Route Administration System for Aalborg Climbing Club

Developing Applications – From Users to Data, Algorithms, and Tests – and Back Again

Group:
DS315e16
ds315e16@cs.aau.dk

Supervisor:
Jane Billestrup

December 20, 2016

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Aalborg University
http://cs.aau.dk

Title:
Route Administration System for Aalborg Climbing Club

Theme:
Developing Applications – From Users to Data, Algorithms, and Tests – and Back Again

Project Period:
Fall 2016

Group:
DS315e16
ds315e16@cs.aau.dk

Participants:
Morten Rask Andersen
Anton Christensen
Christian Mønsted Grünberg
Mathias Ibsen
Mathias Steen Jakobsen
Jacob Svenningsen
Henrik Herbst Sørensen

Supervisor:
Jane Billestrup

Pages:
136

Date of Completion:
December 20, 2016

Number of Copies:
0

Abstract:

This report is about developing a route administration system for Aalborg Climbing Club. By using OOA&D techniques the problem is analysed.
Requirements for the design and scope of the system is defined through user cooperation by performing interviews.
From these requirements follows the problem statement: *How can we develop a mobile system that allows multiple members to easily administrate climbing routes including associated grades, sections, images, betas, ratings and comments?*
The system developed is a web application designed primarily for smartphones, such that the current manual whiteboard system can be moved to the user's pockets.
By working iteratively, the system has gradually been built up from the most important must-have requirements, to the less important should-have and could-have requirements.
Through user evaluations, usability tests, and unit tests, the system has been tested to ensure that the users can use the system, and ensure that the new system fulfils the established requirements.
Finally, it is concluded that the system solves the problem statement. The will-not-have requirements are discussed as possible features that can be included in future development.

Preface

This report is the product of a third semester project by the group ds315e16 consisting of seven Software and Computer Science students attending Aalborg University. The project started in September 2016 and ended in December 2016. The theme of the report is the digitisation of an administration system for Aalborg Climbing Club. The system developed in this project is accessed through a web-application, which can be found at the following URL: `http://ds315e16.cs.aau.dk/`. An account with administrative privileges:

Username: admin
Password: 1234

We would like to thank Aalborg Climbing Club for allowing us to test the system on several of their members, and especially Mattias Hornum for his cooperation with us during the project. We would also like to thank our supervisor, Jane Billestrup, for her advice and guidance during the project.

Reading Guide

The report is the product of several iterative work cycles. Therefore, all chapters in the report have not been written chronologically, but should be read as such. All sections have a short description of the content and purpose of the section. Likewise, most sections end with a short summary. It is expected that the reader possesses a general knowledge of the subjects of object-oriented programming, systems development, object-oriented analysis and design, and design and evaluation of user interfaces.

In the bibliography the following notation for referencing books and articles are used: author initial, author surname, year published, URL/ISBN. To cite these books and articles an author-year citation style is used.

In this report, we conducted an interview. To reference this interview we use the following notation to, for example, reference line 182 of the interview transcription: (Hornum, 2016, L. 182). The full transcription can be found in appendix B.

Unless otherwise stated, the illustrations and figures used in this report have been made by the group ds315e16. In some of the code listings shown in the report, we have only included the essential parts. Removed sections of the code have been replaced by an ellipsis [. . .].

Classes mentioned throughout the report are formatted as `Route`, and when methods or functions are mentioned, they are be formatted as `AddRoute`. The formatting does not differentiate between OOA&D classes and C# classes.

Contents

Chapter 1

Introduction

Today, climbing is a popular sport; in the last decade, the number of members in the International Foundation of Sport Climbing (IFSC) has increased by 25 percent (IFSC, 2016). In Denmark alone, the number of members has increased from 2263 in 2005 to 6942 in 2015, which translates to an increase of approximately 207 percent in a decade (Fester, 2016). This increase in popularity is also reflected by the fact that climbing has been included in the official proposal for the Tokyo Olympics in 2020 (Kunio, 2015).

There exist many different types of climbing such as mountaineering, rock climbing, sports climbing, and indoor climbing. Typically for indoor climbing, the climber uses a specific set of holds to climb a wall. One such set of holds is called a route.

In this project, a route administration system for indoor climbing in Aalborg Klatreklub (AKK) was developed. AKK is a Danish climbing club with 269 members located just outside of Aalborg (Aalborg klatreklub, 2016a). The AKK building is split into two halls: one with rope climbing routes and one with bouldering routes. These two types of climbing, are described in depth in section 3.2. The hall with bouldering routes consists of two floors. Routes in the bouldering hall are regularly added and removed, so to get an overview of the routes, a system of whiteboards has been used in the club. This system has users write the routes they create on different whiteboards depending on the difficulty of the route, also known as the grade.

According to a member of AKK a restriction of their current system is that it was *"a bit difficult to get an overview, especially if you have to go upstairs."* (Hornum, 2016, L.301-302). The users of the system had to go back and forth between the whiteboards and the routes, which is especially bothersome when they are climbing the routes on the second floor, since the current whiteboards are located on the first floor. Another problem we observed while visiting the club was limited space on the whiteboards, resulting in pieces of paper with route information having to hang around the whiteboards. This project aims to help AKK solve the problems with their current system.

In this report, we begin by describing the methodology used in the project in chapter 2. Thereafter, we further analyse the problem and define the system to be developed in chapter 3. In chapter 4, we start to model how the problem domain should be represented in the system, which is followed by an analysis of the users and different use cases of the system in chapter 5. In chapter 6, we create a prioritised list of selected design criteria and describe the technical platform used in the project followed by a description of the system architecture and the underlying components. In chapter 7 the system user interface is explained and presented. We discuss how a presentation prototype was used in the project to get feedback from the client on design ideas. In chapter 8, the implementation of the system is documented with corresponding code snippets. In chapter 9, we explain how we tested essential parts of the system, and in chapter 10 we evaluate the system with the users. The report is then ends with an discussion, conclusion and an assessment of ideas for future development, as described in chapters 11, 12, and 13 respectively.

Chapter 2

Methodology

In this chapter the overall methods for software development and data-collection used throughout the project are described. To do so, the five activities of the Object Oriented Analysis and Design (OOA&D) method are specified. The advantages and disadvantages of two software development methods are discussed. This knowledge is used to choose which one to use, for developing the system and cooperating with users. Lastly, three different interview types are described: structured, semi-structured and unstructured.

2.1 Software Development Methods

In this section, the choice of the overall methods in this project is described. We describe the object-oriented analysis and design (OOA&D) method and how activities within this approach are structured in the traditional and the iterative approach. Finally, we derive the overall method used for this project by looking at the advantages and disadvantages of the two approaches and describe how we include different activities for cooperating with future users of the system.

2.1.1 Software Development Activities

The overall activities in this project are based on the OOA&D method as described by (Mathiassen et al., 2000). The OOA&D method comes with tools and techniques for developing systems in the object-oriented paradigm. In this section, we briefly describe the activities of OOA&D: system choice, problem-domain analysis, application-domain analysis, architectural design, and component design.

System Choice

The purpose of the system choice activity is to determine the overall properties of the system that is going to be developed. This is done by describing the future system in a system definition, which contains a clear description of the system's functionality, application domain, conditions, technology, objects, and responsibilities, also known as the FACTOR criteria (Mathiassen et al., 2000, p. 39-41).

Problem-domain Analysis

Mathiassen et al. (2000) describes the problem domain as the part of the environment, which is administered, surveyed, or controlled by the system. In the problem-domain analysis, the purpose is to both define and describe the problem domain (Mathiassen et al., 2000, p.6). To do this, different types of interviews can be used, as explained in section 2.2. After gaining an initial understanding of the problem domain, we can begin to find and select classes, objects, structures, and events in the problem domain by creating class diagrams and event tables. The end-result of the problem-domain analysis is a model representing the problem domain (Mathiassen et al., 2000, p.45).

Application-domain Analysis

The application domain can be described as the organisation that administrates, monitors or controls the problem domain (Mathiassen et al., 2000, p.6). In the application-domain analysis, the purpose is to analyse and find requirements for the use of the system (Mathiassen et al., 2000, p.115). In this activity, we must first determine different use cases and the types of users that are going to follow these use cases. A use case is a pattern that describe interactions between the system and the users (Mathiassen et al., 2000, p.120). From these use patterns a list of functions for the system can be derived.

Architectural Design

In the architectural design activity, the purpose is to structure the system (Mathiassen et al., 2000, p.173). We start by defining different system criteria and prioritise them based on how important they are for the system. Examples of system criteria could be portability or security (Mathiassen et al., 2000, p.178). The second part of architectural design is to define the underlying system architecture. We do this by specifying which components the system is divided into and how they interact. An example of a system with a simple architecture is a system that consists of three main components: interface, functions, and model.

Component Design

In the component design activity, we use the specifications of the architecture to design components, create component specifications, and in greater detail, design how the components are connected to each other. To do this for the simple architecture, we can look at the model, function and interface components. The result of the model component is a revised version of the class diagram created in the problem-domain analysis. In the function component, we specify components containing all the functions derived in the application-domain analysis. Furthermore, we design our functions like operations, explore patterns and specify complex operations (Mathiassen et al., 2000, p.251-270). Lastly, the result is a component specification that describes each component and its responsibility.

2.1.2 The Traditional Approach

Activities in the OOA&D method can be structured differently. The first of two software development approaches we describe is the traditional approach. The traditional approach describes the process of software development as a series of steps where each step must be completed before moving on to the next (Mathiassen et al., 2000). This means that the developer must obtain a clear understanding of both the problem domain and application domain before trying to design the system. This method only allows the developer to move forward, which means that once an understanding of the requirements of the system has been obtained, the design process starts and no more requirement gathering can take place.

One of the advantages of this method is that it forces the developers to understand the problem in depth before trying to implement a solution. Because of this, more time is spent on the analysis, and therefore implementing the system is easier, since the way different parts of the system interacts with each other, is already known.

Another advantage is that it structures the activities in a chronological order, which means there is no moving back and forth between activities; This makes it easier to figure out which activity the developer should work on at a given time in the project.

Only moving forward introduces some problems as well, as it can be difficult to adapt to changes in the problem domain, if one cannot go back and adjust the analysis of the problem domain after it has been completed.

Another problem that can arise from only moving forward is that later activities are based on the result of previous activities, which can cause problems if mistakes were made in past activities or if these activities were not entirely completed. This illustrates another problem, which is that it can be difficult to determine whether any given activity is completed or not.

Using the traditional approach, time can become a very limiting factor. It may be easier to plan the entirety of the project in the beginning since the order of activities is set in stone, but it also means that if any of the activities take longer than planned, the time schedule quickly suffers. This may result in later activities being shortened to compensate for lost time.

In the next section, we present an iterative approach as an alternative for developing software, which relieves the developer of the problem that the last activities in a project are more likely to suffer under time pressure than the first.

2.1.3 The Iterative Approach

"An iterative approach [...] allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution over multiple iterations." (Software, 1998). In each iteration knowledge is obtained so the perception of the problem is shifted. This means that each iteration brings the developer closer to the final product.

Usability evaluation can be used in the different iterations of the iterative approach where smaller parts of the system can be tested and evaluated without the final product being finished.

The fact that there are going to be possibilities for getting feedback from clients and users also means that problems that would not have been found through analysis alone can be fixed.

If the project is constrained by a time limit, the iterative approach can be used to implement most important features in early iterations and less important features in later iterations and thereby ensure that there is a higher possibility for the most important features to be implemented before the deadline of the project.

A disadvantage of the iterative approach, is that the need to look at smaller parts of the system in each iteration, means that it is necessary to determine which parts the system should be split into. This is more of a disadvantage if there are few iterations, as the division of parts is less flexible than if there are e.g. five more iterations in which the division can be fine-tuned further.

Another disadvantage is, that it can be difficult to keep focus when jumping back and forth between different activities.

In the next section the final method for this project is decided by comparing and selecting how the activities in OOA&D are going to be included, which approach is used to structure these activities and finally, how other activities for user cooperation are included.

2.1.4 Deciding on a System Development Method

The purpose of this section is to decide on a method for system development used throughout this project, while also discussing why we made the choice we did. Furthermore, each activity in the method, which tools we can use to incorporate our clients in our work is illustrated.

In section 2.1.2 and 2.1.3 the advantages and disadvantages of the traditional approach and the iterative approach were described. These advantages and disadvantages of both development approaches are summarised in table 2.1, which can help decide which approach to use.

Approach	Advantages	Disadvantages
Traditional	• Understanding the entire problem makes programming easier • Knowing all connections between components before programming	• Difficult to adapt to changes in the problem domain • Later activities depend on previous activities • Time constraints negatively affect important parts of the system, such as quality insurance and implementation
Iterative	• Enables usability evaluation on smaller parts of the system • If constrained by time the developers can choose to start develop the most important parts of the system	• Difficult to keep focus if going back and forth between different activities • Hard to focus on smaller parts of the system without first getting an overview of the entire system

Table 2.1: Advantages and disadvantages of the traditional and iterative approaches

To decide on a method for this project we look at the advantages and disadvantages and compare them to some of the conditions and constraints we have in this project.

We are limited by time, in that the project has to be turned in on the 21st of December 2016. Additionally there is a demand for our system to be developed in cooperation with future users, and systematically tested to show that it solves the client's problem.

Limited Time

If we intend to use the traditional approach, then there is a risk of spending too much time on the analysis and design activities in which case we do not have enough time to implement and evaluate the entire system, because we are constrained by time. We could therefore, in the worst case, end up with a system that cannot be demonstrated. If we use the iterative approach instead, we can divide the system into smaller individual parts. Because a smaller part of the system does not require as much time to develop as the entire system, it is more likely that we end up with at least some of the parts of the system fully developed and evaluated.

Cooperation with Users

Both the traditional and iterative approach enables user cooperation. The different activities in OOA&D gives us information that can be shown and discussed with users, but some of them are more technical than others. It makes more sense showing the analysis design to the client, rather than to the prospective users. Generally speaking, prototypes and quality assurance are going to be the main points of interaction between the developer and the users of the system. With a traditional approach, these two activities only happen once, and the last one leaves us little room to use the results in the system due to time constraints. With an iterative approach, however, the activities happen once per iteration, which means that it is more suitable for cooperating with the future users, and also use the feedback from them in later iterations.

Systematic Testing and Determining if the Problem has been Solved

To make a systematic test of a system one must first have a system to test, and secondly, the resources to test it and check whether it solves the client's problem or not. This can be difficult with the traditional approach since we only do each activity once and cannot skip to the next activity before finishing the current one. Because of this, we do not know if we have the time or resources left to systematically test and evaluate the system since evaluation is the last activity in the traditional approach. In comparison, if we use the iterative approach, it is more likely that we have a system, or at least a part of a functional system, that can be tested.

Choosing the Methodology for this Project

We have decided to use the iterative approach because it enables us to develop a functional part of the entire system, where we can focus on the most important requirements, before focusing on smaller details. Having a fully functional part of the system allows us to test it and to analyse if it solves the client's problem. By evaluating smaller parts of the system after each iteration, we ensure that we catch usability problems as early as possible, which can then be rectified in the following iteration.

We use the iterative approach in a way, that incorporates principles from the traditional approach, by doing activities in each iteration in chronological order, which helps us maintain an overview of the OOA&D method. Using this method also allows us to develop small parts of the system in each iteration as well as knowing what activity we are at, and what activity that is next.

Figure 2.1 illustrates the method chosen for this project.

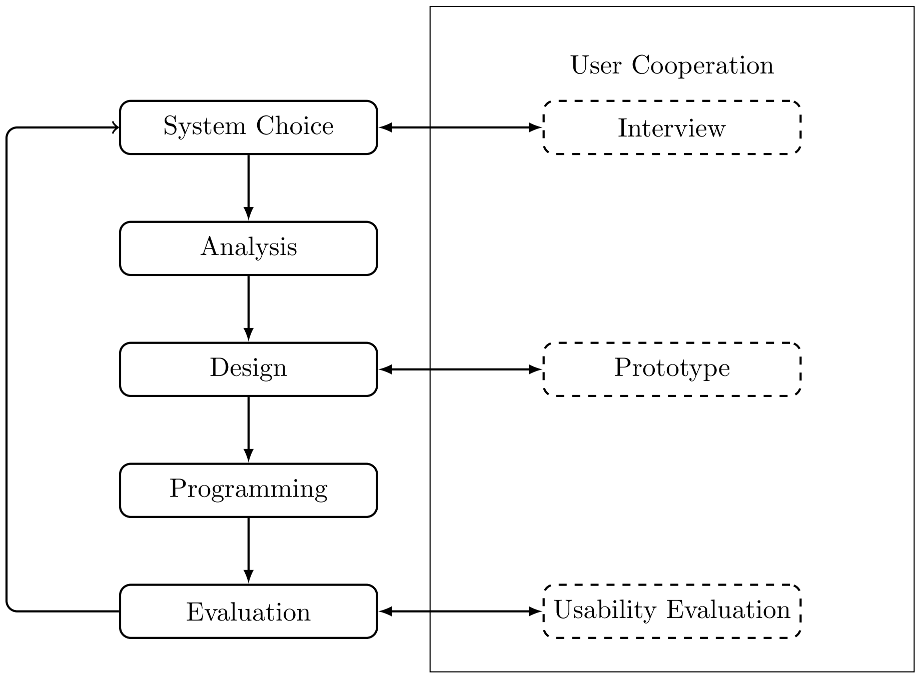


Figure 2.1: Illustrating the method for this project based on the iterative approach of OOA&D combined with activities for user cooperation.

The boxes on the left illustrate the activities we go through in chronological order in each iteration. Here, the system choice covers the initial steps of the system development, where we formulate the system definition. The analysis covers both the problem and application domain, where we identify and model the problem domain, and determine the requirements of the future system in the application domain.

The design activity covers architectural, function and component design, where we structure the system and describe its components. Second to last, we begin programming the system, followed by the last activity, which is an evaluation of the developed system in each iteration, where we conduct usability tests.

The dotted boxes on the right illustrate methods for how we can include users during each iteration to gather information. The activities for user cooperation during the project are described later in the report where we discuss how we have used it.

2.2 Interviewing

Interviewing is our main method for gathering information from our client and future users during the project. Interviewing is specifically used to gather information for understanding the problem domain and application domain. In this section, we describe three different types of interviews: structured, semi-structured, and unstructured. Finally, some of the advantages and disadvantages of the three interview types is compared.

Structured Interview

In a structured interview, all questions regarding the interview are prepared before conducting the interview (Benyon, 2014, p.142-143). When conducting a structured interview, the questions must be asked in the same order and wording as prepared (Benyon, 2014, p.142-143). This type of interview is very close to using a questionnaire, but it allows the interviewees to respond more broadly. A questionnaire must be designed very carefully and all possible ways of answering it must have been thought of beforehand. A structured interview can make this process slightly easier, as the interviewees are not as constrained in a conversation, as if they are filling out a form, be it digital or not.
Advantages:

- Easy to conduct the interview
- Easy to analyse and evaluate the data

Disadvantages:

- No additional knowledge beyond those of the predefined questions can be discovered

Semi-structured Interview

Before conducting a semi-structured interview, different topics or questions should be prepared (Benyon, 2014, p.142-143). When conducting a semi-structured interview the predefined questions or topics serve as a guide for the interview. The interviewer is then responsible for asking follow-up questions to gather all relevant information (Benyon, 2014, p.142-143). This type of interview is useful if the interviewer already has some information about the relevant problem, but wants more in-depth knowledge, while also accepting the fact that he or she, most likely has not thought of everything.
Advantages:

- Additional knowledge beyond those of the predefined questions can be discovered

Disadvantages:

- Difficult to analyse and evaluate the data from different interviews, since most data will differ

Unstructured Interview

In an unstructured interview there are no questions or topics prepared beforehand and the only guideline that the interviewer has, is an overall subject (Benyon, 2014, p.153).
Advantages:

- Virtually no preparation is needed
- The interviewer does not need to know much about the subject

Disadvantages:

- Very difficult to analyse and evaluate the data. Especially to compare data from different interviewees
- Difficult interview to conduct efficiently, as it is easy to get sidetracked

Comparing Types of Interviews

If the interviewer only needs answers to a predefined set of questions, a structured interview makes it easy to conduct the interview. Data from a structured interview is also easy to analyse and evaluate because data from all interviews follow the same template. This also means that statistics can be made directly from the gathered data. It is more difficult to analyse and evaluate data from a semi-structured or unstructured interview because the questions and their format vary from interview to interview.

In a structured interview one cannot get additional knowledge outside the topics or areas that the predefined questions cover, since one is not allowed to ask follow-up questions, which is possible in semi-structured and unstructured interviews.

Unstructured interviews are easy to prepare, compared to structured and semi-structured interviews because the interviewer does not need to have any knowledge or any predefined questions about the subject. It can, however, be difficult for the interviewer to know what information he or she needs to gather, due to the lack of knowledge about the subject they are conducting an interview on.

Recording Interviews

In all three types of interviews, it is important to ensure that all essential information gathered during the interview, is available for later analysis. To ensure this, devices for video and audio recording can be used. When choosing to record the interview with a video camera, there will be a lot of information that needs to be processed afterwards. To make this easier, one of the people conducting the interview can write down notes about important moments of the interview, including times synchronised with the video, which then serve as pointers to highlights in the video (Benyon, 2014, p.145).

Summary

In this chapter, methods for software development and data-collection were described. The activities of OOA&D were described in detail, and because of the time frame of the project and the fact that cooperation with users is an important aspect of this project, the iterative method for software development was chosen. Furthermore, different forms of interviews were described.

The knowledge of interviews that was obtained from this, was used in the next chapter, which includes descriptions of interviews conducted.

Chapter 3

System Choice

In this chapter, the requirements of the system developed in this project, are described. To select the requirements a PACT analysis is used. It is used to achieve an understanding of the people, activities, context, and technologies at Aalborg Climbing club. Throughout the PACT analysis, requirements are derived. Each requirement is then prioritised using the MoSCoW rules. From these prioritised requirements a system definition is created. A FACTOR analysis that summarises the findings in the system definition is also included.

3.1 First Interview with Aalborg Climbing Club

The purpose of this section is to describe the first interview with Aalborg Climbing Club; this includes the goal of the interview and the choices made regarding it.

To prepare questions for the first interview, a preliminary PACT analysis was made, with the purpose of obtaining a basic understanding of Aalborg Climbing Club, before conducting the first interview.

It was decided that a semi-structured interview, which is described in section 2.2, should be conducted. The reason for that, was that very little was known about climbing and the context in which it is performed. Having limited knowledge of a subject can make it significantly more difficult to prepare well-defined questions. This meant that the interview was structured loosely by defining themes we wanted covered during the interview, instead of making a structured interview.

To be able to organise and plan the semi-structured interview well, it was necessary to come up with overall themes that could give us the information we required. First and foremost, it was necessary to get a good fundamental understanding of climbing as an activity and the systems currently in use in the club. Secondly, we needed to know more about the users for which we were designing. To further specify the direction in which the conversation should go when the interview was conducted, each main theme was split into smaller sub-themes, which can be seen in appendix A.

Only three of the total seven group members went to AKK to conduct the first interview. This decision was made to prevent making the project client and interviewee, Mattias Hornum, feel intimidated by a large group of interviewers. The three group members each had a role: an interviewer, a cameraman, and a secretary. The interview was conducted on the 29th of September, 2016. Initially, we asked Hornum demography-related questions such as his age, his interests, his reason for climbing, etc. The conversation quickly gravitated towards climbing and we agreed that the rest of the interview would benefit greatly from seeing the climbing halls and the current system. After a quick tour of the club, we resumed the interview with a better general understanding of the things we were going to cover in the rest of the interview.

After the interview, we used the data collected to create a PACT analysis, as described in section 3.2.

3.2 PACT Analysis

The following sections are based on the PACT analysis model which is divided into four main topics: people, activities, context and technology. A PACT analysis is a useful tool for understanding a situation or problem and see where improvements can be made (Benyon, 2014, p.43). Throughout the PACT analysis, we formulate possible improvements for the situation in AKK as requirements. We differentiate between two types of requirements: direct requirements and derived requirements. A direct requirement is a requirement we received from the client. A derived requirement is a requirement that is found through analysing the information gathered in the PACT analysis. We choose to present requirements directly in the PACT analysis to make it clear what information the different requirements are based on.

3.2.1 People

In this section, the people of AKK are described and analysed. This includes information about their demographic, physical and psychological differences, as well as a look at their common disabilities.

Demographic

According to Hornum, AKK is a club with a wide range of members. From table 3.1, it can be seen that AKK has a total of 269 members and that these differ a lot in age and gender. Furthermore, quite a few of the club's members are foreign exchange students, as mentioned in the interview: *"Well we have quite a few foreign members here. Exchange students."* (Hornum, 2016, L.183).

It is important to develop a system that is intuitive and usable for all members of the club, regardless of their nationality, gender, and age.

Developing a system that needs to be accessible to foreigners as well as locals draws focus to language differences and cultural differences. Language issues may not only be related to foreigners, but also to children. Danish children are taught English from first grade in elementary school, at which point they are usually between six and eight years old (EMU, 2016) (UVM, 2015).

To cater to the needs of not only foreigners, but also children, the system should be available in both Danish and English. Seeing as there are often significant differences in the mental models of people of differing ages, it is also important to consider these when designing the user interface.

> ***Derived Requirement:*** *The system must be available in both Danish and English.*

Membership type	Male	Female	Percentage
Age 0-12	11	22	≈ 12
Age 13-18	10	13	≈ 9
Age 19-24	63	26	≈ 33
Age 25-59	85	39	≈ 46
Age 60+	0	0	0
Total	169	100	100
Instructors 0-24	3		
Instructors 25+	26		
Total combined	269		

Table 3.1: Membership Statistics for AKK. Source: AKK

Physical Differences

Physical differences, such as height, are also relevant to take into account when constructing a new climbing route. Some people may be too tall to climb a specific route, where others may be too short. As both adults and children are present, the importance of differences in height is amplified. The average ten-year-old is about 140 cm tall (Afd. for Vækst og Reproduktion, Rigshospitalet, 2014). However, at the age of 20, the height difference between the genders varies, since men are, on average, 181 cm tall, and women 169 cm tall (Afd. for Vækst og Reproduktion, Rigshospitalet, 2014). There is, for example, a gap of 38 cm between boys who are aged 10 and men who are aged 20. Physical differences like height might be relevant to consider if the future system were to distinguish routes on the distances between holds. If the distance is too large, the route may not be suitable for young climbers. To address this, the system could show routes that are suitable for younger climbers.

Psychological Differences

While people's physical characteristics differ greatly, it is also very common for people to have different psychological traits. There can, for example, be a significant difference in how good people's spatial ability is. Having bad spatial ability can make it difficult to navigate easily in the real world, but also interactive systems can be challenging to navigate successfully (Benyon, 2010, p. 32). Language differences are also categorised as a psychological difference (Benyon, 2010, p. 32).

While there are several differences in the way people understand, behave, and process information, there are certainly also traits that almost all people share. Especially when asking users to perform relatively cumbersome work, most people might appear lazy (M.D., 2014). As pointed out by Hornum, it is tough to make users carry out tasks that requires much more than a couple of seconds of work. *"...people are lazy. There is never anyone who wants to create a cool overview picture. That is why I think that a simple task such as taking a picture, I mean, anyone can go in there [the bouldering hall], snap a picture of a section and upload it. It takes four seconds, it is easy to do. However, if you have to draw instead..."* (Hornum, 2016, L. 577-582).

Hornum also stated in a later interview that: *"Since the people who are creating the routes are volunteers, then using the system should be as problem free and simple as possible."* (appendix C). In this project a simple system is defined as a system that is easy to use. Therefore we derive the following requirement for the system.

Derived Requirement: *The system must be easy to use.*

Common Disabilities

Colour blindness may have to be taken into account. Most often, climbing routes are differentiated by colour, and therefore, the colours being used have to be colour blind friendly, otherwise colour blind people cannot distinguish between routes. According to the National Eye Institute: *"As many as 8 percent of men and 0.5 percent of women with Northern European ancestry have the common form of red-green colour blindness."* (National Eye Institute, 2015). Even though, we are not aware of any current members of AKK being colour blind, it might be relevant to take into consideration.

3.2.2 Activities

In this section, the activities related to climbing will be described and analysed. In this project, we look at two types of climbing: rope climbing and bouldering. Hornum describes the two types of climbing as follows: *"One is rope climbing, [...], where you climb with rope, and then in here [referring to a different room], there is bouldering, where you climb without rope, then it is not as high and then there is a mattress."* (Hornum, 2016, L.155-158).

Bouldering

Bouldering is a short low-height type of climbing, where the climber climbs without a rope, as shown in figure 3.1. The climbers are often faced with difficult challenges that they have to overcome to finish the route(Dansk Klatreforbund, 2016a). In bouldering, the climbers often have no safety line, but instead there may be a spotter that guides the climber to a crash pad if he or she falls (Dansk Klatreforbund, 2016a).

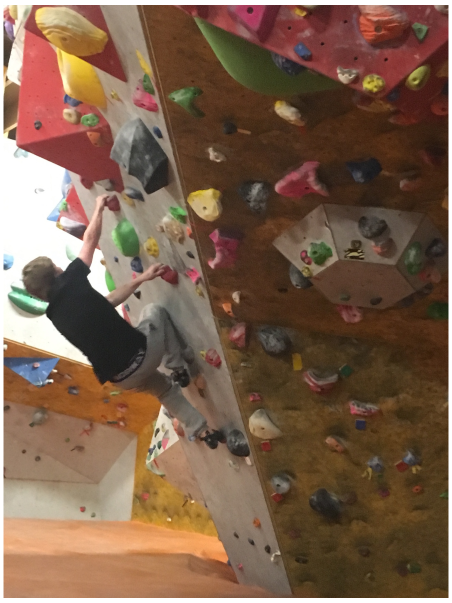


Figure 3.1: Shows a climber bouldering by following a route of holds in a specific colour.

Rope Climbing

There is at least two types of rope climbing: sports climbing and top roping.

In sports climbing there are two people; a climber and a person who is in charge of the security rope. During the climb, the climber attaches the rope to bolts along the route (Dansk Klatreforbund, 2016b).

Top roping is suitable for beginners due to the low risk of injury: this is because the safety line is always attached above the climber, which means the climber rarely falls very far (About Sports, 2016). New climbers can thus focus on learning basic techniques and movements, instead of focusing on attaching the rope to along the way. (About Sports, 2016).

Temporal Aspects

In AKK, the climbers have the option to climb as much as they want: *"We are open 24/7 so people can, in principle, come and leave as they want"* (Hornum, 2016, L. 634-635). If the members of AKK can climb every hour of every day, it should be a requirement that the system is also available at all times. Because the system is being developed for all members of the climbing club and not just one person it should be able to handle multiple members using it at the same time. It is therefore important multiple members can use the system simultaneously. This may be relevant on club nights when there is a lot of members in the climbing club: *"[...] we have club nights Tuesday, Thursday, Sunday and Monday"* (Hornum, 2016, L. 646-647).

> ***Derived Requirement:*** *Be able to handle multiple users simultaneously.*

If a climber only wants to climb the best routes, some sort of rating system should be considered. This could be designed similarly to Minimum Boulder, a climbing administration system used in a climbing club in Switzerland (Minimum-Bouldering, 2016). Hornum describes it: *"[...] the nice thing about it was that then you could rate them with one to five stars [...]. Then you could see if you have limited time which boulder routes that are best. So then you just climb those."* (Hornum, 2016, L. 351-355). Therefore, to create a better experience for the members, we can take inspiration from this and derive the following requirement.

> ***Derived Requirement:*** *Members should be able to rate routes and see ratings of routes.*

Cooperation

Climbing can be a highly social activity, even though the climbing itself is done individually. When climbing with a rope, it is necessary for another person to hold the safety line, but even when climbing without safety gear, a spotter on the ground can still be useful by giving helpful comments and guide the climber safely down again. In AKK, the other person is often a friend, but climbing as a family activity also takes place there (Hornum, 2016, L.628). Hornum stated that bouldering could be more popular than rope climbing: *"[...] it could have something to do with the fact that it is easier and it can be done by yourself."* (Hornum, 2016, L.164-167). This means that bouldering is more independent, but this does not mean that climbers do not talk to each other: *"It is surprising that even though it is done individually, then it is surprisingly social. It is very nice."* (Hornum, 2016, L.649-650). When designing a future system, it might therefore, be an option to include social features such as sharing climbing activities among climbers, plan meetings and send messages. Another social feature could be to add comments to the routes in the system which would let members share their thoughts on a route. We asked Hornum about this feature, to which he replied: *"Yes, or a video. That could also be very cool"* (Hornum, 2016, L. 521). From this, we derived the following requirement that the system could implement.

> ***Derived Requirement:*** *The system could allow members to comment on routes.*

One way of cooperation between climbers is to share what is called beta: *"There is a thing in climbing called beta, [. . .] if you are going to climb and you have many problems with the route, because you try to move your hand in a specific way, then I can come to you and then say, 'That is because you should actually use your left hand'. Then I just gave you a beta to the route."* (Hornum, 2016, L.497-501). A Beta is therefore some kind of hint or instruction on how to overcome difficulties in a climbing route, which could be represented in the system by uploading a video as suggested by Hornum: *"[. . .] it could be a really cool function to be able to upload video beta"* (Hornum, 2016, L. 521). This suggestion leads us to the following requirement.

> ***Direct Requirement:*** *The system could allow members to add beta to the routes.*

Complexity

If we look at the complexity of creating a new route, then one of the reasons that there is no system for the rope climbing activities may be that: *"[. . .] it is more difficult to create routes in here because you need to be held by a rope so it takes a lot more time."* (Hornum, 2016, L. 163-164). Even though it is more challenging to create a route for rope climbing, it is still possible to include these routes in a system.

In bouldering, it is easier to create a new route due to the lower heights where the holds can be attached while standing on a ladder. The difficult part of creating a new route is not just to attach the holds but also that: *"[. . .] it takes a surprising amount of experience to create. If you think, okay I want to create such a move here, so to get the hold placed right and place the feet to fit and something like that. That is surprisingly difficult."* (Hornum, 2016, L.261-264). In the climbing activity, the complexity is enhanced by doing very difficult moves: *"[. . .] it is often that people ask 'Is the goal to climb as fast as possible?', but the real goal is to climb as difficult routes as possible"* (Hornum, 2016, L.102-104). If the goal of climbing is to climb the most difficult routes then the user is not interested in routes within all grades. Therefore it should be a requirement for the system that the user can choose a grade that fits his skill level and view routes in that grade.

> ***Derived Requirement:*** *It should be possible to use the system to find routes by grade*

Safety

Because activity of climbing takes place several meters above the ground, the system can therefore, be seen as safety-critical. It is important that the use of the system does not distract the climber from their activity. In sports climbing it is important that the climber is aware of how to safely attach the rope, along the route. In AKK it is a requirement for members to have taken a safety course in order to be allowed to do sports climbing (Aalborg klatreklub, 2016b). To help prevent members that have not yet completed the safety course, from climbing a sports climbing route, it may be rational to indicate whether a route requires a safety course to have been completed or not.

> ***Derived Requirement:*** *The system must indicate if a route requires that the climber has completed a safety course.*

3.2.3 Context

During the interview, we found that the climbing club is not a commercial organisation, but rather a club of volunteers. Therefore, it does not make a lot of sense to analyse the organisational context of the club. Instead, we have focused on the physical and social contexts of AKK.

Physical Context

The physical context in which the mentioned activities take place, is primarily inside AKK's clubhouse, but sometimes AKK arranges competitions at different locations. For instance, they recently had a competition at Aalborg Waterfront (Hornum, 2016, L.78-81). The context may also change in the future as AKK is currently working on moving to a new building, as Hornum states in the interview: *"We are actually working on a project where we are going to move into new buildings together with Sportshøjskolen [. . .]"* (Hornum, 2016, L.63-68). Therefore, a future system must take this into consideration and be dynamic enough to be used if AKK decides to move their facilities elsewhere. This means that the system must be able to handle new sections as well as editing or deleting old ones.

> ***Derived Requirement:*** *The system must be able to modify sections, i.e. add, delete, edit.*

For both bouldering and rope climbing, the holes for installing climbing holds have the same distance between one another, but the holds themselves are not installed in a way that creates straightforward routes (Hornum, 2016, L. 532-539). When asked if the routes or holds break, Hornum answered; *"No, they do not break."* (Hornum, 2016, L. 440-443). Since the holds and routes do not break down, they are usually only maintained when the staff clear a whole section and create new routes.

Social context

Despite the main point of coming to AKK is climbing, it is entirely possible to sit down afterwards, relax with a cup of coffee or a small meal while talking with other members of the club. Although, when asked if people show up at the climbing club solely for social gatherings, Hornum answered *"That is not my impression."* (Hornum, 2016, L. 638).

Being a part of a climbing club, means being part of a social community, which is built around the interest of climbing. Because of this, it could be relevant for the members of the climbing club to share information about routes with each other. In fact, AKK already has a social media group on Facebook, which they use to communicate with each other. It could, therefore, be relevant for the future system to be integrated with social media sites and thereby allow sharing of routes and route information such as images. During an unstructured interview after the second usability test (see appendix E) a member of AKK said that he wished there was a share button, making it possible to share the image of a route he just took with people on Instagram (see appendix E.2).

> ***Derived Requirement:*** *The system should be integrated with social media, and allow easy sharing of routes and route information.*

3.2.4 Technology and Current System

In the interview, Hornum informed the group that the current system consists of five whiteboards, one for each grade. With the current system, the only way to find a route, is to look on the whiteboard matching the route's grade (Hornum, 2016, L.113-139). Hornum suggested that it would be nice if it was possible to find routes based on the dates they were created, and the section they are built in: *"If you could also sort it by say, date and such. I imagine that would be easy to implement. Also because it is important to be able to sort by section"* (Hornum, 2016, L.397-404).

To make it more efficient for users to find routes in the system, filtering and sorting routes by their section, grade or creation-date should be possible, and we can therefore derive the following requirement.

> ***Derived Requirement:*** *Users should have the ability to find routes based on what section, or date it was created on.*

"In here [the top-rope hall], we have not taken the system into use yet, since there has not been enough motivation." (Hornum, 2016, L.159-160). In this quote, Hornum states that the whiteboard is only used for bouldering problems. As seen in figure 3.2, the whiteboard keeps track of the grade of the route (colour of the wooden block at the top), the section, a number identifying the route, name of the creator, and the date of creation All members can add new routes to the whiteboard, but according to Hornum, it is widely accepted that members do not change other members' routes without permission: *"One must rather not touch or adjust other people's routes."* (Hornum, 2016, L.252-253), (Hornum, 2016, L.285-286).

The current system was initiated by Hornum, and is the first system used in the climbing club (Hornum, 2016, L.79-81), (Hornum, 2016, L.97-99). According to Hornum, developing an app as the first system for AKK would be too big a change for the members. Instead he developed the whiteboard system to use as a stepping stone for a more complex system: *"My first thought was basically that an app would be great, but I think it would be too big of a change for the climbing club."* (Hornum, 2016, L.130-133). The current whiteboard system is written in English since AKK has foreign members (Hornum, 2016, L.181-185).

Since the members already know how to use the current system, and its intention was to be used as a stepping stone, we can therefore derive the following requirement from what Hornum has told us.

> ***Derived Requirement:*** *The system must be able to do the same things as the current system.*

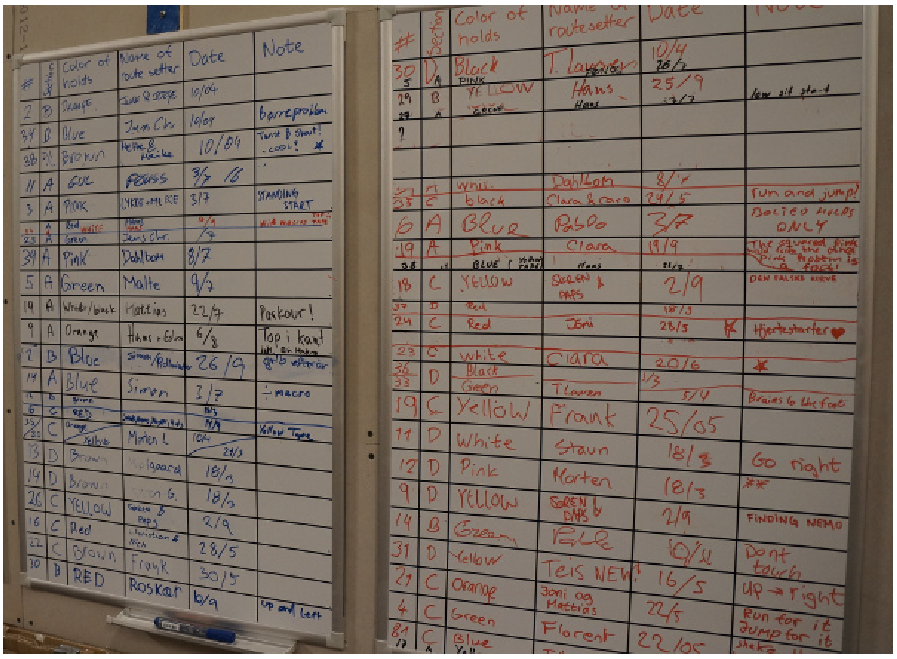


Figure 3.2: The current system at Aalborg Climbing Club using five whiteboards, each using a different colour that represents the grade of the routes listed on it. Each whiteboard has six columns: Number, section, colour of holds, creator of route, date of creation, and note.

When using the system, it is important that the members know the connection between routes in the system and the actual routes on the wall: *"[. . .] It is important to make clear what route it is right? And one of the things are the blocks with the number. But it could also be extremely cool if you just could upload a picture of the entire route, as we talked about, where you could mark the holds. [. . .]"* (Hornum, 2016, L. 545-548). From this we state the following requirement.

> ***Direct Requirement:*** *The system should allow members to add an image of a route.*

From the final usability test, one of the people tested suggested that the system could integrate QR-codes on the routes which could be scanned by the phone to quickly get information about a specific route. The usability test of the person suggesting this feature, can be seen in appendix F table F.3. From this suggestion we derive the following requirement.

> ***Derived Requirement:*** *Find all information about a certain route, by making use of QR-code technologies in the coloured bricks on the wall.*

AKK sometimes clear an entire section at once to make up space for new routes, and therefore Hornum finds it problematic that routes in a specific section can be spread out on the whiteboard, since it is sorted by grade (Hornum, 2016, L.271-273), (Hornum, 2016, L.396-399).

Hornum also finds it problematic to administrate the sections on the upper-level of the AKK building, since the whiteboard is on the ground floor (Hornum, 2016, L.390-395).

For the system to be used different places in the climbing club the system must be mobile, and if we were to develop a mobile or web application, as Hornum suggests, the system would require access to the Internet.

> ***Derived Requirement:*** *Be mobile, such that it can be used anywhere with Internet access.*

If this was to be implemented, the members of AKK would all need to have a smartphone to access the system. As stated in table 3.1, the majority (79 percent) of the members are aged 19-59, and according to Statistics Denmark, 75 percent of people aged 16-89 accessed the internet through a mobile phone in 2015 (Lauterbach, 2015, p.28, figure 40). This 75 percent would be higher, if the age group 16-89 was changed to 16-59, since younger people access the internet more often on their smartphones and because the elder group access the internet on a smartphone less often (Lauterbach, 2015, p.30, figure 42), (Lauterbach, 2015, p.35, figure 50). To further support the claim that most people have a smartphone or cellphone, another survey made by Statistics Denmark states that 77 percent of Danish households has a smartphone and close to all households (approximately 98-99 percent) have a cellphone (Danmarks statistik, 2015). Furthermore, this tendency is also growing, e.g. the amount of households that has smartphones has grown by more that 40 percent since 2011. From this, we can conclude that there is a tendency that most people either have a smartphone, or have access to a smartphone which, makes an application, and smartphones in particular, a realistic choice of technology, even though we cannot state for certain that all members of the climbing club have a smartphone.

3.3 Requirements

According to (Benyon, 2014, 139), a requirement is "something the product must do or a quality that the product must have". In this project the requirements for the system were established through the PACT analysis, and they serve as a contract for what the system should be able to do once the system has been fully developed.

Since the project has a time limit, decisions are made about how we prioritise these requirements. The prioritising of the requirements is used throughout the different development iterations as a tool for knowing which requirements to implement in each iteration, which ensures that important requirements for the new system are implemented first.

The requirements were prioritised by following the 'MoSCoW Rules' which classify the requirements into one of four types of requirements (Must have, Should have, Could have, Will not have) based on how important they are for the system (Benyon, 2010, p.150).

The requirements were first prioritised by the group members, and then with the help of our client, Hornum, during the second interview (see appendix C). The reasoning behind the prioritising of each requirement is described later in this section.
Below, the different 'MoSCoW rules' are briefly described: (Benyon, 2010, p.150)

- **M**ust have - is the essential requirements of the system. Without them, the system would not work to the clients satisfaction.
- **S**hould have - important parts of the system that should be included if there is time to implement them, but the system will still be usable without them.
- **C**ould have - are requirements that would be nice to have, but could be left out in the current development of the system, without making the system less usable.

- **W**ant to have but **W**ill not have this time around - requirements that are not important for the current development of the system, and therefore will be left out to be implemented in later development.

These MoSCoW rules were used to prioritise the requirements found in the PACT analysis. Must have requirements…

- The system must be able to do the same things as the current system.
  - Add, delete, edit and view routes
  - View routes by grade
- The system must have the ability to show routes based on what section or date it was created on.
- The system must be able to handle multiple users simultaneously.
- The system must be easy to use.

Should have requirements…

- The system should allow members to add an image of a route.
- The system should offer the ability to modify sections, i.e. add, delete, and edit.
- The system should be mobile, such that it can be used places in the climbing club where there is Internet access.

Could have requirements…

- The system could allow members to add beta to the routes.
- The system could allow members to rate routes and see rating of routes.
- The system could allow members to comment on routes.

Want to have, but will not have this time around …

- The system will not be available in both Danish and English.
- The system will not be integrated with social media, and allow easy sharing of routes and route information.
- The system will not support QR-codes technologies to find the information about routes.
- The system will not indicate if a route requires that the climber has completed a safety course.

From the interview with Hornum, it was clear that the system needed to have at least the same functionality as the current system. It also had to be able to sort and give an easy overview of routes, since this was one of the problems of the current whiteboard system. After our first usability test, we found that the participants had problems finding routes based on authors, and also, the fourth participant requested a search feature, which we found to be a good idea, since it would also be able to solve the *Find Route by Author* usability problem. The details of the first usability test, can be seen in appendix D. Since there are a lot of members at AKK, it also had to be able to handle multiple members simultaneously. Furthermore, it was also established that the system should be easy to use, as a must-have requirement, since the system is being developed to be usable by all current and future members of AKK independently of their technical knowledge.

We then looked at which requirements that were important for the system, but not essential. These included adding an image of a specific route, as well as being able to administrate sections. Additionally, the system should be mobile by being usable anywhere with an Internet connection. These requirements attempted to solve some of the issues that AKK had with the whiteboard system, such as the lack of mobility and overview of routes. However the system could still be used without implementing these requirements, and they were therefore prioritised as should-have requirements.

A feature that would be nice to have, was to allow members of AKK to help each other by providing tips on how to climb a specific route, known as giving each other beta. Additionally, being able to rate and comment routes would be a useful addition to the system. Common for all these requirements, is the fact that they provide some extra functionality that the current whiteboard system does not have, but was deemed relevant features for the new system, through the PACT analysis. These requirement were all prioritised as could-have requirements since they would be nice to have in a new system, but not not implementing them will not make the system less usable since commenting, rating and giving beta is not directly related to route-administration.

The last group of requirements, the will-not-have requirements, were requirements that would be beneficial to have at some time, but not in this development period. We chose not to include support for both Danish and English, since most users of the system already know English, and the system will not contain a lot of text. We did not include the requirements concerning support for QR-code and integration with social media as these requirements were discovered in the end of the last iteration. Finally we did not choose to include the requirement that the system must show if a route requires that the climber has completed a safety course because we do not see this as part of the primary responsibility for the new system.

Even though all requirements would contribute with functionality to the system, they were not deemed important enough to fit within the limited time frame of the project.

3.4 Problem Statement

From the PACT analysis, we discovered multiple requirements for the system to be developed in this project. Based on the must-have, should-have, and could-have requirements, we delimit the focus of this project and formulate the following problem statement:

How can we develop a mobile system that allows multiple members to easily administrate climbing routes including associated grades, sections, images, betas, ratings and comments?

3.5 Defining the New System

From the requirements and PACT analysis in section 3.2, we start to define a new system for administrating climbing routes in AKK. Figure 3.3 is a rich picture that illustrates the transition from the current system at AKK, to a digitised system. In the current system, the members will have to physically move to the whiteboard-system if they want to modify, add or remove climbing routes. In a digitised system, the members will have the option to administrate climbing routes through their smartphones. This means that the users of the digitised system are not limited to one specific place where they can use the system as is the case with the current system.

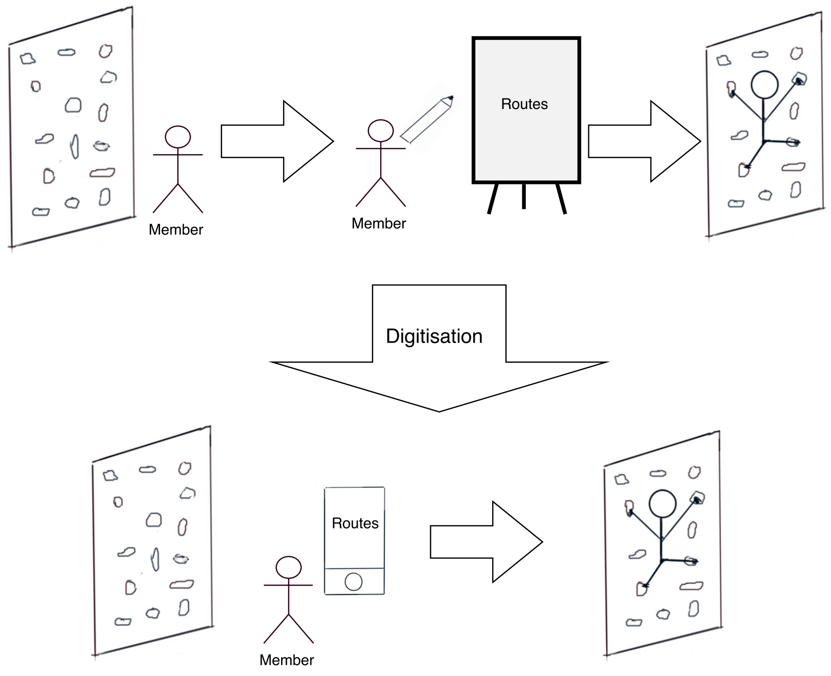


Figure 3.3: Rich picture that illustrates the transition to a digitised system.

From our MoSCoW analysis, we can create a System Definition and the corresponding FACTOR analysis, which describes what system we are going to develop in this project.

3.5.1 System Definition

An IT system for administrating climbing routes at AKK using a smartphone connected to the internet. The system lets members of the climbing club easily administrate routes by adding, removing, and changing them in the system. Furthermore, members can rate, comment, and provide beta to a climbing route. They can also add a note to a route and submit a picture for a route. All routes are located in sections which are controlled by members, who can add, clear, rename and remove an entire section of routes. Members of the climbing club can get an easy overview of the provided information about each route.

3.5.2 FACTOR Analysis

A FACTOR analysis is tied to the system definition and consists of six elements: Functionality, Application domain, Conditions, Technology, Objects, and Responsibility. The elements in a FACTOR analysis should be derived from the system definition, and therefore the process of making the FACTOR analysis and the system definition is an iterative one. It is iterative because we check our system definition against the FACTOR until all elements in the FACTOR is represented satisfactory.

F Add, edit, view, remove sections and routes
Add comments, rating, images and videos, as well as beta and a note, to routes

A Members of the club can administrate routes and sections
Guests can view routes

C Must be usable in a climbing room
Easy to use by members and guests of the climbing club

T Runs on smartphones
Requires internet connection

O Sections, routes, member, rating, note, beta and comment.

R An IT system for administrating climbing routes in the AKK.

Summary

In this chapter, it was explained how and why a semi-structured interview with a member of AKK was conducted. From this, it was possible to create a PACT analysis of the people, activities, context, and technology of AKK. From the PACT analysis, AKK was found to be a climbing club with 269 members that differs in gender, age, and nationality. Furthermore, AKK has two types of climbing: rope climbing and bouldering, which are located in separate halls. Knowledge about how AKK currently keep track of their bouldering routes on five whiteboards, was obtained. Throughout the PACT analysis, two types of requirements were specified: *derived* and *direct* requirements. The direct requirements included the features that the clients explicitly stated they would like in a future system, while the derived requirements included those that could be derived through the PACT analysis. The requirements were prioritised using the MoSCoW rules. From the requirements and PACT analysis, a problem statement specifying the problem that needed to solved be in the future system was made. The transition of AKK's current system to a digitised version of it was illustrated using a rich picture. Lastly, a system definition which specified the system to be developed, was made.

In the next chapter, we will continue on to the next step in the OOA&D method, which is the problem-domain analysis, where we will do the Classes, Structure and Behaviour activities.

Chapter 4

Problem-domain Analysis

In this chapter the problem domain is modelled, based on the system definition and the information obtained in the PACT analysis in section 3.2. The modelling is done by identifying classes and events in the problem domain. From these, an event table, which illustrates how the different classes and events are connected, is created. The classes and their attributes are then structured in a class diagram, which illustrates the relations between them in the problem domain. Using the information contained in the event table, behaviour diagrams for each class are created, which illustrates how events affect the specific classes.

4.1 Classes Activity

The first part of the problem-domain analysis is the Classes Activity, where classes and events are analysed in order to help understand the problem domain.

4.1.1 Selecting Classes

After doing a PACT analysis and analysing the requirements for the new system, it is possible to begin looking at which classes were necessary to model the problem domain in the new system. According to Mathiassen et al. (2000), the problem domain is the part of a context that is administrated, monitored or controlled by a system.

A class is a collection of objects that have the same behaviour, attributes, and structure (Mathiassen et al., 2000, 49). To identify these classes, a brainstorm was conducted using the knowledge gained through the PACT analysis and the interview with Mattias Hornum (2016). Afterwards, we excluded some of the classes based on the system definition, and by following four criteria for selecting classes from Mathiassen et al. (2000, p.61) which can be seen below.

- Can you identify objects from the class?
- Does the class contain unique information?
- Does the class encompass multiple objects?
- Does the class have a suitable and manageable number of events?

Below is a list of suggested classes found after brain-storming. The classes that have not been crossed out represent the selected classes used to model the problem domain.

Classes	
Member	~~Top-roping route~~
~~Admin~~	Route
~~Guest~~	~~Club~~
Comment	Grade
Section	Rating
~~Bouldering route~~	~~Map~~
~~Sports-climbing route~~	Beta
~~Text~~	Hold

Table 4.1: Suggested Classes for the new system. Classes that were discarded at a later time, are crossed out.

Discarded classes

The three classes `Bouldering route`, `Top-roping route`, and `Sports-climbing route` were all combined into a single `Route` class, since Hornum told us that there were no differences between the different types of climbing in their current system and thus no reason to model it (Hornum, 2016, L.213). We removed the `Club` class because of the third criterion since we only focus on AKK in this project and not several climbing clubs. The `Map` class was removed since it is not a part of the system definition.

The classes `Guest` and `Admin`, were removed since we do not need to administrate, monitor, or control these classes in the problem domain. We removed `text` as it has much in common with the `Comment` class.

Describing selected classes

A `Member` is a person that is a part of the climbing club. A `Member` climbs routes, adds comments, rates routes and provides other people with beta to routes. A `Member` can also add additional routes to the system, as it is the members of the club who creates routes in AKK.

A `Section` describes a physical area within the climbing club. Each `Section` is identified by a single letter, and at the time of writing there are four `Section`s in AKK: A, B, C, and D. Each `Section` contains several `Route`s, but a `Section` without `Route`s can exist, e.g. when a `Section` is cleared of all `Route`s.

A `Route` describes a climbing route in the climbing hall. A climbing route consists of a number of `Hold`s located somewhat close to each other in a `Section` of a climbing wall. The beginning of a `Route` is marked on the climbing wall by a small coloured wooden square with a number written on it. A coloured square marks the `Grade` and number of the route. The `Grade` and number is what uniquely identifies a `Route`.

A `Hold` is an integral part of a `Route` and climbing in general. It is a coloured chunk of plastic screwed into a climbing wall that climbers can grab and stand on. They come in a multitude of colours, and a route follows a specific set of `Hold`s indicated by colour. If routes using the same colour `Hold`s are placed too close to each other, coloured tape can be used to indicate which `Route` a `Hold` belongs to.

`Beta` is information that describes how to climb a specific route. For instance, if one climber finds a specific route difficult to complete, another climber can explain how to perform a certain move during the route and thereby provide beta for the route (Hornum, 2016, L.497-501).

A `Grade` is a colour that denotes the grade of a `Route` where each `Grade` represent different difficulties. It is the member that creates a `Route` that determines the `Grade` of a `Route`. AKK have at the time of the writing five different grades: green, blue, red, black, and white.

A `Comment` is an opinion that a `Member` has of a specific `Route`, which the `Member` shares with one or more `Members`.

A `Rating` represents a score given by a `Member` that describes how good a `Route` is according to that `Member`.

4.1.2 Selecting Events

During the brainstorm we also found possible events that, along with the classes, could be used to describe the problem domain further. An event is an instantaneous activity that involves one or more objects and is used to describe how classes can behave (Mathiassen et al., 2000, p.49).

Just like with the classes, we started by brainstorming events and then excluded that on the following three criteria from Mathiassen et al. (2000, p.63)

- Is the event instantaneous?
- Is the event atomic?
- Can the event be identified when it occurs?

The events, before and after the critical selection, can be seen in table 4.2. The discarded events are crossed out.

Events	
~~Uploaded~~	Section added
~~Signed in~~	Section cleared
~~Signed out~~	Section removed
~~Signed up~~	Grade created
~~Route climbed~~	Route commented
Route created	Route rated
Comment removed	~~User deleted~~
Route removed	Beta given
Hold created	Beta removed
Hold removed	Member signed up
Grade removed	Member resigned
Grade created	

Table 4.2: Suggested events for the new system. Discarded events are crossed out.

Discarded Events

The events *Uploaded*, *Signed in*, *Signed out*, *Signed up*, and *Deleted user* were not used, since we realised that they, like the classes `Guest` and `Admin`, were not part of the problem domain nor the system definition. The event *Route climbed* is part of the problem domain because a `Member` can climb a route, however we did not choose to monitor this in the system. Therefore, the event has been crossed out.

The selected events, has been identified from the system definition, requirements, and the above criteria.

Event Table

After having found the classes and events, we constructed an event table that illustrates which events affects which classes. The event table can be seen in table 4.3. The table was revised after we had finished the Behaviour Activity, to include information about how often an individual event can have an impact on a specific class.

	Classes							
Events	Section	Route	Rating	Comment	Beta	Member	Grade	Hold
Route rated		∗	+			∗		
Route created	∗	+				∗	∗	+
Route removed	∗	+	+	+	+		∗	+
Section added	+							
Section removed	+	+	+	+	+		∗	+
Section cleared	∗	+	+	+	+		∗	+
Route commented		∗		+		∗		
Comment removed		∗		+				
Beta given		∗			+	∗		
Beta removed		∗			+			
Member signed up						+		
Member resigned						+		
Grade created							+	
Grade removed	∗	+	+	+	+		+	+
Hold created		∗						+
Hold removed		∗						+

Table 4.3: Event table. A plus (+) means that the event can effect a class zero or once, while an asterisk (∗) means that the event can affect a class zero or multiple times

The event table as seen in table 4.3 helped to give an overview of the relations between classes and events. This overview can now be used to describe the structural relationships between classes and objects in the problem domain.

4.2 Structure Activity

This section describes the structure of the classes in the system. From the classes and events found in section 4.1.1 and section 4.1.2, we construct a class diagram with attributes, as seen in figure 4.1.

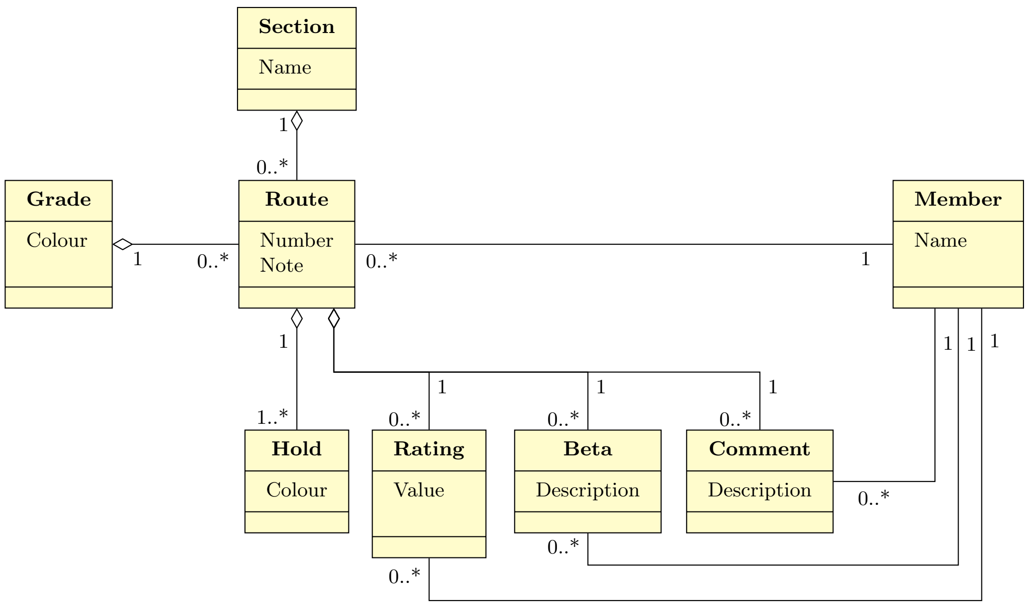


Figure 4.1: Class diagram for the future system. The attributes of the individual classes are values belonging to the objects of the classes, and are used to characterise and identify them.

Since the climbing club requires routes for the climbers to climb, and those routes cannot exist without sections, each `Section` aggregates zero or more `Route`s, and each `Route` aggregates zero or more `Hold`s. The structural relations between `Section`, `Route`, and `Hold` is called a hierarchy pattern because of the hierarchical structure of these classes (Mathiassen et al., 2000, 82). As modelled in the current system at AKK, each `Route` has a relation to a single `Grade` and a `Grade` aggregates zero or more routes. A `Route` also aggregates zero or more `Rating`s, `Comment`s and `Beta`s. This structure was chosen because it does not make sense for either of `Rating`, `Comment`, and `Beta` to exist independently of the specific route.

Looking at the class diagram in figure 4.1, we can see that until a user adds a `Rating`, `Comment`, or `Beta` to a route, there is no real change from the current system.

4.3 Behaviour Activity

After creating the class diagram for the structure of the classes in the problem domain, we move on to make state diagrams based on the selected classes and events found in section 4.1.1 and section 4.1.2 respectively. A state diagram illustrates the behavioural pattern of a class, which is a definition of potential event traces that may affect a class until it changes state and ceases to exist as an active object in the system (Mathiassen et al., 2000, p.89-90).

4.3.1 Section

A `Section` object is added to the system once it is affected by the *Section added* event. When a `Section` is in the system, `Routes` can be added to the `Section`, which does not change the state of the `Section` object, but merely makes it contain more `Routes`. These `Route`s can either be deleted individually by the *Route deleted* event or all routes in a section can be removed by the *Section cleared* event. Clearing a `Section` will not change the state of the `Section` object, nor will it be deleted. The *Route deleted*, *Route added*, and *Section cleared* events can therefore occur multiple times on the `Section` object. The section will cease to exist when it is affected by the *Section deleted* event. The modelled behaviour is shown in figure 4.2.

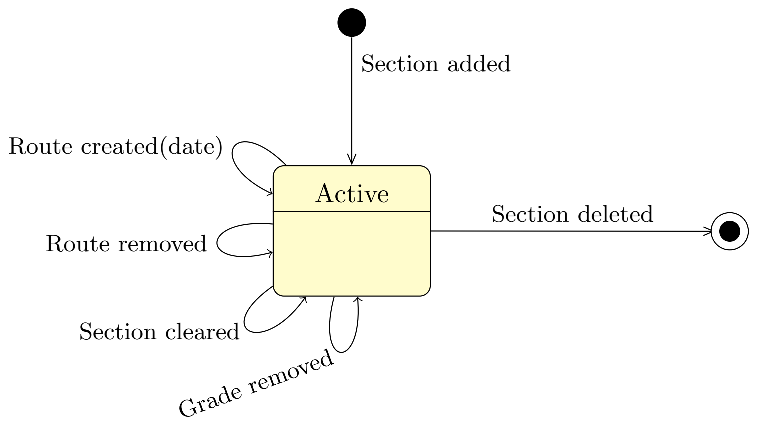


Figure 4.2: State diagram for a section object.

4.3.2 Route

A `Route` is added to the system when it is affected by the *Route added* event. Once a `Route` is in the system, other users can *Comment* on it, rate it and add `Beta` to the `Route` in the form of comments or videos; these events can occur multiple times on a route object, but does not change the state of the object. A `Route` will cease to exist if the `Section` it belongs to is deleted or cleared, or if the `Route` itself is deleted. The state diagram for the `Route` can be seen on figure 4.3.

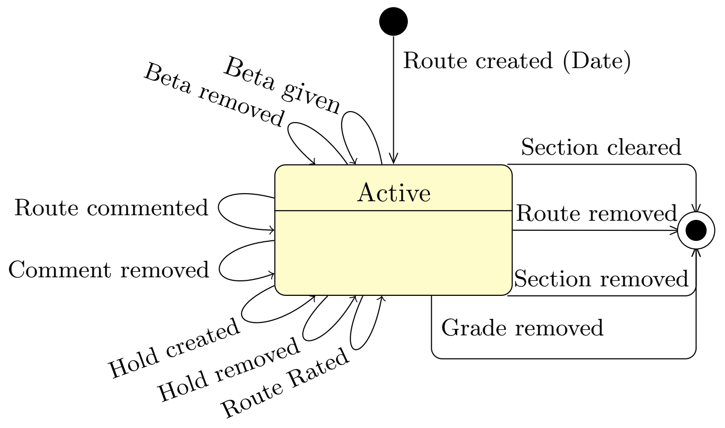


Figure 4.3: State diagram for a route object.

4.3.3 Rating, Hold, Comment, and Beta

When a user rates a `Route` object, the *Route rated* event causes the creation of a `Rating` object. This rating will continue to exist until the route is deleted in either of the possible scenarios previously described. This behaviour can be seen in the state diagram for `Route` in figure 4.4

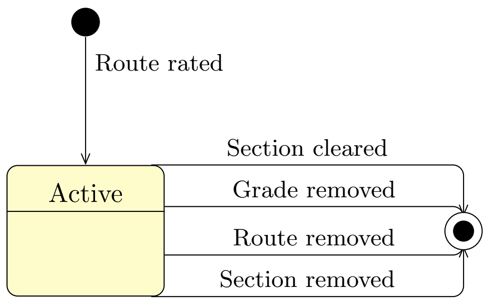


Figure 4.4: State diagram for a rating object.

The only difference between the behaviour of `Comment`, `Hold`, `Beta` and `Rating`, is the events that trigger the creation and deletion of the instances of these classes. The behavioural diagrams for these classes has therefore been places in appendix H.1.

4.3.4 Member

The `Member` object is constructed when a `Member` is added. A `Member` can add routes, `Comment` on them, and add `Beta` as well as add a `Rating` to the route. The `Member` object is deleted when the *Member deleted* event is called on the object.

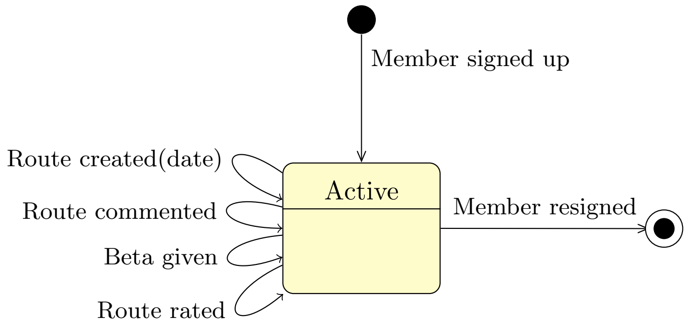


Figure 4.5: State diagram for a member object.

4.3.5 Grade

The `Grade` object is instantiated and added to the system when the *Grade added* event occurs.

The behavioural pattern for the `Hold` object is almost similar to that of the grade object, and can therefore be seen in appendix H.

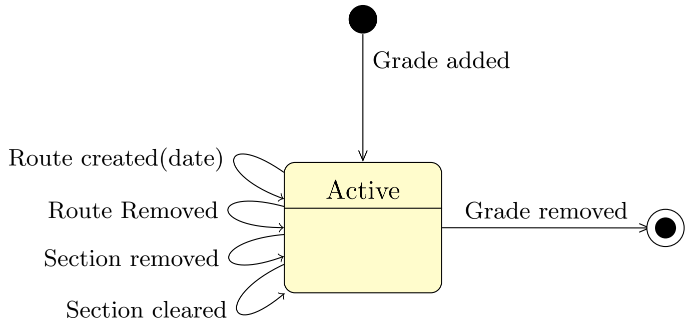


Figure 4.6: State diagram for a grade object.

Summary

To model the problem domain, the following classes were identified and selected: `Section`, `Route`, `Rating`, `Comment`, `Beta`, `Member`, `Grade` and `Hold`. To achieve a better understanding of the behaviour of these classes, different events were identified and listed in table 4.2. In order to show how the different classes and events were associated, an event table was created, as seen in table 4.3. The event table was also created to better understand how many times specific events could affect specific classes, which the class diagram was then updated to reflect.

The knowledge gained from the problem-domain analysis helped identify and describe the actors and use cases in the application-domain analysis, as described in the next chapter.

Chapter 5

Application-domain Analysis

In this chapter, the application domain is analysed. The first step of doing so, is determining who and what interacts with the system, including people and other systems. These are called actors, and actor specifications that describe their goals and characteristics are created for each one. Each actor specification also contains examples of these who these actors are and how they interact with the system.

After defining the actors, the ways in which they interact with the system will be described in detail by specifying use cases. This is done by by using the affected objects and functions, to create statechart diagrams that illustrate how different interactions change the state of the system. Combining the actor and use case specifications allows for the construction of an actor table, showing which use cases can be performed by which actors. Using the system definition, problem-domain analysis and use cases, the different functions of the system are determined and their type and complexity is ordered in a table.

5.1 Actors and Use Cases

In this section, we describe actors and use cases in the application domain. The actor is an abstraction for people, systems, or other entities that interact with the system (Mathiassen et al., 2000, p.119). A use case is a pattern for how the system and actors interact with each other (Mathiassen et al., 2000, p.119). The result of this section will be an actor table and state diagrams for all use cases.

5.1.1 Actor Specifications

In analysing the application domain, we came across three different types of actors: *climber*s, *setter*s, and *administrator*s. These three actors all use the system in different ways, but do not necessarily describe separate people: it is entirely possible for a single person to take on the role of more than one actor. Following is an in-depth description of each actor.

In the actor specification 5.1, the most prominent actor, the *climber*, is described. A *climber* is any person who uses the club for climbing, and while most of these actors are the people who frequents the club, one-time guests are *climber*s as well. This will be the most common actor in the system.

Climber

Goal: A person using the climbing club needs to be able to get an overview of climbing routes as well as a detailed description of each route. A climber's main goal is to find and climb routes. Climbers may also show other climbers how to climb a route by recording a video beta demonstrating certain movements along the route. Finally, a climber may want to express their opinion about a route by commenting or rating the route.

Characteristics: Generally, climbers are members of the climbing club, which means that the full diversity of members in the club is represented in this group of actors. From the PACT analysis in section 3.2, we know that the members are in the ages of 10 to 60 years old and that there is a slight gender bias towards men. Climbers that provides beta are most likely experienced climbers.

Examples: Climber A is a 24-year-old member of the climbing club who visits it once every two weeks. He has two years of experience with climbing and is mainly interested in using the system to figure out which routes, if any, have been created since his previous visit. This means that he only uses the system to get an overview of routes, by sorting them by date.

Climber B is a 16-year-old first-time visitor of the climbing club. She does not have any experience in climbing and, therefore, does not have any knowledge about the different gradings or how to best climb a route. She wants to find routes that fits her beginner level as well as basic instructions on how to climb a certain route.

Climber C is a 42-year-old veteran climber who enjoys climbing difficult routes, especially those made by his good friends. He is interested in finding routes of a certain grade that are also made by specific members of the club. Furthermore, he enjoys rating the routes that his friends have created. Because he is a very skilled climber he also helps other members with both difficult and simple routes by leaving beta for them to see.

Actor Specification 5.1: The Climber actor.

The second actor described in actor specification 5.2 represent the people setting, editing or deleting routes in the system.

Setter

Goal: A member of the climbing club, who creates or sets routes for climbers and enters the route information in to the system is a setter. A setter's main goal is to create routes, both in the real world, but also in the system. If an existing route changes, or a mistake was made when entering the route in the system, a setter is also able to edit existing routes.

Characteristics: While a setter technically can be any member of the climbing club, creating new routes requires a level of expertise only experienced climbers have. As mentioned in section 3.2, creating a route requires plenty of climbing experience in order to place each holds in a way that makes the climber of the route, do a specific move.

Examples: Setter A is a 38-year-old veteran member of the club with nine years of climbing experience. He also has extensive knowledge about setting routes, which he does on a regular basis. He uses the system to add these routes to the system.

Setter B is a 26-year-old member of the club, who recently started creating her own routes. As she is relatively inexperienced in route creation, she has a tendency to misjudge the grade of her routes. This means that she uses the system, not only to add her routes, but also to occasionally edit the routes she previously added.

Actor Specification 5.2: The Setter actor.

Lastly, in actor specification 5.3, the most privileged actor of the system, the *administrator* is described. This will most likely be the smallest group of actors. A reasonable choice for *administrator*s would be the elected board members of AKK.

Administrator

Goal: A member of the climbing club, who helps administrate it, by being in control of sections and all the routes they contain. The main goal of an administrator is to apply changes to the available sections, grades and routes in the system.

Characteristics: Only a few members of the climbing club are administrators. Administrators are most likely long-time members or board members of the climbing club. It is likely that people that acts as administrators also acts as climbers and setters in the system.

Examples: Administrator A is a 31-year-old veteran, a long time member of the climbing club and part of the board. When the club clears a section of all routes every two months, Administrator A uses the system to delete all routes in the section.

Administrator B is a 49-year-old long-time member of the climbing club. He rarely uses the system as he would rather leave the task of administrating sections, grades and routes to other administrators. Administrator B prefers only to use the system if it is necessary.

Actor Specification 5.3: The Administrator actor.

5.1.2 Use Case Specifications

This section aims to describe different use case specifications for the new system. The goal of this activity is to find use cases as described by (Mathiassen et al., 2000, p. 127): “The goal is to collect the many possible ways of using the target system in a few well-chosen use cases”.

Authenticating

From the structure activity in section 4.2, we know that a `Member` is associated with a `Route`, `Beta`, `Comment`, and a `Rating`. To model these associations in the system, we choose to identify `Member`s in the system. To do this, we need to authenticate users of the system. Authentication is also used to restrict access to certain functions of the system. For this reason, we want to authenticate users when they first try to access such functionality. The *Authenticating* use case describes how an unauthenticated user becomes an authenticated `Member`.

Authenticating

Use Case: Authentication is possible for anyone who is not already authenticated. It starts when the actor tries to access a function that is only usable for authenticated users. The users are sent to the Log in page. Here, the actor can enter their user credentials and press the Log in button which, if the credentials are valid, sends the actor on to their intended destination. If the log in fails, the actor will be prompted to try again. If the actor does not yet have a login, they can click the Register button. This sends the actor to the Register page where the actor can create a user in the system. After successfully creating a user, the actor is sent to their intended destination.

Objects: Climber, Setter, Administrator, Member.

Functions: Log in, Log out, Register member.

Use Case 5.1: Describes how a user goes from the unauthenticated state to an authenticated state.

Figure 5.1 shows the statechart diagram for the *Authenticating* use case. In the following statechart diagrams for the different use cases the black dot indicates where the use case is initiated, and the black dot surrounded by a circle describes when the use case is terminated.

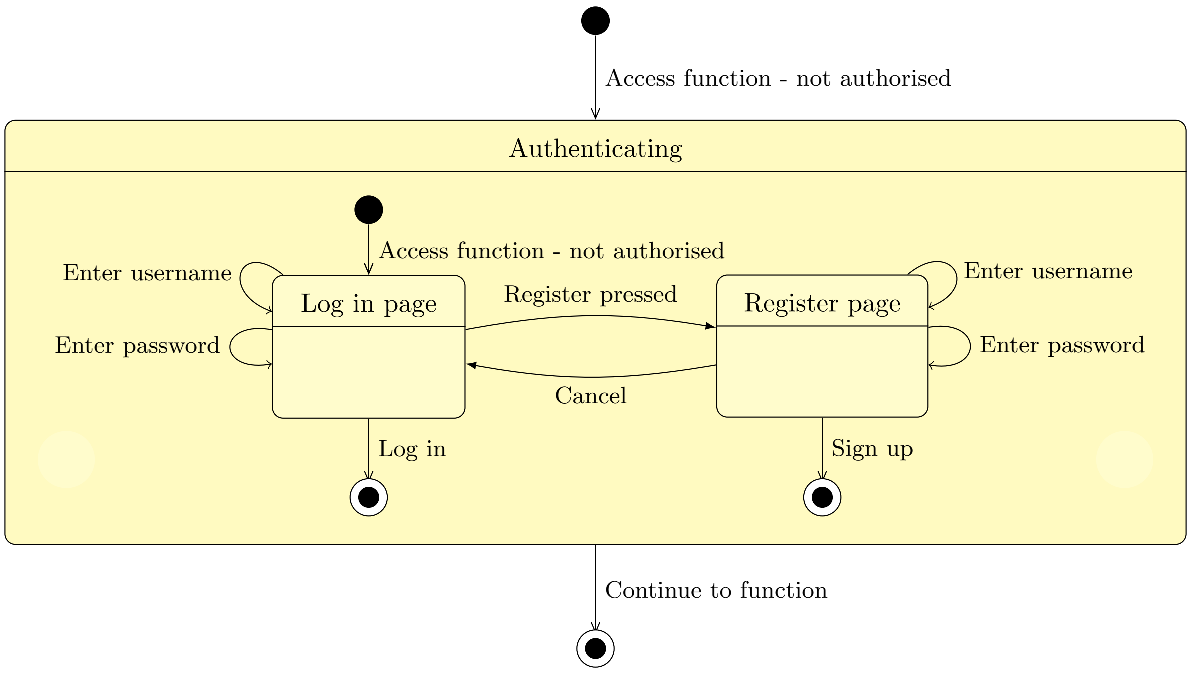


Figure 5.1: Statechart diagram for the use case *Authenticating*.

Viewing Route Information

The *Viewing route information* use case, is the activity of finding information about a route. For most of the users, especially climbers, this will be the key functionality of the system. The actors for this use case are *Climbers*, *Setters* and *Administrators.* The *Viewing Route Information* use case describes how an actor accesses information about specific routes, and how they can make use of filters, searching, and sorting options to help find the wanted route.

Viewing Route Information

Use Case: Viewing Route Information is available to all actors. It is initiated by an actor that accesses the Find route page that lists all routes. The actor can choose to filter routes by section or grade and sort by rating, newest, oldest, grade or author. Actors can also choose to input a string into a search-field, which allows the actor to find all routes which have a certain relevance to the string they are searching for.
At the Find route page the user can select one of the routes which will navigate them to the Route information page that shows all the available information about a specific route including average rating, comments, and beta.

Objects: Climber, Setter, Administrator, Section, Route, Grade, Comment, Beta.

Functions: Get list of routes, Sort routes, Search for routes, Filter routes, Get route information, Calculate average rating, Get route comments, Get route beta.

Use Case 5.2: Describes how to find and view information about a route.

As described in section 3.3 the new system must be easy to use, have the same functionality as the predecessor, and enable users to find routes by section, grade, and date of creation. This use case is related to these requirements by digitising the current whiteboard system and providing filtering, searching, and sorting capabilities to make it easy to find routes.

The statechart diagram in figure 5.2, shows the different states the system can be in, and how the actions performed by the actors changes the state of the system.

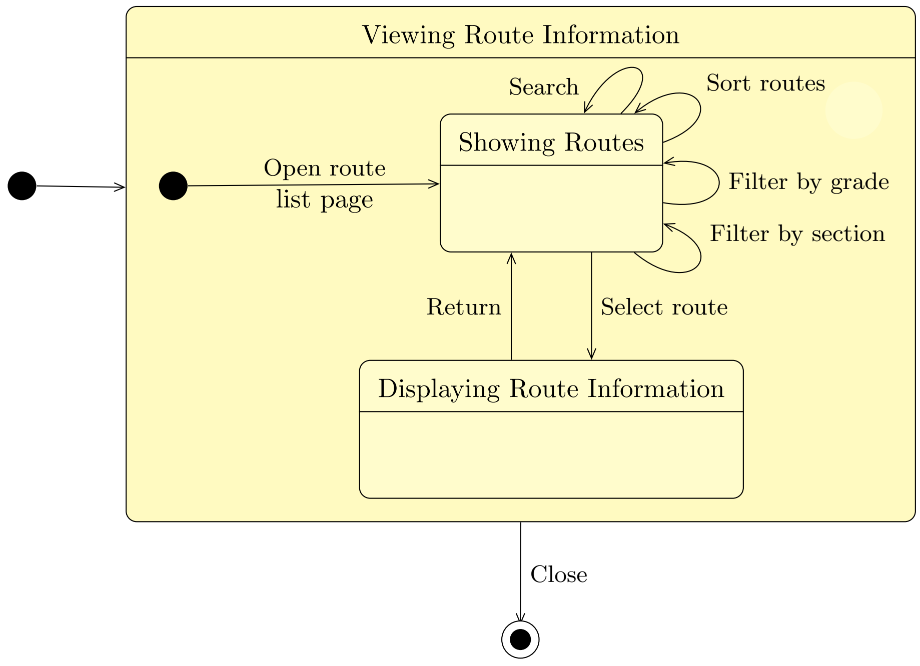


Figure 5.2: Statechart diagram for the use case *Viewing Route Information.*

Adding/Editing Route

This use case describes the activity of adding new routes to the system and helps fulfil the requirement that the system must have the same functionality as the previous.

Adding Route

Use Case: Adding a route is possible for the actors *administrator* and *setter*. The use case is initiated by one of these actors when they open the Add route page. If the actor chooses to add a new route, the system provides input fields for entering information about the new route. This includes the section that the route belongs to, the grade of the route, the number identifying the route, and the colour of holds. It is also possible to select whether the holds have tape on them or not, by pressing a button which changes the holds to include tape. There are no requirements for the order in which this information should be entered.

Objects: Administrator, Setter, Route, Section, Grade, Hold.

Functions: Add route, Get sections, Get grades, Get hold colours.

Use Case 5.3: Describes how to add a new route.

Figure 5.3 shows the statechart diagram of the use case *Adding Route*. Because there are no requirements for the order in which different types of information about the route are entered, then this use case follows the material pattern. The material pattern is a use pattern which is characterised by having little sequence, meaning that the actors have very few limitations in the order in which they perform action. For this use case, the actor can e.g. select grade, colour of holds, section, or enter route number in an arbitrary order. The material pattern is used in situations where no rules limit the way an actor can perform a specific use case. The only difference between adding a new route and editing an existing one, is that the page will already have the current section, grade, hold colour selected, and route number assigned to it, all which can be changed. The *Editing Route* statechart diagram can be seen in figure H.2 in appendix H.

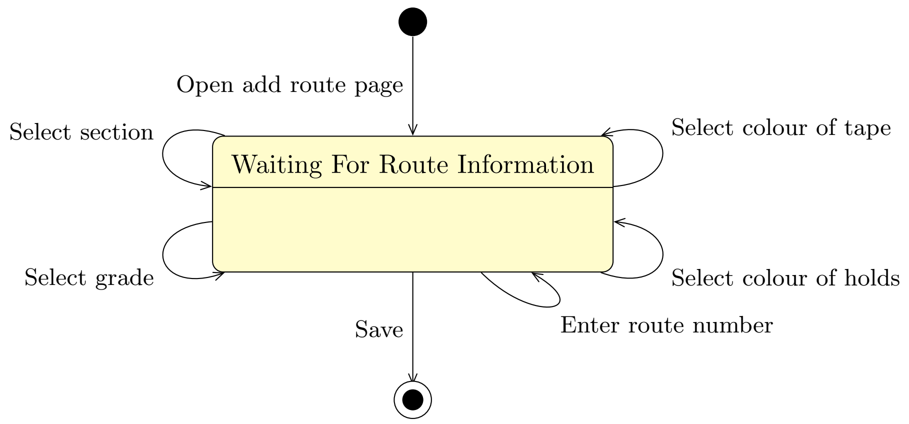


Figure 5.3: The statechart diagram for the *Adding Route* use case.

Adding Additional Route Content

After a route is added, additional content can be attached to the route. This includes ratings, comments, and betas. This use case is based on the could-have requirements in the MoSCoW analysis in section 3.3.

Adding Additional Route Content
Use Case: Adding additional route content is possible for *administrator*, *Setter* and *climber*. This use case is initiated by an actor that accesses the Route Information page. At this page, the actors can rate the route by selecting a score ranging from one to five. The actors can also enter a comment that will be shown below the route information. Finally, the actors can choose to add beta by selecting a video from their device.
Objects: Administrator, setter, climber, route, beta, comment, rating.
Functions: Add rating, Add beta, Add comment.

Use Case 5.4: Describes how actors can add a rating, comment and beta to a specific route.

Ratings can then be used by climbers to quickly find a route that other climbers liked. Comments can be used to discuss and share thoughts about a route. Finally, beta lets climbers who are signed in, share image or video that shows how to climb the route. Use case 5.4 is illustrated with a statechart diagram in figure 5.4.

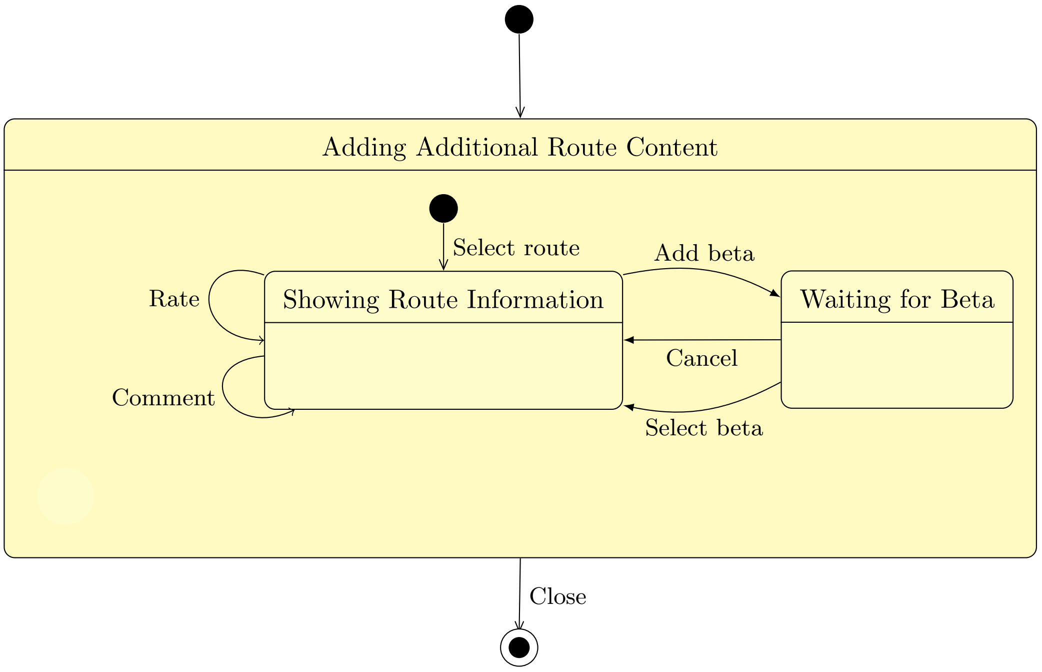


Figure 5.4: Statechart diagram for *Adding Additional Route Content*.

Administrating Sections

Use case 5.5 shows the *Administrating Sections* use case. This use case describes how an *administrator* can control sections by adding, renaming, deleting, or clearing a section. This use case is related to the requirement that the system should be able to handle new sections, meaning that if the physical environment of the climbing club changes, the system should enable *administrator*s to add a new section. This will become important if AKK moves to new buildings.

Administrating Sections

Use Case: Section administration is initiated by an administrator when the Administration page is opened. The system will then list all the sections in the climbing club. The actor can then choose between four actions: add a new section, change the name of a section, clear a section, or delete a section. If the actor wants to add a new section or change the name of an existing section, then the system prompts the user to enter the name of the section. The actor now has the option to add a name or just cancel the requested action. If the administrator chooses to delete or clear a section, then the system will inform the user that all routes within the section will be removed. The actor can then confirm or cancel the requested action.

Objects: Administrator, Section, and Route.

Functions: Add section, Clear section, Delete section, Change section name.

Use Case 5.5: Describes how actors can add, rename, clear and delete sections.

The *Administrating Sections* use case is illustrated in figure 5.5 and follows the material pattern to perform different actions on sections.

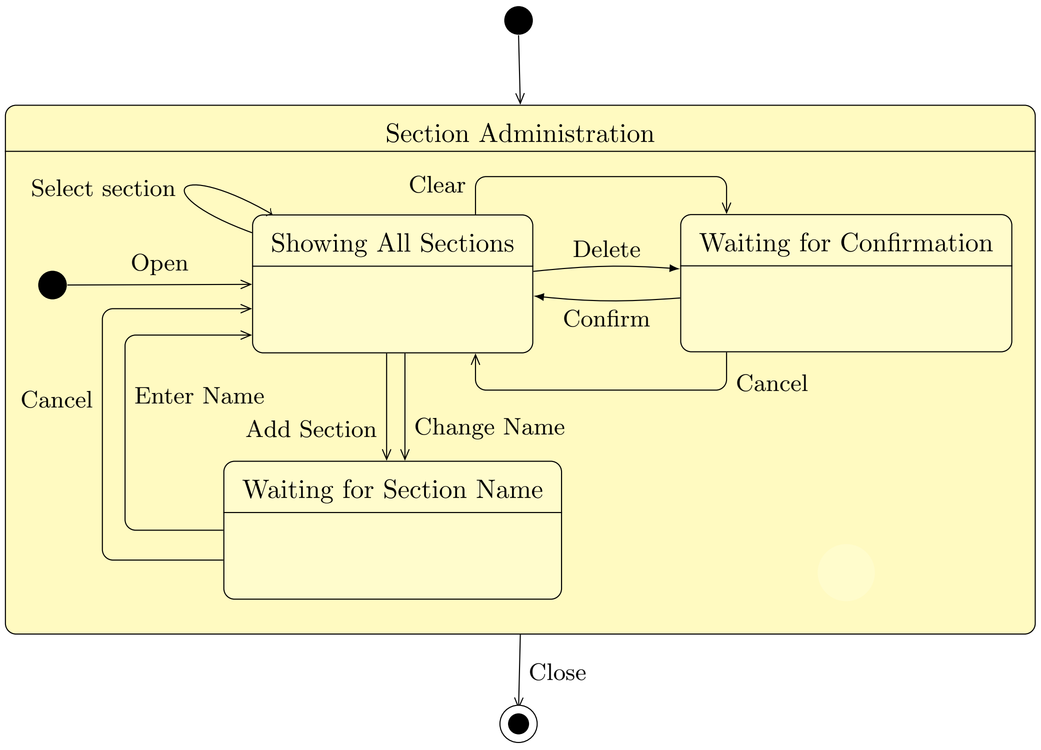


Figure 5.5: Statechart diagram for the use case *Administrating Sections*.

Deleting Route

Use case 5.6 explains how a *setter* or *administrator* can delete a specific route from a section in the system.

Deleting Route

Use Case: Deleting a route is possible for two actors: *administrator*s and *setter*s. On the Route Information page, the actors can delete the route. If they choose to do so, they will be prompted with a confirmation box with OK and Cancel buttons so that they do not delete the route by accident.

Objects: Administrator, Setter, Route.

Functions: Delete route.

Use Case 5.6: Describes how actors can delete a route.

The statechart diagram for the use case *Deleting a Route* is shown on figure 5.6.

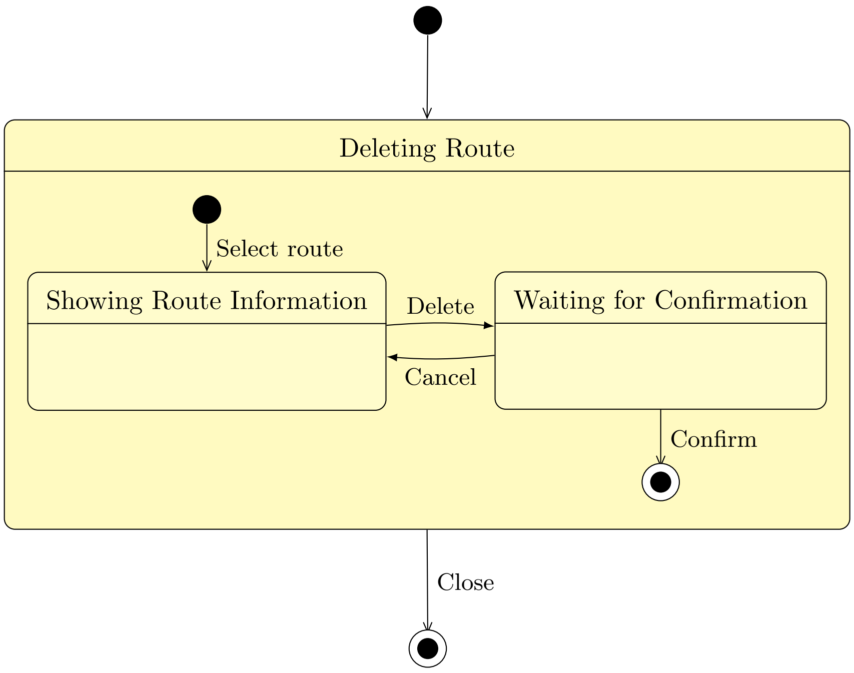


Figure 5.6: Statechart diagram for the use case *Deleting Route*.

Adding Image to Route

Based on the requirement that the system should allow members to add an image of a route, a use case for adding image to a route was introduced. Additionally this use case allows the *climber*, *setter*, and *administrator* to mark where the holds of a route are located on an image of the route.

Adding Image to Route

Use Case: When viewing route information, it is possible to add an image to a route. When choosing to add a new image, the actor has to choose whether to choose an existing photo, or to take a new one with their phone. After the image has been chosen, the actor can click on the holds on the image so that the system can draw the route on top of the image to make it easier for climbers to identify the route.

Objects: Climber, administrator, setter, Route, Image, Hold.

Functions: Add image, Add holds, Delete latest added hold.

Use Case 5.7: Describes how to add image to a route.

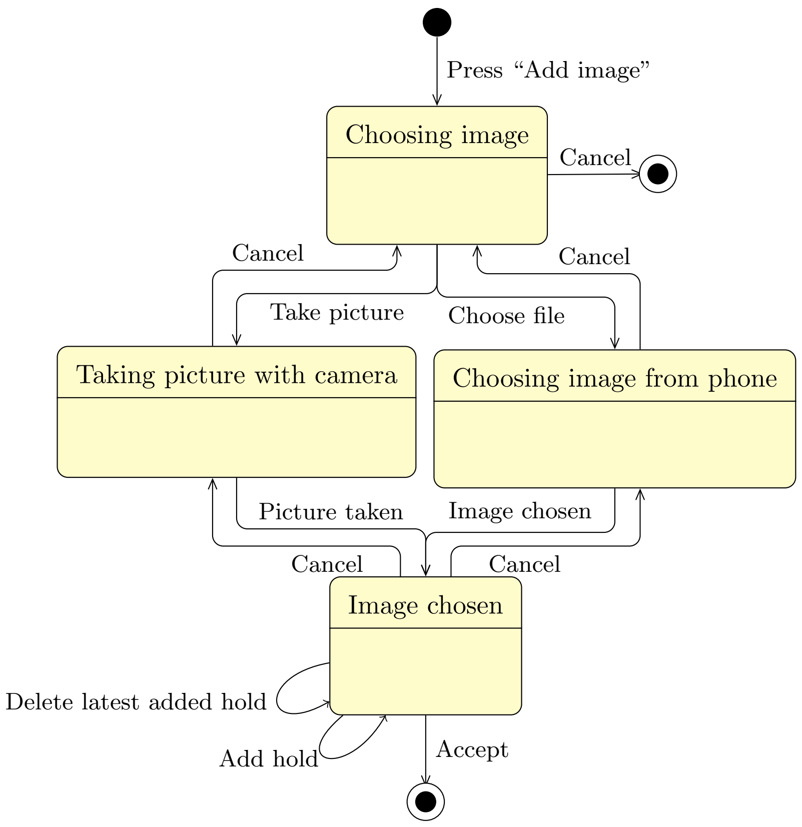


Figure 5.7: Statechart diagram for the use case *Adding Image to Route*.

5.1.3 Actor Table

Table 5.1 shows the link between the identified actors and identified use cases. The table shows which actors can perform which use cases.

	Actors		
Use Cases	Climber	Setter	Administrator
Viewing Route Information (*)	✓	✓	✓
Adding Additional Route Content	✓	✓	✓
Adding Image to Route	✓	✓	✓
Authenticating	✓	✓	✓
Adding Route		✓	✓
Edit Route		✓	✓
Delete route		✓	✓
Section administration			✓

Table 5.1: Actor table. The asterisk indicates that a use case does not require authentication

As can be seen from the table the use case *Viewing Route Information* does not require that the actors is authenticated in the system. The reasoning behind this choice, was that we did not want to force users of the system to login if they just wanted to find or get an overview of routes.

5.2 Functions

In the following section, we determine the functions of the new system and summarise them in a table, showing the category of each function as well as their complexity.

5.2.1 Defining Functions of the System

In this section, we describe functions for the system based on the use cases described in section 5.1.2. We categorise all the functions into three groups of functions, which are read, update, and compute functions. For each function, we also estimate how complex the different functions will be to implement. The result is a list of all primary functions in the system, which will show the type and complexity of each function.

Read functions

The first type of functions to be identified are read functions. The fact that these functions can read and return the current state of the model of the problem domain is what they have in common. From the use cases, we can identify the functions that receive information which are the following get-functions: `Get list of routes`, `Get route information`, `Get route comments`, `Get route beta`, `Get grades`, and `Get sections`. All these functions, except `Get route beta` , are simple because they only read and return text based information. `Get route beta` is estimated to have medium complexity because it requires the function to read and return videos which might be more difficult to handle than simple text-based information.

Update functions

The second type of functions are update functions which change the state of the model. This includes the following functions that add or update objects to the model: `Add route`, `Add hold` , `Update route`, `Add beta`, `Add section`, `Add rating`, `Add comment`, and `Register member` . These add functions, except `Add beta`, are estimated to be simple functions to implement, whereas the `Add beta` function is estimated to be complex because it is more challenging to handle video input than simple text input. In contracst to these add functions are the update functions that clear or delete data from the model: `Delete route`, `Delete latest added hold`, `Delete section`, and `Clear section`. These functions are categorised as medium complexity as they change the state of multiple classes. When a `Section` is deleted, it deletes all the `Route` s contained within that `Section` and deleting a `Route` deletes all the additional content such as ratings, comments and betas. Furthermore, the function `Change section name` is simple to implement because it only updates a single attribute of a section object. The `Add image function` is estimated as medium as it might be challenging to add an image depending on how the data is stored.

Compute functions

The third type of functions are compute functions which perform some calculation in the system. In our system, there are four compute functions: `Sort routes`, `Filter routes`, `Calculate average rating` and `Search for routes`. Sorting a list of items is a common computational problem inside computer science, and therefore there exist several implementations that we can use to sort a list of routes. From this we estimate this function to be simple to develop, which is also the case for `Filter routes`.

The third function is `Calculate average rating` that will be presented to the user together with other route information. This function is also simple as it just calculates the average rating for a specific route using simple math.

The last function, `Search for route`, requires a working search function that searches through each route and their properties to give back a result that best matches the input string. There exists algorithms that can do this, such as Levenshtein, which is an algorithm that we can use to compute the distance which is the similarity between the inputted string, and each route in the model (Allen, 2016). Levenshtein along with other possible methods for our search function will be described in section 8.1.5. The routes can then be sorted by the lowest distance value, which means the routes will be sorted by which routes closest matches the inputted search string, then returned to the caller of the function. Unlike the other functions, we did not know that search functionality was necessary until the first usability test, so this requirement has been derived from appendix D. Even though that searching is a well defined task which can be implemented with the use of already existing algorithms it is still a complex function, since we must take into consideration different search criteria as well as computational time in order to make the search function.

To implement authentication as described in use case 5.1 we need the two functions: `Log in` and `Log out`, which is used to authenticate different actors in the system. These functions have simple complexity under the assumption that we choose a technical platform that already has functionality for authenticating users.

5.2.2 List of Functions

From the previous section, we now combine a list of all read, update and compute functions as shown in table 5.2.

Functions		
Get list of routes	Simple	Read
Get route information	Simple	Read
Get route comments	Simple	Read
Get route beta	Medium	Read
Get grades	Simple	Read
Get hold colours	Simple	Read
Get sections	Simple	Read
Add route	Medium	Update
Add beta	Medium	Update
Add section	Simple	Update
Add rating	Simple	Update
Add comment	Simple	Update
Add image	Medium	Update
Add grade	Simple	Update
Add holds	Simple	Update
Delete route	Medium	Update
Delete grade	Simple	Update
Delete latest added hold	Simple	Update
Delete section	Simple	Update
Clear section	Medium	Update
Change section name	Simple	Update
Register member	Simple	Update
Swap grades	Simple	Update
Sort routes	Simple	Compute
Filter routes	Simple	Compute
Calculate average rating	Simple	Compute
Search for routes	Complex	Compute
Calculate distance	Complex	Compute
Sort routes by distance	Simple	Compute
Log in	Simple	Compute
Log out	Simple	Compute

Table 5.2: List of all functions' name, complexity and type.

Summary

In this chapter three types of actor were identified: *climber*, *setter*, and *administrator*. The primary goal of the *climber* was described as being able to find, rate, view, and add beta to a route. *Setter*s were specified as actors who use the system to create, remove, and edit routes. Finally, the *administrator* was specified as an actor with the goal of managing routes and sections in the system.

Seven use cases were identified in the in the system and specified in use case specifications: *Authenticating*, *Viewing Route Information*, *Adding Route*, *Adding Additional Route Content*, *Adding Route Image*, *Delete Route*, and *Section administration*. An actor table, as seen in table 5.1, was created to illustrate which actors could perform which use cases. Lastly, read, update, and compute functions were identified in the system, as showed in table 5.2.

Having obtained knowledge of which actors and use cases were present within the system, it became possible to formulate and design the architectural design of the system, as described in the following chapter.

Chapter 6

Architectural Design

In this chapter, the architectural design of the system is in focus. First, common criteria for system architecture is looked at and then prioritised according to the established requirements of the system. Then the the choice of which environment to develop in and frameworks to use, is made. The overall architecture layers and components in the system is explained, with the components being described in detail. Specifically, which classes each component consists of. Using the class diagram and event table from chapter 4, the model component, where events are represented by attributes and new classes, is constructed. The information flow of the system is then illustrated, and finally, the connection between different classes in the components is shown in order to illustrate the dependencies between the components and layers of the system.

6.1 Criteria

In this section, we describe each criterion and explain how important it is for our system, then summarise the section with a table showing the same information for a quick overview of each individual criterion's importance.

6.1.1 Prioritising

To figure out what criteria need the most focus, we use a checklist from Object Oriented Analysis & Design (Mathiassen et al., 2000, p.185).

Each criterion is rated on a scale from very important to irrelevant as well as an easily fulfilled option, based on how important they are in the system. The reasoning for the prioritising of the requirements is based on the requirements found in 3.3. Each criterion will be briefly described and given an explanation for their prioritising in table 6.1.

Usable

Can the system be used by the user in the desired context to solve their problem? Based on the requirement that the *system should be easy to use*, we rate this criterion as 'very important', since the two are closely related.

Secure

How important is it that the system is secure against unauthorised access? In our system, we do not have any sensitive personal information, except for passwords, that can be lost, nor any specific requirement for the system to take special precautions against unauthorised access. Other than storing the members passwords securely, we still rate this criterion overall, as being 'less important'.

Efficient

Is it important that the system is efficient to use? Based on the requirement that the *system should be easy to use*, it can be deduced that for this to hold, the system must also be efficient, else it would require many resources to perform a task in the system, and thus contradict the requirement and make the system less easy to use. This criterion is therefore rated as being 'important.'

Correct

Does the system correctly fulfil the requirements? From the requirement that the *system must have the same functionality as the current system*, it can be deduced that the system must work at least as correct as the current system. Therefore, this criterion is 'important.'

Reliable

Is the functionality of the system reliable, can we depend on it to produce a precise output every time and not cause errors or terminate? In the PACT analysis, it was mentioned that the members of AKK were able to go climbing whenever they wanted to, and that the system, for this reason, should be reliable. However, since the members of AKK do not depend on the system in order to climb, but only to administrate routes and sections, this criterion is rated 'important', instead of 'very important'.

Maintainable

Should it be easy to locate and fix system errors? Even though maintainability from a programmer's perspective always seems important, we have no requirements or other things from our analysis that states this, and this criterion is therefore deemed as being 'less important'.

Testable

Should we be able to easily test and determine if the system works as intended? From the study curriculum (Thomsen, 2015), we can see that we must be able to systematically test our program and ensure that it solves the client's problem. This criterion is, therefore 'very important' for this project.

Flexible

Is it important that the system is flexible and can handle changes in the context? In the PACT analysis, we derived a requirement which stated "*the system should be dynamic in regards to the context*". For this to be solved, the system must be able to cope with changes in the context and this criterion is therefore rated as being 'important'.

Comprehensible

Should the users of the system easily be able to get a coherent understanding of the system? Based on the requirement that the *system should be easy to use*, we can derive that the system also must be coherent and for this reason, we prioritise this criterion as 'very important'.

Reusable

Should the system or parts of the system be usable in other systems? There is no requirement that our system should be reusable and the criterion can therefore be deemed as 'irrelevant'.

Portable

Should it be easy to move the system to another technical platform? In section 6.2, it will be mentioned that we want to develop in C# for the web. However, it was also mentioned in the system definition, that it should work on tablets and smartphones, which means that it is not unrealistic that the system will be ported to another technical platform in the future. However, it is not a focus for this particular project and we therefore consider this requirement to be 'less important'.

Interoperable

Is it important that the system can be coupled to other systems? No requirements or analysis states that the system should be interoperable and this criterion is therefore rated as being 'irrelevant'.

Summary

All of the previous requirements and their priority for the system have been summarised in table 6.1.

Criterion	**Very important**	**Important**	**Less important**	**Irrelevant**	**Easily fulfilled**
Usable	×				
Secure			×		
Efficient		×			
Correct		×			
Reliable		×			
Maintainable			×		
Testable	×				
Flexible		×			
Comprehensible	×				
Reusable					×
Portable			×		
Interoperable				×	

Table 6.1: Prioritising of design requirements

6.2 Technical Platform

In this section, we describe how the system is interacted with, as well as on what platform it is interacted through. We also mention what languages the system is programmed in, and why we made this decision.

6.2.1 Platform and Interaction

As specified in section 3.3, the system has to be mobile, and therefore, we develop the system for mobile devices such as smartphones and tablets. If someone at the climbing club does not own such a device, replacing the current whiteboards with one or more tablets is a possibility for the club, so everyone at the club has access to the system. Being able to use the system on a PC would also be ideal, but it is not a necessity.

The system is interacted with using a touch-based device and such a device is therefore a requirement. We focus on developing the application so it works on both Android and iOS. Developing a web-based application makes it work easily on both devices, and even make it work on PCs too. Since the application requires input in form of touch, there is no elements that can be interacted with by hovering.

6.2.2 Programming Languages and Frameworks

We develop the system using Microsoft C#, because we all have experience using that language, and because we develop the system in the object-oriented paradigm according to the curriculum (Thomsen, 2015, p.21). To develop the system, we use the cross-platform framework ASP.NET Core. This is because it enables both Windows, Mac and Linux users in the group to develop, build, and run the application using .NET Core (Microsoft, 2016b). ASP.NET Core also includes the MVC (model-view-controller) framework, which can be used to construct a layered architecture similar to the simple architecture proposed in OOA&D with model, function, and interface layer (Mathiassen et al., 2000, p.10). Because we create a web-based application, we use HTML, CSS, and JavaScript to develop the user interface. We also use Handlebars.js which is a templating engine which is used to render templates for the user interface, that can be reused across multiple pages and show more dynamic content than static HTML pages (Katz, 2016). Finally, we use JQuery for interaction between JavaScript and HTML elements, as well as sending requests to the server-side API.

To handle the data of the system, we use Microsoft Entity Framework Core which is a part of .NET Core, as this offers the possibility of using Code First databases. This allows us to define our database structure directly in our programme, instead of manually writing database queries (EntityFrameworkTutorial, 2016).

6.3 System Architecture

As mentioned in section 6.2, our system is a web-application and thus makes it obvious to base our architecture on the client-server architecture pattern (Mathiassen et al., 2000, p.197-201). This divides the system into two main components; the client and the server, as seen on figure 6.1. The client then consists of three components: ClientService, ViewModel, and View. The server consists of a controller and model component. The architecture for our system is illustrated on figure 6.1, where the arrows indicate that the component pointed by is dependent of the component pointed to.

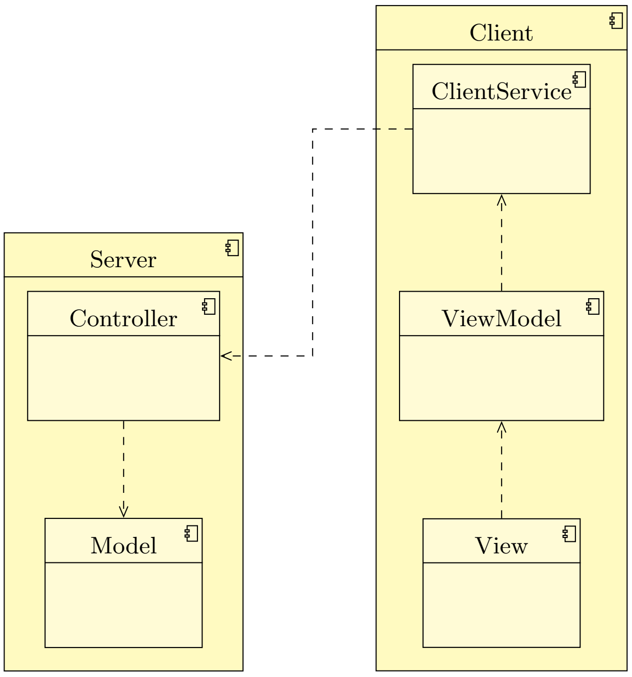


Figure 6.1: Overview of system components. Arrows indicate a dependency.

In the following sections we will describe the design of the five components: `Model`, `Controller`, `ClientService`, `ViewModel`, and `View`.

6.3.1 Model Component

The model component contains the classes, attributes and relations for representing the problem domain in the system. In this section, we design the model component based on the events and classes found in the problem-domain analysis. First, we determine how events and their attributes should be presented in the model based on the event table in section 4.1.2. The only event that has attributes in our system is the *Route created* event which has an attribute representing the date the route was added. Table 6.2 shows how this event affects different classes.

	Classes							
Events	Section	Route	Rating	Comment	Beta	Member	Grade	Hold
Route created	∗	+				∗	∗	+

Table 6.2: Excerpt from table 4.3 showing the event *Route created*.

Because the *Route created* event only occurs once for an object in the `Route` class we can simply add an attribute on the `Route` class to store the date of creation.

Model Component Structure

To design the structure of the model component we make adjustments to the class diagram derived in section 4.2 to use it during the implementation activity.

From the use case specification for adding an image to a route (figure 5.7), we have chosen to model holds as being part of an image of a route, since this allows us to show the holds of the specific route. We therefore introduce a new class called `Image` that aggregates zero or more `Holds`.

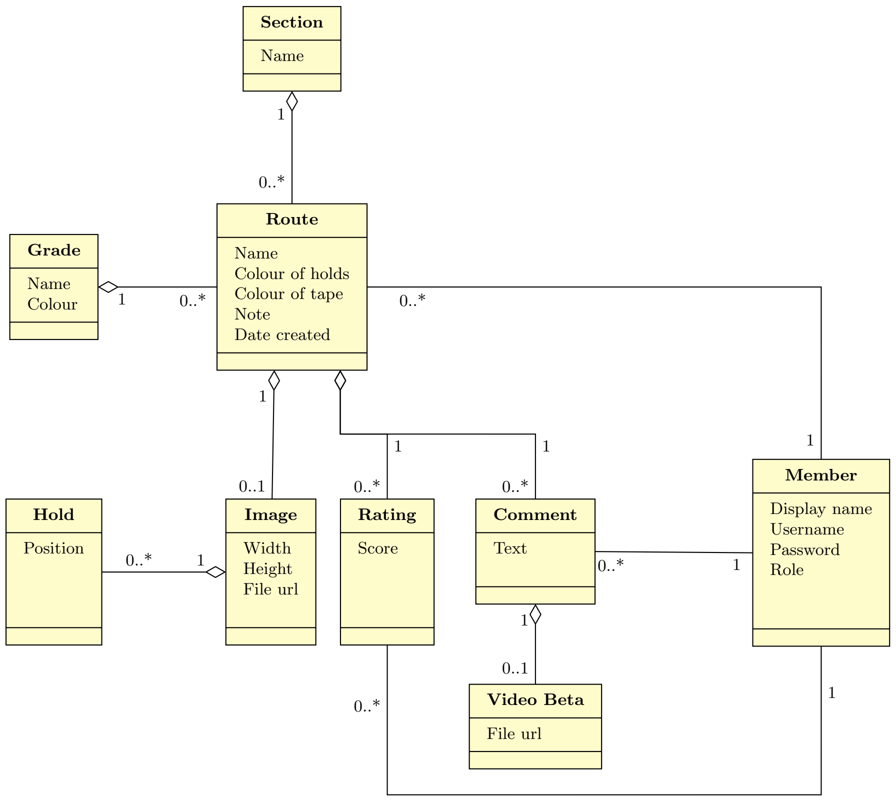


Figure 6.2: Model structure.

In this activity, we made a structural change compared to the class diagram found in section 4.2. After interviews with the climbing club, we found that the only reasonable format that a beta would be given in, was video. However, after our third usability test, as described in 10.4, we found that most members when creating a beta, did so by creating a comment, not uploading a video. Also, we found that beta is information you give to each other, so one climber might ask another to "give the beta" to a certain route. To represent this in our system, we have chosen to model a video beta as being a part of a comment. This means that a beta is now an attachment to a comment, instead of being a completely separate object.

By modelling it this way, we can create a feed for each route where members can share beta in form of comments and videos with each other.

6.3.2 Controller Component

The controller component consists of several controllers, one for each table in the database. Each controller is responsible for keeping the model up-to-date by making functionality available to the client service component which then reports changes that needs to be updated in the model. The controller also interprets information in the model in a way that is advantageous to the client by for example filtering irrelevant routes by request of the user. The different controllers and their functions are illustrated on figure 6.3. The methods in the controllers are based on the functions found in the application-domain analysis in section 5.2.2.

The controllers respond to REST-like HTTP requests. REST (Representational state transfer) is an "architectural style for distributed hypermedia systems". *"The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components"*(Fielding, 2000, Ch. 5). This uniform interface is achieved by following conventions of requests for interacting with the system as a service. For this, the URL is used to indicate what part of the model you want to operate on and HTTP methods or verbs like GET, POST, PATCH, and DELETE are used for the different operations that can be used on the model. These are read, write, update, and delete. The intention is that you are able to interact with any REST service without knowing anything about the service. In reality this goal can be difficult to achieve.

URL	Verb	Method
/api/section	GET	GetAllSections
/api/section	POST	AddSection
/api/section	DELETE	DeleteAllSections
/api/section/{id}	GET	GetSection
/api/section/{id}	PATCH	UpdateSection
/api/section/{id}	DELETE	DeleteSection
/api/section/{id}/routes	GET	GetSectionRoutes
/api/section/{id}/routes	DELETE	DeleteSectionRoutes
/api/route	GET	GetRoutes
/api/route	POST	AddRoute
/api/route	DELETE	DeleteAllRoutes
/api/route/{id}	GET	GetRoute
/api/route/{id}	PATCH	UpdateRoute
/api/route/{id}	DELETE	DeleteRoute
/api/route/{id}/image	GET	GetImage
/api/route/{id}/comment	POST	AddComment
/api/route/{id}/comment/{commentId}	DELETE	RemoveComment
/api/route/{id}/rating	PUT	SetRating
/api/grade	GET	GetAllGrades
/api/grade	POST	AddGrade
/api/grade/{id}	GET	GetGrade
/api/grade/{id}	PATCH	UpdateGrade
/api/grade/{id}	DELETE	DeleteGrade
/api/grade/{id1}/swap/{id2}	PATCH	SwapGrades
/api/holdColor	GET	GetAllHoldColors
/api/holdColor	POST	AddHoldColor
/api/holdColor/{id}	PATCH	UpdateHoldColor
/api/holdColor/{id}	DELETE	DeleteHoldColor
/api/member	GET	GetAllMembers
/api/member	POST	AddMember
/api/member/{token}	GET	GetMemberInfo
/api/member/{token}/ratings	GET	GetMemberRatings
/api/member/login	GET	Login
/api/member/logout	GET	Logout
/api/member/role	GET	GetRole
/api/member/role	PATCH	ChangeRole

Table 6.3: Controller methods with the associated request URL and HTTP verb.

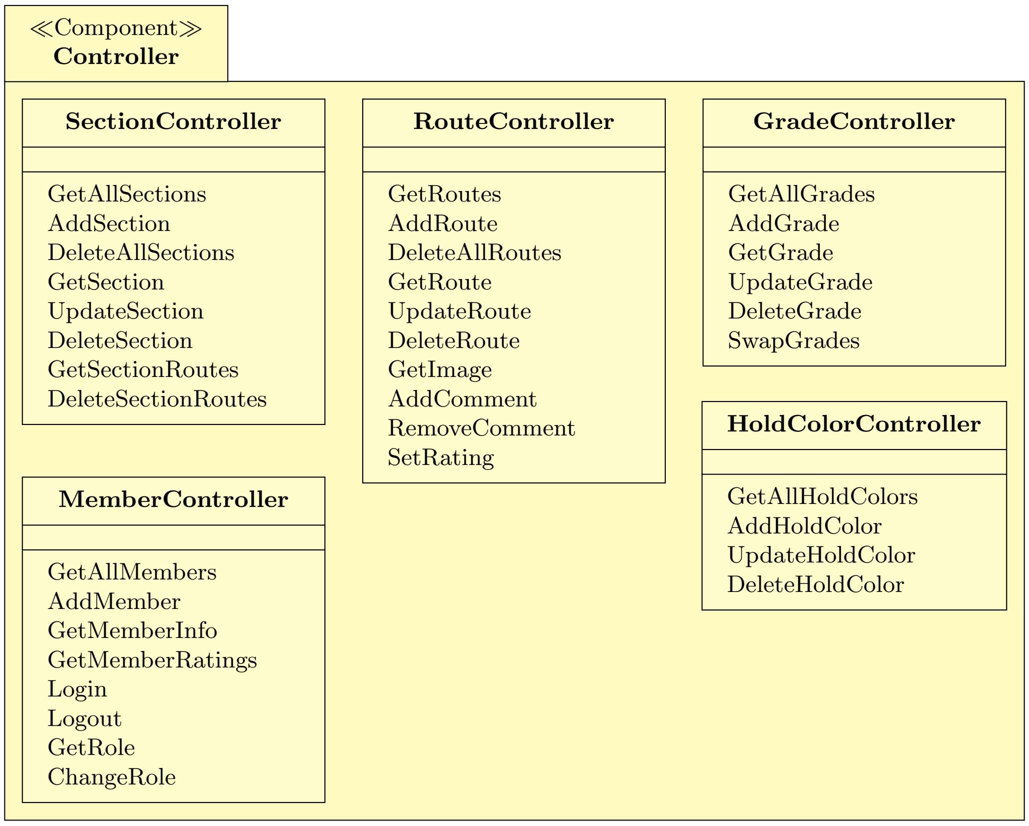


Figure 6.3: Illustrates the different controllers in the system. The methods implements the functionality listed in table 5.2 that was made as a part of the application-domain analysis.

6.3.3 Client Service Component

The client service is responsible for sending and receiving data to and from the model through the controller. The component consists of three clients, each of which is responsible for communication with its corresponding controller. The calls to the client service are facilitated by the view models.

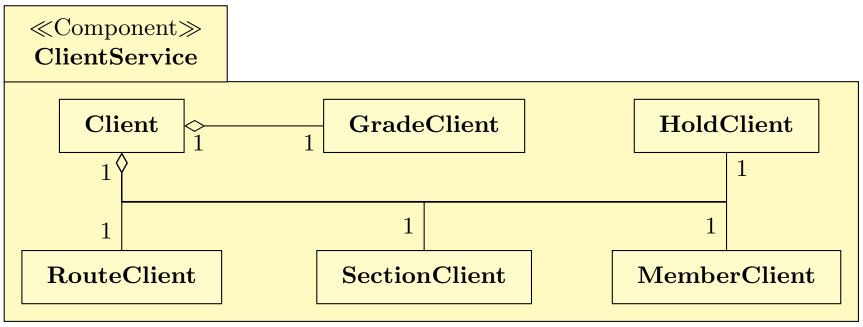


Figure 6.4: Illustrates the clients of the system. The methods of each client are equivelent to the methods on the controllers in figure 6.3

6.3.4 View Model Component

The ViewModel component contains logic behind the user interface and is responsible for updating the state of the data for the user interface. The ViewModel component uses the ClientService component to send and receive data according to the actions performed through the methods in the ViewModel. The second responsibility of the ViewModel is to trigger events when the state the ViewModel changes to make other classes able the observe changes in the ViewModel. Figure 6.5 shows the ViewModel components and their associated methods.

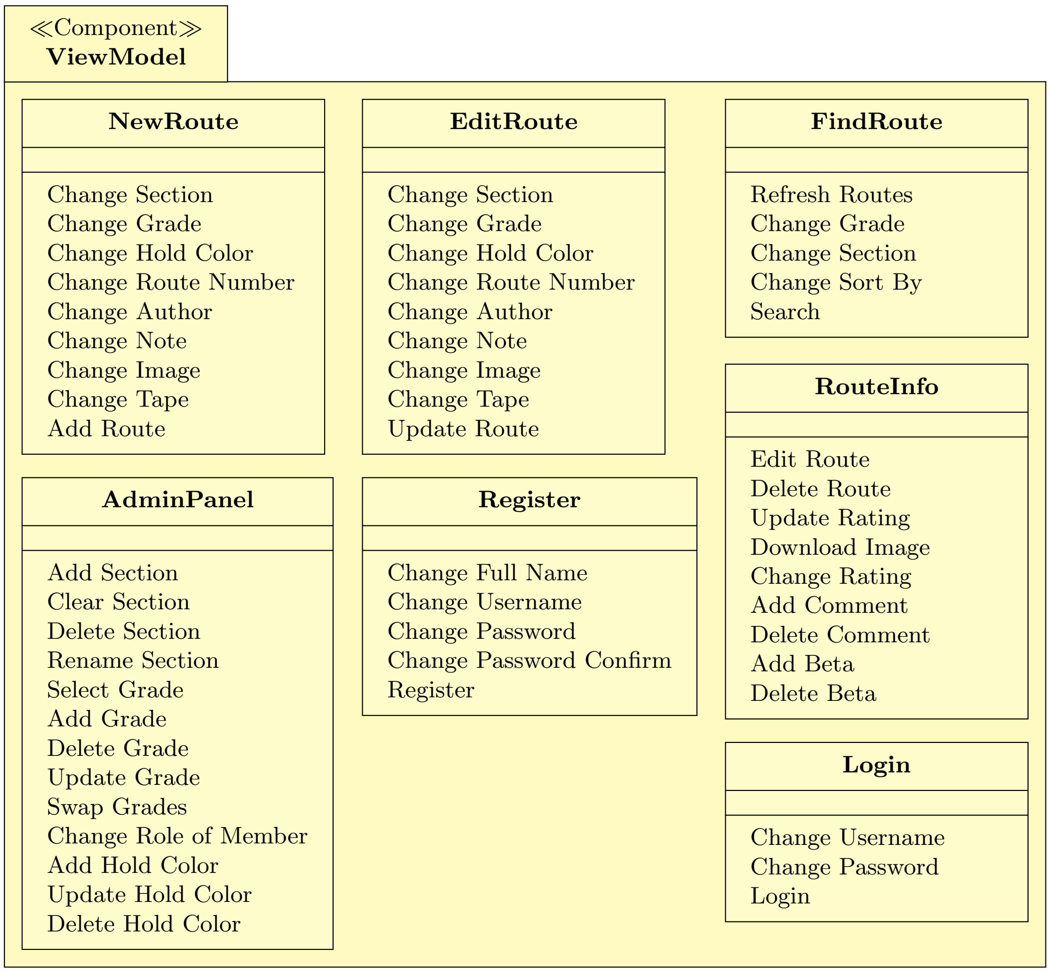


Figure 6.5: ViewModel Component containing different ViewModels and their methods.

6.3.5 View Component

The View component is responsible for presenting the state of a ViewModel component. A view in the View component connects buttons, text-fields, images, and other user interface component to create a page that enables the user to interact with the corresponding view model in the ViewModel component. Figure 6.6 shows the different views in the system which are described below.

The FindRoute view is made from the use case *Viewing Route Information* on figure 5.2, and corresponds to the part of this use case that is showing routes to the actors based on different filters and sort options.

The RouteInfo view is the second part of the *Viewing Route Information* use case and shows information about a specific route.

The EditRoute view corresponds to the use case for editing a route as seen on figure H.2 in appendix H. When the edit button from the RouteInfo view is clicked, the viewmodel will redirect the user to the EditRoute view where the user can change the content of a route e.g. its grade.

The AdminPanel view corresponds to the use case illustrated on the statechart diagram for section administration on figure 5.5. From the AdminPanel view, the user can add new sections. Furthermore, a specific section can be selected, and when selected the user can edit, clear, or remove the section, as well as view all routes that are part of the selected section.

The NewRoute view corresponds to the use case *Adding Route*, and allows a member to add a new route. When the Add New Route button is tapped, the user can add all the necessary information required for the route, then tap the Add button, which sends a request to the controller, and then, adds the new route.

The LogIn view corresponds to the first part of the *Authenticating* use case. The view is accessed, either by accessing the LogIn view by pressing the LogIn button, or by accessing a feature which requires authentication. The user can become authenticated, once they have typed in their username and password, and press the log in button.

The Register view covers the last part of the *Authenticating* use case. If the user taps the Register button, they will see a screen with four fields. This allows the user to type in their display name, their username, and their password twice. In order to register, the username will have to be unique, and the passwords have to match each other, to ensure the user did not mistype their password.

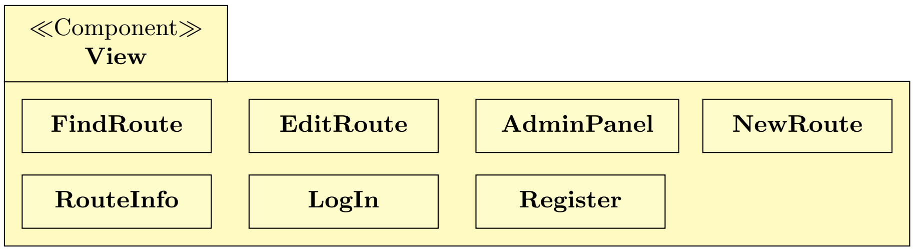


Figure 6.6: A list of the views that make up the user interface.

6.4 Connecting Components

Figure 6.7 shows an example of the flow of information in our system architecture. The process starts with the user selecting a filter on the view to see only certain routes. The event of selecting this filter is handled by the viewmodel which then in turn queries the client to fetch the filtered routes from the server. The client handles the connection to the server, and passes along the query to the correct controller on the server. The controller can then fetch the routes from the model, before returning it down the chain until it reaches the viewmodel again. The viewmodel will then update the Document Object Model (DOM) of the view to display the updated routes.

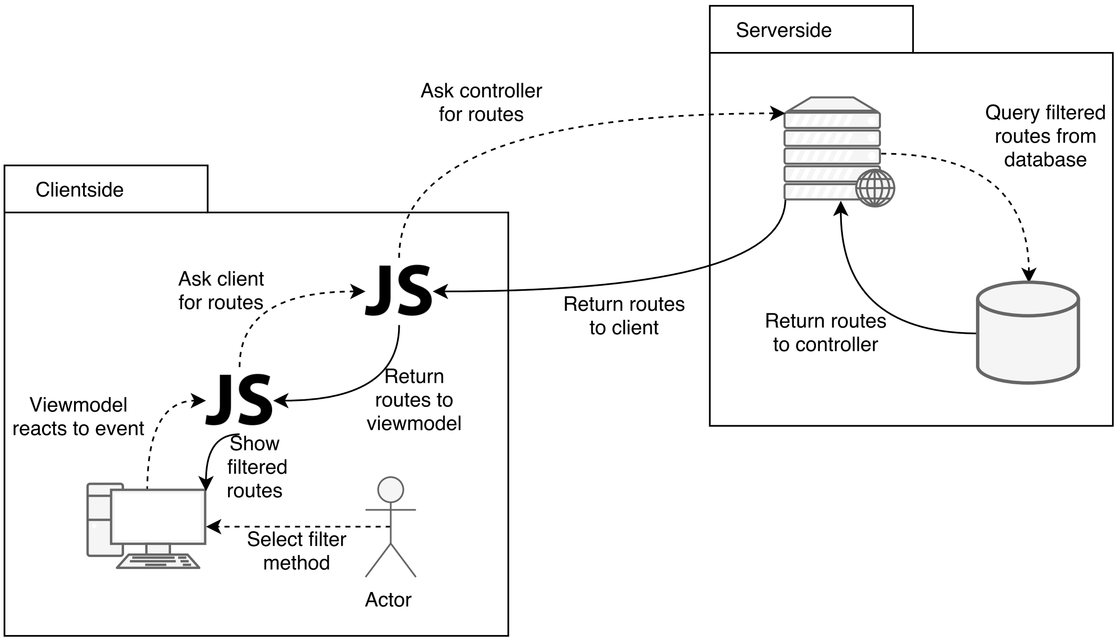

Serverside
Ask controller for routes
Query filtered routes from database
Clientside
Ask client for routes
JS
Return routes to client
Return routes to controller
Viewmodel reacts to event
JS
Return routes to viewmodel
Show filtered routes
Select filter method
Actor

Figure 6.7: Illustration of information flow in our architecture. The process starts at the actor and sends a query following the dashed lines before returning the data represented as the solid lines.

This information flow can also be represented with dependencies between the different components in our architecture. The components with their dependencies are illustrated in figure 6.8. The figure is an extension of figure 6.1, which illustrated the overall system architecture components. This extended figure is an overview of the different classes in each component of the new system.

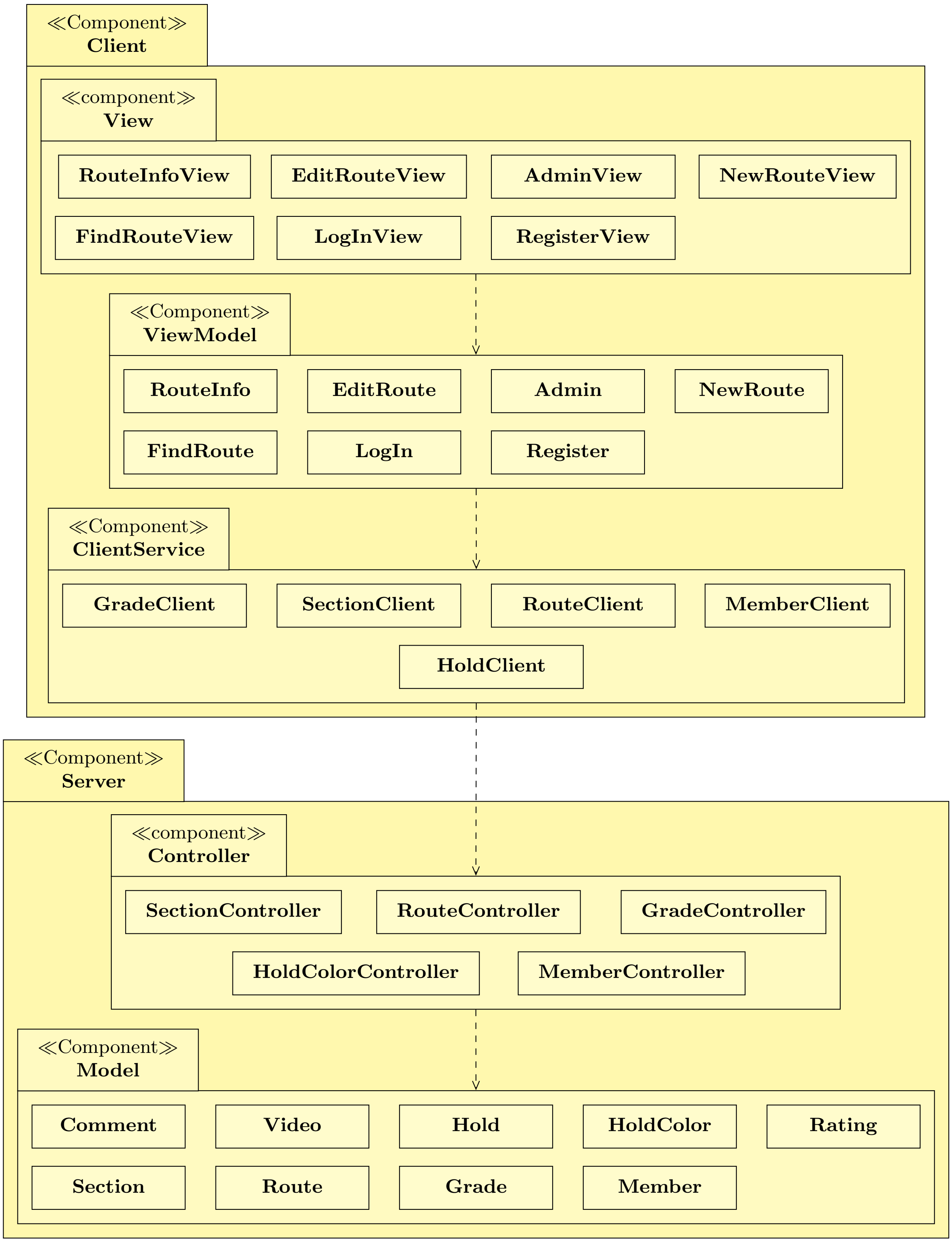


Figure 6.8: Shows the connections between the different components in the components, which illustrates the classes in each component.

Summary

In this chapter, 12 basic design criteria for software development were prioritised based on their importance for the system. The most important criteria for the system were found to be that it needed to be usable, reliable, testable, and comprehensible. Criteria such as security and portability were prioritised as being less important. It was briefly described how the system was developed using the programming languages C# and Javascript as well as the frameworks HandleBars and JQuery. In section 6.3, the server-client architecture used in the system including the five main components of the system, the Model, the Controller, the ClientService, the ViewModel, and the View, was explained. Lastly the flow of information in the system was described: The user interacts with the system, which triggers events in the ViewModel that calls the appropriate methods on the client. The client then communicates with the controller, which is responsible for extracting information from the model, which the controller can then send back to the client and eventually the view.

Having described the architectural design of the system, it was possible to describe the design that was used in the view of the system as described in the following chapter.

Chapter 7

User Interface Design

This chapter contains a discussion on the user interface of the system, including general theory of graphical user interfaces and prototyping, and their use in the project. The first thing described is the purposes and types of prototypes, followed by a description of the prototyping used in this project. An overview of the user interface of the system in its final form is then presented, with illustrations and descriptions on which use cases the different pages relate to. Finally it is described how different key principles and design guidelines are used to ensure usability in the system and thereby solve the requirement that the system must be easy to use.

7.1 Prototyping

This section aims to describe what a prototype is, which types of prototypes exist, and what their purposes are.

A prototype is generally a small and very limited part of an IT-system, where some of the main components (interface, function, model) are used, in order to help system designers build an intuitive and easy to use application for end users (Melissa Mcclendon et al., 2012). A prototype is an early version of the future system and is used to test the system or parts of it while communicating with end-users to get feedback. Since some prototypes focus on specific parts of the system, might mean that other parts of the system may not exist in the prototype, or may just be non-interactive sketches. Other prototypes may focus on providing small insight into the entire system.

In the following sections, several kinds of prototypes are described, the purposes of using them in specific parts of the project, and how they can help overcome problems in the development process. Different types of prototypes are used to achieve different goals, and only some of them will be described.

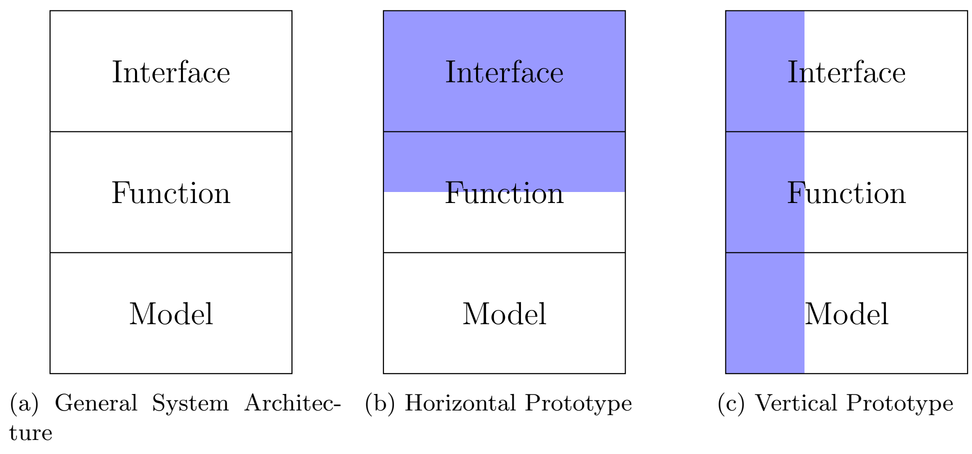
The following sections on prototyping are written on the basis of the article 'Prototyping in Industrial Software Projects - Bridging the Gap Between Theory and Practice' (Lichter et al., 1994, p.826-827) and on the book 'Object oriented analysis & design' (Mathiassen et al., 2000, p.33 & 167).

7.1.1 Purpose of Prototypes

Constructing prototypes can help the development of a system in several ways. They can be helpful in the beginning of the project to confirm the requirements of the system to be developed, but also later in the project by finding and refining problems that may have arisen during development. During early development, a low-fidelity prototype will usually be used in order to show off your ideas to your clients, in order to get feedback. Using a low-fidelity prototype such as a paper prototype is important since a prototype that is done with only pen and paper, makes the client believe that all the progress made can easily be discarded and that nothing significant will then be lost.

7.1.2 Horizontal and Vertical Prototyping

A system with a simple architecture has three layers: an interface, function, and a model, as shown in figure 7.1a. When developing a prototype, it is important for the developers to consider what it is they want to accomplish. They could either want to test a specific part of the system in-depth, or the entire system superficially. Figure 7.1 illustrates this.



(a) General System Architecture (b) Horizontal Prototype (c) Vertical Prototype

Figure 7.1: Figure showing differences between horizontal and vertical prototypes. The blue colour represents the architectural parts involved in the prototype.

As shown in figure 7.1b, a horizontal prototype can involve all of the interface, some of the functionality, and almost nothing, if any, part of the underlying model. This can help test navigation and user experience, and if relevant elements are available, and how intuitive the design is.

A vertical prototype, as shown in figure 7.1c, does not go in-depth with any of the three parts of the system architecture, but instead it implements a chosen part of the system entirely. This means that a fully functional prototype has to be implemented.

7.1.3 Types of Prototypes

Constructing prototypes can help the development of a system in several ways. They can be helpful at the beginning of the project to confirm the requirements of the system to be developed, but also later in the project by finding and refining problems that may have arisen during development. Three different kinds of prototypes exist: the presentation prototype, the prototype proper, and the pilot system.

The presentation prototype is, as its name suggests, used as a way of presenting the future system to a client. This kind of prototype is used early in the system development process and is not very resource-heavy. The purpose of this kind of prototype is to give clients an overview of the system to be developed, which allows them to give feedback on the feasibility of it. When doing a presentation prototype, it is usually done with a low-fidelity prototype such as a paper prototype. The paper prototype is often made on the basis of the first interview between the developers and the client. The presentation prototype is typically a horizontal prototype, since it is an attempt at visualising the application's interface with the goal of finding potential usability problems.

The prototype proper has more functionality than the presentation prototype. It is a *provisional operational software system* (Lichter et al., 1994, p.826), that is used to e.g. reveal problems in the design or illustrate specific functionality. This prototype is used later in the project development process, and the development of the prototype is based on information gathered after showing the client the presentation prototype. A prototype proper is higher fidelity than the previously mentioned presentation prototype, which means it is more resource-heavy since the prototype has more functionality, and could e.g. be a well-made PowerPoint prototype.

A pilot system uses a functional version of the system under development and tests it against the end-users. While the pilot system is a working version of the system, it will not be taken into use. The purpose of this kind of prototype is to get a better understanding of how the system will be used in the proper environment and to refine potential usability problems discovered while prototyping.

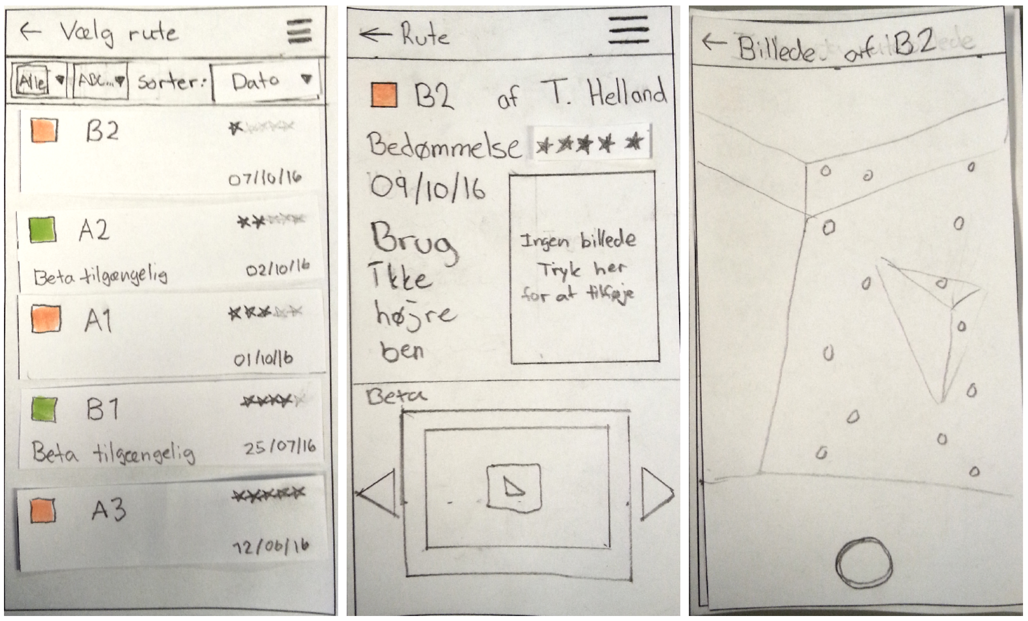
7.1.4 Low-fidelity Presentation Prototype

To design the user interface for the system, we choose to create a horizontal low-fidelity presentation paper prototype as described in section 7.1.3.

We used a horizontal presentation paper prototype because we were at a stage where we wanted to show the client our initial thoughts on the interface of the new system, and for them to comment on it. Hopefully, some of those comments would represent thoughts and ideas of their own, which we can use to adjust and improve the user interface. This prototype illustrated essential parts of the user interface for the new system. Using a paper prototype made it easy for the client to see that what we presented was nothing but initial ideas that could be changed with little to no cost. The prototype is shown on figure 7.2.

Figure 7.2a shows a list of routes that can be filtered and sorted by section, grading, date and rating. When the user selects one of these routes, the user interface navigates to the screen illustrated in figure 7.2b, which shows additional content such as an image of the route, note and beta. Finally, figure 7.2c shows the screen, the user will see when providing an image of the route.

After we created the paper prototype, we visited the climbing club again and showed it to our client. From this visit we discovered some issues with the design as well as ideas for other features. Notes from this visit are shown in appendix C. The following summarises the findings from this visit.



(a) Front page shows a list of routes. (b) Route info for a single route. (c) Taking an photo of a route on the bouldering wall.

Figure 7.2: The paper prototype from the first iteration.

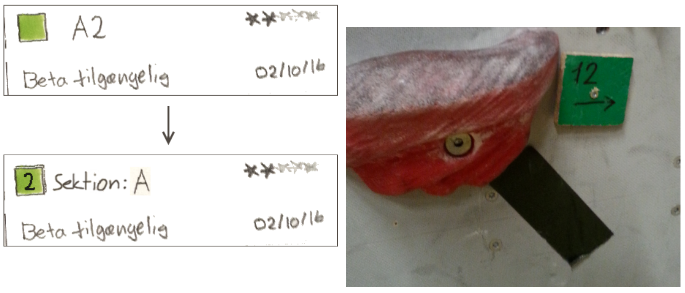


Figure 7.3: Suggestion from the client to represent route numbers in a way which was more representative of how they were already displayed on the climbing wall.

Figure 7.3 illustrates how our client came with the suggestion of changing the presentation of the identifying number of a route in the system. By moving the route number inside the coloured square representing the grade, we were able to get a stronger connection between how the route is represented in the problem domain and how it is represented in the application domain.

Another thing that needed to be more clear is what the three drop-down menus do. To clarify this we talked about adding a word above each drop-down menu that describes what the routes are filtered or sorted by as shown in figure 7.4.

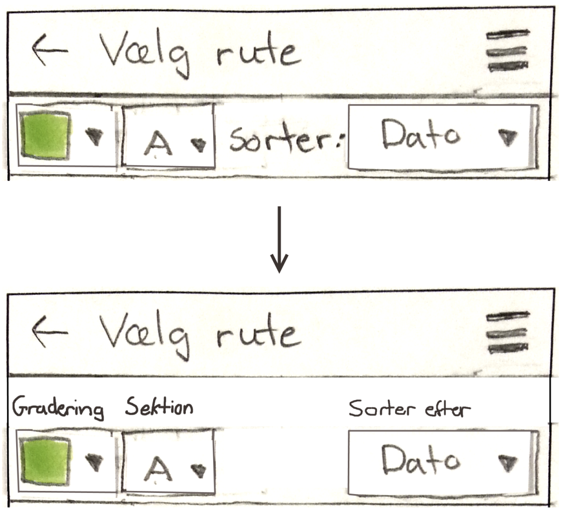


Figure 7.4: Shows how text can be inserted above each drop-down menu to clarify what each drop-down menu represent.

From our visit we got some comments on the design and suggestion for other features.

Comments and Suggested features from the second interview (See appendix C)

- Marking holds on the image of the route with circles.
- For each route in the list indicate with small icons if there are other content available such as betas.
- Rating a route would be nice to have but it is more important that it has the same functionality as the whiteboard system.
- Commenting a route.
- Support for naming a route.

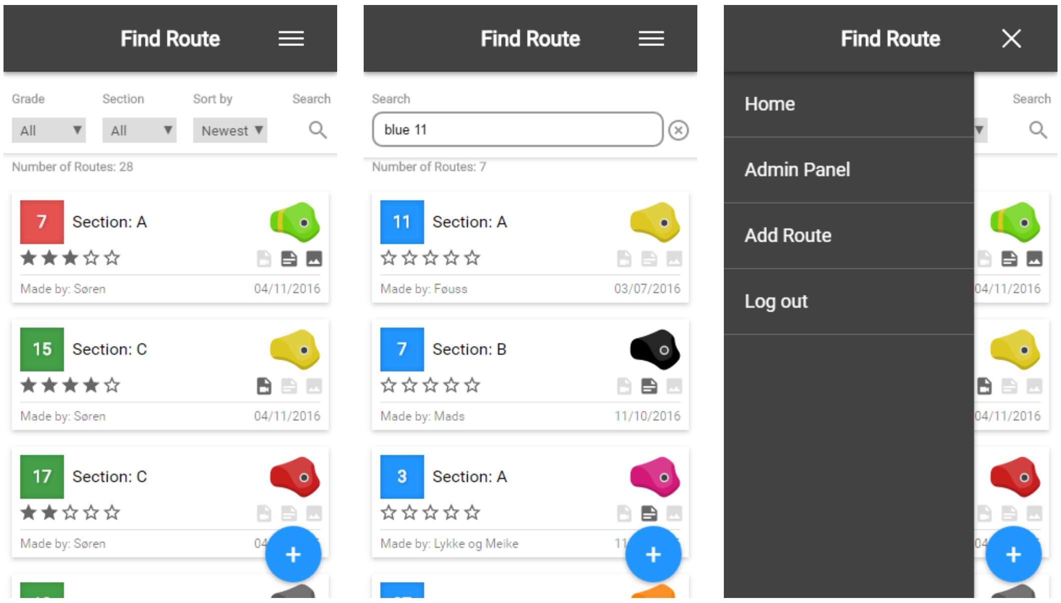
The suggestion: *Marking holds on the image of the route with circles*, was used to adjust our design to be able to add and show circles above the holds in the route image as shown in figure 7.7c. The second adjustment is to introduce icons to indicate all the extra information that are available to a route in the system. We choose to use this in our final design as shown in figure 7.5a where we included icons for showing whether or not a route contains beta, a note or an image. Rating a route was shown in the paper prototype and was approved by the client that it would be a nice feature to have, but it was prioritised as a should-have requirement like adding beta and commenting a route. Thereby rating, adding beta and commenting was prioritised as less important than the must-have and should-have requirements as described in section 3.3. We reached these could-have requirements in the third and final iteration. This is shown in the next section where we describe the final user interface design.

7.2 Page Overview

In this section, we give an overview of the final system and its features. We do this by showing the functionality provided at each page in the system and use this later to discuss how the system solves the requirements in section 3.3.

7.2.1 Find Route Page

The Find Route page is based on the *Viewing Route Information* use case. It lets the user get an overview of the routes available in the climbing club as shown in figure 7.5. The Find Route page provides options for filtering routes by grade and section, and options for sorting by date, grading, rating, or author. Another way to find routes in the system is to click on the search icon and type in a search string containing information about the route(s) that one wants to find. The result of either filtering or searching is a list of elements representing routes. Each element shows the overall information for a particular route such as: number, section, colour of holds/tape, rating, author and date. Each element does also contain three icons that indicate if the routes contains video beta, note or image.



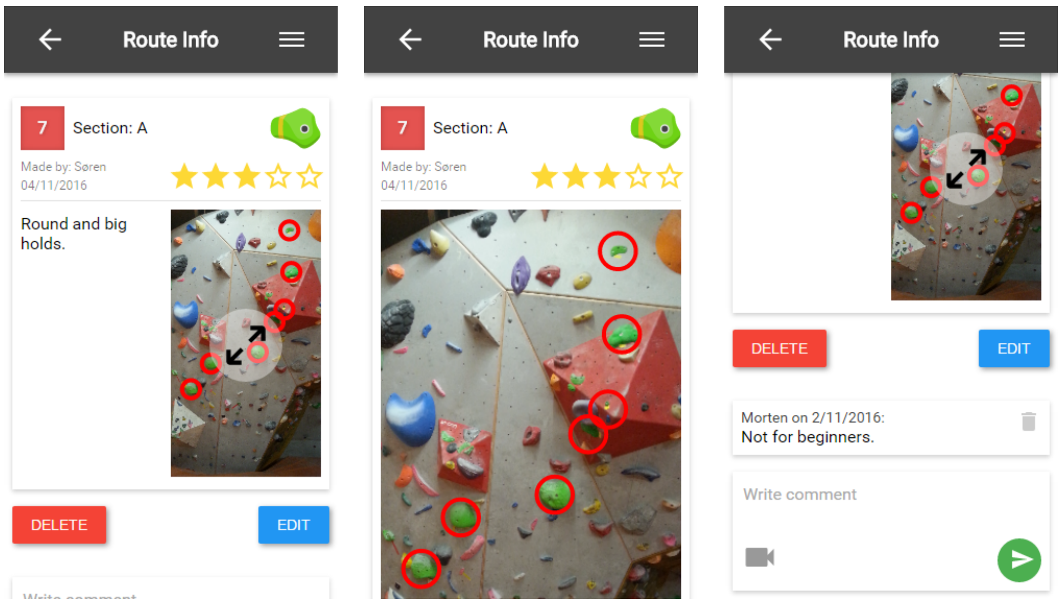
(a) List of all routes. (b) Searching for a route. (c) Menu opened.

Figure 7.5: The Find Route page.

On each page in the system the user can click on the menu icon in the top right corner to open the menu as shown in figure 7.5c. The menu contains a home button that will navigate to the Find Route page. If the user is not authenticated then there will be a button to navigate to the Log In page. Both authenticated and unauthenticated users will have an Add Route button shown, but if the user is unauthenticated, they will first have to log in. Finally if the user is an administrator the menu will contain a button to navigate to the Admin Panel page. The menu can either be closed by clicking the cross in the top right corner or to the right of the menu.

7.2.2 Route Info Page

If the user clicks on one of the routes at the Find Route page the system navigates to the Route Info page as shown in figure 7.6. This represents the second state in the *Viewing Route Information* use case, where all the information that is provided for a route in the system is shown. This includes all the information that was shown at the Find Route page, as well as note, image, comments and betas.



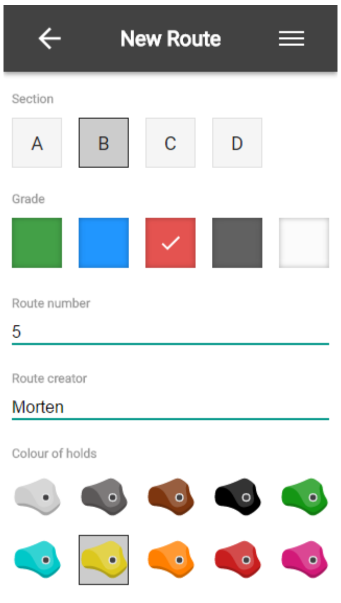
(a) Route information including a note and an image.
(b) Image of the route is enlarged when the user clicks it.
(c) Comments including betas can be added to a route.

Figure 7.6: The Route Info page.

The Route Info Page is also based on the *Adding Additional Route Content* use case where authenticated users can provide additional content to the route. Users can rate a route by clicking on the stars at the Route Info page or add a comment or beta using the input field as shown in figure 7.6c. If there is no image added to the route then there will be a placeholder with the text "Click to add image". If an authenticated user clicks on the image placeholder or on the Edit button then the system will navigate to the Edit Route page where an image can be added. Finally an authenticated user can choose to delete the route by clicking the Delete button.

7.2.3 New/Edit Route Page

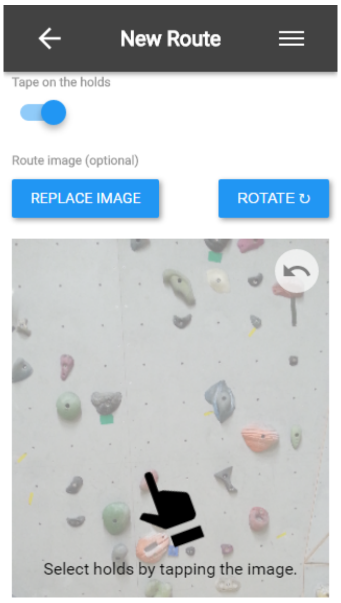
The two pages New Route and Edit Route are almost identical. They both provide input to set information about a route as shown in figure 7.7. This includes section, grade, route number, route creator, colour of holds/tape, and optionally image and note.



Route number
5

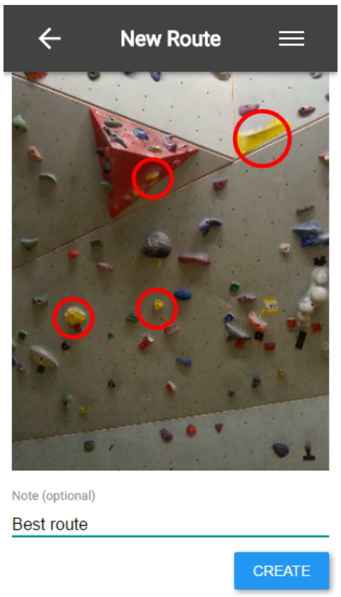
Route creator
Morten

(a) Shows how section, grade, route number, route creator and colour of hold is selected.



Select holds by tapping the image.

(b) Image is added by clicking the Add/Replace Image button.



Note (optional)
Best route

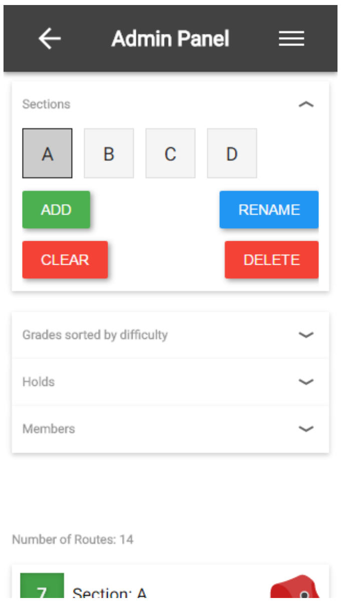
(c) Holds are added to an image by clicking on the holds on the image.

Figure 7.7: The New Route page.

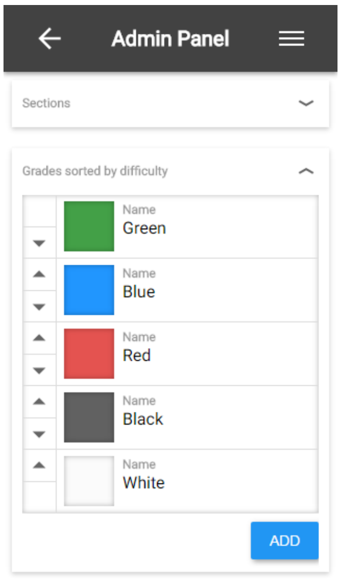
When a user adds an image to a route then the image is shown with an overlay telling the user that they can add holds by clicking the image. If the image is captured vertically then on some smartphones the image may appear with a wrong rotation. To fix this the user can rotate the image by clicking the rotate button above the image. When the user has entered information about the route they can click on the create button if they are on the New Route page, or the save button if they are on the Edit Route page. If the user forgets to specify required information about the route or the provided information is invalid then the system will show a message describing what is wrong. If valid information is provided, the system will navigate to the Route Info page showing the newly changed information.

7.2.4 Admin Panel Page

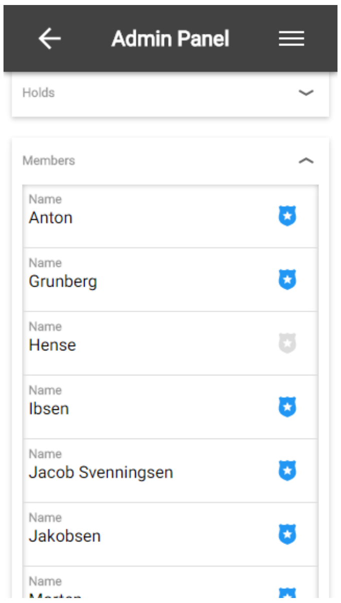
The Admin Panel is based on the *Administrating Sections* use case and is extended to handle administration of grades, holds and other administrators as well. These four categories are represented in collapsed menus as shown in figure 7.8



(a) Sections can be added, cleared, deleted and renamed.



(b) A new grade can be added and existing grades can be edited and reordered.



(c) Members' administration rights can be changed.

Figure 7.8: The Admin Panel page.

If the administrator clicks on one of the menu items it will expand as shown in figure 7.8. Under the sections menu the administrator can select an existing section. This will show all the routes in the selected section at the bottom of the page to make it more clear what is deleted if the section is cleared or deleted. If the administrator clicks the Add button then they will be prompted to enter the name of the new section.

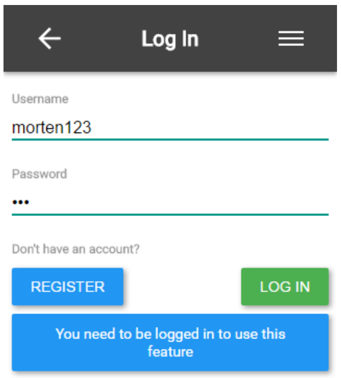
Under the grades menu, the user can re-order existing grades. If they click the Add button a slider will be shown to select the colour representing the new grade and a text input field for typing the name of the new grade.

Under the holds menu the administrator can specify which holds that other members can select from when adding and editing routes.

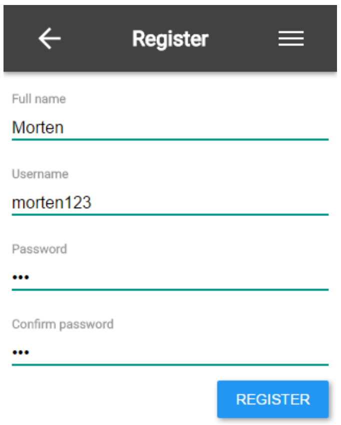
Finally, under the members menu the administrator can view all members and specify which members that should gain administrator privileges by clicking the sheriff icon to the right of the member name.

7.2.5 Log In/Register Page

When an unauthenticated user tries to access the New Route page, Edit Route page or Admin Panel page, they are redirected to the Log In page as shown in figure 7.9a which shows a message stating that the feature they tried to use requires them to be authenticated. If the user does not already have an account they can click on the Register button which navigates to the Register page where the user should provide their name, username and password, as shown in figure 7.9b. Thereafter, they will be authenticated and the system will keep their session until the user explicitly clicks the Log out button in the side menu or the cookie that stores their session token is cleared in other ways. The Log In page can also be accessed by clicking the Log In button in the menu.



(a) Log In page with message is shown if authentication is required to access a certain feature.



(b) A user can register as a new member in the system at the Register page.

Figure 7.9: Log In page and Register page.

7.2.6 Site Navigation Structure

A navigation diagram is used to get an overview of the different pages in the user interface and the connection between them (Mathiassen et al., 2000, p.159).

The navigation for the system has been made from the use cases in section 5.1.2, and can be seen in figure 7.10. From the Home page the user can navigate to any page in the system except the Edit and Register pages, which can only be accessed from the Info or Login page. Once the user has been authenticated through the Login or Register page the user can navigate to the admin panel through the navigation menu which is placed in the top right corner of every page. Each page except the Home page has a back button and to understand how this works we can create a navigation hierarchy for all pages, which is the shortest amount of steps it takes to navigate to a page from the home page. The back button then works by navigating to the first page in the navigation hierarchy from which the specific page can be accessed from. This means that the back button for the Admin, Info, Login and Add page will navigate to the Home page since these all can be accessed from the Home page. Furthermore the back button for the Edit will navigate to the Info page, and for the Register page it will navigate to the Login page.

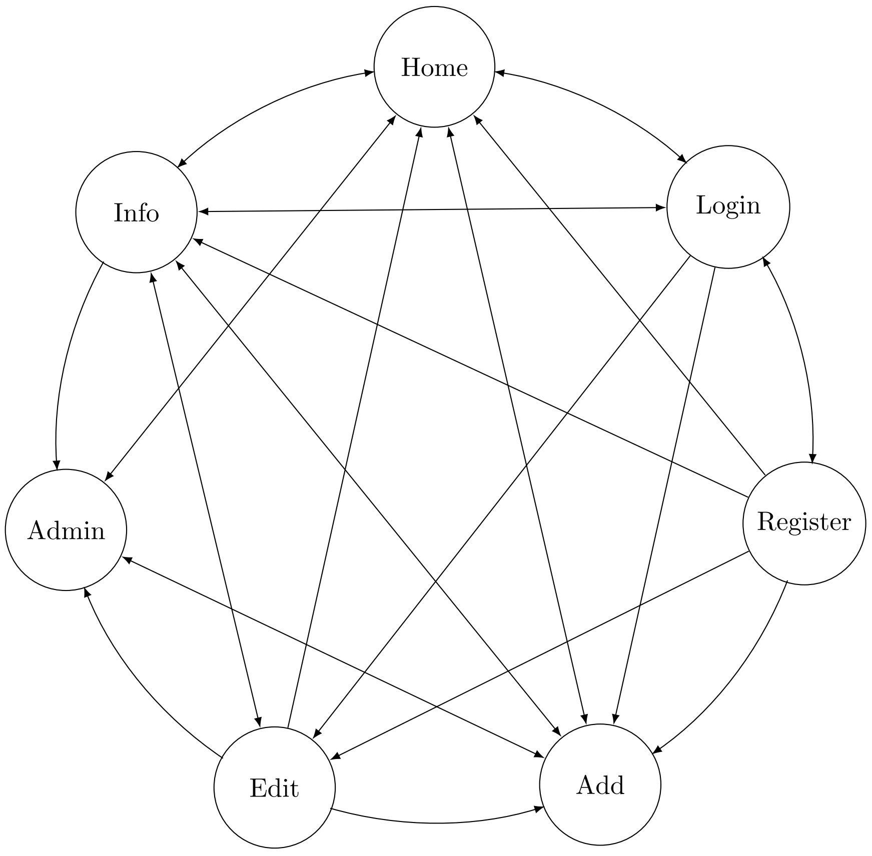


Figure 7.10: Navigation map

7.3 Design Principles and Guidelines

In this section, we describe different design principles and guidelines. We describe these so we get a better understanding of how we can create a user-friendly experience by following these guidelines, and will therefore help us in solving the requirement: *The system must be easy to use.* In Benyon (2010, p. 90) twelve design principles are described and categorised into the following four groups:

- Access, ease of learning and remembering
- Ease of use
- Safety
- Accommodating differences between people and respecting those differences

We chose to focus on the first two design principles: "access, ease of learning and remembering" and "ease of use" because these are related to the requirement: *The system must be easy to use.*

According to Benyon (2010, p. 90), the design principles concerning access, ease of learning, remembering and ease of use are:

Visibility Make sure that the user knows what functions are available and what the system is currently doing.

Consistency Make sure the design is consistent throughout the system.

Familiarity Make sure that the design uses symbols that users are familiar with, especially from the context in which the system will be used.

Affordance Make sure that different parts of the design look the part, e.g. buttons should look press-able and text-fields selectable.

Navigation Make sure that navigation is comprehensible, so that users always know where they are in the system.

Control Make sure that there is a mapping between controls and the result of the controls.

Feedback Make sure that the user knows that the system is working on some input they gave, or actually registered a key-press.

Design principles can lead to more concrete design guidelines about the appearance of elements in the user interface (Benyon, 2010, p. 89). This can include, but is not limited to: size, colour and style of buttons, menus, information boxes, and labels. It is simply a description of the general theme that a design should follow, and can be made as simple or advanced as it needs to. While developing a unique design for every new system would be good news for developers wanting to try out a lot of different design methods, it is generally not a good idea. Using a tried and tested design guideline means that the developers are ensured that their system will have at least some of the key points above. Especially consistency, familiarity and affordance are going to be well-represented in a system that uses a widely known and accepted design guideline.

Apple, for example, has created a design guideline for developers looking to make applications for their devices (Apple, 2016). This guideline is an attempt at making apps for iPhones fit in with the general design theme of the operating system. This means that buttons, navigational menus, settings, dialogue-boxes, and such, all have the same feel to them, which is good for consistency and affordance. It also ensures that as long as Apple only features buttons that they have thoroughly usability tested, in their operating systems, applications that follow their guidelines will also have buttons with a high degree of usability, which is a good thing for not only the developer, but also Apple. This makes writing detailed and thorough design guidelines beneficial to software developers and large companies that deal with platforms for applications.

Even though a lot of companies have their own design guidelines or style guides, they are generally concerned with the same core concepts for designing for Human-Computer Interaction (HCI). Basic ideas of design, such as grouping together elements and controls that are logically connected, are found in all proper design guidelines from major software companies (Benyon, 2010, p. 323, p. 335, p. 215).

Some basic basic concepts and ideas that should be considered when designing a graphical user interface are listed here and described in detail below.

- Interactive elements
- Typography
- Colours

Interactive elements are obviously a hugely important part of any interactive design. It is especially important to consider how the end user is going to experience pressing buttons on a smartphone, when designing them for that platform (Benyon, 2010, p. 217). The size of a button is going to affect how easy or difficult it is to hit the desired one. Generally, the touch target (the zone inside of which tapping should activate the element) should be larger than 40x40 pixels, for users to comfortably activate them (Sun et al., 2007). The graphical item itself can be smaller, but making it less than 24 pixels on any one length, can make it hard to see. While the distance between interactive elements is not as important as the touch target of them, it still has an impact on the user's understanding of the different elements' boundaries (Sun et al., 2007).

Generally, a user interface contains text in many different forms. A good design should try to establish a hierarchy of the text in a specific screen of the system, as it makes it easier for the user to know which things to read first (Benyon, 2010, p. 396). As with interactive elements, size plays a large role in how users perceive text, and making one paragraph larger than another is a straight forward way of making it seem more important to the user. Varying text sizes have its limits as well: if a screen contains too many different sizes of text, it can become overwhelming to the user, who will have trouble differentiating the sizes. (Benyon, 2010, p. 202). Another way of making text more readable is to use a font with serifs. While a sans-serif font can help give a cleaner and more modern look to a design, it is more difficult to read. Therefore, careful consideration should be used in choosing the fonts of a design.

Having a lot of text can also become a problem for a design: filling up the entire screen with text can be overwhelming to users since the task of reading it will seem insurmountable. Trying to make the same text take up less room by reducing the font size could result in eye-strain for the user. All in all, there are a lot of things to consider when it comes to typography in a design.

Colours can be used for a variety of effects in a user interface design (Benyon, 2010, p. 343). Differences in colours can be used to to highlight different parts of a design. This means that they can be used to make it easier to distinguish different logical parts of the design from each other. The colours of interactive objects can also help draw attention to them in the design: a red button in an otherwise colour-neutral design is going to stand out a lot. It is important to note that using colours to draw attention to too many things in a design, diminishes the effect greatly, since this means that the different elements will all be vying for the user's attention.

Depending on how much one element needs to stand out from another, the contrast of the two elements' colours can be tweaked. This is especially important when it comes to text, since choosing two colours with high contrast can help reduce eye strain and make the text easier to read. Contrast can also be used to make it more clear to the user how a page is divided in to different sections or to direct their attention to important elements, such as the main button of the page.

Changing the saturation of the colours in a design can affect the user's emotional response to using the system (Benyon, 2010, p. 344). A colour scheme with a high saturation, for example, can help make a design appear more energetic and alert, as opposed to a more calm appearance if less saturated colours are used. The saturation of the colour scheme should be matched with the content, intended user base, and the context in which the system will be used.

7.4 Using Design Guidelines

As described in 7.3, design guidelines can be helpful when trying to design a user interface with a high degree of usability. To make it easier for ourselves to create a design that is consistent and easy to learn, we decided to use Google's guidelines for Material Design (Google, 2016) for major parts of our design.

7.4.1 Using Material Design

We have chosen to make use of Google's guidelines for Material Design, because as we found out from the PACT analysis, many of AKK's members have a smartphone and according to TNS (2015, P. 14), most either have, or have had, an Android-based smartphone which follows the same design guidelines. Material Design relies heavily on trying to mimic some of the physical characteristics of materials in the real world. Generally, this means that distinct elements and sections within the design are represented a bit like how pieces of paper are in the physical world. This means that information is displayed on sheets of "material" or "cards", which casts a shadow to the material underneath it as if it was actually laying on top of it. All material sheets have the same thickness and are incapable of moving through each other. Some other forms of movement is highly encouraged, as it can help give the user feedback on their actions.

We decided early on that we wanted our system to represent the routes of AKK as items in a list. This idea came before any kind of prototype was made because it fit our mental model of how to most intuitively represent objects in an easily sortable manner. We also knew early on that a significant amount of information would have to present on each item of this list of routes. Given the basics of Material Design it makes sense for routes to be displayed as cards slightly raised from the background.

An example of the usages of these cards is on the Find route page where each card contains the following information:

Grading Difficulty of the route

Number The identifying number of the route

Section The section in which the route is located

Hold-colour The colour of the holds on the route

Rating The average user-rating of the route

Video-beta Whether or not the route has video-beta added to it

Note Whether or not the route has a note attached to it

Image Whether or not an image of the route has been uploaded

Author The name of the person who created the route

Date The date the route was added to the system

An example of a single card can be seen on figure 7.11

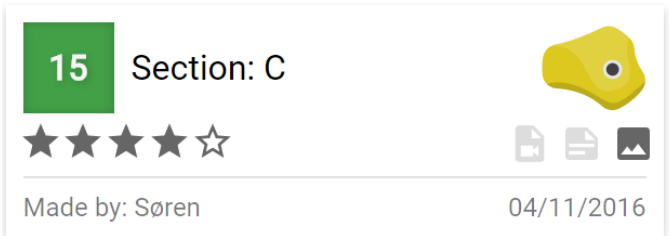


Figure 7.11: Showing a route card.

We chose to represent five of the ten items using icons, which meant that we could drastically reduce the amount of text required, while maintaining the same amount of information and making some of it available in a more intuitive manner. The hold-colour is a good example of this, as it allowed us to show a colour the most intuitive way: by showing the actual colour, rather than writing the name of the colour. Since we also had to display a colour for the grading of the route, using the hold icon made it easier for users to distinguish which of the two colours on each route represented what. A few of the elements would directly allow users to identify the specific route on the wall. We tried to design the route card in a way where these elements took up the most room, and were the first thing the user would see, if they "read" the card from top to bottom.

The author and date were moved to the very bottom of the card, not because they are less important than the rating and the video, note, and image indicating icons, but because having an author at the bottom of the card makes it seem a bit like a signature. Dividing the card in two with the line above the author and date also makes the information below the line stand out more.

The large blue add button which can be seen in figure 7.12 is what is known as a "Floating Action Button" in the material design guideline.

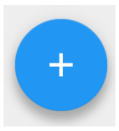


Figure 7.12: The Add Route Button

This button, which is designed as an addition symbol, is also an example of the use of metaphors in the design. The addition symbol tells the user that they can 'add' something to the system. This particular metaphor works because of its familiarity with other usages of the addition sign e.g. it is mathematical notation. Floating action buttons should be used to represent the primary action of an application. The primary focus of our system is to find routes, but one could argue that it is not an action, which would make adding routes the primary action. Having the button be so prominent on the front page is an attempt at making it very clear to users that they have the ability to add routes - even if they need to log in to the system first. In an attempt to show new users that it is possible for them to create routes, we agreed that it should not be hidden away in a menu, but rather be as available as possible.

7.4.2 General Design Considerations

As explained in section 7.3, some of the key principles that make a system have high usability are consistency, familiarity, affordance, navigation, and feedback. In this chapter, we explain how we took these key principles into consideration when designing the user interface.

Visibility

To implement the visibility design principle, we ensured that available features are visible to the user. Because rating, commenting, adding and editing routes require that the user is authenticated there could be a risk that users would never discover that these features are available. To prevent this we made these features visible to the users even if they are not yet authenticated. This is shown in figure 7.13.

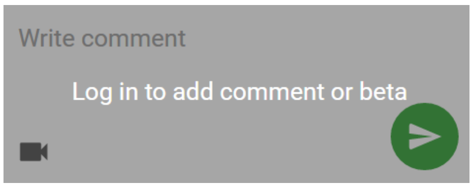


Figure 7.13: This figure illustrates our comment/beta box, when a user is not logged in.

We chose to make the comment and beta input field visible with an overlay showing that the user needs to log in to use this feature. Another way to make features visible to the user is shown in figure 7.7b, where the route image contains an overlay showing that the user can add holds by tapping the image. Without this information the user would not directly know that this feature was available.

Consistency

To implement consistency into the design, we tried to be consistent in the way we used colours, input buttons, error messages, icons amongst others. Furthermore, we tried to provide common characteristics to input elements like radio buttons, and text-fields an example of this could be their common shape. Another example of consistency is the navigation menu that is located the same place on every page. If elements are reused throughout multiple pages they also stay consistent in the way we represent them e.g. how we represent holds and grades in the application. All pages also have the same general consistent design with a black top bar, and buttons are either red or blue - but never various across the pages, which means that the type delete buttons are red on every page, whereas edit and add buttons are always blue.

Familiarity

To make the application seem familiar we try to implement design elements that the users are already familiar with. An example of this could be the usages of metaphors in icons. Figure 7.14 illustrates the usage of such a metaphor in an icon. On the icon there is a paper aeroplane which refers to the actions of sending a message. When the users see such an icon, they should be familiar with the effect of pushing the button, which in this case is used to send a comment so that other users can see it.

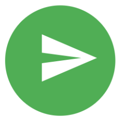


Figure 7.14: Metaphor that illustrates the action of sending a message.

Another example is to use commonly used metaphors like the navigation bar, back button, and image icon where the image icon can be seen on figure 7.15. Another example of familiarity is a loading bar that comes when an action is being performed which e.g. can occur when uploading a video. Stars used to rate something is also an example of familiarity, where the amount of stars indicates how good a route is.

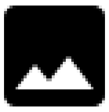


Figure 7.15: Image icon.

Affordance

To implement affordance into the system, we tried to make things do what they look like they do, this means that if something in the system looks like a button it means that the user can afford to click it, and that it performs as is expected of a button. Another example is the way users rate, where empty or grey stars on the route info page means that the users can afford to press them in order to rate a route, whereas they are filled and yellow if the route has already been rated by the user.

Navigation

To make the navigation comprehensible and show the users where they are in the system, we have a title in the header of every page e.g. Edit Route if the users are on the Edit Route page in the system, this also tells the user which action they are performing. To further help the users navigate in the application, we created a navigation menu, where the user can navigate to user pages of the application like the Home Page. On every page except the front page there is also a back button which when pressed navigates you to the previous page you visited, or in some cases the home page.

Control

We ensure that the user know what they are in control of in the system by dividing the user interface into pages that represent control of certain objects from the problem domain. The initial page showing the list of all routes in the system lets the user take control of the appearance of the list by enabling filter and search functions and let the user add a new route by pushing the Add button. At the New/Edit Route page the user controls a single route. At the Admin Panel the user can be in control of sections, grades, colour of hold and members. To make it clear what the user is in control of at the Admin Panel page, we group the content into a collapsed menu containing elements containing sections, grades, colour of hold and members as shown in figure 7.8. The reason we use a collapsed menu is that it only shows one element at the time and thereby makes it clear to the user what they are currently in control of.

Feedback

Feedback is used across the application whenever the users perform actions that might need some feedback; this can be when an user tries to add a route without some of its required fields. Another example of feedback is when an unauthenticated user tries to perform an action that needs authentication, and in some instances they are navigated to the login page, and if so a feedback box provides information on why they were navigated to this page. Another example of feedback is a red error message that occurs when the user enters a wrong username or password when trying to login as figure 7.16 illustrates. The colour red is used to attract attention and is therefore suitable for warnings.

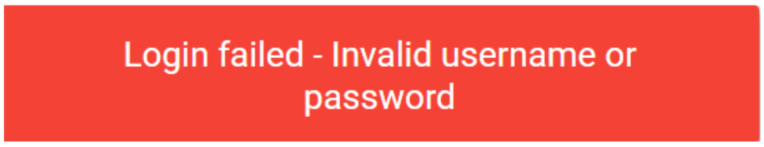

Login failed - Invalid username or password

Figure 7.16: Log in error message.

Summary

In this chapter, general theory of prototyping was explained. How this theory was applied to create a low fidelity prototype for the client at AKK and how the feedback was used to create a list of suggested features and improvements, was also described. Some of the key principles that help increase the usability of an application were also described. The key principles were *consistency*, *familiarity*, *visibility*, *control*, *affordance*, *navigation*, and *feedback*. The point that using design guidelines as a way to ensure that these principles are considered during development, was made. Design guidelines were discussed and found to contain many concepts for how to design well. Four such concepts were focused on in this chapter: interactive elements, typography, contrast and saturation. An overview of the entire finished user interface of the system was also presented along with how the use of design guidelines helped with designing the it.

Having decided on the architectural design of the system and also designed the graphical user interface of the system, it was possible to implement the remaining architectural layers, as described in the following chapter.

Chapter 8

Implementation

In this chapter, the implementation of the system is described. The structure of the chapter follows the overall system architecture, as illustrated in figure 6.1. The first description is that of the model component. Its description includes an explanation of how data is stored in the system and how classes in the model are mapped in an SQLite database. Next, the second server-side component, the controller component and how it makes itself available through its application programming interface (API), is described. A general implementation of an API response is described in connection to this. The implementation of user authentication and searching in the server-side components, is also described. The implementation of the client-side components follows, the first of which is the client component that calls the server-side API. Next, the implementation of the user interface and its corresponding functionality is described. This includes a description of how the code in the user interface is separated into views and view models, and how these are connected. Finally the implementation of the functionality required for users to add images, comments and video betas to routes in the system is described.

8.1 Server-side

In this section, we first describe the Database and then the Data Access Layer, as described in the introduction of this chapter.

8.1.1 Database

To be able to store information in the system, we chose an SQLlite database which is a server-less lightweight SQL database engine that reads and writes directly to disk files (SQLite, 2016). To be able to retrieve data from the database, we used .NET Core Entity Framework (EF) which is an Object-Relational Mapping (ORM) framework that maps classes from the model to the database tables automatically (EntityFrameworkTutorial, 2016).

In EF there are generally three approaches for ORM: Database First, Code First, and Model First.

In the Code First approach, a database is generated from the relations between classes in the system, whereas in Model First the entities and relations for the database are created using a design tool. In Database First the entity classes are created from an existing database.

We chose to use EF Code First since it allowed us to create the entire database structure using C# and Code First conventions, which meant that we did not have to spend time learning new tools or manually writing SQL statements.

In C#, we created classes and relations corresponding to the class diagram for the model structure in section 6.3.1.

Listing 8.1, shows the Section class. The Section class inherits an Id from the Model class and the Id is a globally unique identifier (GUID) that is generated by the database when the row is inserted into the database. Furthermore, the Id also serves as the primary key for any Section object as can be seen by the data annotation `[key]` in listing 8.2. A section also has a Name and aggregates zero or more routes.

```
    public class Section : Model
    {
        public Section()
        {
            Routes = new List<Route>();
        }
        public string Name { get; set; }

        public List<Route> Routes { get; set; }
    }
```

Listing 8.1: Section class

```
    public class Model : IIdentifyable
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public virtual Guid Id  { get; set; }
    }
```

Listing 8.2: Model class

Similar classes were created for all our models in the system such as Routes, Grades and Members. To generate tables for each of these classes, we simply included EF in the project and created a class that inherited from DBContext that comes with EF. In this class, we created a DbSet property for each of our model classes, which Code First then uses to generate tables in the database. On line 7 in listing 8.3, the table for the Route class is created and named after the property name which in this case is "Routes". Line 7-16 in listing 8.3 shows the ten properties used by Code First to generate tables in the database.

```
1       public class MainDbContext : DbContext
2       {
3           public MainDbContext(DbContextOptions<MainDbContext>
                options)
4               : base(options)
5           { }
6
7           public DbSet<Route> Routes { get; set; }
8           public DbSet<Section> Sections { get; set; }
9           public DbSet<Grade> Grades { get; set; }
10          public DbSet<Member> Members { get; set; }
11          public DbSet<Image> Images { get; set; }
12          public DbSet<Hold> Holds { get; set; }
13          public DbSet<Video> Videos { get; set; }
```

```
        public DbSet<Comment> Comments { get; set; }
        public DbSet<Rating> Rating { get; set; }
        public DbSet<HoldColor> HoldColors { get; set; }

        [...]
    }
```

Listing 8.3: Class for the database context

To create the relations between the classes in the model, we used the Code First conventions. Listing 8.4 illustrates an example of a one to many relationships between grades and routes. As it can be seen, the grade class contains a list of `Route` objects. To make the 'many' relation to the `Route` class each `Grade` contains zero or more foreign keys, which is referenced to a `Route`'s primary key. By doing so, Code First keeps track of which `Grade` is associated to which routes. Similarly, the `Route` class contains a reference to one `Grade` object where the `GradeId` is a foreign key to a `Grade`.

```
    public class Grade : model
    {
        [...]
        public List<Route> Routes { get; set; }
    }

    public class Route : model
    {
        [...]
        public Grade Grade { get; set; }

        public Guid GradeId { get; set; }
    }
```

Listing 8.4: Establising the relations between Grades and Routes

An alternative way of making these relations is through either data annotations like `[key]` and `[Foreignkey("")]` or Fluent API where we specify the relations manually by overriding the `OnModelCreating` method in the DbContext class.

Figure 8.1 shows the structure, the dependencies, and the multiplicity between the tables in the database. The arrows indicate that the one pointed by is dependent of the one pointed to. An example is the one to many relationships between `Holds` and `Images`, where one `Hold` has one `Image` and an `Image` has zero or more `Holds`. Furthermore, a `Hold` must have one `Image` and is thus dependent on the `Image` class.

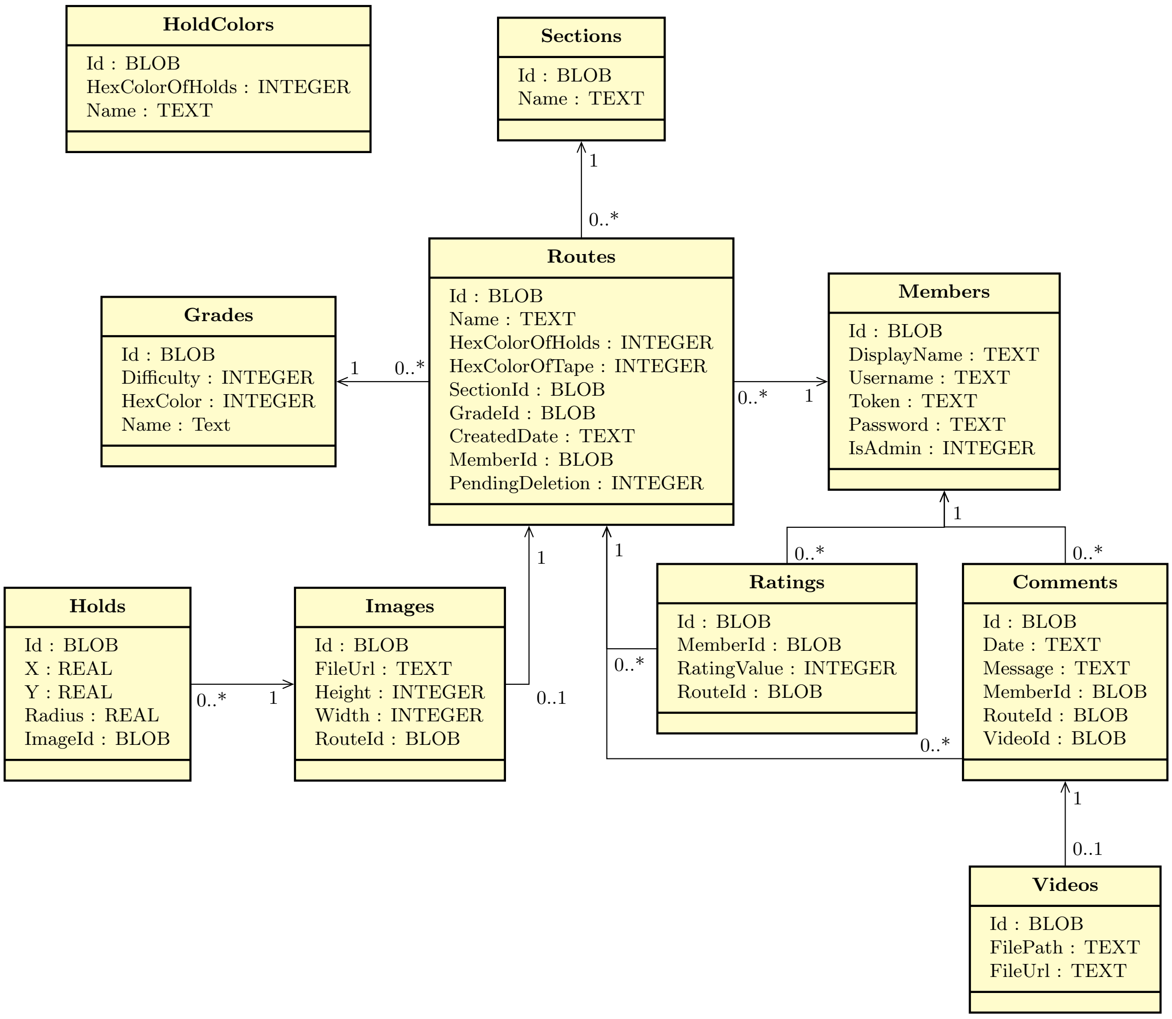


Figure 8.1: Database diagram.

Data access layer

To access the data in the model we use a data access layer (DAL). The DAL abstracts over the logic needed to communicate with the underlying data stores in the system. It creates common data access functionality, which makes maintaining and configuring the system easier (Microsoft, 2016a). To implement a DAL in the system we use the repository pattern where the logic that communicates with the database is separated from the business logic (Microsoft, 2016c) . Listing 8.5 shows the `IRepository` interface which is a generic interface containing the methods for accessing and retrieving information from the database. We then implemented the `IRepository` interface for each class in the model: `GradeRepository`, `HoldColorRepository`, `HoldRepository`, `ImageRepository`, `MemberRepository`, `RouteRepository`, and `SectionRepositry`

```
    public interface IRepository<TEntity> where TEntity : class
    {
        void Add(TEntity entity);
        TEntity Find(Guid Id);
        void Save();
        void Delete(Guid Id);
        IEnumerable<TEntity> GetAll();
    }
```

Listing 8.5: Repository interface

Using the repository pattern enabled us to create test repositories that was used to unit test the controllers as described later in section 9.3.

8.1.2 API

The API functions by sending an HTTP request to one of its defined URLs with the corresponding HTTP method as described in table 6.3. An example of such a method is seen in Listing 8.6 where line 2 denotes the URL used to call this method. Furthermore, the call includes a `name` parameter which is used in the method to identify and return a section with this name or id from the database. This attribute also determines the HTTP verb that has to be associated with the request for the method to be called. The method takes a name or id of a section and returns the entire section if it exists in the database.

```
 1      // GET: /api/section/{name}
 2      [HttpGet("{name}")]
 3      public ApiResponse<Section> GetSection(string name) {
 4          var sections = _sectionRepository.GetAll();
 5
 6          try {
 7              Guid id = new Guid(name);
 8              sections = sections.Where(s => s.Id == id);
 9          } catch(System.FormatException) {
10              sections = sections.Where(s => s.Name == name);
11          }
12
13          if(sections.Count() != 1)
14              return new ApiErrorResponse<Section>($"No section with
                    name {name}");
15
16          return new ApiSuccessResponse<Section>(sections.First());
17      }
```

Listing 8.6: Controller method that handles one type of API request

The API follows a specific pattern when responding, the general implementation of an API response can be seen in listing 8.7. The class simply specify a format where every response contains a success flag that indicates if the request was successful. A response also contains data. As the value property is virtual, this format can be overwritten as it is for the specialisation `ApiErrorResponse<T>`. Here no data is returned but instead an error message that helps the developer identify any mistakes. The type is generic to ensure that the Data property can never be anything but the expected return type of an API request. This is to make it easier to test.

```
public class ApiResponse<T> : IActionResult {

    public T Data {get; set;}
    public bool Success {get; set;}
    public virtual object Value {
        get {
            return new {success = Success, data=Data};
        }
    }

    public ApiResponse (bool success)
    {
        Success = success;
        Data = default(T);
    }

    public void ExecuteResult(ControllerContext context)
    {
        new JsonResult(Value).ExecuteResult(context);
    }

    public Task ExecuteResultAsync(ActionContext context)
    {
        return new JsonResult(Value).ExecuteResultAsync(context);
    }
}
```

Listing 8.7: ApiResponse class

8.1.3 Login

This section describes the implementation of user management in the system and the reasoning behind the choices made during the implementation.

8.1.4 Cookie Authentication or Token Authentication

ASP.NET Core MVC has a built-in authentication service that uses cookies in the browser to keep the user logged in and authenticated. It is then simply a matter of adding the `[Authorize]` attribute to the view controllers, to get ASP.NET to ensure that a user is authenticated when accessing those views.

This approach allows for a very easy implementation of authentication, as there is not much code to be written by the developer, which minimises the risk of bugs in the code.

The downside to this type of authentication is that we require the client to support cookies.

Another approach to authentication, is the use of tokens. In every request sent to the server that requires authentication, the controller requires the client to send an authentication token as a parameter, which it can use to authenticate the user.

The token authentication process is shown in figure 8.2. The client contacts the server with the user credentials, the server then returns a token that the client uses for subsequent requests to the server that requires authentication.

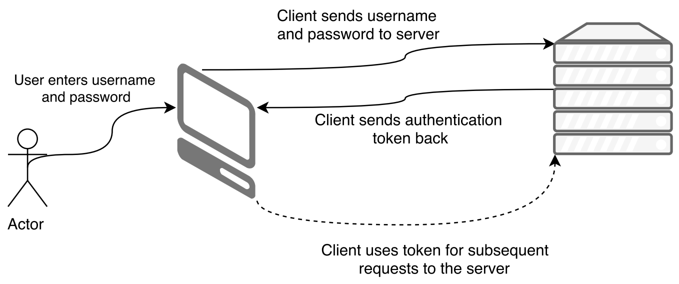

Client sends username and password to server
User enters username and password
Client sends authentication token back
Actor
Client uses token for subsequent requests to the server

Figure 8.2: Illustration of token authentication process.

We chose the latter approach and implemented token authentication in our system. Even though cookie authentication would be much easier to implement, we chose not to use it since we want to keep any client implementation from having to deal with cookies.

To facilitate these authentication tokens, the only change to our Member class was the addition of a property Token as can be seen in listing 8.8, on line 10. This Token then contains the latest authentication token for the user and allows the controllers to confirm that the member is authenticated.

```
1 public class Member : Model
2 {
3     [...]
4
5     public string Token { get; set; }
6
7     [...]
8 }
```

Listing 8.8: Member class

To distribute the tokens, we created an interface `IAuthenticationService` and a class `AuthenticationService` that implements the interface. The interface can be seen in listing 8.9.

```
public interface IAuthenticationService
{
    string Login(string username, string password);

    void Logout(string token);

    bool HasRole(string token, Role role);

    void ChangeRole(Guid id, Role role);

    IEnumerable<Role> GetRoles(string token);

    string HashPassword(string password);

    bool TestPassword(string password, string hashedPass);
}
```

Listing 8.9: IAuthenticationService interface

This service provides an authentication token with the `Login` method, if provided with valid user credentials. It can also invalidate a token with the `Logout` method and it can confirm that a user has a particular role (admin or member). It also allows changing the roles of members in the system.

The actual implementation of the interface uses the Token property on the member class to create and invalidate tokens for the user and uses the property `IsAdmin` to determine the role of the specific `Member`. It also hashes and creates salt for the users' passwords, so we do not store them as plaintext. The implementation of the latter can be seen in listing 8.10.

```
private byte[] Hash(string value, string salt)
{
    return Hash(Encoding.UTF8.GetBytes(value), Encoding.UTF8.
       GetBytes(salt));
}

private byte[] Hash(byte[] value, byte[] salt)
{
    byte[] saltedValue = value.Concat(salt).ToArray();
    return SHA256.Create().ComputeHash(saltedValue);
}

private string GenerateSalt() {
    return Convert.ToBase64String(Guid.NewGuid().ToByteArray());
}

public string HashPassword(string password)
{
    string salt = GenerateSalt();
    string hash = Convert.ToBase64String(Hash(password, salt));
    return salt+":"+hash;
}
```

Listing 8.10: Hashing and Salting the Passwords of Users

In the `MemberController` seen in listing 8.11, we can then add methods to the API to access this authentication token from our AuthenticationServices.

All in all, this allows a client to connect to `SERVERURL/api/member/login` in order to gain an authentication token and `SERVERURL/api/member/logout` to remove an authentication token.

```
[Route("api/member")]
public class MemberController : Controller
{
    private readonly IAuthenticationService _authenticator;
    private readonly IRepository<Member> _memberRepository;

    public MemberController(IRepository<Member> memberRepository)
    {
        _memberRepository = memberRepository;
        _authenticator = new AuthenticationService(
           _memberRepository);
    }

    [...]

    // GET: /api/member
    [HttpGet]
    public ApiResponse<string> Login(string username, string password)
    {
        if (string.IsNullOrEmpty(username))
        {
            return new ApiErrorResponse<string>("Login failed - Invalid username or password");
        }
        string token = _authenticator.Login(username.ToLower(), password);

        if (string.IsNullOrEmpty(token))
        {
            return new ApiErrorResponse<string>("Login failed - Invalid username or password");
        }

        return new ApiSuccessResponse<string>(token);
    }

    // GET: /api/member
    [HttpGet]
    public ApiResponse<string> Logout(string token)
    {
        _authenticator.Logout(token);

        return new ApiSuccessResponse<string>("Logout successful");
    }

    [...]
}
```

Listing 8.11: MemberController class

On the client side, the token is saved in a cookie, such that the user stays logged in if they leave the page and come back later. This also means that if a user is logged in, a token is saved in their cookies. We can then use this to check if they are allowed access to certain parts of the system. To make the process of restricting access to views as simple as possible in our code we created an attribute `RequiresAuthAttribute`, which takes a number of roles as an argument, so only members that have one of the roles are authorised to access the view.

An example of the use of the attribute can be seen in listing 8.12. The attribute on line 3 makes sure that only users who are authenticated (logged in) can access the ”new-route” page of the system.

```
1 // GET: /new-route
```

```
[HttpGet("new-route")]
[RequiresAuth(Role.Authenticated)]
public IActionResult NewRoute() {
    return View("Views/NewRoute.cshtml");
}
```

Listing 8.12: Example usage of `RequiresAuthAttribute`

8.1.5 Search

In this section, the implementation of the search functionality is described. A justification of the algorithm chosen to help the search functionality is also presented.

Comparison of Alternatives

There exist multiple string metrics for calculating the distance between two string sequences. Following is a description of a few such algorithms, along with their advantages and disadvantages.

Knuth-Morris-Pratt

Knuth-Morris-Pratt is a string searching algorithm, that works best on a short pattern (the word you search for) and a long text (the words the pattern is compared to). However, this is not the situation in our system, where neither pattern nor text will be long. This algorithm differs from the naive approach by saving data from comparisons and makes use of this knowledge to skip some iterations of comparisons. It only searches for an exact match of the pattern in the text, which rules out spelling mistakes (of Information and Sciences, 1996).

Boyer-Moore

Boyer-Moore is an algorithm that compares the last character of the pattern instead of the first, since if the last character does not match, it can shift the pattern to the right. How much the pattern is shifted, is based on whether or not the character in the text is in the pattern or not. If it is not in the pattern, then the pattern's first character is shifted and now compared to the character in the text, following the one that produced a mismatch (d'électronique et d'informatique Gaspard-Monge , IGM). Because of these shifts, the algorithm is optimal to use on a long text, since it allows for some large shifts. This algorithm does not tolerate misspellings.

Levenshtein

The Levenshtein distance is a measure of the minimal amount of units it takes to go from one word to another by using additions, deletions and substitutions of characters. The cost of each of these operations can be changed to fit the situation. The Levenshtein distance is not an algorithm as the two previously described, but there exists algorithms, that calculate the distance between two strings according to the Levenshtein principle. There also exists a modification of the Levenshtein distance, called Damerau-Levenshtein, which implements a fourth edit operation called a transposition of two adjacent characters. Damerau only considered words with a single misspelling or missing letter (Damerau, 1964). Since we now look at the edit distance (approximate match, fuzzy logic) and not for an exact string match, we now tolerate misspellings (Allen, 2016). The naive, recursive approach of calculating the Levenshtein distance is not very efficient since it computes the same problems multiple times (of Computer Science Old Dominion University, 2013).

Choosing a Method

Primarily based on the fact, that we want multiple misspellings to be tolerated, we have to disregard the advantages of the Knuth-Morris-Pratt, Boyer-Moore, and Damerau-Levenshtein method. This leaves us with the Levenshtein method.

The mathematical piecewise function can be seen below, assuming that each operation has a cost of 1 (Berger-Wolf, 2016).

$$lev_{\mathrm{a,b}}(i,j) = \begin{cases} max(i,j) & \text{if min(i,j)=0,} \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{a_i \neq b_j} \end{cases} & \text{otherwise} \end{cases} \tag{8.1}$$

Wagner-Fischer - Based on Levenshtein

Wagner-Fischer is an algorithm that makes use of Levenshtein distance together with dynamic programming. The dynamic programming approach makes calculating the edit distance efficient, while the algorithm is still able to be modified according to what it is to be used for. Our implementation of the algorithm can be seen in listing 8.13.

```
    private static int _computeLevenshtein(string pattern, string
        text)
    {
        if (string.IsNullOrEmpty(pattern) || string.IsNullOrEmpty(
            text))
        {
            return 0;
        }

        const int cost = 4;
        int m = pattern.Length;
        int n = text.Length;
        int[,] d = new int[m + 1, n + 1];

        pattern = pattern.ToLower();
        text = text.ToLower();

        //Fill first column
        for (int i = 0; i <= m; d[i, 0] = cost * i++) ;
        //Fill first row
        for (int j = 0; j <= n; d[0, j] = 1 * j++) ;

        for (int j = 1; j <= n; j++)
        {
            for (int i = 1; i <= m; i++)
            {
                if (pattern[i - 1] == text[j - 1])
```

```
26                      {
27                          d[i, j] = d[i - 1, j - 1];
28                      }
29                      else
30                      {
31                          d[i, j] = Math.Min(
32                              Math.Min(
33                                  d[i - 1, j] + cost, //Deletion
34                                  d[i, j - 1] + 1), //Insertion
35                              d[i - 1, j - 1] + cost); //Substitution
36                      }
37                  }
38              }
39              return d[m, n];
40          }
```

Listing 8.13: Our version of the Wagner-Fischer algorithm

As seen in lines 8 and 33-35, we set our cost to be 4 for deletion and substitution, but only 1 for additions. This is because we want to penalise misspellings and typos, while also wanting it to show results while only receiving a substring, e.g.: typing Al should make all routes created by Alice pop up.

To make each route keep track of its own distance, we created a Tuple object, which holds a route object and a float type representing the Levenshtein distance. After computing the Levenshtein distance for all routes based on a specific search input, the routes are then sorted by the one which has an attribute, that has the lowest distance to the sought word. The attributes, that the users can search for are a route number, a section name, a grade colour and a route author. If the search includes multiple words (separated by one or more spaces), the distance for each search term is calculated for each route. These distances is then added to each route, and then sorted as described before.

The system also implements minor methods for improving the search mechanism, such as:

- If a search includes a single letter followed by one or more digits, e.g. A4 (meaning route 4 in section A), a method splits the two and search for each.
- If a search includes a keyword such as author, the following word is only compared with a routes author attribute and not the rest of its attributes.
- A method to make the search input case insensitive.
- A threshold that ensures no routes show up, if the Levenshtein distance between two strings is above the threshold.

Complexity of the Wagner-Fischer Algorithm

The complexity of Wagner-Fischer is $\Theta(qp)$ where p is the pattern length, and q is the search length. In order to find the best matching route, we evaluate every route against q and sort them by the calculated Levenshtein distance which with n routes make the complexity $\Theta(nqp)$. The sorting algorithm used is a quicksort variant implemented in the LINQ library. Quicksort has an average time complexity of $O(n\ log\ n)$. This makes our average-case time-complexity $O(nqp + (n \log n))$ for the entire search. As the amount of search-able content p is determined by the length of four factors:

- The route's number (less than four characters).
- The section's name (very likely one character).
- The name of the grade (usually less than 10 characters).
- The name of the route setter (assumed to be shorter than 40 characters).

All in all p will in a real world scenario, very likely be less than 100 characters. If q is ever longer than p, the search algorithm only gives less results than if it was using the exact number of characters or less. This bounds the length of q to p. This leaves n as the only factor that realistically have any impact, which means that in practice the algorithm is most likely upper bounded by $n \log n$. If we were to enforce limits on the length of q and p, we could theoretically have a correct average time complexity of $O(n \log n)$.

8.2 Client-side

In section 6.3, the client service component and its responsibilities were explained. In this section, we describe how we implemented the client in the system, as well as the user interface which users of the system primarily interacts with. We will also explain how we implemented comments, beta and route images.

8.2.1 Distribution

The specific client we have implemented in this project is distributed as a website. This means that anyone with a web browser can access the service. The specific browser and its version may determine if the website is rendered correctly as some of the features we use to present the page are not available in older browsers. The website is hosted on the same server as the server component of the system. This is not ideal as it makes the server accessible directly instead of through the public interface of the API. We use this on the server side to check if a user is authenticated before sending the view back to the browser. This could be done via the public API and as such this is an example of a bad practice used in the system.

```
public class ViewController : Controller {
    public IAuthenticationService AuthenticationService;

    public ViewController(IRepository<Member> memberRepository)
    {
        AuthenticationService = new AuthenticationService(
            memberRepository);
    }

    // GET: /
    [HttpGet]
    public IActionResult Routes() { return View("Views/Routes.
        cshtml"); }

    ...

    // GET: /new-route
    [HttpGet("new-route")]
    [RequiresAuth(Role.Authenticated)]
    public IActionResult NewRoute() { return View("Views/NewRoute.
       cshtml"); }

    ...

    // GET: /sections
    [HttpGet("admin-panel")]
    [RequiresAuth(Role.Admin)]
    public IActionResult Sections() { return View("Views/
       AdminPanel.cshtml"); }

    ...
}
```

Listing 8.14: The `ViewController` on the server, responsible for serving the views to the users browser

In listing 8.14 it is important to note that it has access to a `IRepository<Member>` which it uses through the `RequiresAuthAttribute` class to directly check if the token in the requesters cookie matches an authenticated user in the model instead of sending a request to the public API.

8.2.2 Client

As seen in listing 8.15, the system has a `Client` object. This object aggregates a `RouteClient`, `SectionClient`, `GradeClient`, `MemberClient`, and `HoldClient` each of whom is responsible for communicating with their respective controllers as previously illustrated in figure 6.8.

```
function Client(routeUrl, sectionUrl, gradeUrl, memberUrl,
   cookieService)
{
    this.routes = new RouteClient(routeUrl, cookieService);
    this.sections = new SectionClient(sectionUrl, cookieService);
    this.grades = new GradeClient(gradeUrl, cookieService);
    this.members = new MemberClient(memberUrl, cookieService);
    this.holds = new HoldClient(holdUrl, cookieService);
}
```

Listing 8.15: Clients

To send, retrieve, and update information in the server, we use asynchronous JavaScript and XML (AJAX). Listing 8.16 shows an example of an AJAX request. In the code snippet, we see that an HTTP GET request is made to the server and the data type response from the server is JSON. Furthermore, we also specify the data to be sent to the server and what URL the request should be sent to. At last, we specify the success parameter which is the function to be executed if the request succeeds.

```
    this.getSection = function(name, success)
    {
        $.ajax({
            type: "GET",
            dataType: "json",
            url: url + "/" + name,
            data:
            {
                name: name
            },
            success: success
        });
    };
```

Listing 8.16: Method for getting a section from the server

The client objects contain a variety of different methods, similar to the one shown above, for sending and receiving data to and from the server.

8.2.3 Implementing the User Interface

Inspired by the Model-View-View-Model (MVVM) pattern (Microsoft, 2012), we chose to separate code for each page in the user interface into two main components: a view and a view model. We did this to separate code that implements the representational design from the code that implements the state and logic of the user interface.

View Models

To enable the use cases described in section 5.1.2, we created view models to store information about the states in the different use cases as well as functions for each action that can be performed in the use cases. We will now, based on the use case 5.2, show how we have implemented view models.

The Viewing Route Information use case has two overall states. In the first state, the actor can search and filter the list of routes that are available in the system, as well as sort the routes presented to them. To represent this state, we create a view model class called `RoutesViewModel`. The information that describes this state is stored as attributes on the `RoutesViewModel` class. The different methods of the `RoutesViewModel` class are shown in figure 6.5.

The list of routes is stored in the route's attribute, which is updated in the `refreshRoutes` function according to the current value of the `selectedGrade`, `selectedSection`, and `selectedSortBy` attributes. The value of these attributes is changed by the `changeGrade`, `changeSection`, or `changeSortBy` functions, to one of the available values in the grades, sections, or sortOptions lists. Instead of filtering routes by specific options, the search function updates the `routes` attribute to a list of routes that matches a given search string without applying any of the filters or search order. To provide these actions, the `RoutesViewModel` is dependent on a client instance that has functionality for receiving routes from the Controller. This dependency is injected by the constructor of the `RoutesViewModel` and enables different implementations of a client instance to be used by the view model. This also enables the use of a mock-up client instance, for unit testing the view model without interfering with other components.

An important part of each view model is the `EventNotifier` function, which has three sub-functions: `addEventListener`, `trigger`, and `removeEventListener`. This enables other components such as views to add an event listener to certain events that are triggered by the view model.

Views

The last component in our system's dependency chain is the view component. This component has the responsibility for the representational design of the system. Representational design concerns the style and layout of how information is presented to the user (Benyon, 2014, p. 54). For each view model there is a corresponding view that visualises the information stored in the view model. In our system, the view consists of two components. The first component is a set of HTML handlebars.js templates (Katz, 2016). The second component is a code behind JavaScript file that adds event listeners to the view model, which renders the template with the latest state of the view model when it triggers certain events.

For the `RoutesViewModel`, the corresponding view is represented by the two files Routes.cshtml and routes.js. A part of the template for the list of routes is shown in listing 8.17.

```
 1 <div class="content-section route-list">
 2     <div>
 3         <h2>Number of Routes: {{routes.length}}</h2>
 4     </div>
 5     {{#each routes}}
 6     <a href="route-info?routeId={{id}}">
 7         <div class="route-card">
 8             <div class="route-card-top">
 9                 <div class="route-grade-number">
10                     <span class="number" style="background-color:
                           rgb({{grade.color.r}},{{grade.color.g}},{{
                           grade.color.b}})">
11                         <div style="{{#if_light grade.color}}color
                               : black{{/if_light}}">
12                         {{name}}
13                         </div>
14                     </span>
15                 </div>
16                 <span class="route-section">Section: {{sectionName
                       }}</span>
17                 ...
18     {{/each}}
```

Listing 8.17: Shows the first part of the route-list-template that renders the route number, grade colour and section for each of the routes in the list.

Listing 8.17 shows how Handlebars.js syntax is used to implement a template for representing the list of routes stored in the view model. At line 3, we use the {{attribute}} syntax to specify that when the template is rendered with the current state of the view model it should replace {{routes.length}} with the current value of `routes.length`. This inserts the number of routes currently stored in the view model. At line 5, we use the {{#each}} attribute syntax to render each route in the routes attribute. Between {{#each routes}} and {{/each}}, we implement a template that is rendered for each route. At line 6, {{id}} now refers to the id attribute of a single route object. This is similar for name/number of the route at line 12 using {{name}} and the name of the section {{sectionName}} at line 16.

The template shown in code snippet 8.17 is rendered when the *routesChanged* event is triggered by the RoutesViewModel. This is configured as shown in listing 8.18.

```
1      var client = new Client(API_ROUTE_URL, API_SECTION_URL,
          API_GRADE_URL, API_MEMBER_URL, API_HOLD_URL, new
          CookieService());
2      headerViewModel = new HeaderViewModel("Find Route", client);
3      viewModel = new RoutesViewModel(client, new LoadingService());
4
5      var configurations = [
6          ...
7          {
8              scriptSource: "js/templates/route-list-template.
                  handlebars",
9              elementId: "routes-content",
10             event: "routesChanged",
11             viewmodel: viewModel
12         }
13     ];
14
15     setUpContentUpdater(configurations, function() {
16         viewModel.init();
17         headerViewModel.init();
18     });
```

Listing 8.18: Shows how the route-list-template is configured to be rendered into an HTML element that has "routes-content" as id when the view model triggers the *routesChanged* event.

For each template used in the view there is an object that stores information about the connection between the template and the corresponding view model. This is shown for the route-list-template at line 7-12, where scriptSource is the path to the file containing the template.

`elementid` is the id of the HTML element that the template is rendered into, located in Routes.cshtml.

`event` is the event(s) triggered by the view model that causes the template to render the newest state of the view model.

The configurations for all the templates in the view is assigned as an array to the variable content at line 5 and passed to the `setUpContentUpdater` function at line 15 which ensures that all the configurations are initialised.

8.2.4 Implementing Route Images

One of the should-have requirements from the MoSCoW analysis was that *"the system should allow members to add an image of a route"*. While designing the model component in section 6.3.1, we modelled a route such that a route aggregates one image, and the image aggregates a number of holds.

This structure is mirrored in our code such that a route contains one image, and the image contains a list of holds. Listing 8.19 shows the Image class' properties. The `Image` class inherits the `FileUrl` property from the `Media` class. The `FileUrl` will contain the reference to the image file, which will either be a URL pointing to a file, or a data URI containing a base64 encoded image.

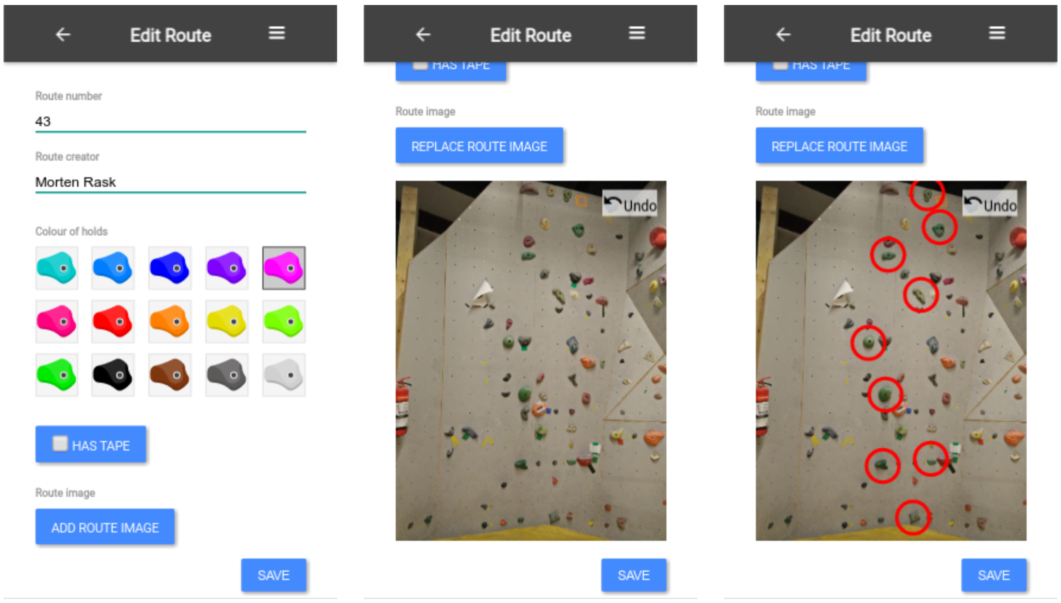
```

public class Media : Model
{
    public string FileUrl { get; set; }
}

public class Image : Media
{
    public Image()
    {
        Holds = new List<Hold>();
        Id = Guid.NewGuid();
    }

    [JsonIgnore]
    public Route Route { get; set; }

    public Guid RouteId { get; set; }

    public uint Width { get; set; }

    public uint Height { get; set; }

    public virtual List<Hold> Holds { get; set; }
}
```

Listing 8.19: Properties in the Image class

While climbing, we noticed that because of the sheer number of holds on the climbing walls, it was sometimes difficult to find the next hold on the route. While some would argue that this is part of the challenge of climbing, we noticed that because we were novices in the climbing club, we accidentally used parts of other routes because we thought that they were part of the route that we were climbing.

To combat this problem, we wanted to give the members of the climbing club the ability to show the locations of the holds on the image of the route. To do this, we created a JavaScript object called `RouteCanvas` which transformed a HTML canvas element into an image of the route where they could click on the holds, and the hold would be added to the image.

When no image is added, the member can choose to add one as shown in figure 8.3a. When the member has chosen an image from their phone or taken one with their camera, that image is shown as illustrated in figure 8.3b. After that, the user can add holds by clicking on them on the image, after which they will show up on the image as illustrated in figure 8.3c.



(a) Before adding an image to the route (b) After choosing an image from phone storage or camera (c) After adding holds to the image

Figure 8.3: Process of adding an image to a route.

The holds are saved separately to the image, which makes undo-functionality very easy to implement, as we can treat our holds as a stack, and when the undo button is pressed, we simply pop our stack. The code for this functionality is shown in listing 8.20.

```
RouteCanvas.prototype.undo = function() {
    if (this.viewModel.HoldPositions.length){
        this.viewModel.HoldPositions.pop();
    }
    this.latestClick = null;
    this.DrawCanvas();
}
```

Listing 8.20: Undo funcionality of RouteCanvas

To make sure that the RouteCanvas always represent the ViewModel that is in charge of the current view, we added the following line to the RouteCanvas:

```
this.viewModel.addEventListener("HoldsUpdated", this.DrawCanvas);
```

What this does is that whenever the hold positions are changed in the ViewModel and the ViewModel triggers the `HoldsUpdated` event, the canvas will redraw with the newest holds. This also means that instead of keeping track of the hold positions in the RouteCanvas, we use the ViewModel to add and remove holds to make sure that there is no syncing problem between the objects. In listing 8.21, it can be seen how the ViewModel triggers the `HoldsUpdated` event that the RouteCanvas is subscribed to.

```
this.addHold = function(hold)
{
    self.HoldPositions.push(hold);
    self.trigger("HoldsUpdated");
}
```

Listing 8.21: Viewmodel triggers custom event

Originally, we implemented the RouteCanvas such that to draw a circle on the image, you would touch where the centre of the circle would be, and then drag your finger such that the distance from the centre to your finger would be the radius of the circle. This worked very well when we tested it locally on our computers with a mouse, but when we tested it on a smartphone, we found that it was not intuitive, it was difficult to see how large the circle was due to your finger blocking your vision, and lastly; it was very easy to accidentally add a hold, when you actually meant to scroll up or down on the page.

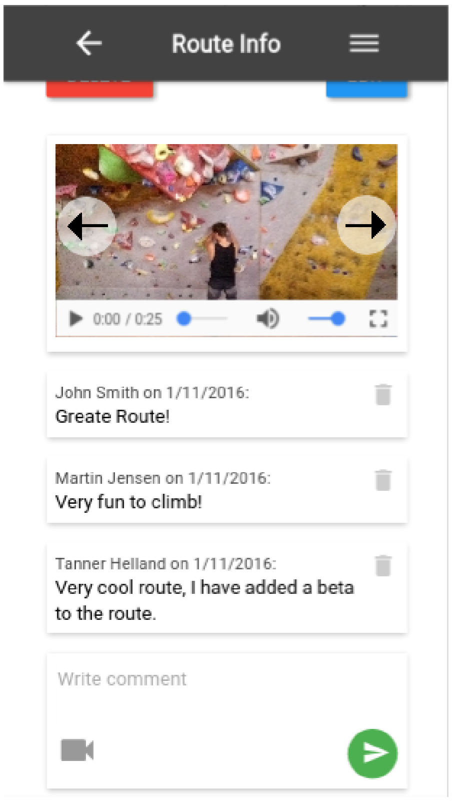
So we chose to implement the canvas in such a way that the holds are added when you touch, and then every subsequent touch on the hold will make the radius larger.

8.2.5 Implementing Comments and Betas

As mentioned in section 6.3.1, we wanted to represent video betas as parts of comments, instead of individual objects. The reason for this choice was that we wanted to be able to merge a text comment and video beta in a shared feed. We wanted to have this functionality to promote the use of the comments for being more than commenting on route design. We wanted the comments to be a place where members of the climbing club can ask for help as well as receive it.

Originally, we designed the video betas as being on top of the page, where the users could scroll through them. This is illustrated on figure 8.4a. Under the route information would be the video betas that the member could scroll through, to easily find videos that would help them climb a route.

The reason we only have video betas and not audio or image betas, is because a video is the only medium that makes sense to use for teaching other members how to climb a route, other than a comment. To maximise the ability for members to help each other in the comments, we needed video betas to be a part of those instead of being separate.


Route Info

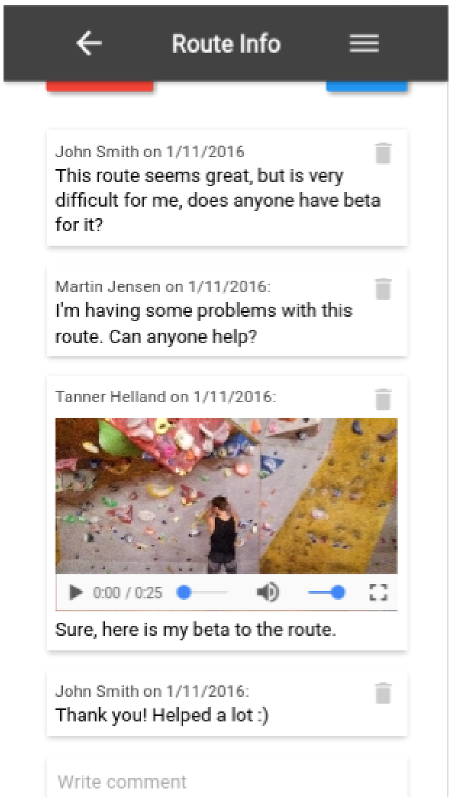
John Smith on 1/11/2016:
Greate Route!

Martin Jensen on 1/11/2016:
Very fun to climb!

Tanner Helland on 1/11/2016:
Very cool route, I have added a beta to the route.

Write comment

(a) First idea layout containing comments and betas


Route Info

John Smith on 1/11/2016
This route seems great, but is very difficult for me, does anyone have beta for it?

Martin Jensen on 1/11/2016:
I'm having some problems with this route. Can anyone help?

Tanner Helland on 1/11/2016:
Sure, here is my beta to the route.

John Smith on 1/11/2016:
Thank you! Helped a lot :)

Write comment

(b) Better layout containing comments and betas together

Figure 8.4: Different layouts of comment/beta section.

Figure 8.4b shows the other layout where comments and betas are merged in the feed. The figure illustrates how having the video embedded in the comments offers the ability to use the video as a response to other members' questions about a route, instead of just adding general betas to the route.

Implementing this in the system was simply a matter of adding a `Video` object to the `Comment` object, and display the video object on the front-end if it was present.

The layout made it possible to use a single place for uploading both betas and comments. We simply added a video icon to the comment form, and upon the button being pressed, it would open a dialogue window, asking to specify which application to use for uploading a file. This could e.g. be opening their camera. If the camera was selected, the user would be able to select/record a video, that would then be embedded in the comment.

Summary

In this chapter the description of how the model classes were created using C# and mapped to tables in an SQLite database using Entity Framework, Code First, was presented. Furthermore, logging in to the system was described along with the token authentication process related to it. Through code listings it was shown how an `IAuthenticationService` interface in the

`MemberController` class stores a token in the database, in order to authenticate users. In section 8.1.5 the three algorithms, Knuth-Morris-Pratt, Boyer-Moore, and Wagner-Fischer, all related to searching were analysed. Based on the advantages and disadvantages of each algorithm the Wagner-Fischer algorithm was chosen. It relies on dynamic programming to compute the edit distance of two strings. A general implementation of an API response was shown, and it was explained how the client uses AJAX requests to send and receive data to and from the server. Furthermore, the user interface, and how different events in the viewmodels update the views, were explained. In section 8.2.4 support for images in the system was illustrated. In addition to this, it was shown how that that the system also supported the marking of holds in the images.

As the implementation of the system became realised, it also became necessary to test the code regularly, as described in the next chapter.

Chapter 9

Testing

In this chapter, some theory behind software testing is covered and the way in which the different parts of the system were unit tested, is described. This is done by selecting different unit test frameworks for testing. Furthermore, how the frameworks were used to create tests for the system, is shown using code snippets and test results.

The choice of unit testing both the front and back-end is explained and described. Reasons for using unit testing, such as to make sure new changes did not break existing functionality before usability tests, are also described in the chapter.

9.1 Purpose of Testing

It was decided to do unit testing for both the front-end and the back-end of the system, since we did not want existing features to break when we added new features or changed our implementation in each iteration. The exception to this, was when we changed or added features that we knew would cause existing tests to fail, in which case, the affected tests needed to be updated.

The intention of unit testing is to split up the code into smaller parts and test those independently from the rest of the project to see if they behave as expected (Microsoft). Unit tests were especially useful to check if new changes broke existing functionality in a way we did not expect them to, before conducting our usability tests. Using unit tests also saved a lot of time that we would otherwise have spent checking if our programme worked as intended, since unit tests automates this process (Burke and Coyner, 2003).

Testing our software, is also part of this semester's curriculum (Thomsen, 2015, p. 22).

9.2 Unit Testing Frameworks

According to the curriculum, the code written must be tested to ensure that the code works (Thomsen, 2015). We decided to unit test the project as this will allow us to pinpoint where the errors in our codebase are, by testing each individual part of the system. To unit test our C# backend, there are multiple frameworks that can provide ways to write unit tests efficiently, examples of these frameworks are NUnit, MSTest, and xUnit.net. We decided to use NUnit to unit test our system, as it integrates nicely with ASP.NET Core, and all group members have worked with this framework before. After installing this framework into our project, it can be run easily with the command "dotnet test".

We can use NUnit to test our APIs and database, but as substantial part of our system is in our viewmodels, we need to test those as well. Here we have decided to use the JavaScript unit testing framework QUnit.

9.2.1 NUnit

NUnit is a testing framework for the .NET platform (NUnit, 2016), and can be used for ASP.NET Core projects. The testing framework provides easy ways to determine whether the output of the system is as expected, and more advanced methods to determine if a method throws the correct exceptions, given a certain input.

For .NET Core, the testing framework can be included to the project by adding a few additional lines into the Project.json file, since the testing framework is a nuget package (NUnit, 2016). This allows for testing by adding `Using NUnit.Framework;`. Before the test class, an attribute called `[TestFixture]` needs to be added in order to tell the framework that the class contains unit tests.

NUnit also allows for setup and tear down methods, which can be used to initialise variables used for each test, and remove the values from each variable once each test has run. This is necessary since each test needs to run in a clean environment. The attribute that needs to be added to each method in order to specify whenever the method is used to set up the environment, and to tear it down, is called `[SetUp]` and `[TearDown]` respectively.

Each test needs an attribute called `[Test]` in order to specify that the method is a test.

A simple test class without making use of the optional `[SetUp]` and `[TearDown]` attributes, would look like listing 9.1

```
using NUnit.Framework;

namespace Tests
{
    [TestFixture]
    public class TestClass
    {
        [Test]
        public void Method_Input_ExpectedResult()
        {
            var result = Method(Input);
            Assert.IsTrue(result == ExpectedResult);
        }
    }
}
```

Listing 9.1: Simple test class with one test

9.2.2 QUnit

QUnit is a JavaScript unit testing framework created by the JQuery foundation and is used for testing JQuery, a JavaScript library which was used on 61 percent of the most popular websites in 2014(Methvin, 2014).

The framework itself is a webpage, which will execute the unit tests and show the results. Figure 9.3 shows an example, of how this page can look.

The QUnit tests are very versatile. They work by calling the QUnit.test function that takes another function as a parameter. That function will then be executed in the test, and in that function you can make calls to the assert function which will be used to determine if the test succeeds or fails. The example provided by the QUnit documentation can be seen in listing 9.2.

```
    QUnit.test( "hello test", function( assert ) {
        assert.ok( 1 == "1", "Passed!" );
    });
```

Listing 9.2: QUnit example from documentation

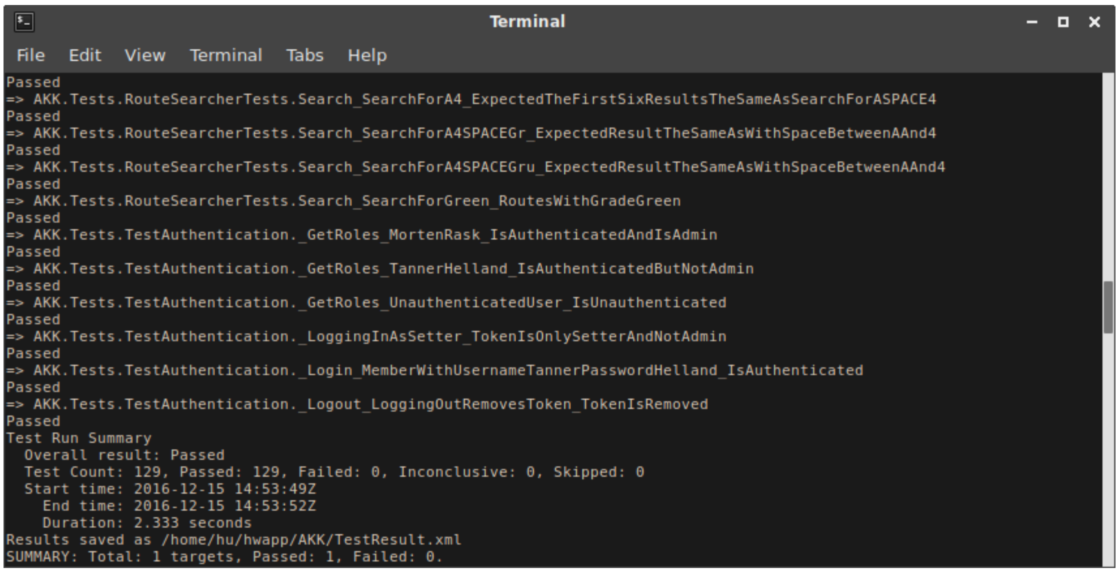
9.3 Controller Test

In this section, selected unit tests for the controller methods is shown and explained. These tests help us solve and discover previously unseen bugs.

When creating a programme, it is important that it works, and works well. We can help ensure this by creating unit tests for each public method in our programme, to make sure these methods work as intended.

The programme contains the following controllers: Section Controller, Route Controller, Grade Controller, Member Controller, and Hold Colour Controller.

To create the tests, we used the NUnit testing framework. All of the unit tests can be found in the electronic appendix inside the Tests directory.



```
Passed
=> AKK.Tests.RouteSearcherTests.Search_SearchForA4_ExpectedTheFirstSixResultsTheSameAsSearchForASPACE4
Passed
=> AKK.Tests.RouteSearcherTests.Search_SearchForA4SPACEGr_ExpectedResultTheSameAsWithSpaceBetweenAAnd4
Passed
=> AKK.Tests.RouteSearcherTests.Search_SearchForA4SPACEGru_ExpectedResultTheSameAsWithSpaceBetweenAAnd4
Passed
=> AKK.Tests.RouteSearcherTests.Search_SearchForGreen_RoutesWithGradeGreen
Passed
=> AKK.Tests.TestAuthentication._GetRoles_MortenRask_IsAuthenticatedAndIsAdmin
Passed
=> AKK.Tests.TestAuthentication._GetRoles_TannerHelland_IsAuthenticatedButNotAdmin
Passed
=> AKK.Tests.TestAuthentication._GetRoles_UnauthenticatedUser_IsUnauthenticated
Passed
=> AKK.Tests.TestAuthentication._LoggingInAsSetter_TokenIsOnlySetterAndNotAdmin
Passed
=> AKK.Tests.TestAuthentication._Login_MemberWithUsernameTannerPasswordHelland_IsAuthenticated
Passed
=> AKK.Tests.TestAuthentication._Logout_LoggingOutRemovesToken_TokenIsRemoved
Passed
Test Run Summary
  Overall result: Passed
  Test Count: 129, Passed: 129, Failed: 0, Inconclusive: 0, Skipped: 0
  Start time: 2016-12-15 14:53:49Z
    End time: 2016-12-15 14:53:52Z
    Duration: 2.333 seconds
Results saved as /home/hu/hwapp/AKK/TestResult.xml
SUMMARY: Total: 1 targets, Passed: 1, Failed: 0.
```

Figure 9.1: The last part of the output of ”dotnet test”.

For each controller in the system we created multiple unit tests. Figure 9.1 shows the output from the terminal when running “dotnet test”, to run the tests.

9.3.1 Section Controller

The section controller is what allows a client to manipulate sections and their routes in the model. One of the methods of the `SectionController` is the `GetAllSections` method, which returns all sections in the system, ordered by their name. To test if this method works as intended, we compare the value we get after calling the method, with the actual repository, to see if we got the expected amount of sections returned. Another test, tests to see if we also got the same amount of routes, because if we do not, the system is not working as intended. We then use the method `CollectionAssert.AreEquivalent` on the two collections, in order to check if they are equivalent. If both tests pass, then we assume that the method `GetAllSections`, works as intended.

9.3.2 Route Controller

The route controller, is what allows a client to manipulate routes. This controller specifies methods such as `AddRoute`, `DeleteRoute` and `GetRoutes`.

The last-mentioned method might at first glance seem like it takes many arguments, but most of them are null-able. By default, the `GetRoutes` method returns all routes if everything but the sortOrder is null.

To test this method, several tests have been created to see if only the routes specified by the input values, get returned. In the first test, only the sortOrder is specified. In the next, grade has also been specified, and so on. The test, where sectionId has been specified, can be seen below:

```
    [Test]
    public void _GetRoutes_GettingRoutesFromSectionWithID
                _ExpectOnlyRoutesFromThatSection()
    {
        var section = _sectionRepo.GetAll().First();
        var response = _controller.GetRoutes(null, section.Id,
           null, 0, SortOrder.Newest);
        var routes = response.Data;

        Assert.IsTrue(response.Success);

        CollectionAssert.AreEquivalent(section.Routes, routes);
    }
```

Listing 9.3: RouteController unit-test for the GetRoutes method

In listing 9.3, we find the first section in the system, then call the `GetRoutes` method with the found section.Id. We then check to see if the method actually did what was required of it, by asserting that the `Success` property is true. Finally, we expect the routes in the section and the data we received through the return-value to be equal, so we compare the two collections.

9.3.3 Grade Controller

The grade controller handles the grades in the system and specifies methods such as `AddGrade` and `DeleteGrade`.

Testing these methods required creating a new `Grade`, so to test the `DeleteGrade` method, it was easier to first test the `AddGrade` method. We did this, by creating a `Grade` object, then calling the method and checking its return-value to see if it corresponds to the `Grade`, we just added.

Since these tests passed, we could just try and call the `DeleteGrade` method to remove the `Grade`, then try and find it in the grade repository. If no such grade could be found, the grade got deleted from the system.

9.3.4 Member Controller

The member controller handles the registered users of the system and specifies methods such as `Login` and `Logout`.

Testing these methods can be done by inputting a username and a password for an already existing member in the system, then check if the token received gets assigned to that member. This token is proof that the member is now authenticated, and allows them to call methods such as `AddRoute`, from the route controller. Testing if the token allows a member to do so, is tested in tests for the other controllers, since those methods requires an authenticated user to call them. The `Logout` method can be tested by calling the method for an already authenticated user, and see if the `Token` property, gets assigned to null.

9.3.5 Hold Color Controller

The Hold color controller handles the the colours of the holds which a user can choose between, for a route. This controller was necessary to create so that a user of the system, would not be able to add an arbitrary hold colour to a route. This also allows an administrator to add new hold colours and delete existing ones.

The hold color controller specifies methods such as `AddHoldColor` and `DeleteHoldColor`. These methods can be tested in similar way as the `AddGrade` and `DeleteGrade` methods, and how to do this, is described previously in section 9.3.3.

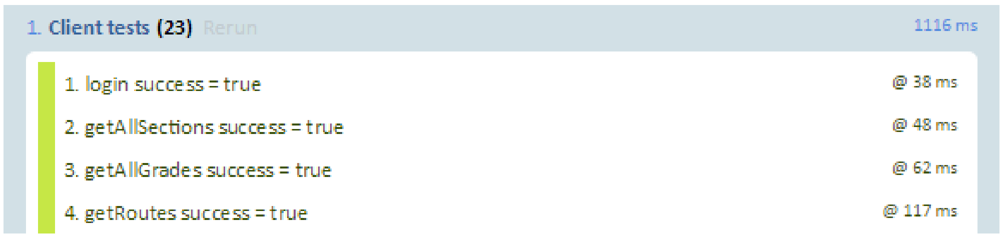
9.4 Client Service Tests

In this section we describe how we tested the client service component. The responsibility of the client service component is to make the interface for the server-side API available at client-side. Because each method in the client service component is just a single web request to a service-side controller method then it is difficult to unit test the client service component alone. Instead we are interested in testing the connection between the client service component and the controller component. We also chose to use this test to validate changes in the API by checking the data responded from the API. We do not test all the 30+ methods in the client service component. We focused on testing the methods that concerns authentication and administration of routes because these are the methods that affects most users if there should be any bugs in these.

```
QUnit.test("Client tests", function( assert ) {
  var client = new Client(API_ROUTE_URL, API_SECTION_URL,
      API_GRADE_URL, API_MEMBER_URL, API_HOLD_URL, new
      CookieService());
  var done = assert.async();
  login(assert, done, client);
});
```

Listing 9.4: Instantiating the client service component for testing and invokes the register test function.

Because all methods of the client service component runs asynchronously we use `assert.async()` to store a function that is executed after the last test to inform QUnit that all tests are done. The tests is divided into the following functions: `login`, `getAllSections`, `getAllGrades`, `addRoute`, `updateRoute`, `getRoutes`, `getRoute`, `getImage` and `deleteRoute`. A part of the test result is shown in figure 9.2. A more comprehensive list of the test results is shown in appendix G.



```
1. Client tests (23) Rerun                                  1116 ms
  1. login success = true                                   @ 38 ms
  2. getAllSections success = true                          @ 48 ms
  3. getAllGrades success = true                            @ 62 ms
  4. getRoutes success = true                               @ 117 ms
```

Figure 9.2: Shows a part of the test result for the client service component.

9.5 View Model Unit Tests

In this section, we explain how we conducted tests for the view model component using the QUnit framework. The view models in the system, are: NewRouteViewModel, EditRouteViewModel, FindRouteViewModel, AdminPanelViewModel, RegisterViewModel, RouteInfoViewModel, and LoginViewModel. In order to test the view models, we focused on two things; checking if events got triggered in the view model and that functions worked as expected e.g. that properties changed correctly.

To demonstrate how we tested the view models, we look at an example from the tests we conducted on the `FindRouteViewModel`. On lines 3-5 in listing 9.5 we add an eventlistener that sets a local boolean variable to true if the event *sectionsUpdated* is triggered. On line seven we call the function `downloadSections` which in the system requests the client to get all sections from the database. However for this test case, we made a test client that returned a number of predefined routes. We did this because we did not want our view model tests to be dependent of the client, and because the client uses asynchronous functions, which meant we do not know when the functions in the view model, gets triggered. After having called the `downloadSections` function to trigger the *sectionsUpdated* event, which downloads a list of sections from the database to the view model, we check how many sections we have in our view model that correspond to the predefined sections (line 11-14). On line 16, we then check if the number of sections in the view model is the same as the number of predefined sections, and on line 18 we check if the event *sectionsUpdated* was successfully triggered.

```
 1      var sectionUpdatedTriggered = false;
 2
 3      viewModel.addEventListener("sectionsUpdated", function () {
 4          sectionUpdatedTriggered = true;
 5      });
 6
 7      viewModel.downloadSections();
 8
 9      var times = 0;
10      if (viewModel.sections.length > 0) {
        for (var i = 0; i < viewModel.sections.length; i++) {
            if (viewModel.sections[i] == TEST_SECTIONS[i]) {
                times++;
            }
        }
        assert.equal(times, viewModel.sections.length, "
           RouteViewModel downloadsections");
    }
    assert.equal(sectionUpdatedTriggered, true, "RouteViewModel
       sectionsUpdated triggered");
```

Listing 9.5: Unit test example for the FindRouteViewModel

For each view model in the programme we conducted similar tests. Figure 9.3 shows a part of the rest result.

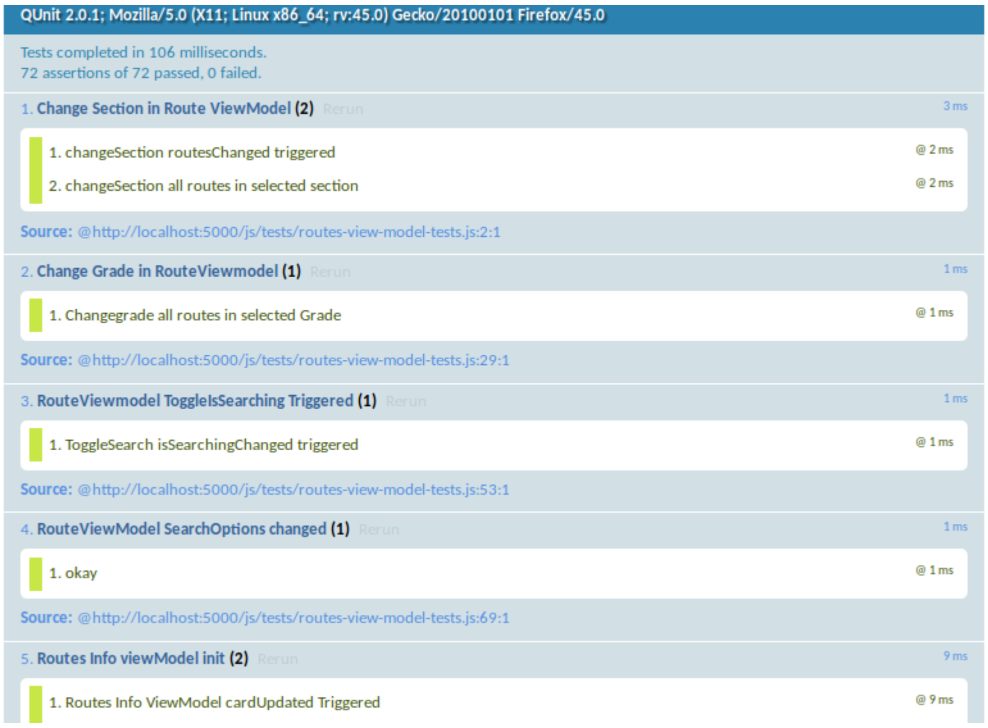

```
QUnit 2.0.1; Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0

Tests completed in 106 milliseconds.
72 assertions of 72 passed, 0 failed.

1. Change Section in Route ViewModel (2) Rerun                                    3 ms
   1. changeSection routesChanged triggered                                       @ 2 ms
   2. changeSection all routes in selected section                                @ 2 ms
Source: @http://localhost:5000/js/tests/routes-view-model-tests.js:2:1

2. Change Grade in RouteViewmodel (1) Rerun                                       1 ms
   1. Changegrade all routes in selected Grade                                    @ 1 ms
Source: @http://localhost:5000/js/tests/routes-view-model-tests.js:29:1

3. RouteViewmodel ToggleIsSearching Triggered (1) Rerun                           1 ms
   1. ToggleSearch isSearchingChanged triggered                                   @ 1 ms
Source: @http://localhost:5000/js/tests/routes-view-model-tests.js:53:1

4. RouteViewModel SearchOptions changed (1) Rerun                                 1 ms
   1. okay                                                                        @ 1 ms
Source: @http://localhost:5000/js/tests/routes-view-model-tests.js:69:1

5. Routes Info viewModel init (2) Rerun                                           9 ms
   1. Routes Info ViewModel cardUpdated Triggered                                 @ 9 ms
```

Figure 9.3: The unit tests for the ViewModels.

Summary

In this chapter, it was briefly described which unit testing frameworks were used to test the system and its methods. These unit tests allowed focusing on developing without fear of accidentally breaking existing features without realising it. The fact that both the front end and back end of the system were unit tested was discussed, and the reason behind it explained.

Testing was described as a way to make sure existing features worked the way they were intended, even when new features were implemented, so there was a reliable way to find out if the system worked, when conducting the usability evaluations as described in the next chapter.

Chapter 10

Usability Evaluation

In this chapter, the system is evaluated. First, different methods for evaluating a software system are described. It is then explained how some of those methods are used to test our system. The usability evaluations of the system that were conducted are also described and their findings shown and analysed.

10.1 Usability

Usability is essential for good interaction design. An example of good usability through affordance is a button, which looks like other buttons, be it in the real world or in software. We know from experience that buttons afford being pressed. If an interactive element in software resembles a button, we expect that it is press-able. As described in section 7.4.1 the design of the system is based on Google's Material Design, which means that users are at least somewhat familiar with the design of the interactive elements in the system. It follows that a user interface element that looks press-able but is not, is considered bad design. According to Benyon (2010, p.80-94), a system is often defined as usable if it possesses the following properties:

- Efficiency: achieving results in an appropriate amount of time.
- Safety of use: can safely be operated in the contexts it is being used in.
- Utility: the ability to solve a given problem.
- Ease of use: how easy the system is to use for people new to the system and people experienced with it.

10.1.1 Usability Testing

Testing for good usability has become essential in later years since nobody wants to buy a piece of software, which its users find frustrating to use. During the development of a system, it is important that the developers conduct usability testing during the different phases of development. If they do not, they run the risk of having a finished design, that does not fulfil the requirements of a system with good usability. This may severely limit the competitiveness of the system and potentially mean having to re-do several parts of it.

Testing for usability becomes relevant almost as soon as the requirements of the system have been found. The testing should then happen throughout the preliminary design phase, and ideally every time major changes are made to the system. There exist a few methods for doing this, two of which is described below.

Heuristic Inspection can be done by usability experts, which means that there is a possibility for the developers of the system to do this type of testing themselves. This can save the developers a lot of work and resources, since arranging meetings with future users can be both expensive and difficult. By using a checklist of common usability problems and analysing the system in great detail, the usability experts will be able to find a lot, if not most, usability problems. It can be problematic for the developers to be a part of the usability test of their own system, since they are easily biased towards their own system.

Even though they are following a checklist and should strictly test within the confines of this checklist, it is easy for them to miss some glaring mistakes, in the same way that it is possible to miss grammatical errors, when proof-reading one's own text, despite knowing what mistakes to look for. It is often cosmetic mistakes, such as the size of a click-able button, in the system that are identified this way (Rubin, 2008, p.19).

User-based Evaluation requires the end-users of the system to test it during the development to find the areas of the system that are lacking in usability. Finding people and a date where they are available, and willing to help out with the development can be extremely difficult, since not everyone is willing to spend their spare time. This can mean that it might be necessary to pay prospective users to have them test the system.

An integral part of the evaluation is to ask the users to complete certain tasks, perhaps giving them some data they need to enter into the system or ask them to trigger or use a certain functionality in the system. The user is then asked to think aloud so the observers can get an understanding of exactly what the user is thinking and how they are trying to accomplish the given task. Frustrations that the user might have when using the system should be identifiable in this manner. It is important to repeat the same tasks with each different test-person, not only to confirm that several users identify the same problem, but also to determine the severity of the problem.

Usability problems are not the only kind of problems that can be identified with this method of evaluation, despite it being the main focus. If the tasks for the user are well thought out, the evaluation can also be used to see if core functionality is present and in working order in the system.

It is the test monitor's job to make users feel as comfortable as possible while they are trying out the system. Failure to do so, could lead to users being frustrated about parameters that are outside the scope of the system.

Since the system is tested on the potential user-base, there is a greater chance for the issues to be found to be more severe usability problems, that are hard to identify by looking for the most commonly occurring ones, as a Heuristic Inspection does (Benyon, 2010, p.232-235).

Location of usability evaluations is something to consider when performing user-based evaluations. The usability tests can either happen in a laboratory or as field studies (Kjeldskov, 2016). The strengths of evaluating in a laboratory is that the laboratory grants a very high level of control of the situation, to the ones performing the test. In the laboratory the observers are hidden behind a one-way mirror which means that a lot of people can be watching without the test subject being overwhelmed. The laboratory also removes disturbances from the outside world, which means that the test conditions can be closely replicated for each test subject. The downside of using the lab is that it is difficult to give the test subject a sense of realism (Kjeldskov, 2016). It is possible to replicate parts of the use context in the lab to enhance the realism, but it will never be exactly the same as using the system in real life.

If a greater sense of realism is needed, the usability test can be performed as a field study, meaning that the test takes place in the actual location of where the system will be used after development. This type of usability testing gives less control of the situation to the people performing the test, but ensures a high level of realism for the test persons. This includes distractions from the surrounding environment, which may make the test more difficult to analyse, but also to test that the system works in a workplace where distractions inevitably will happen.

10.1.2 Analysing Usability Testing

Analysing the usability testing is essential to understanding which kinds of usability problems exist, why the users find them problematic and what their thoughts were when they encountered them. Two methods of analysing the gathered information can be used to identify problems:

Video Data Analysis (VDA) requires a camera and microphone setup to capture picture and sound from the usability test, which can then be analysed after the conclusion of the test. The user's facial expressions, behaviour, and comments can be used to learn which parts of the system the user finds problematic and why. To document the VDA, log files of identified problems, both major and minor are created.

Instant Data Analysis (IDA), is another type of analysis which has the advantage that it requires significantly less time in analysing the usability testing than other more traditional approaches since it can be done on the run (Kjeldskov et al., 2004). Compared to VDA, it only requires one more person who can hear what the tester is saying, can see how they react and can watch what the tester is doing while interacting with the software. While observing they can document everything while the testing is in progress and everything is fresh in their mind. After the tests, the analysers can discuss what happened and since the severity of usability problems can be defined by how frequently it is being discovered, the more severe usability problems are far more likely to come up. This method finds most of the severe problems that VDA also finds, but not as many minor cosmetic issues, which are not that important, to begin with (Kjeldskov et al., 2004). To efficiently use IDA, one must have quite a lot of experience with the concept of usability testing. It is recommended that people have spent several hours doing VDA on separate projects before attempting IDA, since it is based largely on quickly being able to stop problems that are similar to ones that have been seen in other projects (Kjeldskov et al., 2004).

10.1.3 Usability Testing during the Project

Since the project had a relatively short timescale, we only conducted three user-based tests. This means that a lot of the usability testing was done with heuristic inspection. Of course, as one of the requirements was that the system should be easy to use, we also had to conduct usability tests with the users. After each iteration we performed a field study usability test with members of the climbing club, while standing in the bouldering hall next to the walls. We considered doing a laboratory test, but decided that doing a field study would allow us to test the system while other climbers were in the hall, as well as granting us the ability to use the actual routes on the walls in our tasks.

For our first usability test, two people were performing the test, the roles of these two people were:

A test monitor who stood next to the test subject during the test and made sure that the test subject understood the task, as well as made sure that the test subject were thinking aloud.

A data logger who stood out of the way of the test subject, and took notes during the test about the tasks as the test subjects were solving them.

From our first experience with usability testing we learnt that it would be preferable to have a video of the usability test. So we added another role to the test:

A video operator whose task was to ensure that a camera was recording at all time during the usability test, so that parts of the usability test could be watched again at a later time.

For each test subject we had two recordings, one recording of the subject recorded with a video camera, and one screen recording of the device they were using. The device was set up in a way such that each tap on the screen was easily visible, giving us the ability to easily see what the user was doing on the device.

After the tests we then synced up the two recordings into a single video, where the device was visible next to the real life recording of the test subject. A single frame from a recording can be seen in figure 10.1.

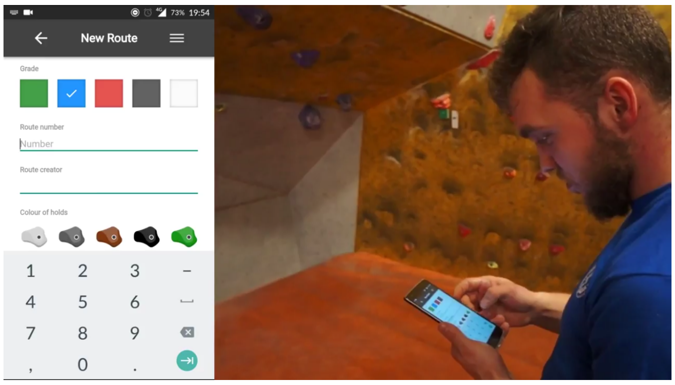


Figure 10.1: A single frame from the video recording of a usability test

For our final usability test, we used VDA to make sure we found as many usability problems as possible. Since IDA relies too heavily on the previous usability experience of those conducting it, and none of the group's members have had any experience with usability testing, it was decided that we would not use it.

10.2 First Usability Test

To test the system that was created during the first iteration we conducted a user-based evaluation. At that point the system could show the complete list of routes, filter the routes by section, grade and change the sort order. It was also possible to add new routes to the system as well as edit existing ones. The views that the user could interact with are seen in figure 10.2.

Because of the limited scope of the system at this point, there was only four tasks:

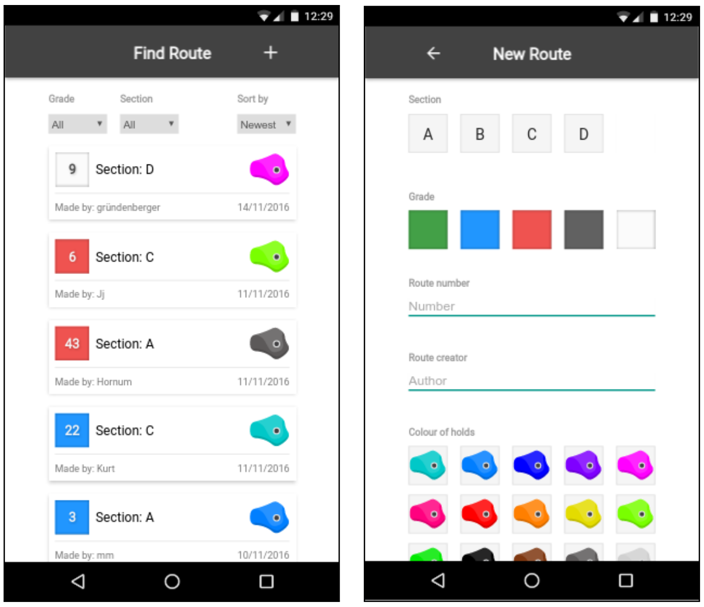
Tasks

a) In the system you see a list of routes - find route blue 38 in section c.

b) Find a route that is created by Hans.

c) Find a route on the wall and add it to the system.

d) You find that the route you have just created should have had the black grade. Change this in the system.


12:29
Find Route +
Grade: All
Section: All
Sort by: Newest

9 Section: D
Made by: gründenberger 14/11/2016

6 Section: C
Made by: Jj 11/11/2016

43 Section: A
Made by: Hornum 11/11/2016

22 Section: C
Made by: Kurt 11/11/2016

3 Section: A
Made by: mm 10/11/2016

12:29
New Route
Section: A B C D
Grade
Route number: Number
Route creator: Author
Colour of holds

(a) Route List View (b) Add Route View

Figure 10.2: View in the app after the first iteration.

Test person	Gender	Age	Climbing experience
1*	Male	23	3 years
2	Female	24	4 years
3	Female	19	7 years
4	Male	47	3 years
5	Male	14	1 month

Table 10.1: Overview of the people tested during the usability test. *Client Mattias Hornum

We tested our system on four people and our client, Mattias Hornum, with the demographics shown in table 10.1. For every person we wrote down what difficulties they had with each task, so that we later could identify the different usability problems in our system. The notes for every person can be seen in appendix D. For our test we tried to pick different types of people from the climbing club. We chose people of different genders, in the age range of 14-47 and with climbing experiencing ranging from 1 month to 7 years.

In total we identified seven usability problems, most of which were cosmetic but others were critical. The complete list can be seen in table 10.2. The problem encountered by the most people was P7. P7 was the problem "Difficult or cannot find a route by author", which was encountered during task **b)**. The people could not figure out how organise the routes in a way such that the system facilitates the search of routes by a particular member (author). The problem was classified as a catastrophic problem, since some of the users actually gave up on the task. P3 also addresses this issue since the users experiencing this problem actually clicked the "Add routes" and entered the author field on the Add New Route page. The people tested thought this was a way they could search for an author, so they typed in Hans into the Author field. So while they thought that they were searching for a route by a given author, they were actually adding a new route by that author. This problem was classified as serious, since the users perception of the system state was wrong, but they figured it out fairly quickly, once they saw the Add button, then went back to the previous page without adding the route to the system.

Usability problem	**Category**	**Experienced by**				
		1	**2**	**3**	**4**	**5**
1. Do not realise that it is possible to sort routes.	Cosmetic	x	x			
2. The user does not know that they have forgotten to specify all information during route creation.	Cosmetic	x				
3. Thinks that searching for a specific route can be done by accessing whats actually the new route page by pushing the '+' button.	Serious		x		x	x
4. Do not realise that it is possible to filter routes.	Cosmetic			x		x
5. Cannot find a route that they just created	Critical					x
6. Thought that changes in the edit route page was saved automatically	Serious				x	
7. Difficult or cannot find a route by author	Catastrophic	x	x		x	x

Table 10.2: List of usability problems.

Table 10.3 shows the number of usability problems in each category, discovered in the usability test.

	Cosmetic	Serious	Critical	Catastrophe	**Sum**
Number of usability problems	3	2	1	1	7

Table 10.3: Number of usability problems by category

To solve P1 and P4, we moved the panel with sort and filter (and later search) options to the top of the screen. So even when users scrolled down the list, the options are always located in the top of the screen.

For P2, we introduced alerts to the user when attempting to save a change that did not provide the sufficient information.

P3 and P7 is addressed by adding search functionality in the second iteration, which also introduces a more intuitive icon for search than the plus icon which may have been mistaken for search by some, the magnifying glass.

For P5 it was more natural to, instead of leaving the user back at the route list page, send them to the route info page for the newly created route.

10.3 Second Usability Test

After the second iteration, we conducted a second user evaluation. The users of the system was now able to also upload video beta, add comments, upload an image to the route and highlight holds that are part of the route, which this image is attached to. The users are also able to rate each route, and the system now has restrictions on what a climber is allowed to, and what setters are allowed to do, which we specified in section 5.1.2.

To test these new features, we created a list of tasks which each person during our testing had to go through.

Tasks

a) In the system you see a list of routes - find the oldest route with a blue grade.

b) Find a route that is created by Hans.

c) Use the search bar to find a route of white grade and a route number of 6, in section B.

d) Find a route on the wall and add it to the system. You should include an image of the route, and mark the holds that are part of the route.

e) You want to add tape to the route you just added. Find the route you just added, and add a tape colour.

f) Find the route with a black grade and route number 10 located in section A, and add a video beta to that route.

g) Imagine you just climbed one of the routes located in the system. Give the route a rating and add a comment to the route, based on on your rating of the route e.g "Great route".

Test person	Gender	Age	Climbing experience
1	Male	22	2 years
2*	Male	?	? years
3	Male	?	? years
4	Male	27	7 years
5	Male	29	6 years

Table 10.4: Overview of the people tested during the usability test. * indicates the member who did not partake in all tasks.

We tested the system on four male climbers, all in their twenties, but with experience in climbing ranging between 2-7 years. The last test person (marked by an asterisk in table 10.4) was someone who had almost finished building a new route. We decided to have him add the new route to the system, to get as close to the real context as possible. He did not solve all tasks in our usability tests but he did help us find some usability problems, so we chosen to include those findings.

For each person, we noted what they did during the testing and noted what trouble they had. We also recorded each usability test, so we could re-watch the footage, to identify as many usability problems as possible.

We identified 12 usability problems, most of them cosmetic. The complete list of usability problems can be seen in table 10.5, which also lists each person and what problems they encountered. The complete details of this evaluation, can be seen in appendix E.

The more serious problems we discovered, was an issue with the search functionality we implemented, which took far too long for anyone to find it useful, especially because there was no feedback if the application was actually searching for what each person put into the search-bar, or not. It was so serious, that we decided to tell the user, that the string they searched for was correct, but our algorithm was too slow to wait for the result.

Usability problem	**Category**	**Experienced by**				
		1	**2***	**3**	**4**	**5**
1. Does not know the difference between the colour of holds and grades.	Cosmetic	x				
2. Does not realise he can search/filter/sort	Cosmetic	x		x		
3. Tries to add image instead of comment, to add a video beta	Cosmetic	x				
4. Does not know how to navigate to the add new route page	Serious		x			
5. Toggling tape is not explicit enough	Cosmetic		x			
6. Zooming on pictures with the zoom gesture	Cosmetic		x			
7. Searches with terms not in English	Cosmetic			x		
8. Thinks the circles on the picture highlights the starting hold, not all the holds in the route	Cosmetic			x		
9. When pressing the enter button after a search, the keyboard does not get hidden	Serious	x		x	x	x
10. Afraid to add details to a route during the creation of it, because the route is not fully built yet	Cosmetic				x	
11. Thinks a rating has to be submitted	Cosmetic					x
12. Search functionality is far too slow	Serious	x		x	x	x

Table 10.5: List of usability problems.

Table 10.6 shows the number of usability problems in each category, discovered in the usability test.

	Cosmetic	Serious	Critical	Catastrophe	**Sum**
Number of usability problems	9	3	0	0	12

Table 10.6: Number of usability problems by category. *Only performed one task as he was building a route.

To access the add route page in this iteration, you would have to log in and then select the *Add route* item from the main menu or alternatively click the plus icon in the lower right corner and then enter your log-in information. The menu does not contain the *Add route* menu item if the user is not logged in but this was added after the second iteration to try and fix problem P4.

P9 is simply resolved by making the smartphone keyboard hide away when hitting enter when performing a search.

P12 is addressed by improving the search algorithm and also displaying a loading animation at the top of the screen whilst the system is performing the search.

The rest of the issues are not addressed but possible solutions would have been attempted if more time had been devoted to the issues. Instead we continued on to the third iteration.

10.4 Third Usability Test

During the third usability test, we prepared a list of scenarios and tasks. Each scenario had a number of tasks, and can be seen in appendix F. These scenarios and accompanying tasks was created using the following guidelines for a good task presented by Kjeldskov (2016):

- Represent real use of the system
- Describe the end result
- Motivate (Why should they be solved?)
- Include relevant data
- Group small subtasks together

An excerpt of task 5 of the usability test can be seen below. In the task we describe real scenarios, which then translates to a task in the system. For example, we learnt in our PACT analysis that the club is in the process of moving the climbing club to another location. We include the needed account information in the task, so that the test subject has all of the relevant information to perform the task. Lastly this excerpt shows how we have grouped tasks together, as task a) and b) are both solved in the administrator panel.

Task 5

The climbing club has moved to a different location, and now has enough room for another section. You have been given the task to add the section to the system.
To solve this task, you need an administrator account. Log in with the following information:
Username: admin
Password: 1234

a) Add the new section to the system

A member with the username **TannerH** has joined the board of directors, and wants to help administrating the system. To do that, his account must have its privileges elevated to administrator rights.

b) Give the user TannerH administrator rights

Figure 10.3: Excerpt of a task from usability test 3

We tested our system at the end of the third and final iteration to test the latest added features as well as some of the adjustments of existing features to see if previous usability problems has been solved, namely the issues we discovered with the search functionality we discovered during our previous usability test. All the tasks for the third usability evaluation are listed below.

Tasks

1. Find the oldest route with a green grade in section A.
2. Find out who made that route.
3. Use the search function to find the blue route in section A, created by Hans.
4. Find a route on the wall and add it to the system. Add only the necessary information.
5. Add a note to the route stating that you start the route from a sitting position.
6. Add a image of the route, and mark the starting hold.
7. Add yellow tape to the route you just created.
8. Add beta to the route you created along with a comment.
9. Find the top rated route in the system.
10. Rate the route yourself.
11. Log in as an administrator with the username ”admin”, and password ”1234”, and create a new section.
12. Give the user TannerH administrator rights.
13. Add a new hold colour to the system.

Test person	Gender	Age	Climbing experience	Smartphone owner	How easy 1-6
1	Female	46	1 year	Yes (Many years)	5
2	Male	27	3.5 years	Yes (7 years)	6
3	Male	30	1 year	Yes (5 years)	6
4	Male	24	1.5 years	Yes (6 years)	5
5	Male	34	3.5 years	Yes (8 years)	5

Table 10.7: Overview of the people tested during the usability test.

The final test was conducted on five members of the club, one of them an experienced setter who had just created four routes before going through our usability testing. An overview of the people we tested, can be seen in table 10.7. We tried getting people of different ages, gender and cultures. We ended up testing four danish males, one foreign male, and a danish female. The test people were within the age groups late 20’s and mid 40’s, with climbing from a year and up. Everyone happened to own a smartphone, and they were all quite experienced in handling such a device. Every test was recorded so we could conduct VDA in order to identify the usability problems each person ran into.
During our analysis, we identified a total of 12 usability problems, all which can be seen in table 10.8, most of them cosmetic. One of the more serious problems we encountered, was usability problem 6. The problem, however, did get solved in other ways by the people who encountered this problem, for example by adding a note, an action they already do in their current system, and when others are checking the route information page, everyone can read the note, stating it has tape and what colour the tape has.

Another problem we encountered, was more a bug if anything, which we listed as usability problem 11. The person added a section with a very long name which caused the name of the section to go outside the section-name box. But he also edited the section right away when he discovered that bug.
The previous issues we had with the search-bar, where it took far too long for any routes to appear after a search, has been resolved based on this usability test. The only issue that remains here, was the fact people thought they had to write in Danish words to do a search for what they want, but this is a cosmetic issue since they realised this right away and replaced the Danish words with English. We also will not fix this issue, since this requires adding support for Danish, and we specified that functionality as a requirement we will not have, as can be seen in our MoSCoW analysis in section 3.3.

Usability problem	**Category**	**Experienced by**				
		1	**2**	**3**	**4**	**5**
1. The user tries the zoom gesture instead of pressing the image once	Cosmetic	x				
2. The user tries editing information after having pressed the save button	Cosmetic	x		x		
3. The user has problems finding where to add beta	Serious	x				
4. The user is unsure, if the ratings update immediately	Cosmetic	x				x
5. The user forgot to change the name from default, when creating a hold	Cosmetic	x		x	x	
6. The user is unsure how to add tape	Serious		x		x	
7. The user tries swiping away the sidebar panel	Cosmetic			x		
8. The user goes to a route page/search bar, and when they goes back, filters are reset	Cosmetic					x
9. User adds comment instead of note	Cosmetic					x
10. User is unsure if admin rights changes after he clicks admin-button	Cosmetic	x				x
11. User adds a section with a name, that is too long for the box it is in	Cosmetic					x
12. Has difficulty believing the search functionality only accepts English terms	Cosmetic	x			x	

Table 10.8: List of usability problems.

Table 10.9 shows the number of usability problems in each category, discovered in the usability test.

	Cosmetic	Serious	Critical	Catastrophe	**Sum**
Number of usability problems	10	2	0	0	12

Table 10.9: Number of usability problems by category

At the end of the test, each participant had to fill out a questionnaire of the form seen in appendix F. Two out of the five people tested, gave our system a six-star rating, showing that they strongly agree that the system was indeed easy to use. The remaining three did not wholly disagree with the other two people, and gave it a five-star rating. Overall this shows that the

people tested, finds it an easy experience to use the system we have created, which was one of the key requirements listed in the MoSCoW analysis.
The two people who gave the system a 6/6 rating, were participant two and three. As can be seen in table 10.8, both participants experienced less serious issues with the system, than the rest. Participant two only experience one issue, which was P6, while participant three only experienced cosmetic issues, which they quickly worked around themselves. Two people who experienced more issues than the rest, were participant one and five, both whom gave the system a five-star rating. Participant one had trouble understanding that the search functionality only accepted English terms, and also, had trouble finding out how and where they could add beta. Participant five was annoyed with how the filters they applied got reset each time a search was conducted, and also, discovered a bug in the system (P11).
Overall, the ratings given by the participants, does comply with the amount of usability issues they ran into. The people who discovered the least and only minor issues, gave it a higher rating than those who discovered more issues, that were also more severe.

Summary

This chapter described four properties that a usable system should have: efficiency, safety of use, utility, and ease of use. Then, the two methods for conducting usability tests, heuristic inspections and user-based evaluation, were described. Two methods for analysing the results of a user-based evaluations were also covered. These two, instant data analysis and video data analysis, were described and it was explained why only video data analysis was used to analyse the usability tests of the system.

The usability tests done on the system introduced some sources of errors, which was cause for concern, as discussed in the next chapter along with the discussion of the requirements.

Chapter 11

Discussion

In this chapter, general concerns about the project are discussed. It is described how the users on whom the usability evaluations were conducted, were representative of AKK's members. Furthermore, the way in which the system solved the requirements specified in section 3.3 is described.

11.1 General concerns

In this section, general concerns about the cooperation with users is described. In order to get a better idea of what problems should have been a priority and also to simply find more, possibly undiscovered, usability problems we would have wanted more than five people for each usability test. Nine to twelve people would be a good target to try and reach in future projects as roughly 90 percent of problems should be discoverable with nine or more participating users (Kjeldskov et al., 2004). To get this done, it would be ideal to find a set of people beforehand instead of relying on them to be available when the test was being executed. This would, of course, require more preparation.

The number of iterations and their focus was chosen to fit the MoSCoW prioritisation, meaning that in every iteration we added more features to the system according to their MoSCoW priority. With more new features, we should expect to see new problems in the usability tests. Old problems should disappear or have a reduced impact if addressed well, and this is what we experienced. In the first iteration, a usability test was performed, as seen in section 10.2. Here we found that five of seven problems had to do with finding routes in the system. In the third and final iteration, we only observed one or two cosmetic issues with the route list page overall, and found that most previously discovered usability problems, had been resolved.

In the first usability test, we observed two critical or catastrophic problems. In the second iteration this was reduced to zero and only cosmetic and serious problems remained. In the third and final iteration, we saw that there were still usability problems but the main part of these problem were cosmetic as well and two serious usability problems.

As we go through more iterations, we should expect problems to be fewer and less serious in nature, but as we expanded the system, new problems appeared and as such we could not expect the number of total problems to always be decreasing. It should also be taken into account that the number of tasks the users have to go through, increases with the expansion of the system.

	Membership statistics			Usability test people statistics		
Membership type	Male	Female	Percentage	Male	Female	Percentage
Age 0-12	11	22	≈ 12	0	0	0
Age 13-18	10	13	≈ 9	1	0	≈ 7
Age 19-24	63	26	≈ 33	3	2	≈ 33
Age 25-59	85	39	≈ 46	6	1	≈ 47
Age 60+	0	0	0	0	0	0
unknown	0	0	0	2	0	≈ 13
Total	169	100	100	12	3	100

Table 11.1: Membership Statistics for AKK compared to test persons. Source: AKK

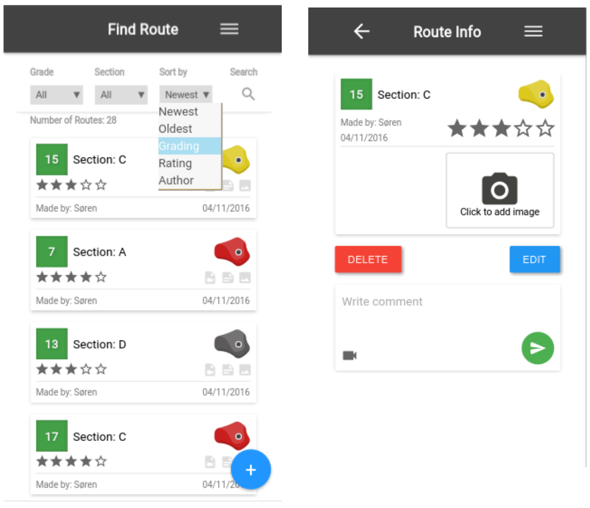
The table 11.1 shows the age distribution of the members in the climbing club alongside the distribution of the people who participated in our usability tests. The most underrepresented group are the females, the percentage of females in AKK are approximately 37 percent and the percentage of females who participated in our test were only 20 percent.

11.2 Fulfilment of Requirements

In section 11.2.1, we look at the must-have requirements concerning the administration of routes, finding a route in the system, handling multiple users simultaneously, and the ease of use. The should-have requirements is discussed in section 11.2.2 and concerns adding an image to a route, modifying sections and the mobility of the system. In section 11.2.3, the could-have requirements concerning rating a route, adding comments to a route and adding a beta to a route, is described. The requirements that we prioritised as will-not-have, are not discussed in this chapter, instead they are discussed in chapter 13 concerning future development of the system.

11.2.1 Must-have Requirements

The first and most comprehensive must-have requirement was that the system should have the same functionality as the current system. This included the ability to add, delete, edit, and view routes, and also the ability to find routes based on their grade, as it is possible on the whiteboards where the routes are grouped by grade. To see that this requirement is fulfilled, we now show how these operations are available and how they work in the system.



(a) Route list screen, showing the ability to filter and sort all routes.

(b) Route info screen, showing the ability to edit and delete a route.

Figure 11.1: Shows how we have implemented the whiteboard system's basic functions.

The ability to scroll through and sort the list of all routes, can be seen in figure 11.1a. In figure 11.1b, we can see that the task of deleting or editing a route is possible by clicking the edit or delete button. Adding a route is possible by clicking the plus icon in the lower right corner of the screen, seen in figure 11.1a. More details about adding, deleting or editing a route in the system are described earlier in section 7.2.

It is possible to filter routes by their grade, which corresponds to the action of looking at a single whiteboard in the climbing club. The second must-have requirement was that the system must be able to show routes based on section and the date it was created on. This requirement was solved by adding the option to filter routes by section as well as two sort options to list routes with the newest or oldest date of creation. In the system, authenticated users are able to edit the section, grade, number, colour of hold, image and note of a route, however, functionality for editing the creating date was never implemented. This means if someone edits a route significantly, it can be argued that the changed route effectively is a whole new route, and therefore, should have an updated date of creation.

The functionality behind adding, editing, deleting and finding routes has been tested internally by unit testing as described in chapter 9. Besides internal testing, the functionality was also tested through the tasks conducted in the usability tests. From the usability tests, we have shown that the test persons were able to add and edit routes. As well as finding routes by section, grade and date of creation. Through the usability evaluations, we have not tested whether the users can delete a route. This has been tested internally by the group members and through unit tests.

Through our choice of developing a web-based application using ASP.NET Core, we fulfil the next must-have requirement, which states that the system must be able to handle multiple users simultaneously. When users perform actions in client-side, it sends requests to the server-side API. ASP.NET Core assigns a thread from its thread pool to handle the request. The thread then executes the code in the controller, and returns the response to the client (Cleary, 2014).

The last must-have requirement is that the system must be easy to use. This is not a technical requirement and cannot be tested without involving the future users of the system. After each iteration of the system, we conducted usability tests to find problems that made the system difficult or frustrating to use. The use of multiple usability tests allowed us to fix problems from the earlier test, and also make sure that the changes we made did not introduce new problems.

In section 10.1, we mentioned four characteristics of a usable system: efficiency, safety of use, utility, and ease of use. The safety of use was also tested in our usability tests, as they were all done as field studies in the climbing club. We found no problems in regards to using the system while being in the climbing club.

In our final usability test, we discovered ten cosmetic, two serious and no critical usability problems. The first serious usability problem, as shown in 10.8, was that one participant had problems finding where to add beta. The second serious usability problem was that two of the participants had problems adding tape to a route in the system.

In the final usability test the five participants were between 24 and 46 years of age which means they fall into the two largest age groups. These people rated the system 5.4/6 on average on the scale of being easy to use (6 being the easiest). This group was the most important as we expect younger people to be more comfortable with technology in general and thus likely experience less problems during use (Lauterbach, 2015).

The score of 5.4 out of 6 and the low number of serious and critical usability problems give an indication that the system is easy to use which is one of our must-have requirements. In order to increase the certainty of the system's usability, we could test the system on a larger amount of people and continue to reduce the number of usability problems.

11.2.2 Should-have Requirements

The goal of the should-have requirements was to solve some of the challenges of the whiteboard system. The first requirement was that the system should allow members to add images of the routes to the system. This requirement was met by allowing the users of the system to take a photo with their phone, and add it to a route. We also added functionality for marking the holds on the image, to make it easier for other members to see exactly where the route was in a picture containing several routes. In our second usability test, we found that the users had difficulties realising that they were able to add these holds. In the third usability test, we found that after adding a guide as seen in figure 11.2, the users no longer had any issues using the feature to mark holds.

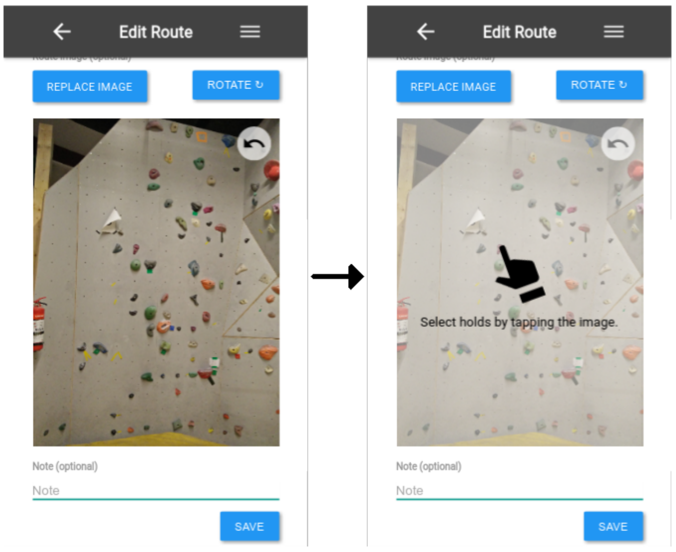


Figure 11.2: Edit route screens before and after the added guide on how to mark holds.

The next should-have requirement was that the system should offer the ability to modify sections. We implemented this by adding administrators to the system, allowing them access to an administrator panel where they could modify the structure of the climbing club. In this panel, they are able to modify the sections, grades, holds, and members. Although the requirement was only to manage sections, managing other administrators was also a necessity to allow more than one person to manage the sections. After adding the user management and section management, the structure was already in place to add grade and hold management as well, which when added, makes the system more versatile.

The last should-have requirement was that the system should be mobile such that it could be used whenever the user has Internet access. Similarly to the must-have requirement for handling multiple users simultaneously, this requirement was largely fulfilled by our choice of technical platform. As we chose to make a web-based application, the application is essentially just a website that the user visits, and as such does not require anything other than an Internet connection.

11.2.3 Could-have Requirements

The goal of the third iteration and the could-have requirements was to add a social aspect to the system. The first requirement was to allow members to add beta to routes. Another requirement was to allow members to comment on routes. Both of these requirements were fulfilled simultaneous, by implementing a comment feed, which allowed comment to contain a video beta. In the third usability test, we learnt that while it was not always clear to the users that the video attachment was considered beta, by the system, adding a video when being asked to do so, was little or no problem for the users.

The last could-have requirement was the ability for members to rate routes and view the ratings. We implemented a star system where each member can rate a route from one to five stars. In the route list a user can then sort by the average rating of routes or simply see them on each route's information card. From the third usability test, we learnt that the feature was very simple to use, although the user needed feedback when a rating was saved. This has been added since the usability test.

11.3 Choice of Software Development Method

As described in section 2.1.4, we worked iteratively in this project. Two of the reasons for this choice were the possibility for more interaction with the future users of the system during the development process, and the limited time frame of the project.

The choice of develop method definitely allowed us to work closely with the climbing club, as shown by the number of tests and interviews we performed in the club. Working closely with the club helped us make sure that the things we were working on corresponded to problems in the club, so that we did not do unnecessary work.

We wanted to use an iterative method such that we would have a working, albeit simple system, early in the project. This was not as easy as we thought it would be. In the beginning of a project, there are a lot of unknown factors that need to be cleared before a system can be developed. Because of our lack of experience with the iterative method, we had problems determining the parts of the problem domain that needed to be analysed in first iteration. This caused us to analyse a large part of the problem domain in the first iteration, causing our first iteration to be much larger than the other two. Even if we had previous experience with the method, the first iteration would probably still be the largest iteration, but we would have less trouble determining what parts to analyse.

Because of the amount of analysis we performed in the first iteration, we found that in the later iterations, a large part of the analysis needed had already been covered in the first iteration. This caused the focus of the later iterations to move from analysing to implementing and testing.

Summary

In this chapter some general concerns about sources of errors in the project were discussed. How the system solved the requirements it should, was shown and explained in detail. Based on this knowledge, it was possible to make a final conclusion on the system and project as a whole, as done in the next chapter.

Chapter 12

Conclusion

In this project, a system for administrating routes at Aalborg climbing club was developed. The project tried to answer the problem statement: *how can we develop a mobile system that allows multiple members to easily administrate climbing routes including associated grades, sections, images, betas, ratings and comments?*

To solve this problem, the classes found in the problem domain were modelled in the system. The classes found were: `Section`, `Route`, `Grade`, `Member`, `Comment`, `Beta`, `Rating`, and `Hold`. In the application domain, three types of actors were identified: the *climber*, the *setter*, and the *administrator*. The goal of the *climber* is to get a simple overview of routes in the club and give beta to other members in the club by comments, videos, or ratings of routes. The goal of the *setter* is to set and manage routes in the club, and the goal of the *administrator* is to handle adding new sections to the system and administrating grades. In order to create a digitised version of the old system, functionality for finding, adding, editing, and deleting `Route`s was added. Each `Route` in the system can be uniquely identified by their `Grade` and `Route` number. To find routes in the system, `Members` can filter routes based on their `Grade` and `Section`, as well as use one of the five different sort options: newest, oldest, grading, rate, and author. As a more advanced search option, a search bar was implemented using the Wagner-Fischer algorithm for calculating the edit-distance from search terms to route attributes. When authenticated in the system, a `Member` can add, edit, and delete `Routes`. The authentication ensures that only registered users of the system can remove, add, and edit information in the system, while still allowing unauthenticated `Member`s to find routes. Furthermore, an administration panel page was created in the user-interface to allow administrators of the system to add, edit, and delete `Grade`s and `Sections`. To support the creation of new administrators, `Member`s that already have administrator privileges can appoint other `Member`s as administrators through the administration panel. To support `Holds` in the system an `Image` class was created, which allows `Member`s of the system to add an `Image` of a route, and thereafter marking the `Hold`s by tapping on the image. Furthermore, `Comment`s and video `Beta`s were supported in the system by allowing a `Member` to attach a video `Beta` to a `Comment` on a `Route`.

To develop the `Route` administration system for Aalborg climbing club, a client-server architecture with five components was used: model, controller, client, view model and view. A big part of the system was developed using C# and .Net Core along with the Entity Framework Code First, which allowed us to store data in the system by mapping classes in the system to tables in a SQLite database. The user-interface was implemented using HTML, CSS and JavaScript along with the frameworks Handlebars and JQuery. The choice of architecture, languages and frameworks, allowed us to create a system able to handle multiple users simultaneously. The choice of making a web based application for smartphones, allowed us to create a system that was mobile and could be used anywhere in AKK where an Internet connection was available. To test the functionality of the system, different unit tests were conducted for the ViewModel, Client, and Controller components using the testing frameworks QUnit and NUnit.

To evaluate the system with the future users, three usability tests were conducted, one for each software development iteration. A total of fifteen different members, between the ages of 14 to 47, attended the usability tests. With the use of notes and video taken during the tests, different usability problems were found during each test. The usability problems were categorised as being either catastrophic, critical, serious, or cosmetic. After the third iteration, we found ten cosmetic and two serious usability problems. The first serious usability problem was experienced by two test persons that were unsure how to add tape to a route in the system. The second serious usability was that one test person had problems finding where to add beta to a route in the system. The five users that took part of the third usability test were given a questionnaire after performing the tasks, where they were asked to rate the ease of using the system on a scale from one to six (six being the easiest). Three out of five users answered five while the remaining two users gave it a score of six. This indicates that the system fulfils the requirement that the system must be easy to use. However, to further conclude on this requirement, the system must be tested on more members of AKK, as well as a more diverse group of people, over a longer time. This ensures a more representative result, as well as allowing the users to become familiar with the system which may create previously unseen usability problems.

The few usability problems found in the last usability test should be fixed in later development, although as most of the problems were cosmetic and no critical usability problem were discovered, then they did not hinder the users much.

As an answer to the problem statement, we can, based on the discussion in chapter 11, see that the system fulfils all must-have, could-have, and should-have requirements stated in section 3.3, and thereby the system is an answer to the problem statement.

Chapter 13

Future Development

While the finished product fulfils all of the requirements it should, there are still several ways in which it could be improved given more resources, especially time. In this chapter, we will discuss two distinct ways of improving the system: how it could be used in other climbing clubs and how it could be improved for use in AKK alone.

13.1 Using the System Outside of AKK

The result of this project is a web application for administrating routes in AKK. As the system has been developed with the single climbing club in mind, some parts of it may not work in another climbing club. A good example of this, is the grading of routes. AKK, and by extension our system, uses five different colours for representing the different grades. These five grades do not follow any official system, but AKK has been moving towards a more standardised grading system: the French Numerical Grades became available in the rope-climbing hall towards the end of the project (Larsen, 2016). This could potentially be a problem, if our system was to be used in another climbing club, but it is worth noting that the system supports custom grades as well, which certainly opens up for a lot of possibilities. The user interface layer of our system forces the users to make some choices, when it comes to creating new grades: a grade consists of a name and a colour along with information of how it compares to the other grades. If a different climbing club needs to represent their grades as a number, they would find that the current user interface does not allow grades to be identified by anything but their colour. The same kind of problem can be found with sections, whose names should not be longer than four characters, since the text would otherwise overflow the boxes it is supposed to stay inside of.

Since the system is created in a strict-closed manner, meaning that any architectural layer can only use operations from the layer directly below it, it is entirely possible to change the top-most layer without breaking any of the lower layers. Both of the previously mentioned problems exist within the top-most layer, and could therefore potentially be fixed by changing parts of the user interface or even creating a new one. This is especially true, since the lower layers are less specialised, and are able to represent grades and sections in ways that are less limiting.

Even then, it is entirely possible that there are some parts of other climbing clubs' problem domains that cannot be modelled sufficiently well in our system as it is now. The lower architectural layers of the system have been created in a manner so that they have low coupling, which means that adding to the model or functional layers is not a cumbersome task. This means, that if we were to visit other climbing clubs, analyse their problem and application domains, and identify the differences from AKK, it would, conceivably, be relatively easy to implement the changes in a way that could make the system usable in other clubs.

13.2 Improving the System for AKK

The modular way, in which the system has been developed means that it is relatively easy to implement new features and improve the system. This means that, given enough resources, it would be entirely possible to take more suggestions from climbers in AKK and provide them with an even more customised route administration system. Our usability tests have provided us with several ideas for future development of the system. Because of our limited time with the project, most of these suggestions ended up in the will-not-have part of the MoSCoW requirements. We did receive several suggestions, but some of them were deemed to be too specifically tied to the people who made them. The three ideas that did make it to the requirements are as follows:

- Localisation
- Social media integration
- QR-codes or NFC
- Indication whether a safety certificate is needed to climb a route

Localisation was one of the very first requirements we received from AKK, since Hornum talked about it in our first interview. We had been expecting to make a system in Danish, because of the location of the Climbing Club, but as it turned out, several members of the club are foreigners and are not very skilled at the Danish language. Because of this, we made it a requirement that the system should be available in both Danish and English, but when Hornum reviewed the requirements he felt that having the system available in Danish was of little importance. Instead the system could be made solely available in English, and the majority of its users would have no problems using it. Given the fact that AKK would like their system to be used by all their members including children, it would seem reasonable to implement localisation, given more time.

Social media integration was a suggestion we received from more than one member in the club. One thought it would be nice if every image taken of routes would automatically be shared on AKK's Instagram profile, while another would just like the ability to share routes and video-beta on various social media websites. Given that the club has both a Facebook and an Instagram account, the idea of integrating the system with social media could seem like a good idea if development was to continue

Using additional technology to allow users to more easily identify a route in the system was suggested by a climber. His idea was that a little QR-code sticker could be attached to the wall, and that the system should be able to identify the routes from the stickers. In essence, this idea is actually a continuation of one of the core ideas of the system, which is to make it easy for people to find routes. We did not develop it in the first place, because we felt it was slightly superfluous, as the system already allowed for people to identify the routes by filtering and searching, which our usability tests proved they could. Besides, our system did not require AKK to change anything about their routes, while this idea would. Given more time for development it would be entirely possible to implement this, and with more resources, NFC seems a viable candidate for electronically identifying routes.

Given more time it would be possible to implement support for warning climbers if a specific route requires a safety certificate to climb. As mentioned in section 3.2 it is only sports climbing routes in AKK that requires a certificate to climb, so by adding this functionality we would also be able to distinguish between sports-climbing and bouldering routes in the system.

Bibliography

Aalborg klatreklub, 2016a. URL `http://njklatreklub.dk`. Accessed: 2016-09-16.

Aalborg klatreklub, 2016b. URL `http://njklatreklub.dk/vil_du_klatre/kom-i-gang/`. Accessed: 2016-12-14.

About Sports, 2016. URL `http://climbing.about.com/od/topropeclimbing/a/TopRoping1.htm`. Accessed: 2016-09-16.

Afd. for Vækst og Reproduktion, Rigshospitalet. The 2014 danish references from birth to 20 years for height, weight and body mass index., 2014. URL `http://www.xn--vkstkurver-d6a.dk/index.html`.

S. Allen. C# levenshtein distance, 2016. URL `https://www.dotnetperls.com/levenshtein`.

Apple. ios human interface guidelines, September 2016. URL `https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/`. Accessed: 2016-12-06.

D. Benyon. *Designing Interactive Systems : A comprehensive guide to HCI and interaction design*. Addison Wesley, Harlow, England New York, 2010. ISBN 978-0-321-43533-0.

D. Benyon. *Designing interactive systems : a comprehensive guide to HCI, UX and interaction design*. Pearson, Harlow, England, 2014. ISBN 9781447920113.

T. Berger-Wolf. Cs 502: Algorithms in computational biology, 2016. URL `http://compbio.cs.uic.edu/~tanya/teaching/CompBio/scribe/TimLuciani_scribing-0119.v3.pdf`.

E. M. Burke and B. M. Coyner. Top 12 reasons to write unit tests, 2 2003. URL `http://www.onjava.com/pub/a/onjava/2003/04/02/javaxpckbk.html`.

S. Cleary. Async programming : Introduction to async/await on asp.net, October 2014. URL `https://msdn.microsoft.com/en-us/magazine/dn802603.aspx`.

F. J. Damerau. A technique for computer detection and correction of spelling errors, 3 1964. URL `http://dl.acm.org/citation.cfm?doid=363958.363994`.

Danmarks statistik, 2015. URL `http://www.dst.dk/Site/Dst/Udgivelser/GetPubFile.aspx?id=20737&sid=itbef2015`. Accessed: 2016-10-10.

Dansk Klatreforbund, 2016a. URL `http://klatreforbund.dk/bouldering/`. Accessed: 2016-09-16.

Dansk Klatreforbund, 2016b. URL `http://klatreforbund.dk/sportsklatring/`. Accessed: 2016-09-16.

I. d'électronique et d'informatique Gaspard-Monge (IGM). Boyer-moore algorithm, 1 1997. URL `http://www-igm.univ-mlv.fr/~lecroq/string/node14.html`.

EMU. Engelsk - fælles mål, læseplan og vejledning, 2016. URL `http://www.emu.dk/modul/engelsk-f\OT1\aelles-m\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{a\global\mathchardef\accent@spacefactor\spacefactor}\accent23a\egroup\spacefactor\accent@spacefactorl-l\OT1\aeseplan-og-vejledning`.

EntityFrameworkTutorial. What is entity framework, 2016. URL `http://www.entityframeworktutorial.net/what-is-entityframework.aspx`.

M. Fester. Medlemstal 2015, 2016. URL `http://www.dif.dk/da/om_dif/medlemstal`.

R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. URL `http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf`. AAI9980887.

Google. Material design, August 2016. URL `https://material.google.com/`. Accessed: 2016-12-06.

M. Hornum. Interview with mattias hornum, 2016.

IFSC. Key figures, 2016. URL `https://www.ifsc-climbing.org/index.php/media-centre/key-figures-2`.

Y. Katz. Handlebars, 2016. URL `http://handlebarsjs.com/`.

J. Kjeldskov. Usability and usability evaluation, September 2016. DAT-3/SW-3/IDA-7. DEB. Electronic appendix "DEB usability guest lecture - pt 1.pdf".

J. Kjeldskov, M. B. Skov, and J. Stage. Instant data analysis: Conducting usability evaluations in a day. In *in Proceedings of the Third Nordic Conference on Human-Computer Interaction*, pages 233–240. ACM Press, 2004.

Y. Kunio. Jma official statement, 2015. URL `https://www.ifsc-climbing.org/images/media-centre/press-releases/JMA_statement_Tokyo_2020.pdf`.

M. F. Larsen. Ny & opdateret oversigts skema over ruterne i den store hal, December 2016. URL `https://www.facebook.com/groups/aalborgklatreklub/permalink/1236330866460009/`. Accessed: 2016-12-13.

T. Lauterbach. It-anvendelse i befolkningen 2015, December 2015. ISSN 2245-4152. URL `http://www.dst.dk/Site/Dst/Udgivelser/GetPubFile.aspx?id=20737&sid=itbef2015`.

H. Lichter, M. Schneider-Hufschmidt, and H. Zullighoven. *Prototyping in Industrial Software Projects-Bridging the Gap Between Theory and Practice*, pages 825–832. IEEE, 1994. doi: 10.1109/32.368126.

L. Mathiassen, A. Munk-Madsen, P. Axel Nielsen, and J. Stage. *Object-oriented analysis design*. Marko, Aalborg, Denmark, 2000. ISBN 9788777511509.

N. B. M.D. The psychology of laziness, October 2014. URL `https://www.psychologytoday.com/blog/hide-and-seek/201410/the-psychology-laziness`.

C. Melissa Mcclendon, L. Regot, and G. Akers. What is prototyping?, 2012. URL http://www.umsl.edu/~sauterv/analysis/prototyping/proto.html.

D. Methvin. The state of jquery 2014, 1 2014. URL https://blog.jquery.com/2014/01/13/the-state-of-jquery-2014/.

Microsoft. Unit testing. URL https://msdn.microsoft.com/en-us/library/aa292197.aspx.

Microsoft. The mvvm pattern, 2012. URL https://msdn.microsoft.com/en-us/library/hh848246.aspx.

Microsoft. Data layer guidelines, 2016a. URL https://msdn.microsoft.com/en-us/library/ee658127.aspx.

Microsoft. What is asp.net core?, 2016b. URL https://docs.asp.net/en/latest/intro.html#what-is-asp-net-core.

Microsoft. The repository pattern, 2016c. URL https://msdn.microsoft.com/en-us/library/ff649690.aspx.

Minimum-Bouldering, 2016. URL http://ba.iff.im/. Accessed: 2016-10-04.

National Eye Institute. Facts about color blindness, February 2015. URL https://nei.nih.gov/health/color_blindness/facts_about.

NUnit. Nunit 3 test runner for .net core, 2016. URL https://www.nunit.org.

D. of Computer Science Old Dominion University. Dynamic programming - example: Edit distance, 7 2013. URL https://secweb.cs.odu.edu/~zeil/cs361/web/website/Lectures/styles/pages/editdistance.html.

D. B. S. of Information and C. Sciences. Knuth-morris-pratt string matching, 2 1996. URL http://www.ics.uci.edu/~eppstein/161/960227.html.

J. Rubin. *Handbook of usability testing : how to plan, design, and conduct effective tests.* Wiley Pub, Indianapolis, IN, 2008. ISBN 9780470185483.

R. Software. Rational unified process best practices for software development teams, 1998. URL http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf.

SQLite. About sqlite, 2016. URL https://sqlite.org/about.html.

X. Sun, T. Plocher, and W. Qu. *An Empirical Study on the Smallest Comfortable Button/Icon Size on Touch Screen*, pages 615–621. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-73287-7. doi: 10.1007/978-3-540-73287-7_71. URL http://dx.doi.org/10.1007/978-3-540-73287-7_71.

L. L. Thomsen. Studieordning for bacheloruddannelsen i software, September 2015. URL http://www.sict.aau.dk/digitalAssets/203/203759_software-bachelor-e16.pdf.

TNS. Mobile devices 2015 - en undersøgelse om danskernes brug af mobile enheder, 2015. URL http://danskemedier.dk/wp-content/uploads/2015/11/Mobile-Devices-2015.pdf.

UVM. Skolestart, 2015. URL http://www.uvm.dk/Uddannelser/Folkeskolen/Fag-timetal-og-overgange/Skolestart-og-boernehaveklassen/Skolestart.

Appendices

Appendix A

First Iteration: Preparations for Semi-structured Interview

Semistruktureret interview til klatreklubben

Tema	Undertema		Forslag til opfølgende spørgsmål
Demografi	Aldersforskel Psykologisk forskel Kønsfordeling	• Hvad hedder du? • Hvor gammel er du? • Hvilken rolle har du i klubben?	• Hvilke andre roller er der i klubben?
		• Hvor mange medlemmer er der i klubben? • Hvad er det for nogle mennesker er medlemmer i klubben?	• Deres aldre • Deres erfaringer • Deres køn
Klatring	Typer af klatring Involverede roller Planlægning	• Hvilken type klatring udfører I?	• Hvilken type klatring foretrækker du? ○ Hvorfor? • Kan du vise os de forskellige typer af klatreruter? Både indendørs og udendørs.
		• Hvordan forløber en typisk dag/aften i klatreklubben?	• Er der socialt samvær i løbet af dagen? • Har man altid en spotter/hjælper med? • Hvad er spotterens rolle?
		• Er der et system i ruternes placering?	• Er der et kort?
		• Hvordan planlægger I brugen	• Hvordan ved man hvornår en rute er ledig?

		af ruter?	• Hvem skal være spotter?
System	Systemer til bouldering Systemer til længere ruter med reb Systemer til evt. andre typer klatring	• Hvordan administrerer I ruter?	• Kan du vise os hvordan det nuværende system benyttes? • Hvem har rettighed til at ændre ruter i systemet? • Hvilke problemer er der ved nuværende system? • Hvor ofte ændres klatreruterne? ○ Hvorfor? • Er det et system du er tilfreds med? • Er medlemmerne tilfredse? • Er det nemt og overskueligt at benytte? ○ Hvorfor? ○ Hvorfor ikke? • Hvad gør i hvis en del af en rute går i stykker?
		• Er der ét samlet system til at håndtere alle typer ruter?	• Hvis ikke, hvorfor er de delt op? • Hvad er ligheder og forskelle mellem de forskellige typer ruter?
		• Hvis der skal laves et nyt forbedret system, hvilke ønsker og krav har I så til dette?	• Booking af ruter, videoer, brugervurderinger, kortoversigt, billeder af ruter mm.
Ekstra		• Vil det være muligt at teste forskellige prototyper i klatreklubben?	• Hvem har mulighed for at teste vores prototyper og hvornår?

Appendix B

First Iteration: Interview Transcriptions (29th of September 2016)

Title: Interview #1 with Mattias
Duration: 00:37:00
File: interview1-del1.mp4 & interview1-del2.mp4

Henrik: Ja, jeg ved ikke. Vi har sådan en helt del ting vi gerne vil spørge dig om.

Mattias: Ja da.

Henrik: De er ikke sådan super struktureret, det er mest bare nogle temaer og emner vi gerne vil ind over.

Mattias: Det er helt fint.

Henrik: Og så Mathias for bordenden tager nogle noter for at skrive ned hvad hvad du fortæller os. Så vil Anton filme dig, og så vil jeg spørge dig om ting.

Mattias: Fedt.

Henrik: Og så håber jeg at vi også kan få en rundvisning eller et eller andet.

Mattias: Ja ja, helt sikkert. I har ikke været inde og kigge endnu?

Anton: Vi har lige været inde og se hallen.

Mattias: Ja.

Henrik: Vi ville heller ikke være i vejen, så vi turde ikke gå for langt ind. Men vi vil gerne se jeres nuværende systemer.

Mattias: Ja der er ikke meget system over det lige nu.

Henrik: Men det kan være at vi kan komme omkring det.

Henrik: Er du klar?

Mattias: Ja.

Henrik: Til at starte med, vil vi gerne vide sådan en lille smule om dig, og hvem du er som person, hvor gammel er du, hvad går du og laver.

Mattias: Ja. Jeg er 23 og jeg læser til maskiningenør ude på universitetet, og jeg har siddet i bestyrrelsen her i halvandet år nu, eller sådan noget, og jeg begyndte at klatre for det er vel tre år siden nu, fordi jeg besteg Mont Blanc og så gik det op for mig, at for at gøre det lidt mere sikkert, end det vi gjorde, så var det nok en god idé at lære at klatre.

Henrik: Okay.

Mattias: Ja det sådan meget kort.

Henrik: Ja okay. Fair nok. Hvad så med de andre der er her ude? Hvor mange mennesker er der egentlig i klubben?

Mattias: Jamen jeg tror at i sidste opgørelse vi havde, så var der 270 medlemmer.

Henrik: Hold da op.

Mattias: Så vi er vokset. Jeg tror vi er vokset med 70 eller sådan noget i den her sæson. Så det er meget godt. Og nu kan jeg ikke lige huske det helt præcist, men det kan jeg sende til jer bagefter hvis I gerne vil have det, men det er et eller andet med at 40% er unge sådan i alderen 20-28(-30) stykker eller sådan noget. Så er der 40% der er over den alder, og så er de sidste 20% børn, sådan et eller andet.

Et andet medlem af klubben (Hans-Christian) træder ind i lokalet.

Henrik: Kønsfordelingen, er det noget du sådan kan sige noget om eller er det for. . .

Mattias: Kan du sige noget om det, Hans-Christian?

Henrik: Altså ikke sådan i præcise tal.

Mattias: Der er overraskende mange piger.

Hans-Christian: Det er steget vil jeg sige, det har sådan typisk været at vi har haft piger under 14 år, drenge over 20. Men det har ændret sig lidt. Det er rigtigt at vi har fået lidt flere piger ind i klubben også, også de der studerende årgange men der er nok to tredjedele mænd, sådan overordnet set.

Mattias: Men det tror jeg også vi kan finde noget om til jer.

Hans-Christian: Ja det kan vi jo trække ud.

Henrik: Jamen det kunne da egentlig sådan være okay.

Mattias: I kan bare få hele vores medlemsstatistik.

Henrik: Ja det kunne vi da selvfølgelig godt, det er måske også svært at sige meget om det når der er 270 medlemmer. Det var mere hvis der var 20, så kunne det være at du kendte dem alle sammen personligt. Men det kan jeg godt se.

Mattias: Vi er en del.

Henrik: Væsentlig større end vi troede. Nå ja, du sagde at du var i bestyrelsen. Hvad inkluderer det sådan af forskellige roller?

Mattias: Jamen jeg tror vi sidder seks eller otte mand afhængigt af hvor mange der kommer. Vi har jo sådan nogle suppleanter og sådan noget. Jamen vi står for alt lige fra daglig drift. Lige for tiden er vi faktisk ved at arbejde på sådan et flytteprojekt sammen med sportshøjskolen. De skal til at bygge en kæmpestor hal ovre på deres grund. Og de har interesse i at gøre det sammen med os fordi vi har en masse vægge og greb og sådan noget. Og vi har en interesse i at gøre det sammen med dem fordi de har en masse penge. Ej, det er meget simplificeret, men ja. Så vi er i gang med at arbejde sammen med dem. Og vi er gået videre, så jeg tror der er nogle arkitekter i gang med at tegne lige nu. Nogle forslag til den hal her. Så jeg tror faktisk at de regner med at have det færdigt om et års tid, så det er sådan det store projekt sådan.

Henrik: Betyder det noget for din daglige dag her i klubben at du sidder i bestyrelsen. Er det nogle specielle roller der har med klatring at gøre, at du har?

Mattias: Jamen jeg står for eksempel for alle konkurrencerne, så jeg står både for at informere folk om de konkurrencer der er i Danmark, men også for at arrangere dem vi holder. Vi har lige holdt en stor konkurrence nede på havnen for eksempel, og vi holder en igen her til november og så er det også mig der har lavet det system vi skal ind og se bagefter, med whiteboardséne.

Henrik: Okay, hvad så med... Er der andre der har nogle... Nej, det var egentlig det vi gerne ville ind på, det system med whiteboard. Er der nogen der står for det specifikt, hvilke roller er der tilknyttet det?

Mattias: Altså da jeg kom i klubben for snart to år siden, der kom jeg lige fra sådan et kommercielt... Altså man skelner mellem kommercielle klatecentre og foreningsdrevne klatrecentre, og det her er jo et foreningsdrevet, så alt det er frivilligt.

Hvis I tager til Aarhus... I sådan noget som Aarhus Boulders, det er kommercielt, det vil sige, at der er nogen der tjener penge på det. Så der har de ansatte til at gøre rent og holde styr på tingene og lave ruter, og det er jo det vi gerne vil snakke om.

Her der er det sådan klubbens medlemmer der laver ruterne og vi har nogle folk der er rigtig dygtige til at skrue, vi har også nogle der ikke er så dygtige. Så det der whiteboardsystem... altså da jeg kom her var der ikke noget system, så du kunne ikke... altså når man klatrer... undskyld... det er sådan lidt svært at vide hvilket niveau man taler på, men når man klatrer...

Henrik: Vi ved ingenting.

Mattias: Nej lige præcis. Og tit så spørger folk: "Hva' går det ud på at klatre hurtigst op?", men det det egentlig går ud på, det er egentligt at klatre sådan sværest mulige ruter.

Henrik: Ja.

Mattias: Så man laver en rute i én farve greb og så kan man lave dem i forskellige sværhedsgrader, og så afhængig af hvor dygtig man er, så klatrer man ligesom den sværhedsgrad. Men hvis du så kommer ind og der ikke... Der ikke ligesom er nogen sværhedsgrad... Du kan ikke se hvor svær den er, du kan ikke... Der er ikke engang lavet ruter i samme type greb, så er det utrolig svært at... Så skal man ligesom lave sit eget hver gang.

Henrik: Ja, ja. Okay.

Mattias: Så det vi startede med at gøre, det var ligesom at sætte det i system. Vi lavede sådan nogle små træklodser, i... Jeg tror vi har fem forskellige farver, der ligesom indikerer sværhedsgraden. Og så startgrebet, det greb man ligesom skal starte i, det har sådan en farvemarking på sådan en klods.

Og da vi så ligesom fik implementeret det system, og folk begyndte at gøre det lige så stille... Det er sådan med foreninger, der skal man jo... Små skridt. Keep it simple.

Så fik vi lavet de her whiteboards. Så har vi markeret alle klodserne med et nummer fra 1 til x. Og så det man gør, hvis man har sat en ny rute, så sætter man en "rød klods nummer seks", så går man op til den røde tavle, så skriver man rute nummer seks. De er så delt ind i fire sektioner. Den her sektion, den her dato, hvem der har lavet den, og så en lille note, for eksempel hvis man skal hoppe langt eller et eller andet.

Og tanken med de whiteboards, det var sådan set ikke... Min første tanke, det var sådan set, at det kunne være fedt med en app, men jeg tror det har været et for stort et spring at gøre. Så det var lidt ligesom skridtet før den her app. For nu er folk ligesom blevet vant til, "okay der er et system i det, man skal ligesom registrere når man laver en rute", og endnu federe, så endelig hvis man for eksempel ikke har været i klubben i to uger, og man gerne vil se hvad der er blevet lavet nyt, så kan man gå hen og kigge på den sværhedsgrad man klatrer og så se okay: "datoer, her er tre nye ruter. Dem skal jeg prøve". For det kan godt være lidt svært at overskue når der er sådan 50 ruter.

Henrik: Det forstår jeg godt. Jeg ved ikke om vi måske skulle prøve at kigge på det nu?

Mattias: Det kan vi fint.

Henrik: . . . eller om det ville være helt åndsvagt at afbryde midt i det hele og sådan noget.

Mattias: Jeg tror det giver god mening, for at give jer en forståelse. Det tror jeg giver god mening. Nu tror jeg også at kaffen er færdig.

Vi går ind i den første klatrehal, hvor der klatres med reb.

Anton: Så før der var noget system, var det så bare at der bare var ruter i farver? Altså var det ligesom bare farverne nogenlunde indikerede.

Mattias: Ja, og så var det. . . der var ikke så meget udskiftning, så grebene blev meget kalkede, og der var rigtig mange greb oven i hinanden, så det var meget svært at finde ud af.

Okay, så vi har to typer klatring her. Den ene er rebklatring eller ruteklatring, det er det man laver herinde, hvor man klatrer op med reb, og så herinde, der har vi bouldering hvor man klatrer uden reb, så er det ikke lige så højt og så er der en madras.

Herinde har vi ikke fået sat systemet i gang endnu, fordi der ikke har været super meget motivation for det.

Henrik: Må jeg have lov at spørge hvorfor?

Mattias: Jamen, det er en del mere besværligt at lave nye ruter herinde fordi man skal ligesom til at hænge i et reb så det tager meget længere tid. Og så er der ikke lige så mange der laver den her form for klatring, som der er i den anden. Og det er en lang historie hvorfor, men det er nok noget med at det er nemmere og man kan gøre det selv.

Men i hvert fald min tanke dengang jeg lavede det gamle system, det var ligesom at starte med bouldering, og hvis det kom til at køre rigtig godt, så kunne man altid prøve at få det herind.

Henrik: Ja.

Vi går nu ind i boulder-hallen.

Mattias: Det her er bouldering. Man kan se der sidder sådan nogle farvede klodser nede i bunden, ved nogle greb. Sådan nogle træklodser, det er dem der indikerer sværhedsgrad.

Mathias: Okay, så de forskellige farver er forskellige sværhedsgrader eller hvordan?

Mattias: Ja de forskellige farver klodser, lige præcis.

Vi går hen til det nuværende tavlesystem.

Mattias: Og så kan man se herovre... Her kan man så se hvad for nogle ruter der er lavet hvornår af hvem.

Henrik: Og det er simpelthen på engelsk?

Mattias: Jamen vi har en del udenlandske medlemmer heroppe, udvekslingsstuderende.

Henrik: Ja.

Mattias: Så vi kører det på [**UFORSTÅELIGT**]

Henrik: Er det fordi, det er mere populært i de lande de kommer fra, end det er her?

Mattias: Det ved jeg... Det ved jeg sgu ikke. Altså mange spaniere kommer her. Jeg tror der er rigtig mange der klatrer i Spanien.

Henrik: Okay. Så sværhedsgraden er grøn og helt op til hvid?

Mattias: Yes, lige præcis. Vi prøver nogenlunde at normalfordele det.

Henrik: Okay, cool. Det ser ret fedt ud.

Mattias: Har I taget noget idrætstøj med?

Henrik: Nej desværre, men jeg kunne dælme godt tænke mig at prøve det alligevel.

Mattias: Det kan vi lige gøre... For der er jo faktisk mulighed for at komme ud at prøve det.

Henrik: Ja.

Mattias: Det kunne I jo...

Henrik: Jamen vi har også kigget på der er... på søndag, ikke? Den første søndag i måneden?

Mattias: Jo, der er også noget. Det er nok en lidt nedern dag at komme på, fordi der er sygt mange mennesker.

Henrik: Okay.

Mattias: Det er bedre bare at komme en dag i klubbens åbningstider og så betale en halvtredser for indgang. Så kan man bare få at lov til at gå amok.

Henrik: Okay. Det kan da godt være. Det virker meget interessant.

Vi går tilbage til køkkenet hvor interviewet forsætter.

Henrik: Jo. . . Så nu har vi jo så set de to forskellige typer der er, og set systemet også, men vi vil gerne høre lidt mere om de forskellige typer. Men nu kan vi jo selvfølgelig godt forstå, at hvis vi skal udvikle noget, er det primært bouldering det kommer til at handle om.

Mattias: Ja, ikke nødvendigvis. Altså, systemet som I laver kan jo i princippet fungere til begge dele, fuldstændig på lige vilkår. Men grunden til, at jeg måske siger, at I skal fokusere på bouldering, det er fordi at der er ligesom et system derinde. Der kommer klodser op, ruterne bliver skiftet ud systematisk. Og hvis vi så har lyst til at bruge appen inde i rutehallen, altså så kan vi jo bare gøre det. Men det ville være rigtig besværligt for jer. . . For der er intet system derinde, overhovedet, altså.

Henrik: Men altså så, fra systemets synspunkt så er der ikke rigtig nogen forskel på hvad det egentlig var man skulle kunne?

Mattias: Overhovedet ikke.

Henrik: Nå men så er det jo rimelig nemt. Helt fint.

Hvad foretrækker du selv, egentlig? Nu snakkede du sådan lidt om at. . .

Mattias: Ja, alle har nok en. . . Altså herhjemme, der boulder jeg mest, inde i hallen. Men det er fordi det simpelthen er nemmest at træne derinde, efter ens egen dagsorden. Men når jeg er udenfor, det er jo det man rigtig gerne vil når man klatrer, komme ud på rigtige klipper, så er det mest rute. Meget alpin-klatring, altså bjergbestigning, kalder man det også.

Henrik: Så det er en stor høj klippevæg, hvor det går lige så stille opad?

Mattias: Lige præcis.

Henrik: Cool. Er det sådan lidt at der måske er lidt aldersfordeling så? At det måske er de lidt ældre der. . . Der klatrer med reb herinde, eller hvad?

Mattias: Ja det er det faktisk. Bouldering det er den nyeste, ligesom diciplin, inden for klatring. Så ja.

Henrik: Det virker måske også sådan lidt mere, hvad skal man sige, friskt, eller sådan. . . Det er lidt mere bare til at gå til, sådan. . .

Mattias: Lige præcis. Ja, lige præcis. Det er sådan lidt mere FitnessWorld, man kan komme når man har lyst. . .

Henrik: Ja okay.

Mattias: . . . Og fyre en workout af.

Henrik: Jamen det kunne godt være at vi skulle overveje sådan noget, nemlig med at hvis det er masser af unge mennesker der gør det, at så skal vi have et lidt mere ungt design på en eller anden måde.

Mattias: Nå, ja.

Henrik: Så det er sådan set spændende nok. Fair nok.

Henrik: Hvem kan... Kan alle gå ind og skrive noget på tavlerne derinde, eller er det kun nogen i tillader?

Mattias: Ja. I princippet kan alle. Men man gør det jo kun, hvis man laver en rute. Men ja, alle skal have mulighed for det.

Henrik: Okay, men hvad så når man skal lave sådan en rute. Det kræver jo noget værktøj.

Anton: Kan du sige hvad det er, det indebærer at lave en rute?

Mattias: Jamen altså så skal man jo finde nogle greb i en eller anden farve, der ikke lige er på væggen. Og så skruer man dem bare op. Vi har værktøj liggende derinde til det. Bolte og det hele. Men det er ikke så meget det at skrue man skal være god til. Det er mere det at... Altså det tager jo overraskende meget erfaring at lave... Hvis du tænker, okay jeg vil gerne lave sådan en bevægelse her, så for at få grebet til at sidde rigtigt og sætte føderne så de passer og sådan noget. Det er overraskende svært.

Henrik: Ja, det kunne jeg godt forestille mig.

Mattias: Så det er derfor at det typisk er erfarne folk, der ligesom sætter ruterne op. Men vi prøver hele tiden at invitere nye folk til de der rutebygningsdage. For ligesom at lære folk op, fordi jo flere der gider at bygge, jo flere ruter kommer der.

Henrik: Er der ikke lidt en grænse for hvor mange ruter i kan have på samme tid? I løber vel tør for plads, gør i ikke? Eller er det ikke et problem.

Mattias: Jo. Jo, men vi skifter regelmæssigt ud. En gang hver anden måned siger vi, "nu ryder vi en fjerdedel af de boulder derinde". Så bliver de ikke fyldt op.

Anton: Så det er sådan nogle dage I har, hvor det er I skal... "I dag så laver vi en helt masse om", og eller sker der ikke så meget. Sådan det er ikke noget folk bare sådan gør, når de lige kommer ind en eftermiddag?

Mattias: Jo, det er det faktisk. Altså det der sker er ligesom, på én dag, det er at vi ryder det. Så er der sådan en håndfuld eller to håndfulde mennesker der går amok, og bare tager alle grebene ned i en sektion.

Anton: Og så starter forfra nærmest. Altså bare ryder tavlen ren.

Mattias: Lige præcis.

Henrik: Men når man sætter en enkelt ny rute op, kunne man så godt finde på at fjerne noget også? Så man fjerner ikke andres ruter eller?

Mattias: Det er sådan. . . Det er der lidt tabu over. Man skal helst ikke. . . Man må helst ikke justere eller røre andres ruter.

Henrik: Okay.

Mattias: Og det er faktisk en anden grund til, at vi lavede systemet herinde. Det er for at kunne se, okay, hvis der nu er kommet en ny rute op, som er helt af helvedes til. . .

Henrik: Ja, så kan man se hvem det er der har lavet den.

Mattias: Lige præcis! Så kan man nemlig se hvem der har lavet den. Ja, så kan man nemlig se hvem der har lavet den. Og så kan man tage hen og spørge folk: "Hey, er det okay, at jeg lige drejer det greb, så jeg ikke brækker min finger?"

Henrik: Så altså, der kan godt være to ruter nærmest oven i hinanden, sådan hvis de bare har forskellige farver så, det er ikke sådan. . . ?

Mattias: Ja, det er ikke noget problem. Man kan faktisk også godt lave dem oven i hinanden med samme farve, så det man gør det er, at man bare taper dem. Så taper man den ene med rødt tape eller et eller andet.

Henrik: Så det er ikke sådan, at I løber tør for farver?

Mattias: Nej, nej, nej.

Henrik: Okay, det havde jeg egentligt troet at der var rimelig begrænset hvor mange farver I kunne finde på.

Mattias: Ja, men så. . . Ja. . . Det. . . Nej, det er intet problem.

Henrik: Hvis du. . . Ja. . . Er det dig der har fundet på systemet. Fordi altså det lyder som at du i hvert fald har været en rigtig stor del af det, men er det noget du, sådan på egen hånd har fundet på?

Mattias: Altså med farverne derinde?

Henrik: Ja, og med lave et system ud af det.

Mattias: Jamen det. . . Nej, nej. . . Altså i alle sådan veletablerede klatrehaller der er der jo et system. Og altså, der er jo ikke super meget man kan variere på sådan et system. Tingene skal graderes og der skal være en eller anden måde at holde styr på det.

Henrik: Ja. Er der tilfredshed med det?

Mattias: Det. . . Det tror jeg. Der har i hvert fald været rigtig meget positiv feedback på at det er blevet mere organiseret. Folk er også. . . Altså folk bruger det til det tror jeg. Og jeg tror hvis interfacet ligesom bliver endnu mere brugervenligt, som f.eks. med en app, så tror jeg det vil blive helt suveræent.

Henrik: Ja.

Anton: Har folk deres telefoner med sig derinde? Altså det var mere det, at hvis vi laver en app, så er det også nødvendigt, at teknologien er der.

Mattias: Det tror jeg helt sikkert. Der er i hvert fald... Musikken kører altid på telefoner derinde. Og der ligger altid telefoner overalt på bordene og sådan noget.

Henrik: Ja, det lagde jeg godt mærke til.

Mattias: Et lille trick er jo, at vi ikke har nogen skabe herude... Man kan ikke låse sine ting inde. Derfor tror jeg også folk de tager deres telefoner med ind.

Henrik: Okay.

Anton: Men de klatrer ikke med dem. De lægger dem på jorden.

Mattias: Ja.

Anton: Ja.

Mattias: Ja.

Henrik: Eller så kunne det jo være det var noget man skulle investere i nogle skærme eller et eller andet.

Mattias: Jeg ved ikke... Har i kigget inde på den der hjemmeside jeg sendte til jer? Minimum Boulder i Schweiz?

Henrik: Ikke jeg personligt, nej.

Anton: Nej, det er der vist ikke nogen af os der har.

Mattias: Nej, okay. Det var faktisk dernede, at jeg fik idéen til hele den her app. Altså stjal idéen. Det er ikke mig der har fundet på det. Men de havde nemlig... De havde iPads i sådan nogle... Låste iPads stående rundt omkring i deres klatrehal. Og så havde de sådan en app, og derinde, der kunne man gå ind, så kunne man vælge en sektion, så havde de sådan nogle billeder hvor man kunne trykke på... Så kunne man vælge en sektion og så kunne man... Ja, så kunne man se hvad for nogle ruter der var...

Anton: Ja, der var tilgængelige i det område.

Mattias: Lige præcis. Så det rigtig fede ved det var, at så kunne man sådan give dem en stjerne fra et til fem, så man kunne, når man havde klatret en rute, sige den var fed, den var ikke så fed. For så kunne man gå ind og se, okay hvis man har begrænset tid, hvad for nogen er de fedeste boulders. Og så er det bare dem man klatrer.

Anton: Ja, eller hvis man kommer ind, som en ny og skal se, hvad skal jeg lige prøve.

Mattias: Lige præcis.

Anton: Hvad eksisterer der ellers af sådan nogle systemer heroppe... Hvad kender du til, der sådan... Nu snakker du om, at der er kommercielle klatreklubber, bruger... Hvordan gør de f.eks.?

Mattias: Altså... I Danmark der bruger de det ikke. Jeg ved at der er en klatreklub i... Som er... Den er stort set kommerciel, men det er en forening, og på Sjælland, der prøvede at indføre det og jeg har faktisk været ved at kigge efter det, for at sende det til jer, men jeg tror sgu... Jeg tror sgu de har annulleret det igen.

Anton: Snakker du så om det her system vi snakkede om lige før? Som du havde set?

Mattias: Nej, det er så et andet system. Men jeg går ud fra, at det er nogenlunde det samme, men jeg har aldrig set det.

Anton: Men de har et eller andet system også?

Mattias: Ja, men tror så måske bare de har fjernet det. Jeg kunne ikke finde noget om det på nettet.

Anton: Så generelt, så bruger folk ikke et eller andet system.

Mattias: Ikke i Danmark, i hvert fald. Men i udlandet der er det min helt klare opfattelse, at det er meget populært at bruge.

Henrik: Okay, det der med ratings af dem f.eks. er det noget I har overvejet at gøre. Det er selvfølgelig ikke lige som I har skrevet det op derinde... Lige nu er det jo ikke sådan umiddelbart så nemt at lave det.

Mattias: Nej, vi har sådan at man kan gå ind og sætte en stjerne.

Henrik: Nå, kan man godt det, derinde?

Mattias: Ja, men det er jo ikke sådan... Når der er én der har sat en stjerne, så kan man ikke sætte flere.

Henrik: Nej, okay, fair nok.

Henrik: Er der andre af sådan nogle ting ved systemet, som kunne være... Du sagde noget før om, at det var egentlig det det var... Altså systemet var hvad det skulle være, så skulle det bare være digitalt i stedet for. Det lyder lidt sådan, men det er ikke sådan at du kunne forestille dig, at der var nogen...

Anton: Er der nogle begrænsninger i jeres tavlesystem?

Mattias: Ja. Altså jeg synes begrænsninger er at det er sådan... Det er lidt svært at overskue. Altså specielt, hvis man nu skal oven på. Hvis man skal kontrollere, at det er opdateret. Så skal man enten tage et billede af det og gå op, og så skal man notere ved siden af ens telefon, mens man kigger på billedet: Den rute er der ikke, den rute er der ikke. Så skal man nedenunder igen og slette og det ene og det andet.

Mattias: Altså det ville fandme være nemt med sådan en app. Altså hvis man så også kunne sortere det efter dato og sådan noget. Det er jo nemt at implementere, kunne jeg forestille mig. Så kunne man gå ind... også fordi det er vigtigt, at man kan... at man kan sortere det i sektorer. Vi har en sektor A, B, C og D derinde. Fordi så kan man gå ind, når man f.eks. ryder sektor D, så i stedet for, inde på tavlen, der er sektor D jo spredt over hele tavlen. Hvis man så i appen bare kunne gå ind og trykke slet det hele.

Interviewet sættes på pause da en anden person kommer ind i lokalet.

Mattias: Hvad skal I egentlig kode det i?

Anton: Pålagt C#.

Henrik: Skal vi godt nok det?

Anton: Det sagde semesterkoordinator som det allerførste, tror jeg.

Mathias: Men det er jo så åbent for alt stadigvæk...

Anton: Ja, ja altså om det er apps eller web eller desktop... Der er ikke rigtig så mange begrænsninger i det. Det er svært at lave embedded...

Mattias: Altså det behøver heller ikke være en app. For mig er det fint nok med...

Anton: Nej, nej, men det giver jo god nok mening, som en idé i hvert fald.

Henrik: Ja, det er jo noget alle brugerne på sin vis har adgang til...

Anton: Fordi at det er den platform man har med sig.

Henrik: Ja.

Mathias: I forhold til administration lød det også meget smart det med... Altså bare at kunne tage den med rundt og så se, hvad er det der skal gøres...

Mattias: Kan man ikke også det i en browser?

Mathias: Jo.

Mattias: Ja, lige præcis.

Anton: Men det er jo det, altså det kunne også sagtens blive en hjemmeside, fordi at... At hvis bare det layout kan tilpasses skærmen, så er der ikke særlig stor forskel på en hjemmeside og en mobilapp.

Henrik: Det er jo lidt det du siger med, at i ikke har nogen skabe og så kan folk ikke låse dem inde. Hvis nu en dag i besluttede jeg for at få skabe så. . . Men det kan man jo selvfølgelig også tage til den tid. Men så kunne det jo være at man blev nødt til at investere i en skærm.

Mattias: Men altså, hvis. . . Det har vi sådan set talt om i bestyrelsen. Altså hvis der kommer et godt system op og køre. . . Vi har også talt om at gøre det selv, vi har nogle stykker der er software-data og sådan noget.

Anton: Jeg ser lige bestyrelsen, der er to ude fra Computer Science der.

Mattias: Ja, der er nemlig så.

Anton: Det er sådan lige ved, at "er det sådan en hel Computer Science-klub i har?".

Mattias: Så det er bestemt ikke udelukket, hvis der kommer et godt system, at vi kunne investere i sådan noget.

Henrik: Sker det nogensinde, at der går noget galt med ruterne? At de går i stykker eller hvad skal man sige. . . Falder der et greb af eller?

Mattias: Nej, det gør der sgu ikke.

Henrik: Så det er ikke sådan at de skal vedligeholdes?

Mattias: Nej.

Anton: Kan du liste hvad er af forskellige informationer, som skal være i det her system? Altså så er der sådan noget som hvor den starter, hvornår den er oprettet og hvem der oprettede den. Hvad for nogle forskellige dele består systemet af?

Mattias: Altså som udgangspunkt de ting der står inde på tavlen. Jeg kan ikke lige huske dem allesammen i hovedet. . .

Anton: Jeg tror jeg har et billede af det, så. . .

Mattias: Det er sådan grund tingene. Men noget jeg har set. . . Altså i kan også. . . Jeg har set nogle ret fede systemer. . . Jeg har faktisk set sådan noget VR, hvor man kan tage sådan en VR på, og så kan man sådan kigge rundt. . .

Anton: Og så er det sådan markeret op eller hvad?

Mattias: Lige præcis. Så kan man sådan skifte imellem ruterne og så kan man faktisk se én klatre ruterne.

Henrik: Okay, ja det kunne være fedt.

Anton: Jeg tror også vi så, at man kunne putte sådan nogle LED'er i klatregreb og så kunne man jo lave alt muligt fancy der.

Mattias: Men en anden ting jeg tænkte på, det var også noget. . . Altså jeg ved overhovedet ikke. . . Jeg er meget til det der med, at jo simplere og nemmere det er at bruge, jo nemmere vil folk have med at tage det til sig.

Henrik: Det er også meget af det vores semester handler om sådan set, så det er jo fint nok.

Mattias: Men features man kunne tænke på det var sådan noget som f.eks. hvis man tager et billede af væggen. Jeg ved. . . Der findes faktisk nogen apps nu, hvor man kan tage et billede af en væg og så kan man sådan trykke. . . Eller sådan markere grebene. Put sådan nogle cirkler om dem og så kan man vedlægge dem ruten.

Henrik: Okay.

Anton: Altså nærmest bare tegne ruten og så er den defineret og så kan du smide den ind i systemet.

Mattias: Lige præcis, for hvis man så går ind på appen der og kigger: "okay den her rute har fået gode anmeldelse", så kan man faktisk se et billede af væggen og se grebene, ligesom markeret. Så man ikke er i tvivl om det.

Mathias: Men når man så skal op og klatre en rute, kan det være svært sådan at vide hvordan man skal gå til ruten, altså hvad ham der har lavet den, har tænkt eller er det sådan meget personligt med hvordan man tackler ruten.

Mattias: Det er meget individuelt. Men det er også en stor del af udfordringen faktisk, når man klatrer.

Henrik: Men det er ikke sådan at der skulle være en guide til hvordan man klarer ruten så?

Mattias: Nej, nej. Altså maks. en note, hvor der står, du må ikke bruge højre ben, eller lignende. . . Sådan nogle ting kan der godt være. . . Sådan noget med at du ikke må bruge kanterne.

Anton: Nu ved jeg at Bøgholm f.eks. snakkede om at det kunne være man kunne vedlægge en video af en der klatrer ruten eller sådan noget.

Mattias: Ja. Det er i hvert fald en fed feature at have muligheden for det.

Henrik: Men det var det der med, at det var en form for guide til hvordan man skulle gøre det så, men hvis folk hellere selv vil finde ud af hvordan man skal gøre det så. . .

Mattias: Nårh! Ja, men så lader man være med at se videoen.

Henrik: Ja, fair nok.

Mattias: Det. . . Man har en ting i klatring man kalder beta, som er ligesom. . . Så hvis du nu skal op og klatre og du har sygt mange problemer med ruten, fordi du prøver at flytte din højre hånd på en eller anden bestemt måde, så kan jeg komme hen til det og så kan jeg sige, "det er fordi du faktisk skal bruge venstre hånd!". Så har jeg lige givet dig beta til ruten.

Anton: Okay.

Mattias: Det taler man rigtig meget om, ligesom at give hinanden beta og beta-videoer og sådan noget. Man kan gå ind på sådan nogle populære udendørsruter, der kan man gå ind på nettet og finde video-beta, til dem. Så man kan gå ind og se hvordan det er nemmest at klatre.

Anton: F.eks. sådan nogle alpinruter eller et eller andet.

Mattias: Nej, lige præcis ikke alpinruter.

Anton: Nej, men sådan noget som stenen derude måske eller et eller andet, lad os sige det var en populær rute.

Mattias: Ja, hvis den var super populær, ja så kunne man helt sikkert det ja.

Mattias: Så det kunne være en rigtig cool funktion at kunne uploade video-beta.

Henrik: Men der snakker vi bare enkelte greb eller enkelte... fra det ene greb til det andet? Eller skulle det være hele ruten?

Anton: Det ved jeg ikke, altså det er jo bare hints i det hele taget, vel. Altså sådan helt generelt.

Mattias: Ja, ja.

Anton: Altså en eller anden form for brugerinteraktion, hvor det er man kan vedhæfte en eller anden kommentar eller et eller andet. Så det kunne bare være sådan et kommentarsystem f.eks.

Mattias: Ja, eller en video. Det kunne også være meget cool.

Henrik: Ja. Umiddelbart vil jeg mene, at vi er kommet omkring de fleste af ting vi har planlagt. Så var der noget af det, som røg i vasken. Vi havde regnet lidt mere med, at der var... At vi kunne fokusere lidt mere på sådan, at der var bouldering, så var der reb og så var det sådan at skulle lave noget til det ene og lave noget til det andet.

Anton: Ja, hvis der nogen forskel på det, så havde det jo selvfølgelig været interessant, men det er jo fedt nok egentligt, at fra et systemudviklingsperspektiv at de er ens.

Mattias: Ja, I kan godt se, at hvis I laver et system hvor vi kan gå ind og tilføje sektorer, så deler vi jo bare der, ind i fem sektorer og så tilføjer vi fem nye sektorer.

Anton: Grebene, er de... Nu holdt jeg ikke lige øje med det derinde, men er de på sådan et regulært grid? Altså er de sådan f.eks. lige afstand fra hinanden alle de huller, hvor der er mulighed for at sætte greb i?

Mattias: Jaah.

Anton: Og er det forskelligt... Er det f.eks. regulært herinde i reb-hallen, men noget andet inde i boulder-hallen?

Mattias: Jeg tror det er rimelig... At grid'et er rimelig ens på pladerne, men pladerne sidder jo sådan i sådan noget. . . triangel-noget.

Anton: Så de sidder i nogle sjove vinkler?

Mattias: Ja.

Mattias: En anden ting til. . . Lige til beta-funktionen jeg tænkte på der. Det er ikke så meget beta, det er mere det der med med ligesom at kunne. . . Altså det er vigtigt at gøre klart, hvad for en rute det er, ikke? Og en af tingene det er jo de der klodser med nummeret på. Men det kunne også være enormt cool, hvis man bare kunne uploade et billede enten af hele ruten, som det vi talte om, hvor man kunne markere grebene. Eller måske bare start grebet, for inde i boulder og sådan set også rutehallen, der er det sådan at man starter altid i ét greb eller to greb og så er det ligesom derfra det begynder. Så det kunne være meget cool, hvis man kunne. . . Måske bare tage. . . Uploade et billede af start-grebet.

Anton: En anden måde at markere det på.

Mattias: Ja, lige præcis, fordi så kan man når man så er inde i appen, så kan man ligesom se, “okay, det er start-grebet”, så kan man kigge, “okay, så er dér den starter, ingen tvivl!”.

Henrik: Men ville der ikke være en tvivl om det? Ville det ikke være svært at finde udfra et billede af et enkelt greb. . .

Der er mistet omkring 30 sekunder af interviewet, da optagelsen blev for lang for kameraet, og der skulle startes en ny optagelse

Mathias: . . . systemet hvis nu I sådan. . . I snakkede om nye lokaler og ny hal, med det nye der. Så det skal jo noget man ligesom kan tage med sig, og oprette en ny sektion, hvis nu hallen bliver større, så på den måde bliver der også noget på den måde. Lidt administrering. . .

Mattias: Jamen det kunne være super ideelt hvis den måde man bare oprettede en sektion på, altså sådan noget som at oprette sektioner og sådan noget, det skal måske være forbeholdt administratorer, men at oprete ruter, skal alle brugere kunne. Men sådan noget som at oprette en sektion. . . at man måske bare uploader et billede af sektionen; et stort billede, og så kan give den et navn, mere behøver der ikke være i det.

Anton: Ellers så ændrer hallerne sig jo ikke sådan lige, så der er også, hvis det var man havde sådan et eller andet ovenfra-kort eller et eller andet, en eller anden kort beskrivelse af det måske.

Mattias: Ja. Det kunne også være lidt cool

Anton: Hvis der var en nemmere måde end bare et billede, og give en indikation af hvor i rummet det var

Mattias: Jeg tror bare at det er rigtig vigtigt I har i baghovedet, at folk er luddovne. Der er aldrig nogen der vil gide og lave et fedt sådan overview-billede. Altså det er derfor jeg tænker sådan noget som bare at tage et billede, altså enhver kan jo gå derind lige nu, snappe et billede af en sektion, uploade. Det tager fire sekunder, det er nemt at gøre. Men hvis man skal til at ind og tegne...

Anton: Nej, det var mere hvis det var sådan noget sektion der skulle gøres én gang i tre år, så kunne man... Altså der ville jo ikke være rigtig nødvendigvis nogen forskel i systemet hvis der var at der alligevel bare var support for et billede om man så havde brugt lang tid på det billede eller om det bare var et der lige var tegnet, sådan en hurtig tegning

Mattias: Helt sikkert

Henrik: Men øhh... Jamen det er jo sådan at vi gerne, vi kunne godt tænke os at den måde vi kunne arbejde på, det var at vi går hjem nu, eller ikke nødvendigvis lige nu...

Mathias: ... og så er vi klar i morgen

Henrik: Ja, lige præcis... så laver vi en eller anden form for prototype af det her, på en meget simpel måde, evt. nok bare på papir, med en helt masse sider vi klipper ud og nogle knapper og sådan noget, og at vi så kan komme tilbage en anden gang...

Mattias: Ja da, selvfølgelig

Henrik: ... og teste, ikke nødvendigvis bare på dig, men måske også på nogle andre i klubben. Det er ikke noget vi behøver, men det kunne være meget interessant at se hvad...

Matthias: Jeg synes da det er en genial idé at komme herop når der er allerflest mennesker, og så dele sådan en prototype der ud på papir, eller hvad det er.

Henrik: Ej vi kan nok ikke dele den ud, det er vist noget værre rod, det er sådan noget med at så har vi sådan en masse små knapper vi har klippet ud...

Anton: Og så siger du hvor du vil trykke og så flytter vi rundt på papirene

Henrik: ... det er noget værre arbejde, men vi kan gøre det med et par stykker, måske endda 10 hvis vi er heldige, det kunne være super fedt. Og så få noget feedback på, altså hvad for noget der virker og hvad for noget der ikke virker, hvad der er brugervenligt og hvad de synes var helt hen i vejret

Mattias: Det tror jeg helt sikkert at folk gerne vil hjælpe med

Anton: Det er i hvert fald en stor del af vores semesters undervisning

Henrik: Det er primært det, det handler om

Mattias: Altså interfacet?

Anton: Det her med at have en iterativ proces og lave nogle små ændringer, se hvordan de virker med brugerne og så arbejde videre derfra. Også så vi hele tiden er på sporet af: hvordan når vi til det bedst muilige resultat?

Henrik: Yes, og involvere brugeren i det, klienterne må det så være

Mattias: Jamen der er 270 medlemmer der gerne vil hjælpe, så. . .

Henrik: Jamen det er jo det. Det kunne være super fedt

Anton: Hvordan fungerer det altså, fordi jeg tror da ikke. . . er I 270 medlemmer herinde hver anden torsdag eller hvad?

Mattias: Overhovedet ikke, jeg tror vi er en hård kerne på en 10-15 mennesker der er her en fire gange om ugen eller sådan noget

Anton: Så andre de dropper sådan ind?

Mattias: Jeg tror de er her en gang hver anden uge eller sådan noget, og så er der mange af medlemmerne der har børn, eller forældre.

Anton: På sådan en typisk mødeaften, hvor mange er i så?

Mattias: Der er nok en 50-70 mennesker. Men det er jo så over en hel aften.

Anton: Og så en hel aften, det er fra hvornår til hvornår?

Mattias: Det er vel fra. . . Nu omkring og så til klokken en ti-11 stykker.

Anton: Okay, og så dropper folk bare ind og går, som det passer.

Mattias: Ja, altså vi har 24/7. . . 24/7-åbent, så folk kan i princippet komme og gå, som de har lyst.

Anton: Er der nogen der dropper herind for at sidde og arbejde på deres skolearbejde f.eks. . . Bliver det brugt som sådan et fritidsrum?

Mattias: Det er ikke mit indtryk.

Anton: Nej.

Mattias: Det tror jeg sgu ikke.

Anton: Men folk dropper jo så ind for at træne, når det så lige passer dem f.eks. Og bruger det som sådan et fitnesscenter måske?

Mattias: Ja, helt sikkert. Helt sikkert.

Mathias: Kommer man alene eller kommer man i små grupper. . . Altså hvis nu jeg vil træne klokken seks om morgenen alene, er det så fint?

Mattias: Ja, ja. Det kan du gøre som du har lyst. Men det er jo sådan lidt. . . Vi har jo klubaften tirsdag, torsdag, søndag og mandag. Og der er det typisk, at man kommer sådan lige omkring nu her. Så samles man ligesom derinde og hygger mens man klatrer. Det er overraskende. . . Selvom det er individuelt, så er det overraskende socialt. Det er meget fedt.

Henrik: Ja, det kan jeg godt se. Der er meget med noget feedback og hjælpe hinanden og sådan noget. Lyder det til.

Mattias: Men jeg synes da i skal, hvis i har lyst selvfølgelig... Så skal i da bare komme herop en dag. En mandag, en tirsdag, en torsdag eller en søndag. I kan bare skrive til mig inden, så skal jeg nok lukke jer ind. I behøver heller ikke betale for indgang. Og så kan i prøve at klatre. Så kan i selv opleve, hvordan systemet er derinde.

Henrik: Ja, det kunne være super fedt.

Mattias: Altså fordi, i kommer helt sikkert til at tænkte: "det er fandme lidt svært at finde rundt i". Og det er jo suverænt, fordi så kan i ligesom...

Henrik: Ja, ja. Så kan vi få nogle idéer.

Mathias: Ja, ja. Så ved vi hvad vi skal.

Anton: Så har vi noget at arbejde med.

Henrik: Cool. Det kunne jeg i hvert fald godt tænke mig.

Mathias: Ja, det snakkede vi om, at det kunne da være super fedt at prøve det på et eller andet tidspunkt i projektet.

Henrik: Det eneste, er at det ligger lidt langt væk.

Mattias: Det er I meget velkomne til.

Anton: Tak.

Mattias: I må også gerne tage de fire andre med.

Henrik: Ja, så må vi jo se om vi kan lokke dem til det så.

Anton: Vi har talt lidt om at prøve og finde en dag i hvert fald. Men altså... Ja, skal vi ikke sige, at vi prøver at aftale en dag. Måske.

Mattias: I skriver bare. Jeg er her tirsdag, torsdag, lørdag, søndag. Så der kan i...

Mathias: Men det kunne jo også eventuelt også være i forbindelse med en prototype. Hvis vi nu kom herud og skal prøve det lidt.

Henrik: Ja, så kunne vi bruge en hel aften her og så kunne vi måske skiftes lidt til at... Til at sidde med den og så er der nogen der klatrer. Vi kan heller ikke være syv mennesker der sidder omkring én stakkels person.

Mattias: Når man ikke er vant til det, så kan man ikke klatre meget mere end 20 minutter alligevel, så begynder ens fingre at være ødelagte.

Henrik: Fair nok.

Anton: Så begynder armene helt at ryste. Ej, jeg har været ude at klatre lidt med reb før, ja. Jeg har været her nogle gange før. Andreas Bagger, ham ved jeg ikke om du kender?

Mattias: Jo, jo, jo.

Anton: Min storebror, som har været...

Mattias: Jo, jo, men han bor i København nu, gør han ikke?

Anton: Han er flyttet til Roskilde. Ja... Ham... Jeg presser på min søster, for at få dem til at flytte tilbage igen. Men ja.

Mathias: Jamen godt så. Var det det?

Henrik: Ja, jeg tror det var det. Du skal have tusind tak for din hjælp i hvert fald.

Mattias: Jamen jeg synes da bare det er fedt. En fed mulighed.

Appendix C

First Iteration: Notes From Second Interview With Paper Prototype (25th of October, 2016)

System definition Hornum synes at systemdefinitionen lød fyldestgørende og han havde ikke yderligere kommentarer.

Citater:

- Spurgt ind til, hvilken platform, der sigtes efter. Hertil svarede Hornum, at PC umiddelbart ikke er nødvendig, da alle har en smartphone med i klatreklubben, om de så efterlader den i omklædningsrummet eller tager den med ind i hallerne er irrelevant. Han nævnte dog at AKK har én PC stående, som evt. kan benyttes af de, som ikke har en smartphone med. Der kunne også bruges tablets, men det er noget, som AKK skal vælge at investere i først.
- Hornum havde problemer med at forstå kravet: "be dynamics in regards to context", dette kunne derfor uddybes, så man forstår at der menes, at systemet skal kunne tilpasses, hvis sektioner ændres (slettes eller oprettes).

Minimumskrav til rute Sværhedsgradsblok og nummer, som markerer startblokken af ruten.

Citater:

- "Da dem, som skal gå ind og lave ruter er frivillige, så skal det [systemet] være så problemfrit og simpelt som muligt.". Der skal altså være så lidt tvang ift. Fx minimumskrav til en rute.
- I forbindelse med ovenstående, nævnte Hornum også, at det ville være en dårlig idé, hvis alle medlemmer skulle logge ind, da det ikke er nødvendigt, og da medlemmerne ikke gider. Administrator-brugere har brug for et log-in. De kan fx cleare en sektion, slette ruter mv.
- Så få begrænsninger som muligt, derfor skal alle have lov til at oprette ruter mv.
- Sværhedsgraden på en rute ændres sjældent. Man har ofte en idé om, hvilken sværhedsgrad en rute har, ved dens oprettelse. Den, der laver ruten, bestemmer sværhedsgraden.
- Én gang årligt ryddes hele boulder-hallen i forbindelse med konkurrence. Derudover er det ikke ofte, at en rute fjernes, men dette kan begrænses til en administrator-rettighed, så man undgår at ruter bliver fejlagtigt slettet. Det kan være en mulighed at anmode

en administrator om at få en specifik rute slettet, hertil kan man skrive en besked, der begrunder forespørgslen.

Papirprototypen

- Da Hornum så papirprototypen, kunne han umiddelbart ikke se, hvad fx A2 betød. Dette mente han kunne gøres simplere og mere pædagogisk, hvis man fulgte metoden, de allerede bruger på deres whiteboards. Dvs. at lave en farvet firkant med et tal indeni, der symboliserer hhv. sværhedsgrad og begyndelsesblok/greb. Til højre for denne firkant ville der stå, hvilken sektion banen er på.
- Hornum synes umiddelbart papirprototypen var nem og intuitiv, men ville fx gerne uddybe, hvad sorterings-funktionerne betød. Dette kunne gøres ved at skrive et ord ovenfor dropdown menuerne, der forklarede, hvad man sorterer efter.
- Hornum sagde at alle i AKK godt ved, hvilken farve passer med hvilken sværhedsgrad, derfor skal vi ikke uddybe dette! (vigtigt!)
- Tekst-beta er ikke en god ide. "Man skal være god til at formulere sig, hvis man skal give beta på tekst".

.

Ekstra funktioner

- Hornum nævnte, at det kunne være nice med en funktion til at sætte cirkler omkring alle greb/blokke i en bestemt rute.
- Hornum mente, at det ville være smart, hvis alle ekstra-funktioner fik et logo. Fx, hvis der var beta tilgængelig, ville der være et farvet logo med et B i, hvorimod, hvis beta ikke er tilgængelig, ville dette logo være gråt. Dette mener han ville være smart for alle ekstra-funktioner.
- Hornum mener at muligheden for at bedømme en rute ville være "en rigtig god idé". Han nævnte dog desuden også, at sådanne ting vil være en 4'er på requirements, og at det vigtigste er at systemet får den funktionalitet, som tavlesystemet i forvejen har.
- Det kunne være fedt, at kunne kommentere på ruter. Dette er ikke det samme som tekst-beta.
- Kunne være en meget grineren feature at kunne give ruten et navn.

Appendix D

First Iteration: Usability Test (9th of November 2016)

Task 1

a) In the system you see a list of routes - find route blue 38 in section c.

b) Find a route that is created by Hans.

Task 2

a) Find a route on the wall and add it to the system.

b) You find that the route you have just created should have had the black grade. Change this in the system.

Test person 1 (Customer)		
Task	**Description**	**Usability problems**
1a	Found the route in about five seconds. Did not encounter any problems finding said route.	
1b	To find the route Hans had created, he scrolled down the list of all routes and found Hans's route. Did not realise he could sort after route's creator alphabetically. When he asked if there was an easier way to do it, and was told there was, he found out it was possible to do exactly that.	P1, P7
2a	He quickly realised how to create a route but forgot to add a section to it, and did not know why it could not be created, but found out on his own relatively quickly, that he forgot to add a section.	P2
2b	No problems editing the created route.	
Comments		

Table D.1: Descriptions and usability problems for tasks performed by test person 1.

Test person 2		
Task	**Description**	**Usability problems**
1a	Found the route by filtering by sections. Did not sort after grade.	
1b	Had problems finding a route created by Hans. She scrolled through the list but could not find him at first glance, so went to the top and attempted to create a new route. When going to the create new route page, she found a field called Author, and mistakenly thought that for a way to search for the author. When she scrolled to the bottom, she realised she was about to create a new route, so went back and scrolled again, and found the route this time. She did not notice that it was possible to sort by author.	P1, P3, P7
2a	Did not have any problems at all.	
2b	Did not have any problems at all.	
Comments	The system gives a better overview than their existing system (the whiteboards). The idea behind rating is a good idea. She does not think the route needs an author.	

Table D.2: Descriptions and usability problems for tasks performed by test person 2.

Test person 3		
Task	**Description**	**Usability problems**
1a	Scrolls down to find the route, and finds it easily. When later asked if she can do it more easily, she finds out she can sort and filter, then does so.	P4
1b	Sorts by author and finds a route by Hans.	
2a	When creating the route, she gets an error-message, stating that the route already existed. She did not have any problems understanding the error-message, She asked if she should just change the number and the facilitator answered yes. Then she just changed the route number and created the route.	
2b	Found out how to edit the newly created route and was able to change the grade. Did not encounter any problems.	
Comments	Works well.	

Table D.3: Descriptions and usability problems for tasks performed by test person 3.

Test person 4		
Task	**Description**	**Usability problems**
1a	Filters grade, then section, and finds the specific route.	
1b	Right as he begins the task, he complains about a missing clear-filters button. He has problems finding Hans's route since he believes the + is a search function, and he believes finding the route created by Hans is too difficult, so he decides not to solve the task. Mentions it would be nice with a search function that can search for whatever string you put in.	P3, P7
2a	+ in his mind means "add", so he presses the + button, and creates the route without any troubles at all.	
2b	Edits the route but did not save the changes. When asked if the route had now been edited, he scrolled to the bottom, where he found a save button, then mentioned that it was too old-school, and the route should be edited right away as soon as you made any change to it.	P6
Comments	Old-school save button. When editing the route, the edits should be enabled right as something got edited, not when the save button is pressed. No need for a save button when it's not a critical system. Needs a button for more advanced searches / sorts, so he can search for any possible string. He wants more advanced searches and recommends elastic sort. He wants a button for every possible search, so he does not have to do multiple touches to do a search or apply a filter. The layout was very intuitive however.	

Table D.4: Descriptions and usability problems for tasks performed by test person 4.

Test person 5		
Task	**Description**	**Usability problems**
1a	Scrolls down and finds the route. When asked if he can do it a different way, he finds out it is possible to filter by grade and section, so chooses to do so to find the route.	P4
1b	Scrolls down to find Hans's route, but cannot seem to find any route by him. So he tries creating a route, and realises he is about to make a route, then goes back and gives up.	P3, P7
2a	Could without a problem create a route. When he went back to the list of routes, he could not find his own since the filters was applied and the list of routes had not been updated to list the recently created route, even though it should have.	P5
2b	Did not encounter any problems editing the specific route.	
Comments		

Table D.5: Descriptions and usability problems for tasks performed by test person 5.

Appendix E

Second Iteration: Usability Test (1st of December 2016)

In our second usability test, we got 5 people to test it, all of them males.

Task 1

a) In the system you see a list of routes - find the oldest route with a blue grade.

b) Find a route that is created by Hans.

c) Use the search bar to find a route of white grade and a route number of 6, in section B.

Task 2

a) Find a route on the wall and add it to the system. You should include an image of the route, and mark the holds that are part of the route.

b) You want to add tape to the route you just added. Find the route you just added, and add a tape colour.

Task 3

a) Find the route with a black grade and route number 10 located in section A, and add a video beta to that route.

Task 4

a) Imagine you just climbed one of the routes located in the system. Give the route a rating and add a comment to the route, based on on your rating of the route e.g "Great route".

Test person 1		
Task	**Description**	**Usability problems**
1a	Could not find the route. Thought the colour of the hold was the grade colour. Kept scrolling but only found the wrong routes, one of them with a green grade.	P1, P2
1b	Searches for Hans, and finds a route made by him.	P9, P12
1c	Does not use the search functionality, but uses the filters instead. When explicitly asked to use the search bar, he searches for ”6 White”, and finds the route.	P9, P12
2a	Finds a route, adds it to the system with everything asked of him without issue. He is not sure about tape, so he scrolls past that.	
2b	Realise he can edit a route. He adds tape, but is unsure if toggling tape disables the hold colour.	
3a	Picks a route in Section B and opens his camera instead of pressing the ”add beta button” on the route.	P3
4a	Rates and add a comment to a route without any problems.	
Comments	Wishes there was a better explanation of what the search functionality is as well as what it is capable of and not capable of. Finds a bug, where two elements of each sections appear in the filter drop-down menu.	

Table E.1: Descriptions and usability problems for tasks performed by test person 1.

Test person 2		
Task	**Description**	**Usability problems**
2a/2b	Test person was in the middle of screwing his own route, so he was asked if he could possibly add the same route in the system. He opens section C, does not realise the + button is present for him to add a new route. When he is told about what he is doing incorrectly, he presses the plus button. He adds everything and wishes to add tape, but misses the toggle button. He takes a picture of the route and adds that to it without highlighting the holds. He is then told it is possible to manipulate the picture he just added, so he then finds the route, but tries to zoom using the standard zoom gesture. He also records video beta and adds that to the route without trouble.	P4, P5, P6
Comments	He wishes there was a share button so it is possible to share it with people on instagram.	

Table E.2: Descriptions and usability problems for tasks performed by test person 2.

Test person 3		
Task	**Description**	**Usability problems**
1a /1c	Scrolls to find the oldest route with blue grade and makes a guess. Scrolls after that. Finds the route without using filters, sorting and searching. After being asked if he could search, he did so, but inputs a Danish word instead of English.	P3, P7, P9
2a	Finds a route on the wall and registers himself. He puts in all the information except for section, and gets the error three times before noticing it, then corrects it by selecting the right section.	
2b	Finds the edit route page and adds tape to the route. He thinks the circles are to mark the start hold, but still marks them all on the picture he added.	P8
3a	Searches for ”black 10” and finds the button to add a beta, after looking a bit for it.	P9, P12
4a	Rates a route and adds a comment without trouble.	
Comments	Wishes the Return key hides the keyboard. He does not think the small icons on the Find-Route page is intuitive.	

Table E.3: Descriptions and usability problems for tasks performed by test person 3.

Test person 4		
Task	**Description**	**Usability problems**
1a	Scrolls down and reads the date of each routes. Realise afterwards he can sort, then does so.	
1b	Presses the search-bar and inputs "Hans", and finds a route by him without any issues.	P9, P12
1c	Searches for "White 6", and finds the right route without any issues.	P9, P12
2a	Presses the plus button, and is surprised that he has to log in. He logs in and creates a new route. Afraid to add a number to the route because it does not yet have one.	P10
2b	Sighs and thinks he has to create a new route to add tape to the route he just created. Thinks that it might be possible to edit it, which he finds out, then adds tape.	
Comments	He is pleasantly surprised about the program. He wishes there was functionality to tell the user what routes he has climbed, especially for the more difficult routes.	

Table E.4: Descriptions and usability problems for tasks performed by test person 4.

Test person 5		
Task	**Description**	**Usability problems**
1a	Finds the right route by sorting after oldest routes.	
1b	Searches for "Hans" and finds a route.	P9, P12
1c	Searches for "White section b route number 6". Will not give any results.	P9, P12
2a	Presses the plus button and registers. Adds the system and happy about the picture functionality of the system.	
3a	Adds beta to the route and finds out how to upload it on his own.	
4a	He adds a rating, but is looking around on how to upload that change, since he does not know rating gets updated immediately.	P11
Comments		

Table E.5: Descriptions and usability problems for tasks performed by test person 5.

Appendix F

Third Iteration: Usability Test (8th of December 2016)

Opgave 1

Du kommer ud i klubben og hører at sektion A snart skal ryddes. Før sektionen ryddes, vil du gerne klatre den ældste rute i sektionen en ekstra gang, inden det er for sent.

a) Find den ældste rute med grøn gradering i sektion A

b) Find ud af hvem der har lavet ruten

Din ven Hans har lavet en rute med blå gradering i sektion A.

c) Brug søgefunktionen til at finde ruten

Opgave 2

Du har lige skruet en rute op på væggen, og vil gerne tilføje den til systemet.

a) Find en rute på muren og tilføj den til systemet. Tilføj kun nødvendig information

b) Tilføj som note til ruten, at man skal starte siddende

c) Tilføj et billede af ruten hvor du markerer startgrebet

Du finder ud af, at du har lavet en fejl under indtastningen af ruten, da du har glemt at der er gult tape på din rutes greb.

d) Ret fejlen i systemet, så der er gult tape på ruten

Opgave 3

Du har klatret en rute, og er den første der har gennemført ruten. Du vil gerne hjælpe andre klatrere med at klatre ruten, ved at give dem beta.

a) Tilføj beta samt en kommentar på ruten, du lavede tidligere

Opgave 4

De andre medlemmer i klubben har været vilde med én bestemt rute, og det er derfor blevet den rute med flest stjerner. Du beslutter dig for at klatre ruten, og bestemmer dig også for at give den en bedømmelse.

a) Find ruten med flest stjerner

b) Vurdér ruten

Opgave 5

Klubben har fået nye lokaler og har fået plads til en ekstra sektion. Du er blevet givet opgaven at tilføje denne sektion til systemet.

For at løse denne opgave skal du være administrator af systemet. Log ind med oplysningerne:
Brugernavn: admin
Kode: 1234

a) Tilføj en ny sektion til systemet

Et medlem med brugernavnet **TannerH** er blevet en del af bestyrelsen, og vil være med til at administrere systemet.

b) Giv brugeren TannerH administratorrettigheder

Du vil gerne tilføje en ny farve som grebene kan have i systemet.

c) Tilføj en ny grebsfarve til systemet

Task 1

You arrive to the club, and discover that section A will soon be cleared. Before the section is cleared, you want to climb the oldest route in the section one extra time before it is too late.

a) Find the oldest route with a green grade in section A

b) Find out who made the route

You friend Hans has created a route with a blue grade in section A.

c) Use the search function to find the route

Task 2

You just added a route to the wall, and want to add it to the system as well.

a) Find a route on the wall and add it to the system. Add only the necessary information

b) Add a note to the route stating that the route should be started from a sitting position

c) Add an image of the route, and mark the starting hold

You realise that you made a mistake when adding the route, in that you forgot to add yellow tape to the holds.

d) Correct the error in the system, by adding yellow tape to the route

Task 3

You have been climbing, and you are the first to complete a route. You want to help other climbers who have struggled with the route, by giving them beta to the route.

a) Add beta to the route you created along with a comment

Task 4

The other members of the club loved one route in particular, and it has quickly become the top rated route in the system. You decide to climb the route as well and after doing so, feel that should share your opinion.

a) Find the top rated route in the system

b) Rate the route

Task 5

The climbing club has moved to a different location, and now has enough room for another section. You have been given the task to add the section to the system.

To solve this task, you need an administrator account. Log in with the following information:

Username: admin Password: 1234

a) Add the new section to the system

A member with the username **TannerH** has joined the board of directors, and wants to help administrating the system. To do that, his account must have its privileges elevated to administrator rights.

b) Give the user TannerH administrator rights

You want to add a new colour for the holds in the system.

c) Add a new hold colour to the system

Questionnaire

Name:	____________________
Age:	____________________
Climbing Experience:	____________________

Do you have a smartphone?

☐ Yes

☐ No

If yes, for how long have you had one? ____________________

	Strongly disagree					Strongly agree
	1	2	3	4	5	6
It was easy to use the system	☐	☐	☐	☐	☐	☐

If you have any comments or suggestions, please write them below.

Test person 1		
Task	**Description**	**Usability problems**
1a	Uses all filters to find the oldest route	
1b	No problems finding out who the creator of the route was	
1c	Uses the filters to find the route. Uses the search functionality later when asked if she could do it in some other way	P12
2a	She uses the burger menu to navigate to the add route page. She has no problems inputting the correct information	
2b	No issues adding a note	
2c	No issues adding an image, however, when she attempts to add the circle around the starting hold, she tries expanding the picture by using the zoom gesture, even though the image is meant to be pressed. She then pressed the edit button and marked the starting hold	P1
2d	No issues editing the route and adding the tape	
3a	She has issues finding where she can add beta. She tried herself first but later when she got hint about of what she was doing, was correct, she could add a video beta without issues. She later said she did not know that she had to add a video beta. She also adds a comment to the video once she has submitted the beta. She tries adding the comment when she is submitting the video	P2, P3
4a	Uses sort to find the highest rated route	
4b	Adds a rating to the route. She is not sure if the rating gets updated immediately	P4
5a	Logs out and logs in with the given username and password, and adds a section	
5b	Makes TannerH an administrator but needs feedback whenever the user is now an administrator or not	P10
5c	Adds a new hold colour but did not change the name	P5
Comments	Knows what beta is, but she had trouble understanding what was meant by adding beta during the given task	

Table F.1: Descriptions and usability problems for tasks performed by test person 1.

Test person 2		
Task	**Description**	**Usability problems**
1a	Sorts by oldest and filters by green grade to find the route	
1b	Knows who the route is created by, since the author is mentioned	
1c	Searches by Hans, then restricts his search to search for section a as well	
2a	Presses the + button at the bottom right, then registers and inputs all the information for the route	
2b	Presses the edit button and adds a note	
2c	Adds a beta and when he sees the image text, he adds a circle for the starting hold	
2d	Presses the Has Tape on Route button, but do not know how to add the yellow tape to the route, so he adds a note with ”yellow tape”. When he receives a hint about the changed holds, he then realises what it means but finds it distasteful	P6
3a	Adds a comment to the route as beta, then asks what is meant by beta in this context, and adds a video beta without issue	
4a	Sorts by adding and finds the route	
4b	Adds his own rating to the route	
5a	Logs out and logs in with the given username and password, and adds a section	
5b	Sees TannerH already has administrator rights, then makes himself an admin	
5c	Adds a hold and changes its name	
Comments	Tape should be added next to the hold, instead of on it. Tape was difficult to see	

Table F.2: Descriptions and usability problems for tasks performed by test person 2.

Test person 3		
Task	**Description**	**Usability problems**
1a	Finds the route quickly and without issue	
1b	Sees who the author is	
1c	Filters then searches ”Blue Hans A” and finds the route	
2a	Tries adding a new route, then gets greeted by the login screen. Says he does not have a username so tries logging in as default, but since there is no default user, he creates a new user and adds a route	
2b	Presses the edit button and adds a note	
2c	Adds an image to the route and adds the starting hold to the image	
2d	Edits the route and adds yellow tape	
3a	Tries finding how to add a beta, then opens the navigation bar, and tries swiping it away, which does nothing. Adds a comment as beta, then adds a video beta	P7
4a	Sorts by rating and finds the highest rated route	
4b	Adds his own rating to the route	
5a	Logs out and logs in with the given username and password, and adds a section	
5b	Changes the rights of TannerH	
5c	Adds a new hold colour and presses save, but realise he forgot adding a name to it, so edits the name. However since he already pressed the save button, it hold colour was already getting uploaded, so the name change did not go through	P2, P5
Comments	Support for QR-codes on walls. Keywords or tags on routes.	

Table F.3: Descriptions and usability problems for tasks performed by test person 3.

Test person 4		
Task	**Description**	**Usability problems**
1a	Needs to get an overview of the system, so it takes a bit of time for him to find out he can filter, then finds the route	
1b	Reads the author name	
1c	Searches for "Hans" and finds the route. Then restricts his searching once asked if he can do so. He has troubles restricting his search since he thinks it is in Danish, so searches for "Sektion" rather than "Section", but realise his mistake	P12
2a	Registers without issue, but has trouble reading the number located on the brick on the wall. He adds the route without encountering any other problems	
2b	No problems adding a note to the route	
2c	Adds an image and selects the starting hold	
2d	Has trouble finding where he can select a tape, but after a bit of help, he sees the "Has Tape" button, says it makes sense and then adds a yellow tape to the route	P6
3a	Adds a comment beta, but not a video. When asked if he could do so, he does it without problems	
4a	Sorts by rating and finds the top-rated route	
4b	Adds his own rating to the route	
5a	Logs out and logs in with the given username and password, and adds a section. He is not sure if the section has been added, but believes it has	
5b	Gives administration rights to TannerH	
5c	Adds a new hold colour, but with the default name	P5
Comments	He likes it is possible to rate routes, and thinks it is very nice that it is possible to use the application at home	

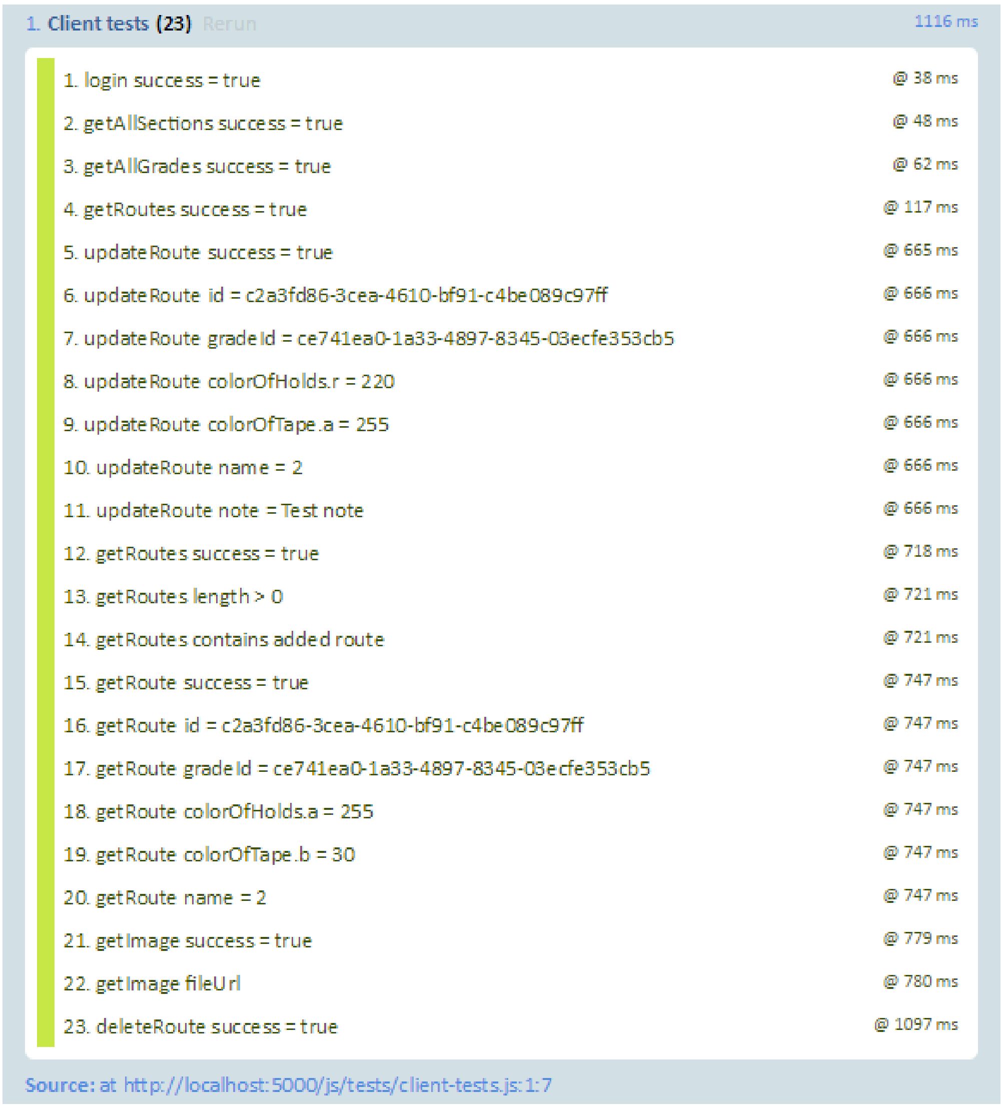
Table F.4: Descriptions and usability problems for tasks performed by test person 4.

Test person 5		
Task	**Description**	**Usability problems**
1a	Filters, finds the route and presses the route	
1b	Finds the route author, then goes back and finds that the filters and sort options has been reset, which he finds irritating	P8
1c	When searching, he finds that the search resets the filters, but manages to find the requested route	
2a	Adds a route with the necessary information	
2b	Adds a comment instead of a note, but when hinted at that, he realise his mistake, then edits the route and adds a note and proceeds to delete the comment he created by mistake	P9
2c	Adds an image and selects the starting hold	
2d	Edits the route and adds yellow tape	
3a	Adds a comment beta, and when asked if he could add a video beta, he did so without issues	
4a	Sorts by rating and finds the top rated route	
4b	Adds his own rating but is not sure if the rating has been submitted	P4
5a	Logs out and logs in then gets to the admin panel by using the browser's back button. Adds a long name to the route which causes the text to exit the boundaries of the section-name box. He then tries deleting it, but nothing happens, so he renames the section name	
5b	Changes the rights to TannerH. Needs search functionality to find the member he is looking for. After changing the rights, he needs some feedback to know whenever TannerH's rights has been changed or not	P10
5c	Adds a new hold colour and gives it a new name	P11
Comments	Wishes the filters and sort order did not change to default when searching or leaving the page. Also wishes it was possible to search for a member at the admin panel	

Table F.5: Descriptions and usability problems for tasks performed by test person 5.

Appendix G

Client Service Test Result

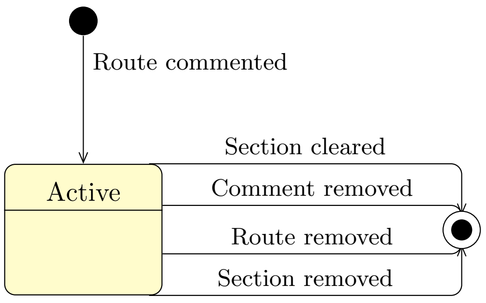


```
1. Client tests (23) Rerun                                              1116 ms

1. login success = true                                                 @ 38 ms
2. getAllSections success = true                                        @ 48 ms
3. getAllGrades success = true                                          @ 62 ms
4. getRoutes success = true                                             @ 117 ms
5. updateRoute success = true                                           @ 665 ms
6. updateRoute id = c2a3fd86-3cea-4610-bf91-c4be089c97ff                @ 666 ms
7. updateRoute gradeId = ce741ea0-1a33-4897-8345-03ecfe353cb5           @ 666 ms
8. updateRoute colorOfHolds.r = 220                                     @ 666 ms
9. updateRoute colorOfTape.a = 255                                      @ 666 ms
10. updateRoute name = 2                                                @ 666 ms
11. updateRoute note = Test note                                        @ 666 ms
12. getRoutes success = true                                            @ 718 ms
13. getRoutes length > 0                                                @ 721 ms
14. getRoutes contains added route                                      @ 721 ms
15. getRoute success = true                                             @ 747 ms
16. getRoute id = c2a3fd86-3cea-4610-bf91-c4be089c97ff                  @ 747 ms
17. getRoute gradeId = ce741ea0-1a33-4897-8345-03ecfe353cb5             @ 747 ms
18. getRoute colorOfHolds.a = 255                                       @ 747 ms
19. getRoute colorOfTape.b = 30                                         @ 747 ms
20. getRoute name = 2                                                   @ 747 ms
21. getImage success = true                                             @ 779 ms
22. getImage fileUrl                                                    @ 780 ms
23. deleteRoute success = true                                          @ 1097 ms

Source: at http://localhost:5000/js/tests/client-tests.js:1:7
```

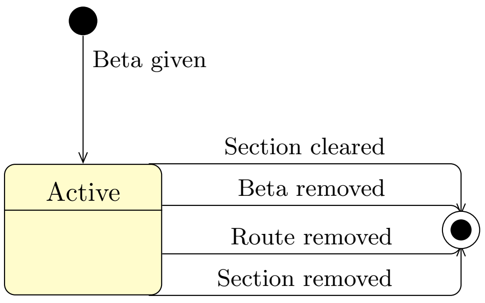
Figure G.1: Shows test result for the client service component.

Appendix H

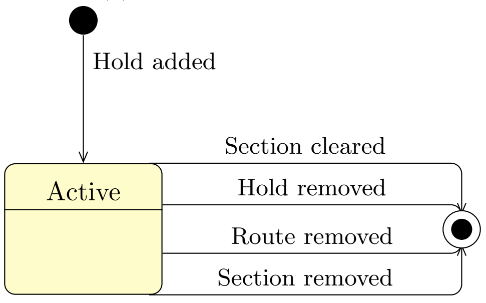
Additional Classes



(a) Behaviour for Comment class.



(b) Behaviour for Beta class.



(c) Behaviour for Hold class.

Figure H.1: Similar behaviour for several classes.

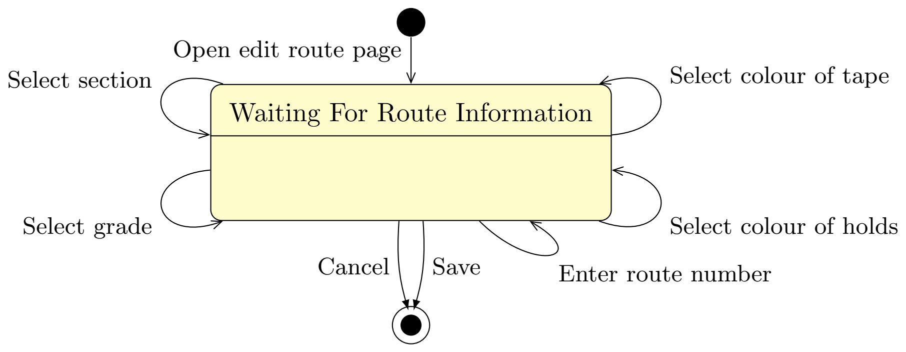


Figure H.2: The statechart diagram for “edit route” shows the procedural pattern of the use case.