndown chart for sprint 5.

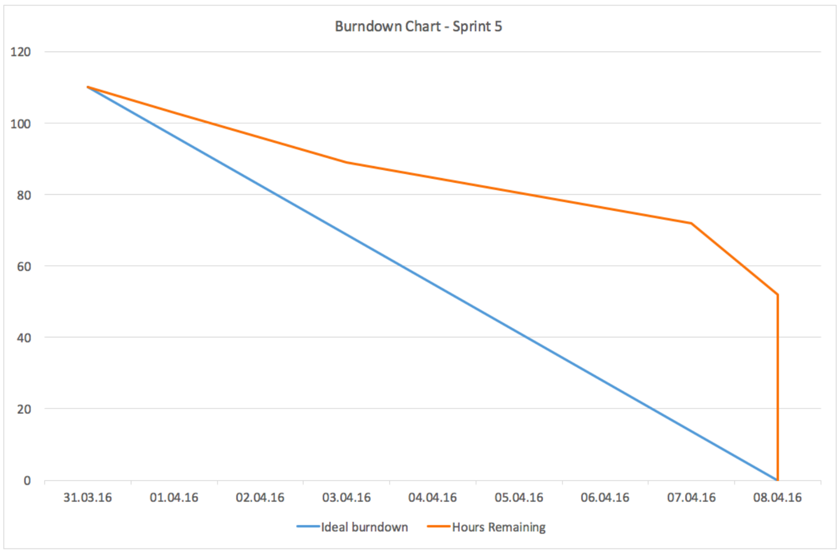


Figure 29: Burndown chart sprint 5

B.6.3 Retrospect and result

In this sprint there were a few problems that occurred. One problem was the feedback that was given on the midterm report. This required the team to prioritize the work hours slightly different, and work more continuously with the report. In this sprint there was only planned tasks for 110 hours, and the rest of the hours was used to writing and improving the report. All the tasks assigned to this sprint was finished. The total hours used on tasks in this sprint was 210 hours, and the estimated time was 200 hours.

B.7 Sprint 6

Sprint 6 were in week 15 and 16, and the main goals were the log-in function and develop an intuitive result table.

B.7.1 Goals

In this sprint the focus was implementing the log-in function, set up a working map, work on the report and testing. The table 27 shows the tasks planned for this sprint.

Table 27: Sprint 6 Backlog

ID	Prioritet (1-10)	Task	Time estimated	Actual time
65	1	Setup Log-in function	20 hours	22 hours
66	4	Working Map	20 hours	23 hours
67	3	Scrolling	8 hours	10 hours
68	3	Advanced search	15 hours	12 hours
69	2	Intuitive result table	30 hours	30 hours
70	1	Report	96 hours	100 hours
71	4	Testing	4 hours	6 hours

B.7.2 Burndown chart

Figure 30 shows the burndown chart for sprint 6.

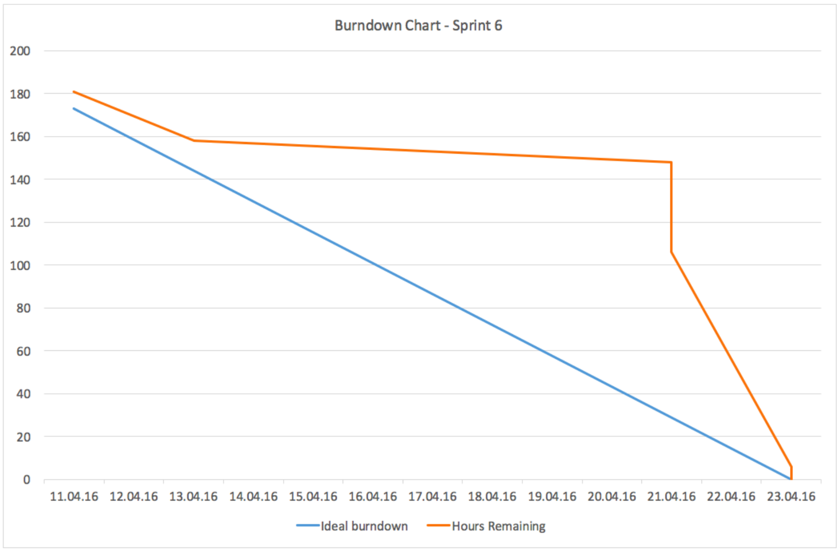


Figure 30: Burndown chart sprint 6

B.7.3 Retrospect and result

In this sprint there were no big problems that occurred. Implementing a working map was more difficult than expected, and was not completed in this sprint. This task was continued in the next sprint. In this sprint there was estimated 193 work hours, but the actual time used on tasks was 203 hours.

B.8 Sprint 7

Sprint 7 was in week 17 and 18, and the main focus were mobile design and finalize functions

B.8.1 Goals

The goal for this print were to make the application more mobile friendly, perform the rest of the tests that are remaining. The log-in function and the map were also to be finished. The customer was also supposed to test the system and give feedback that could be fixed before delivery of the application. The tasks are shown in Table 28.

Table 28: Sprint 7

ID	Prioritet (1-10)	Task	Time estimated	Actual time
72	2	GUI polish	24 hours	27 hours
73	3	Mobile Design	20 hours	24 hours
74	7	Burndown charts	6 hours	4 hours
75	5	Unit testing	8 hours	5 hours
76	5	System testing	4 hours	3 hours
77	5	Integration testing	4 hours	5 hours
78	1	Working Map	24 hours	27 hours
79	5	Usability testing	4 hours	3 hours
80	3	Finalize Log-in function	8 hours	10 hours
81	4	Report	120 hours	126 hours

B.8.2 Burndown chart

The burndown chart for sprint 7 is shown in Figure 31.

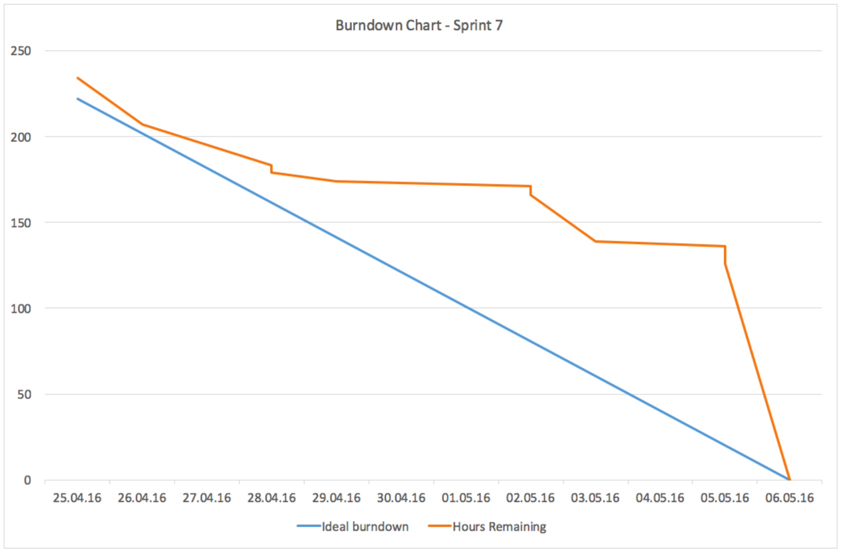


Figure 31: Burndown chart sprint 7

B.8.3 Retrospect and results

During this sprint the customer gave some feedback after some employees at Kommunal Rapport had tested the product. Some fine adjustments and fixing were done. The mobile design were fixed and the rest of the tests were done. The map were also finished and the product were ready to be delivered to the customer. There were estimated 222 hours in sprint 7, and the actual time were 126 hours.

B.9 Project finalization

This phase of the project is not categorized as an own iteration. The project finalization lasted from 09.05.2016 to 22.05.2016, and the main focus was to finish and deliver the product to the customer. The mobile design was approved, the test results were finished and there was made a page on the website called "About". This page is an explanation of the property database and the website. In cooperation with the customer it was decided to stop the coding at Friday 6th of May.

B.10 Report finalization

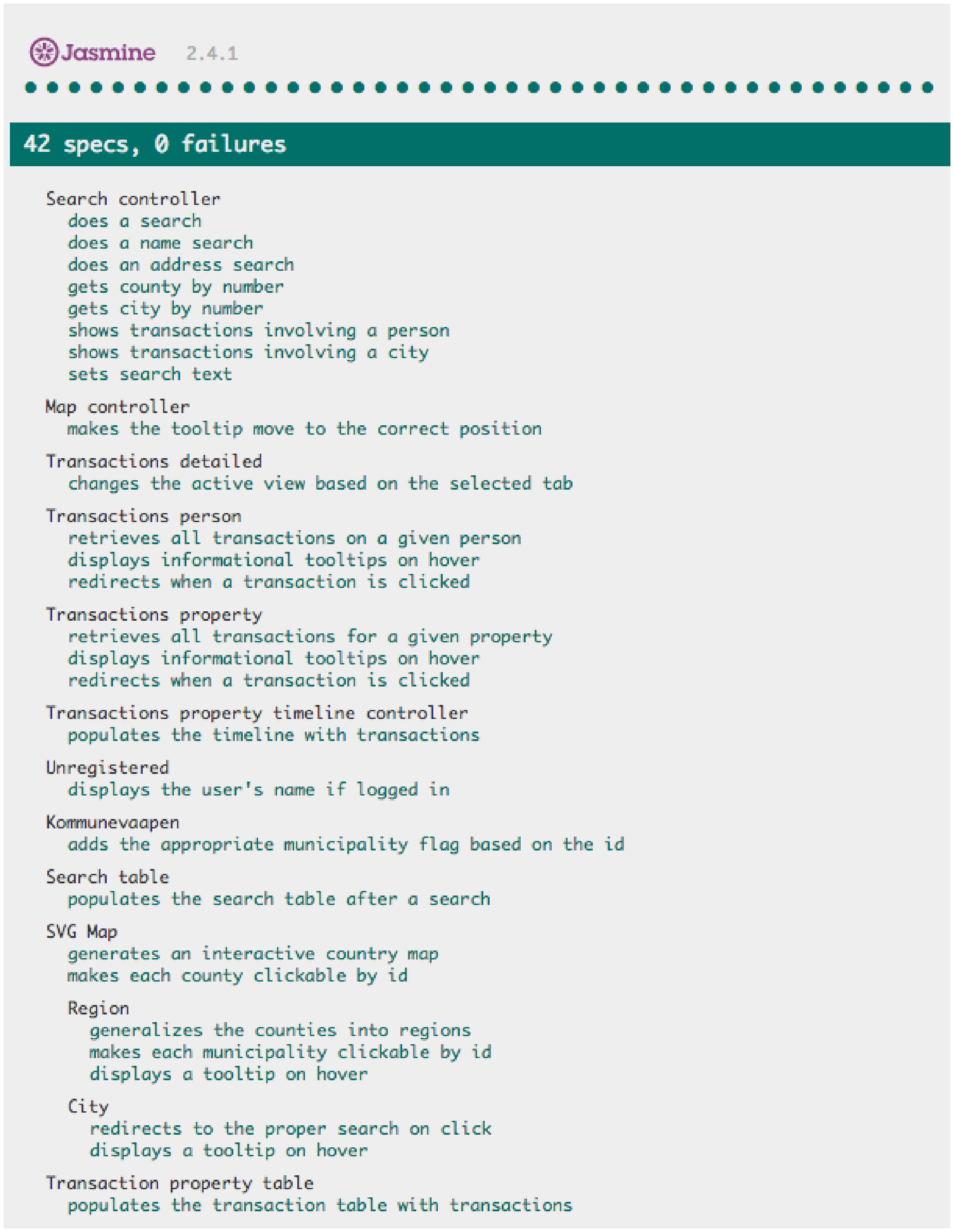
The report finalization lasted from 23.05.2016 to 30.05.2016. Neither this phase is categorized as a own sprint, because the whole team was focused on finishing the report. The team members worked more on their own this period, and the scheduled work hours were not used. The main reason for this was that all the team members had other exams in this period, and the ability to be flexible was necessary. Even though the scheduled work hours was not used, the team scheduled three afternoons from 16:00 to 20:00 to go through, discuss and work together on the report.

C Test Results

This section contains some of the test results regarding the product.

C.1 Unit testing

Figure 32 shows the reslt for the unit testing that were done.



```
Jasmine 2.4.1

42 specs, 0 failures

Search controller
  does a search
  does a name search
  does an address search
  gets county by number
  gets city by number
  shows transactions involving a person
  shows transactions involving a city
  sets search text
Map controller
  makes the tooltip move to the correct position
Transactions detailed
  changes the active view based on the selected tab
Transactions person
  retrieves all transactions on a given person
  displays informational tooltips on hover
  redirects when a transaction is clicked
Transactions property
  retrieves all transactions for a given property
  displays informational tooltips on hover
  redirects when a transaction is clicked
Transactions property timeline controller
  populates the timeline with transactions
Unregistered
  displays the user's name if logged in
Kommunevaapen
  adds the appropriate municipality flag based on the id
Search table
  populates the search table after a search
SVG Map
  generates an interactive country map
  makes each county clickable by id
  Region
    generalizes the counties into regions
    makes each municipality clickable by id
    displays a tooltip on hover
  City
    redirects to the proper search on click
    displays a tooltip on hover
Transaction property table
  populates the transaction table with transactions
```

Figure 32: Screenshot of the unit test results

D User manual

This section will present how to use the product that was created in this project. The product can be found on http://bachelor.kasperrt.no/

D.1 Log-in

To log in for using the property database, click the log-in button in the upper right corner (Figure 33).

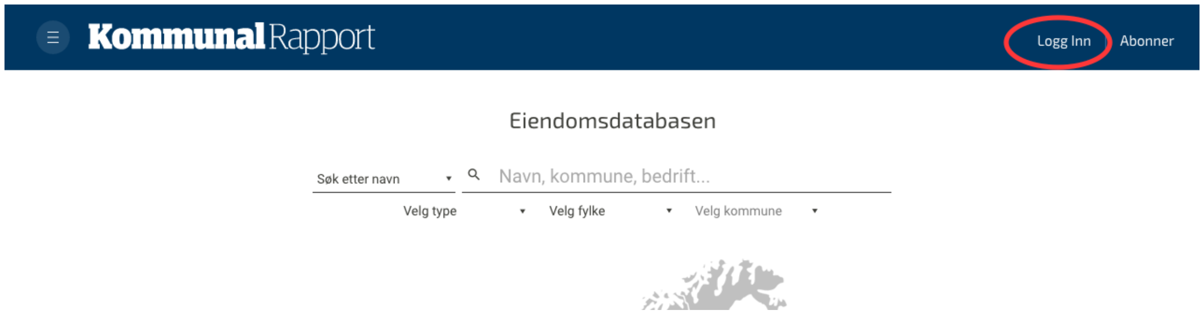


Figure 33: Log-in button

Type in "kasper@kasperrt.no" and the password is "test".

D.2 Search by name or address

At the front-page of the property database one can choose between name of a type (person, organization, county, or municipality), or an address for a property. The dropdown is placed to the left for the search field (Figure 34).

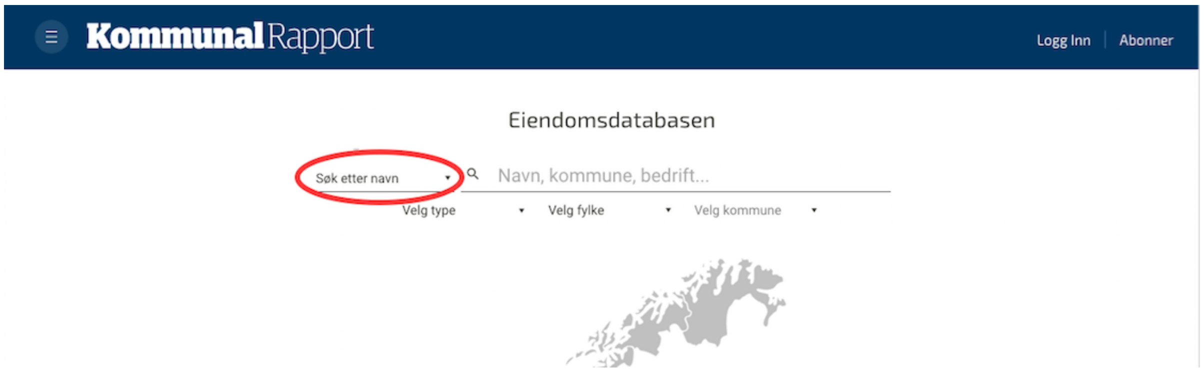


Figure 34: Choose name or address

When the dropdown is selected, one can choose to search by name or the address (Figure 35).

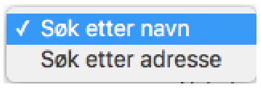


Figure 35: Dropdown for name or address

D.3 Select type, county and municipality

Under the search field there exist three dropdown menus, shown in figure 36.

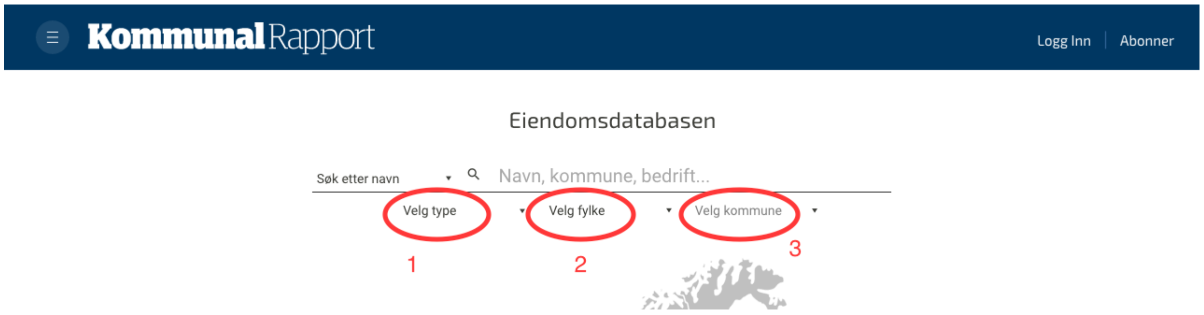


Figure 36: Select type, county and municipality

The first circle, marked with (1), is the dropdown where one can choose between "Person", "County", "Undefined", and "Organization". The second dropdown (2) contains all the counties in Norway. When chosen the county, the third dropdown (3) becomes available and shows all the municipalities in the given county.

D.4 Select county and municipality by map

To select county, one can choose a specific county by the interactive map on the front-page (Figure 37).

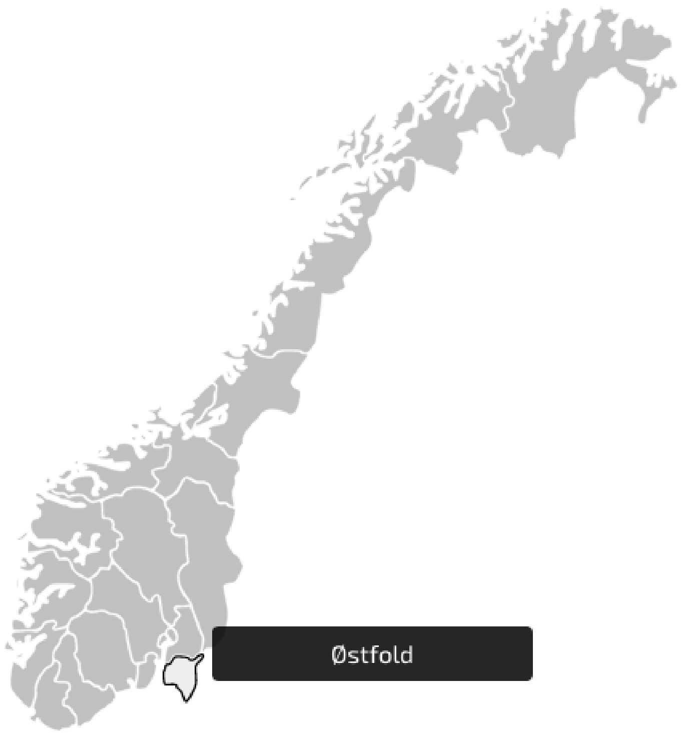


Figure 37: Map of Norway

After choosing a county, a map of the giving county will be shown (Figure 38) and one can choose a municipality.

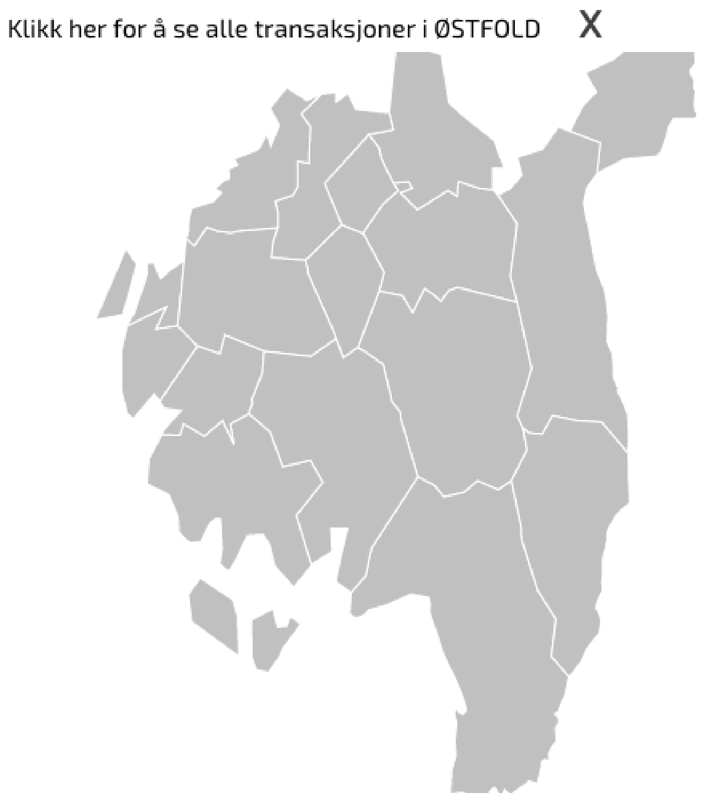


Figure 38: Map over chosen county

If it is desired to see the result table for the given municipality, one can click on "Klikk her for å se alle transaksjoner i ØSTFOLD" at the top of the page.

E Eiendomsdatabase Installation Guide

This installation guide assumes the reader has basic knowledge on how to install and configure programs on a Linux machine, and have root privileges. The system has been verified to work with Apache2 [7], PHP7.0 [24], MySQL5.5 [22]. Prerequisites are composer [11].

E.1 Step-by-step guide

1. Install Apache2, and make sure it is running.
   (a) Enable the module rewrite:
   ```
   $ sudo a2enmod rewrite
   ```
   (b) Restart apache2:
   ```
   $ sudo service apache2 restart
   ```
2. Install PHP7.0 with all the required extensions:
   ```
   $ sudo apt-get install php7.0 php7.0-cli php7.0-fpm php7.0-gd php7.0-json php7.0-mysql
   php7.0-readline php-curl
   ```
3. Install MySQL5.5:
   ```
   $ sudo apt-get install mysql-server
   ```
4. Import data from sql file:
   ```
   $ mysql -u root -p kommunalrapport < data.sql
   ```
5. Copy contents of zipped folder of project to `/var/www/` (or your apache2 html/www server that serves files)
6. Create an initialization file for PHP configuration and fill in the database password and hostname of the server:
   ```
   $ sudo echo ";Config kommunal rapport db_name = kommunalrapport hostname = HOSTNAME_OF_SERVER
   username = DB_USERNAME password = DB_PASSWORD" >> YOUR_APACHE_SERVER_DOCUMENT_PATH
   ```
   Replace `HOSTNAME_OF_SERVER` with the server hostname, `DB_USERNAME` with MySQL username, `DB_PASSWORD` with MySQL password and `YOUR_APACHE_SERVER_DOCUMENT_PATH` with the location of apache2s location for http-file serving.
7. Install needed components with composer:
   ```
   $ php composer.phar install
   ```

The system should now be installed and ready for use.

F Important Souce Code

This section shows some of the important source code from the project. All of the listings are referenced throughout the report.

```php
<?php

ignore_user_abort(true);
set_time_limit(0);

$post = file_get_contents('php://input');

$json = json_decode($post);

if($json->{"repository"}->{"full_name"} == "odden/bachelor"){
        $git_pull = shell_exec("cd .. && git stash && git pull 2>&1");
        echo "Git Pull: $git_pull".PHP_EOL;

        $git_pull = shell_exec("cd .. && cd test && git stash && git pull 
            && pwd 2>&1");
        echo "Git Pull: $git_pull".PHP_EOL;

} else {

        echo "Stop trying to get me to pull, I won't do it!";

}

?>
```

Listing 1: pull.php

```php
$lifetime=432000;
$cookie_lifetime = time() + $lifetime;
session_set_cookie_params($lifetime, "/", "", false, true);
session_name("__komra");
session_start();

if(isset($_COOKIE['name'])){
    setcookie(session_name(),session_id(),$cookie_lifetime, "/", "", false,
         true);
    setcookie("name", $_COOKIE["name"], $cookie_lifetime, "/");
}
```

Listing 2: login.php; Cookie creation

```php
$provider = new \ChrisHemmings\OAuth2\Client\Provider\Drupal([
  'clientId'          => 'bachelor.dev.id.ramsalt.com',
  'clientSecret'      => '*******',
  'redirectUri'       => 'http://' . $_SERVER["HTTP_HOST"] . '/api/login.
     php',
  'baseUrl'           => 'http://komrap.dev-id.ramsalt.com/',
]);
```

Listing 3: login.php; initial creation parameters

```
        $options = [
                'scope' => ['user_profile']
        ];

        $authorizationUrl = $provider->getAuthorizationUrl($options);

        $_SESSION['oauth2state'] = $provider->getState();

        header('Location: ' . $authorizationUrl);
        exit;
```

Listing 4: login.php; scope and redirecting

```
describe('Capitalize first letters filter', function() {
    'use strict';

    var $filter;

    beforeEach(module('kommunalApp'));

    beforeEach(inject(function (_$filter_) {
            $filter = _$filter_;
    }));

    it('capitalizes all words in an input string', function() {
        var capitalized = $filter('capitalFirstLettersFilter');
        expect(capitalized("word and word")).toEqual("Word And Word");
    })
});
```

Listing 5: Example of a Jasmine test

G Figures

This section contains important figures for this project, and the class diagram will be presented.

G.1 Class Diagram

The class diagram was an essential part of the product. It shown of the different classes are connected with each other.

G.1.1 Back-end

Figure 39 shows the class diagram for the back-end.

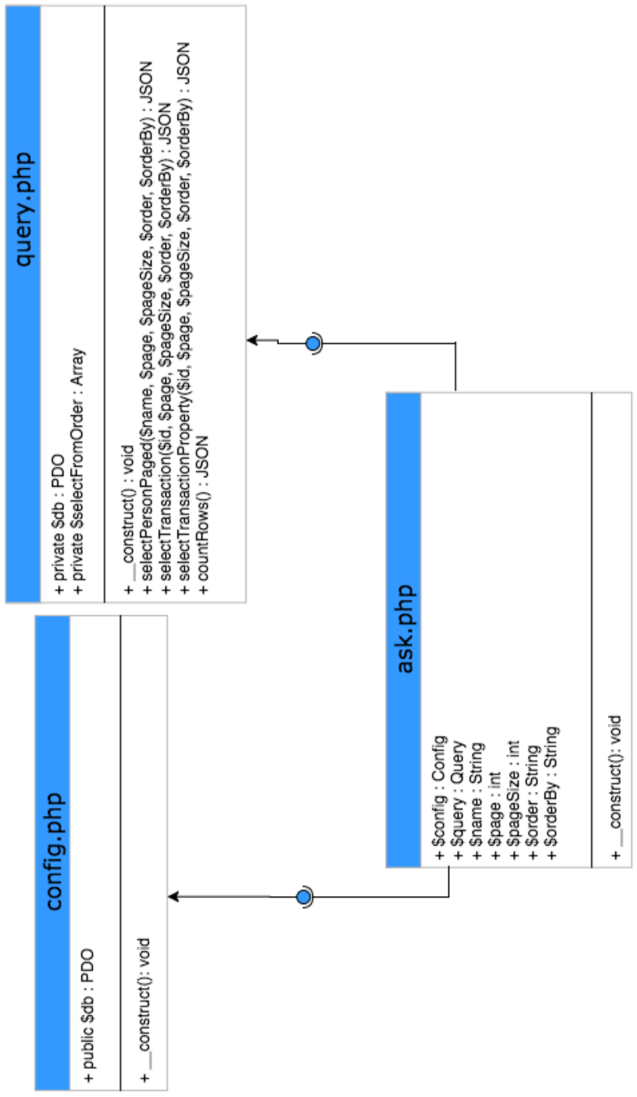


Figure 39: Class Diagram

G.1.2 Front-end

Figure 40 shows the class diagram for the front-end.

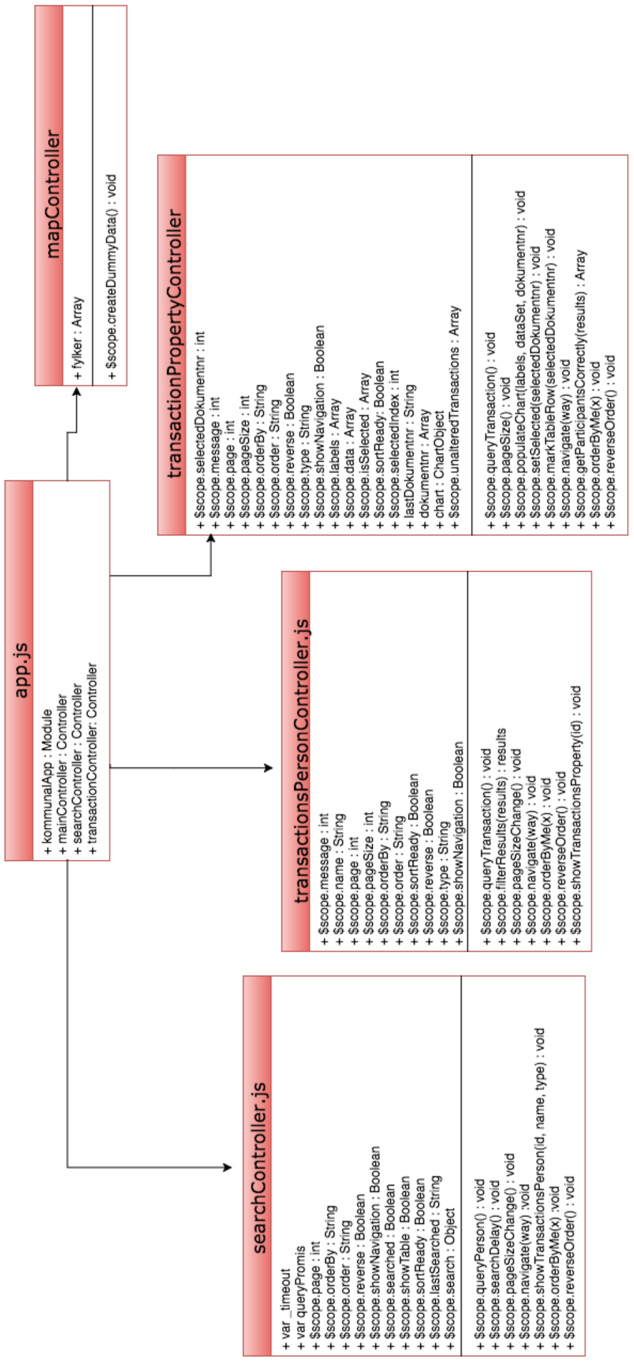
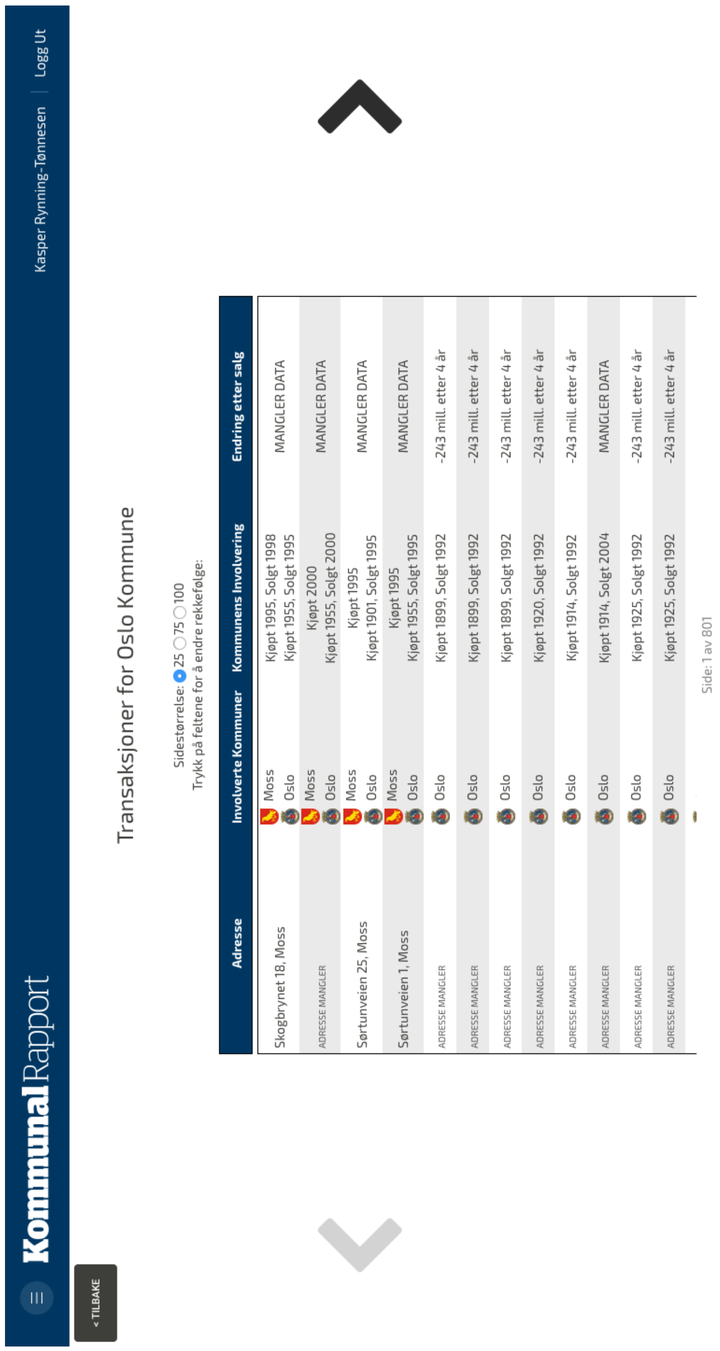

mapController
+ fylker : Array
+ $scope.createDummyData() : void
app.js
+ kommunalApp : Module
+ mainController : Controller
+ searchController : Controller
+ transactionController: Controller
transactionPropertyController
+ $scope.selectedDokumentnr : int
+ $scope.message : int
+ $scope.page : int
+ $scope.pageSize : int
+ $scope.orderBy : String
+ $scope.order : String
+ $scope.reverse : Boolean
+ $scope.type : String
+ $scope.showNavigation : Boolean
+ $scope.labels : Array
+ $scope.data : Array
+ $scope.isSelected : Array
+ $scope.sortReady: Boolean
+ $scope.selectedIndex : int
+ lastDokumentnr : String
+ dokumentnr : Array
+ chart : ChartObject
+ $scope.unalteredTransactions : Array
+ $scope.queryTransaction() : void
+ $scope.pageSize() : void
+ $scope.populateChart(labels, dataSet, dokumentnr) : void
+ $scope.setSelected(selectedDokumentnr) : void
+ $scope.markTableRow(selectedDokumentnr) : void
+ $scope.navigate(way) : void
+ $scope.getParticipantsCorrectly(results) : Array
+ $scope.orderByMe(x) : void
+ $scope.reverseOrder() : void
transactionsPersonController.js
+ $scope.message : int
+ $scope.name : String
+ $scope.page : int
+ $scope.pageSize : int
+ $scope.orderBy : String
+ $scope.order : String
+ $scope.sortReady : Boolean
+ $scope.reverse : Boolean
+ $scope.type : String
+ $scope.showNavigation : Boolean
+ $scope.queryTransaction() : void
+ $scope.filterResults(results) : results
+ $scope.pageSizeChange() : void
+ $scope.navigate(way) : void
+ $scope.orderByMe(x) : void
+ $scope.reverseOrder() : void
+ $scope.showTransactionsProperty(id) : void
searchController.js
+ var _timeout
+ var queryPromis
+ $scope.page : int
+ $scope.orderBy : String
+ $scope.order : String
+ $scope.reverse : Boolean
+ $scope.showNavigation : Boolean
+ $scope.searched : Boolean
+ $scope.showTable : Boolean
+ $scope.sortReady : Boolean
+ $scope.lastSearched : String
+ $scope.search : Object
+ $scope.queryPerson() : void
+ $scope.searchDelay() : void
+ $scope.pageSizeChange() : void
+ $scope.navigate(way) :void
+ $scope.showTransactionsPerson(id, name, type) : void
+ $scope.orderByMe(x) :void
+ $scope.reverseOrder() : void

Figure 40: Class Diagram

G.2 Screenshots

This section contains the screenshots of the product.

G.2.1 Transaction-page



Transaksjoner for Oslo Kommune

Sidestørrelse: ◉ 25 ○ 75 ○ 100

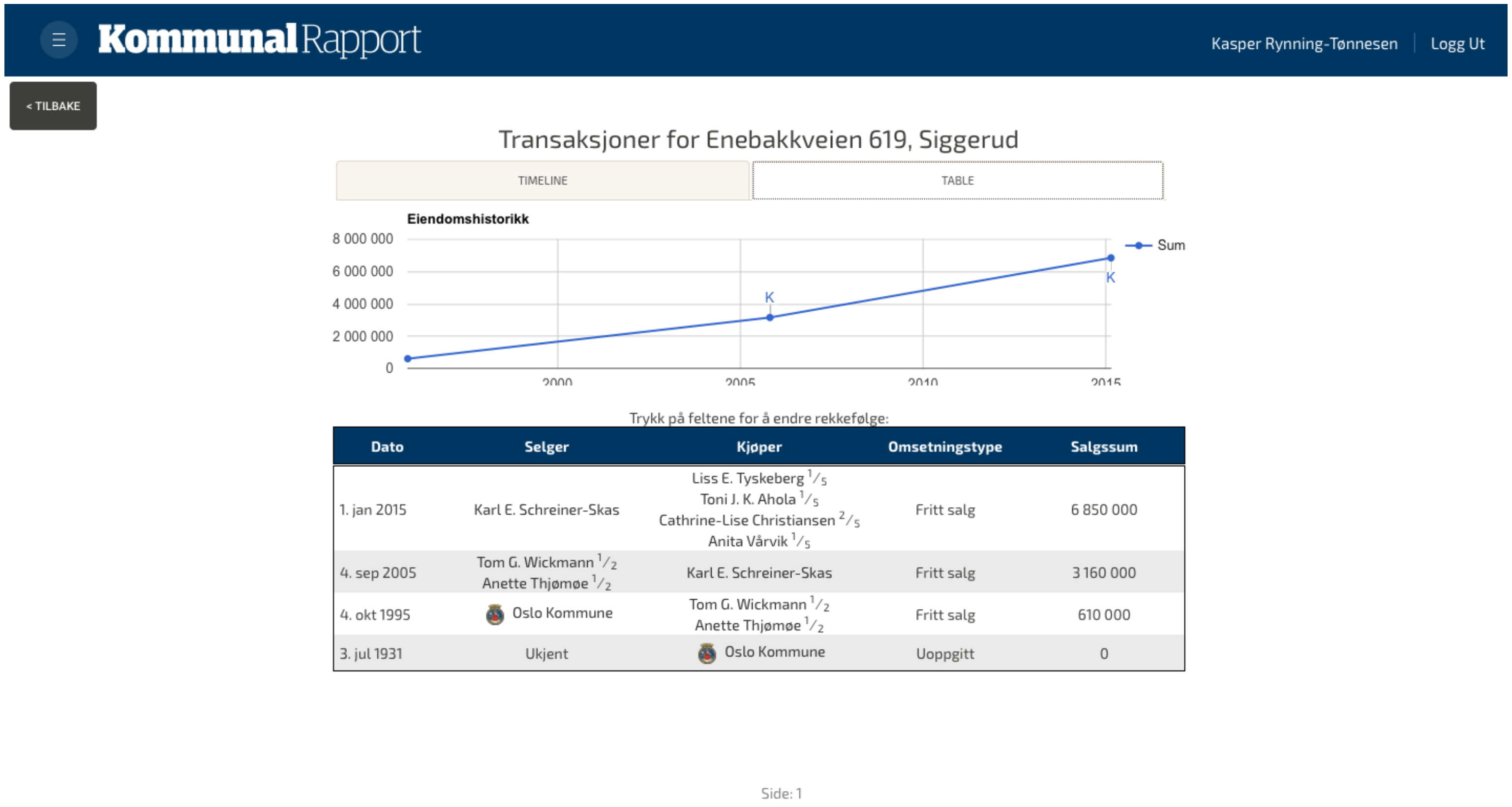
Trykk på feltene for å endre rekkefølge:

Adresse	Involverte Kommuner	Kommunens Involvering	Endring etter salg
Skogbrynet 18, Moss	Moss Oslo	Kjøpt 1995, Solgt 1998 Kjøpt 1955, Solgt 1995	MANGLER DATA
ADRESSE MANGLER	Moss Oslo	Kjøpt 2000 Kjøpt 1955, Solgt 2000	MANGLER DATA
Sørtunveien 25, Moss	Moss Oslo	Kjøpt 1995 Kjøpt 1901, Solgt 1995	MANGLER DATA
Sørtunveien 1, Moss	Moss Oslo	Kjøpt 1995 Kjøpt 1955, Solgt 1995	MANGLER DATA
ADRESSE MANGLER	Oslo	Kjøpt 1899, Solgt 1992	-243 mill. etter 4 år
ADRESSE MANGLER	Oslo	Kjøpt 1899, Solgt 1992	-243 mill. etter 4 år
ADRESSE MANGLER	Oslo	Kjøpt 1899, Solgt 1992	-243 mill. etter 4 år
ADRESSE MANGLER	Oslo	Kjøpt 1920, Solgt 1992	-243 mill. etter 4 år
ADRESSE MANGLER	Oslo	Kjøpt 1914, Solgt 1992	-243 mill. etter 4 år
ADRESSE MANGLER	Oslo	Kjøpt 1914, Solgt 2004	MANGLER DATA
ADRESSE MANGLER	Oslo	Kjøpt 1925, Solgt 1992	-243 mill. etter 4 år
ADRESSE MANGLER	Oslo	Kjøpt 1925, Solgt 1992	-243 mill. etter 4 år

Side: 1 av 801

Figure 41: Transactions-page Screenshot

G.2.2 Transaction Chart Page



Transaksjoner for Enebakkveien 619, Siggerud

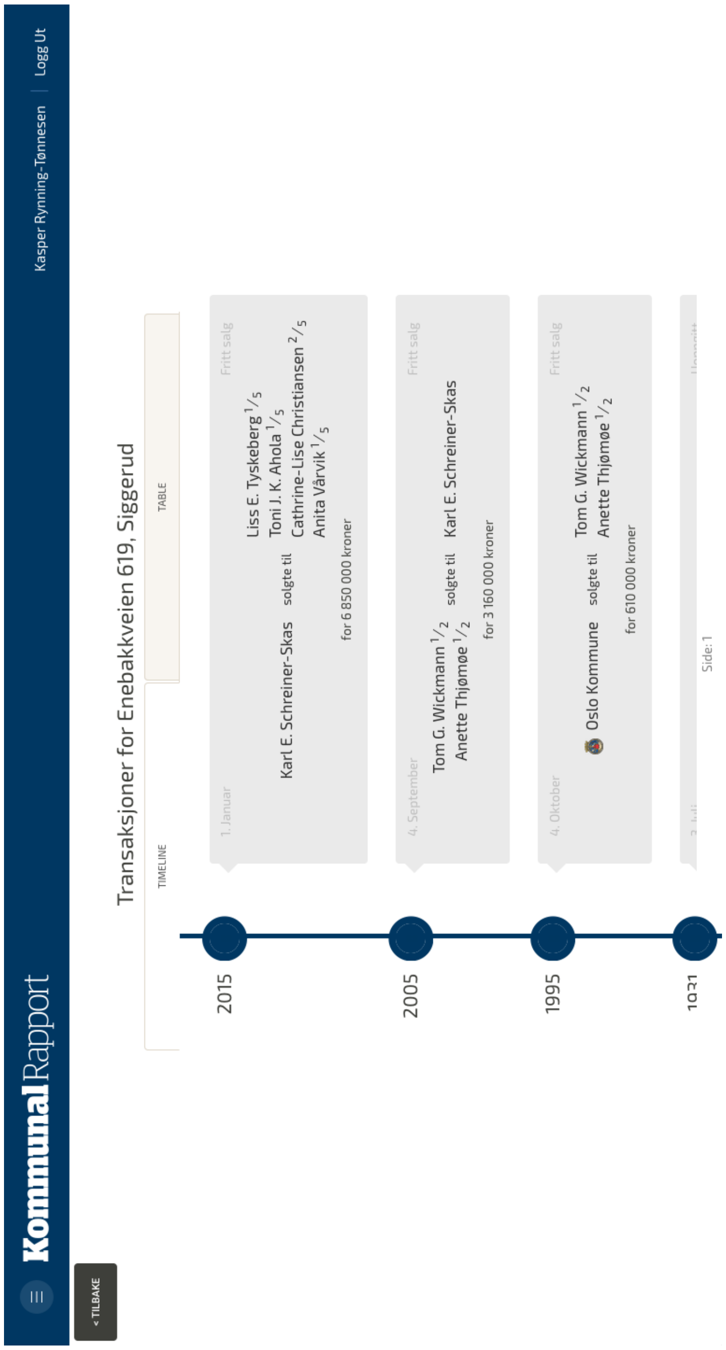
Trykk på feltene for å endre rekkefølge:

Dato	Selger	Kjøper	Omsetningstype	Salgssum
1. jan 2015	Karl E. Schreiner-Skas	Liss E. Tyskeberg $^1/_5$ Toni J. K. Ahola $^1/_5$ Cathrine-Lise Christiansen $^2/_5$ Anita Vårvik $^1/_5$	Fritt salg	6 850 000
4. sep 2005	Tom G. Wickmann $^1/_2$ Anette Thjømøe $^1/_2$	Karl E. Schreiner-Skas	Fritt salg	3 160 000
4. okt 1995	Oslo Kommune	Tom G. Wickmann $^1/_2$ Anette Thjømøe $^1/_2$	Fritt salg	610 000
3. jul 1931	Ukjent	Oslo Kommune	Uoppgitt	0

Side: 1

Figure 42: Transaction Chart Screenshot

G.2.3 Transaction Timeline Page



Transaksjoner for Enebakkveien 619, Siggerud

2015 — 1. Januar — Fritt salg
Karl E. Schreiner-Skas solgte til Liss E. Tyskeberg $^1/_5$, Toni J. K. Ahola $^1/_5$, Cathrine-Lise Christiansen $^2/_5$, Anita Vårvik $^1/_5$
for 6 850 000 kroner

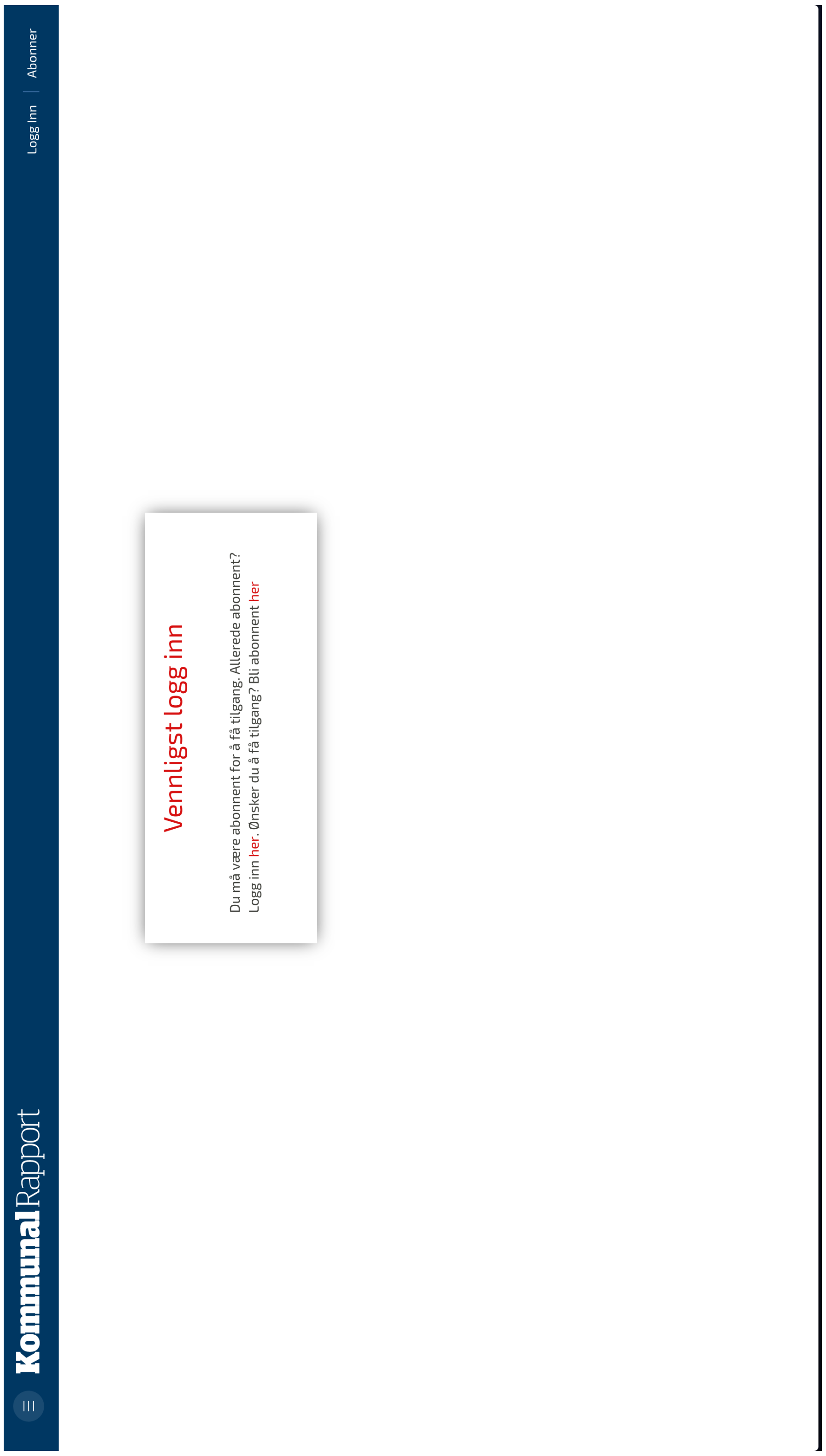
2005 — 4. September — Fritt salg
Tom G. Wickmann $^1/_2$, Anette Thjømøe $^1/_2$ solgte til Karl E. Schreiner-Skas
for 3 160 000 kroner

1995 — 4. Oktober — Fritt salg
Oslo Kommune solgte til Tom G. Wickmann $^1/_2$, Anette Thjømøe $^1/_2$
for 610 000 kroner

Side: 1

Figure 43: Transaction Timeline Screenshot

G.2.4 Error-page



Vennligst logg inn

Du må være abonnent for å få tilgang. Allerede abonnent? Logg inn her. Ønsker du å få tilgang? Bli abonnent her

Figure 44: Error-page Screenshot

H Database Tables

This section contains the different database tables that were created from the csv-file.

Table 29: Document Table

Table Header	Description
InterntDokumentnr	Unique ID identifying the different documents
Dokumentnr	Number that is applied to each document entered in “grunnboken” in accordance with “Tinglysingsforskriften § 38”
Embetenr	The id of the office responsible of registering the document the standard is 200 which is the id of “Statens Kartverk”
Dokumentdato	The date in which the document was registered
Rettsstiftelsesnr	Generated numbering of “rettsstiftelse” within each document
Rettsstiftelsestype	A code for conditions causing certain legal effects listed in “grunnboken”

Table 30: Property Table

Table Header	Description
Eiendomsid	Unique ID for each property generated by the dbms
Kommunenr	The municipality-number of the municipality in which the property is situated
Gaardsnr	The number of the partition within the municipality in which the the property is situated
Bruksnr	The number of the partition “bruk” within the Gaardsnr in which the the property is situated
Festenr	The number of the partition “feste” within Gaardsnr and/or Bruksnr in which the the property is situated
Seksjonsnr	The number of the partition “feste” within Gaardsnr and/or Bruksnr and/or Festenr in which the the property is situated
AntallTeiger	The number of subsections within the property
NordSor	North-South coordinates
OstVest	East-West coordinates
AdresseKommunenr	The number of the municipality which is part of the address of the property
Gatenr	Streetnumber part of the property address
Gatenavn	Streetname part of the property address
Husnr	Property number part of the property address
Bokstav	Property letter part of the property address
Postnr	Property post number part of the property address
Poststed	Post place number part of the property address

Table 31: PropertyInvolvedMunicipality Relation

Table Header	Description
Eiendomsid	References "Eiendomsid" in "Eiendommer"
Kommunenr	References Kommunenr in Kommuner indicating that this municipality at one point has been involved with this property.

Table 32: Property-history Table

Table Header	Description
Eiendomsid	References Eiendomsid in Eiendommer
ForstRegistrert	Documentdate for the first time this property was registered
SistRegistrert	Documentdate for the last time this property was registered
AntallTransaksjoner	Number of transactions sets. Times this property or partitions of it has changed owner
Historie	Summary of the transaction history for this property on the format Price : Date : Internal Document number : Part type (Buyer / Seller) : Participant type : Municipality number (Zero if this is not a municipality). The actual format based on column names are: Salgssum: Dokumentdato: InterntDokumentnr: PartType: DeltagerType: Kommune.

Table 33: Transaction Table

Table Header	Description
Id	Unique ID for each transaction generated by the dbms
Eiendomsid	References Eiendomsid in Eiendommer
Personid	References Personid in Personer
InterntDokumentnr	References InterntDokumentnr in Dokumenter
Omsetningstypekode	Code indicating what kind of transaction this is
Salgssum	Sales price can be zero as this is not an obligatory field
kjop_salg	Describes if the transaction is a purchase or sale
AndelTeller	Numerator of the fraction describing the fraction of the sale/purchase
AndelNevner	Denominator of the fraction describing the fraction of the sale/purchase
Eiendomsnivaa	Describes the ownership role of the associated agent

Table 34: Participants Table

Table Header	Description
Deltagerid	Unique ID assigned to every agent
AnonymisertPersonnr	Part of a person's social security number it is anonymous and not unique
Deltagertype	Describes the type of agent whether it is a person organization or if it is unassigned
Navn	Name of the agent

Table 34: Participants Table

Table Header	Description
Kommune	Specifies the municipality number where a participant is a municipality or county number where the participant is a county administration

Table 35: "Rettstiftelsestypekode" Table

Table Header	Description
Kode	Codes for “rettstiftelser"
Beskrivelse	Description of the different types of “Rettstiftelser”, given by transaction documents

Table 36: ParticipantInvolvedMunicipality Relation

Table Header	Description
Deltagerid	Refers to the participant number in the table “Deltagere"
Kommunenr	Refers to the municipality number in the table “Kommuner”

Table 37: Transaction-type Table

Table Header	Description
Kode	Code for the type of transaction listed in the dataset
Navn	Name of the type of transaction

Table 38: County Table

Table Header	Description
Fylkenr	County number of the current county
Fylkenavn	County name

Table 39: CountyToMunicipality Table

Table Header	Description
Kommunenr	Municipality number of the current municipality
Fylkenr	Refers to the county number the municipality is part of
Kommunenavn	Name of the municipality