Jørgen Eriksen, Elvis Arifagic, Markus Strømseth

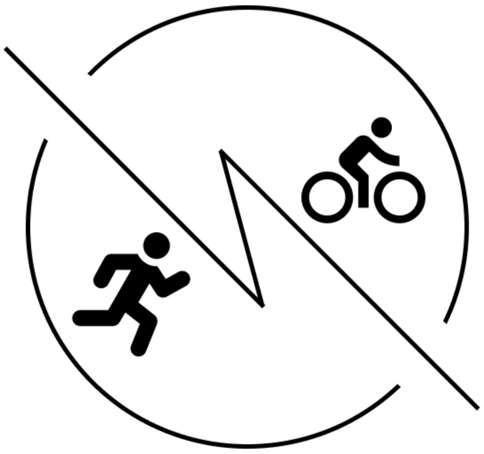
Bike & Run

Administration and display service for relay races.

Bachelor's thesis in Programming
Supervisor: Tom Røise
May 2022

Bachelor's thesis

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



NTNU
Kunnskap for en bedre verden

Jørgen Eriksen, Elvis Arifagic, Markus Strømseth

Bike & Run

Administration and display service for relay races.

Bachelor's thesis in Programming
Supervisor: Tom Røise
May 2022

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

NTNU
Norwegian University of
Science and Technology

Sammendrag av Bacheloroppgaven

Tittel:	Bike & Run
Dato:	20.05.22
Deltakere:	Elvis Arifagic, Jørgen Eriksen, Markus Strømseth
Veileder:	Tom Røise
Oppdragsgiver:	Headit
Kontaktperson:	Magne Johansen
Nøkkelord:	Kryss-platform, Mobil App, REST API, SCRUM, Fullstack
Antall sider:	96
Antall vedlegg:	8
Tilgjengelighet:	Åpen

Sammendrag:

Bike & Run er en folkehelse stafett bestående av fire etapper; two løpsetapper og to sykkeletapper. Headit har deltatt de siste årene i stafetten og ønsket å få utviklet et system for sporing av deltakerne og fremvisning av løpet. Vår løsning er ett integrert system med et REST API som benytter seg av en avansert algoritme for vekslinger. WebSocket forbindelse mellom backend og frontend for fremvisning av posisjoner i nær sanntid. En frontend tjeneste med en moderne industristandard protokoll for autorisering. Avslutningsvis en kryss-plattfrom mobil applikasjon medl okasjons sporing bakgrunnen.

Summary of the Bachelor Thesis

Title:	Bike & Run
Date:	20.05.22
Authors:	Elvis Arifagic, Jørgen Eriksen, Markus Strømseth
Supervisor:	Tom Røise
Employer:	Headit AS
Contact Peson:	Magne Johansen
Key Words:	Cross-platform, Mobile App, REST API, SCRUM, Full stack
Pages:	96
Attachments:	8
Availability:	Open

Abstract:

Bike & Run is a public health relay race consisting of four parts; two running stages and two biking stages. Headit as a company has participated in the relay race for the past several years and wanted a system developed for administrating tracking of participants and displaying the relay race. Our solution to this was an integrated system with a REST API utilizing an advanced algorithm for stage switches. WebSocket connections between the backend and frontend for displaying positions in near real-time. A frontend service with a modern industry-standard protocol for authorization. Lastly a cross-platform mobile application with background location tracking.

Preface

We would like to thank everyone who was involved and contributed to this bachelor thesis. A special thanks to Tom Røise who has been very active as a supervisor and as a provider of invaluable feedback during the project. We would like to thank Headit AS for providing us with a task that was both challenging and rewarding. With special thanks to Magne Johansen for his vital effort, engagement and friendship as the Product Owner on behalf of Headit AS as well as his mentorship who helped us deliver a product to our highest possible potential.

Contents

Figures

Tables

Acronyms

API Application Programming Interface.

CLR Common Language Runtime.

CPU Central Processing Unit.

GDPR General Data Protection Regulation.

GUI Graphical User Interface.

HTTP HyperText Transfer Protocol.

IoT Internett of Things.

JVM Java Virtual Machine.

MUI Material UI.

NTNU Norges teknisk-naturvitenskapelige universitet.

PO Product Owner.

PS Product Supervisor.

REST REpresentational State Transfer.

SDK Software Development Kit.

WCAG Web Content Accessibility Guidelines.

Glossary

Adobe XD Tool for UX/UI collaborative work..

Backend The part of a computer system or application that is not directly accessed by the user..

Bitbucket Git-based source code repository, designed to manage git repositories, collaborate and review code..

Cross-Platform Able to be used on different types of computers, mobile devices or with different software packages..

DevOps DevOps is a set of practices that combines software development (Dev) and IT operations (Ops)..

Docker Image A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform..

Frontend The part of a computer system or application with which the user interacts directly..

Hash A mathematical function that converts a numerical input value into another compressed numerical value.

Haversine Formula for calculating the distance between points on spheroid objects.

Keycloak Identity and Access Management Server which uses OAuth 2.

OAuth2 A industry-standard protocol for authorization.

Overleaf Collaborative latex writing tool.

Real-Time Relating to a system in which input data is processed within milliseconds so that it is available virtually immediately as feedback to the process from which it is coming..

Relay Race A race between teams of two or more contestants with each team member covering a specified portion of the entire course..

WebSocket WebSocket is a computer communications protocol, providing bi-directional communication channels over a single TCP connection..

Yarn Package manager for installing javascript packages.

Chapter 1

Introduction

1.1 Background

This bachelor's project was provided by Headit AS, and our contact person from the company has been Magne Johansen. Headit is an IT- and software company based in Hamar, founded in 2001 and has both regional, national and international clients.

Headit has for the past several years participated in a relay called Bike & Run. This relay is hosted annually by Bedriftsidretten Innlandet as a low threshold public health event. The relay consists of two running stages and two biking stages. In conjunction with this year's relay, Headit is interested in developing a software solution that allows spectators to watch participants in the relay in real-time on a display. The previous relays have not had any digital aids to help facilitate the relay so Headit initially wanted to use it as their tool in hopes that it can be seen as an interesting approach for Bedriftsidretten Innlandet to further use it for the entire relay, which in turn would help Headit attain a positive reputation and create some attention around the company.

A more technical aspect Headit wanted to research was the use of WebSockets and how one would integrate it with such a solution. Providing push notifications to the frontend client, thereby having near real-time updates for a precise overview of the relay and minimizing database strain during the relay. As a part of this project, Headit wants us to develop an administrative panel where they can create the relays and facilitate the creation of the teams among other tasks. They also would like for us to create a mobile app where people join/sign into their respective teams and using that app send their position to an API. That same API will push that data to a website where you can view the relay by plotting the coordinates onto a map display.

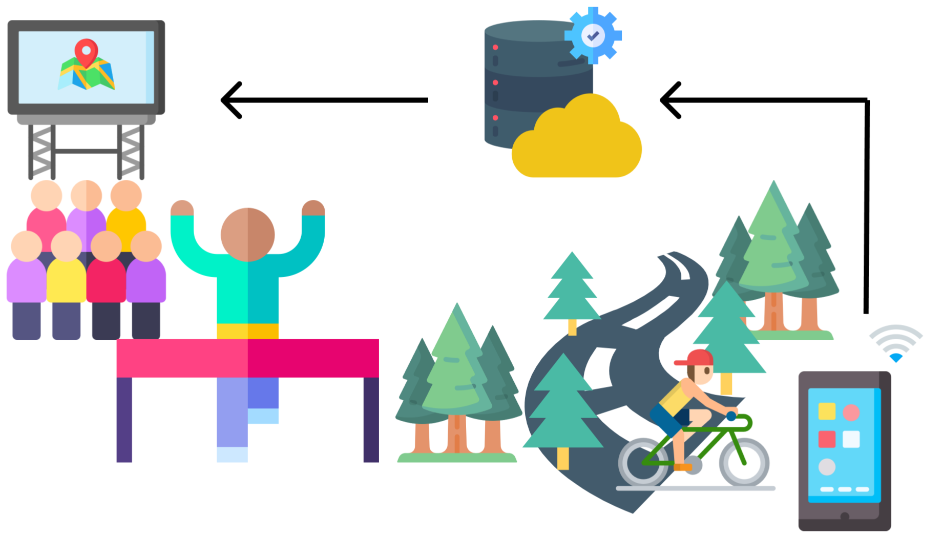


Figure 1.1: A scenario showing our system in use.

1.2 Project Description

Our goal was to develop a backend service, a frontend service for displaying the relay on a map, a frontend service for an administrative panel, a mobile application or IoT device application for tracking the participants during the relay, and to come up with a recommendation for a display medium.

The backend service should be an API. The API should be connected to a database that saves positions from the participants, both for a real-time relay and to save the relay as a history. It should be able to push participants' positions to multiple applications, support the admin panel application, and be ready to be distributed in a docker image. The API should be expandable for future work after we have delivered the project so it can be implemented with features from the participant device like speed and pulse, and also from other device sources.

The frontend service for the admin panel should be able to manage relays with teams and the number of participants per team. It should also display the invite code for each participant in every team.

The frontend service for displaying the relay should display each participant in the relay. The relay could have up to 40 teams with 4 participants each which it needs to handle. The frontend should also be able to be displayed in fullscreen.

The mobile application or IoT application reads the GPS positions of the participant's device and sends them to the API. The device must be able to be linked to a team, and each stage of the relay should be identifiable. If the solution is a mobile app it should be available on both Android and iPhone. If the solution is an IoT device, it must have GPS and low band, while also being low cost.

The recommendation for a display medium should be an analysis which takes into consideration all external forces affecting the display of the race. It should serve the frontend with a Raspberry Pi, or a similar type of mini-computer. We have to take into account that there is no power on the site and the multiple precarious weather conditions that can occur, so things such as battery and a waterproof display are relevant. The solution should be low-cost hardware although no hard limit on the budget is set.

1.3 Target Audience

We can divide the people with interest in this project into three categories. Those who are reading this thesis, the people at Headit who will be users of the administrative panel and the people participating in the relay.

Thesis

The readers of this thesis would be anyone who has an interest in reading about the technologies used in this project. The thesis delves into subjects such as OAuth 2.0 authorization, the WebSockets protocol used in a modern ASP.NET REST API, cross-platform mobile application development with background location tracking with React Native and lastly web application development with React.

The thesis expects the readers to have an understanding of software development, programming and at least a small understanding of modern web frameworks and the structure of REST APIs.

Product

The first target group of users are the administrative users who will use the systems admin panel to manage relays. The second target group would be the participants who will mainly use the mobile application to track their location during the relay as well as the frontend service displaying the relay to see their teammate's location in the track and prepare for exchanges. The last target group is the audience of the relay who will have an interest in the same frontend service displaying the relay.

1.4 Group Background

The group comprises of Jørgen Eriksen, Elvis Arifagic and Markus Strømseth.

Academic Background

We are all students taking the bachelor's degree in programming at NTNU Gjøvik. The program focuses on a practical approach to programming with a wide range of topics from math specific subjects to cloud services and advanced programming. Relevant courses to this project specifically are Cloud Technologies, Mobile Programming and WWW-Technologies. Elvis Arifagic had the extra course called Robust and Scalable Systems which directly helps in creating the docker environment we are expected to create. Jørgen Eriksen previously studied a year of Web Design and is currently employed as a Full Stack developer at EC-Play AS. Elvis Arifagic and Markus Strømseth both attended the Game Programming course which provided a high knowledge of the C# language used in the development of the API.

Motivations

The main standing point for choosing this project was the broad tech stack that was described in the project description which we found interesting. Considering the team's familiarity with the technologies we were requested to use, we saw the opportunity to expose ourselves to more advanced and challenging aspects. As it is a full stack project the scope would give us more experience with projects closely mirroring how we would work in the industry. We highly valued the experience we would get from working with a consultant company based on the discussions and feedback we would have with them. In discussions with Headit when they visited NTNU we discussed ideas we had for how to solve the tasks and they were open to our suggestions for technological choices and methods of approach.

Keeping in mind the modularity of the project it fits well within our vision of how we wanted to approach working for the project. We wanted all team members to work together on every aspect of the project but at the same time allow for specialization in specific areas. We did not want to inflict rigid control of that dynamic and hoped that having a varied set of tasks open for everyone in the team at all times led to a fluent and dynamic approach to development.

1.5 Delimitations

Since Bike & Run is not an isolated event and knowing factually that there are many parties involved in the setup and execution of it, we will be catering to Headit's concerns specifically. Our sole focus will be to facilitate whatever Headit wants us to do even though there is a possibility that there exist other parties involved in this public event that might have conflicting interests. Our correspondence is with Headit only and discussions about software choices, design choices and others will solely be done with Headit.

The deployment and maintenance of the software were outside of our preview for this project. We are tasked with creating all docker files that are needed to deploy the software, but we will not ourselves do the actual deployment, as is specified in the project description given by Headit, seen in appendix A.

In discussion with Headit we decided that an approach should be taken where we seek to minimize server load as much as possible.

1.6 Project Goals

The overall project goal is to create a holistic software solution for managing tracking and displaying relay races with a mobile application for the participants and a display site for the spectators. Beneath is the list of specified result goals we wanted to achieve by the end of this project.

1.6.1 Result Goals

- To develop a backend service that can receive, process, store and distribute coordinate data, relay information and team information. The backend service must be expandable to allow support for different types of devices and further data such as pulse and heart rate.
- To facilitate the deployment of the backend and frontend services as a docker image.
- To develop a cross-platform mobile application with background location tracking. The mobile application must be identifiable by the team and relay stage.
- To develop a frontend service for displaying the relay in near real-time.
- To develop a frontend service acting as an admin panel for managing the relays and teams.
- To provide an analysis on what display medium would be best suggested for the solution.

1.7 Thesis Structure

In the first parts of the thesis, readers will find a glossary and acronym list that they should familiarize themselves with. This document is made by following the template as described here by NTNU[1].

- **Introduction** Defines the task that is meant to be solved. Also goes over the team and Headits motivations.
- **Requirements** Details the requirements of the whole system.
- **Development Plan** This chapter describes how we planned to work with the project and details choices made early on.

- **Development Process** For this section we go over how we worked with Scrum and how we executed the project plan.
- **Technical Design** This chapter discusses the technological choices and their motivations. It goes over the choices for all the different system modules.
- **Graphical User Interface Design** Here we describe the evolution of the design of the admin panel and the mobile app. We also discuss WCAG and universal design.
- **Implementation** This chapter goes over the fine details of what we implemented to achieve our result goals.
- **Testing** Here we discuss testing from a user testing perspective, software and performance tests.
- **Discussion** Is an analysis chapter where we discuss the project result, and method, reflect on technological choices and our experience working on the project.
- **Conclusion** Summarizes the project and goes over the result goals as well as further work Headit can do to improve the system.

Chapter 2

Requirements

When creating any type of product, a considerable amount of time ought to be dedicated to figuring out the requirements that are imposed on the design and the verification of the product. It helps give a complete picture of the project and provides a plan of action and keeps all interested parties on the same page.

In this section, we will look at the requirements for the Bike & Run project.

2.1 Constraints

Constraints were set in discussion together with Headit in early meetings and best attempts were made to consider these points strongly during the development.

Hardware/Software Constraints

- Reducing calls to the database was important to Headit because of the performance costs associated and the likelihood that the race will not be held where there is great internet connectivity at all times.
- As the app was supposed to be running in the background of phone devices, an effort needed to be made to ensure that it does not drain a lot of battery.
- Scalability needed to be considered during development and how the system was expandable post-deployment as Bike And Run might increase its amount of participating teams in the years to come.
- The API, admin panel and map display needed to be deployable to a docker environment.

Time Constraints

- The finished project plan and the signed work contract has to be delivered before the 31st of January 2022.

- User testing needs to be performed per the scheduled date and can not be changed due to people travelling far to come to the premises where it will be held.
- The finished product along with the finished bachelor thesis needs to be delivered before the 20th of May 2022.

Legal Constraints

- The overall software must conform to the General Data Protection Regulation [2].

2.2 Use Case

To help model the various ways a user can interact with the system, we used a Use Case Diagram to help us solve that problem. It breaks down the interaction into Actors interfacing with Systems. We used UML specialized symbols and connectors to then describe these relationships.

It was important to establish a clear understanding of the use cases as it helped scope the system, defined goals that actors can achieve and allowed us to figure out how the system interacts with users and external systems.

2.2.1 Use Case Diagram

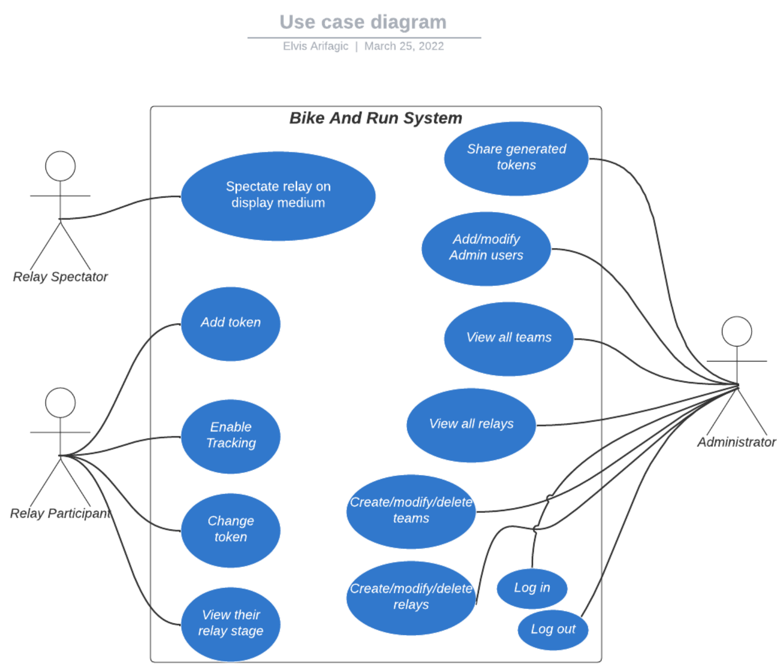


Figure 2.1: Use Case Diagram for the system.

2.2.2 Actors

Relay participant: Member of a team that participates in a relay, participating in one relay stage.

Administrator: Creates and manages relays.

Relay Spectator: The people who are viewing the relay and not participating in it.

2.2.3 Connection Between Use Cases and Issues

The figure below is not 100 percent accurate to how issues were split up because of size limitations. The core is there in regards to how we divided up the issues into multiple cards in Trello and how one issue evolved into another. Images of the Trello board can be viewed in figure 4.4. In Trello we had a total of 67 cards with 600 actions spanning all the cards, which includes all modifications to cards.

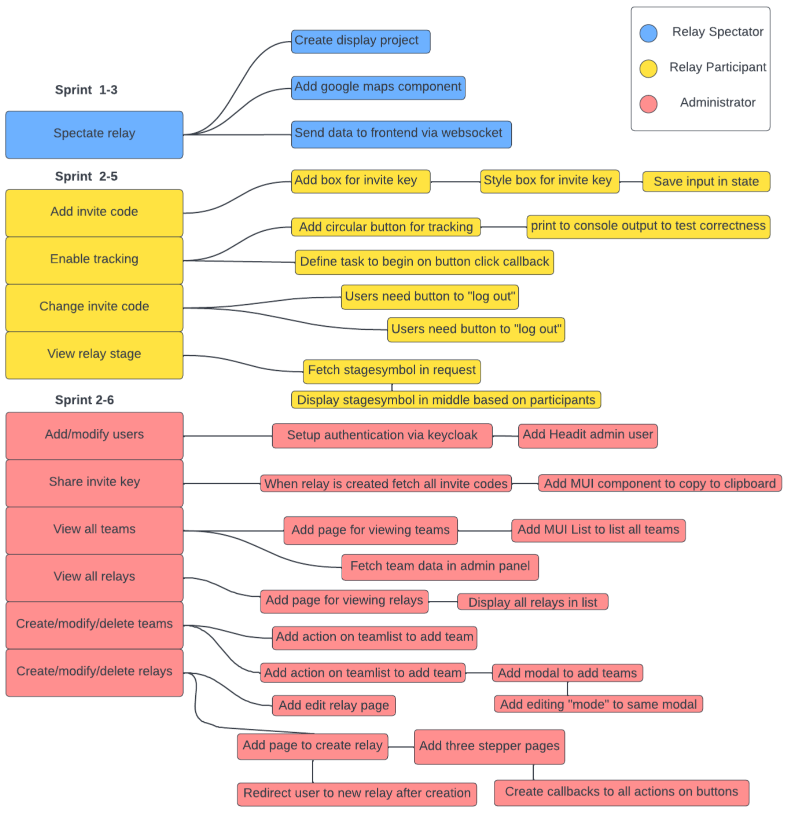

Relay Spectator
Relay Participant
Administrator
Sprint 1-3
Spectate relay
Create display project
Add google maps component
Send data to frontend via websocket
Sprint 2-5
Add invite code
Add box for invite key
Style box for invite key
Save input in state
Enable tracking
Add circular button for tracking
print to console output to test correctness
Define task to begin on button click callback
Change invite code
Users need button to "log out"
Users need button to "log out"
View relay stage
Fetch stagesymbol in request
Display stagesymbol in middle based on participants
Sprint 2-6
Add/modify users
Setup authentication via keycloak
Add Headit admin user
Share invite key
When relay is created fetch all invite codes
Add MUI component to copy to clipboard
View all teams
Add page for viewing teams
Add MUI List to list all teams
Fetch team data in admin panel
View all relays
Add page for viewing relays
Display all relays in list
Create/modify/delete teams
Add action on teamlist to add team
Add action on teamlist to add team
Add modal to add teams
Add editing "mode" to same modal
Create/modify/delete relays
Add edit relay page
Add page to create relay
Add three stepper pages
Redirect user to new relay after creation
Create callbacks to all actions on buttons

Figure 2.2: How the use cases manifested in our issue board.

2.3 Domain Model

The domain model was created as part of our figuring out more closely how to think about the problem space. Early meeting discussions with the product owner influenced how this diagram was structured such as the inclusion of the mobile as an entity. This breakdown heavily influenced how we set up the table relations of the database later on in the project.

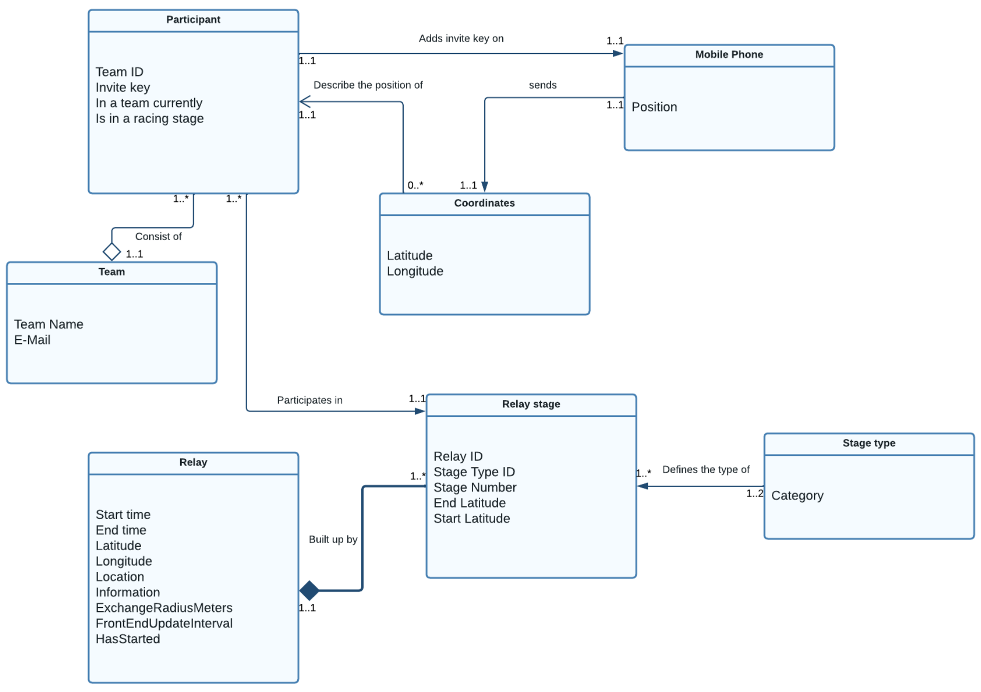


Figure 2.3: Domain space for the system.

2.4 Operational Requirements

Based on the project descriptions, project goals, discussions with the product owner as well as research done on the operation of such an event as Bike & Run we have set some operational requirements standards the software system should uphold. The biggest point of contention is that this is a yearly event which happens within a time frame of a day. So while the race is ongoing we wanted to strive to develop a system that would handle relatively high pressure over a short amount of time compared to other similar solutions.

- The system should support 80 teams and thereby 320 participants.

  The last event had 44 teams and 176 participants. To maintain a buffer with regards to performance we set the requirement to be able to handle almost double the amount of current participating teams to maintain the near real-time display considering the amount of data in transition during a relay.

- The solution should support push notification time intervals as low as 1 second per push.

  The admin panel allows for dynamically choosing the time interval of push notifications based on the scale of the event. Taking into consideration the previously mentioned operational requirement with regards to event scale the system needs to be able to handle low push notification time intervals to maintain near real-time race display.

- The solution should have fail-safes in place to prevent or notify admins of possible fails in the system.

- In the case of a crash the system's total reboot time frame should be between 5-20 minutes depending on the device used to run the program.

- Based on the fact that the application will not be used more than once a year. The user's low exposure to the application sets a high demand for the GUI in regards to clarity. A user should not have to spend more than 5 minutes figuring out how to accomplish a task.

2.5 Security Requirements

Security is a crucial part of all modern software and we have done our best to keep this in mind during the project. The system can be subject to attacks such as brute force and SQL injection so we had to be mindful of this and establish clear requirements to keep the system secure.

2.5.1 Functional Security Requirements

- Admin users have to be authorized and authenticated before they can access the administrator panel.
- In cases of an invalid login, there has to be a prompt that informs the user that the login attempt was invalid.
- An email verifying the creation of a user should be sent the first time an admin user is created.
- Admin users must be blocked from making requests where data is malformed/not filled out.
- The client-side must not store any information or pattern of use that can identify a user.

2.5.2 Non-Functional Security Requirements

- Any data that is received on the client-side should be verified according to a rule set before it's used anywhere in the application.
- Admin panel should not display restricted pages to users who are not authorized to view them.
- The admin site should display which admin user is currently logged into the system in the open browser.

Chapter 3

Development Plan

As we during this bachelors project both were responsible for development and operations in the DevOps environment. Determining what development methodology to use was integral for our success as it automates and integrates the two branches making time consuming tasks faster. Further on we will highlight our needs and preferences both with regards to development and operations.

3.1 Development Model

3.1.1 Determining Development Methodology

To remind the reader we want to briefly mention some important points that were crucial in determining the method for development.

- The product owner's wishes to be heavily involved in the process and discussion. Thereby requesting weekly meetings.
- There were few bounding frames that limit what and how we could implement the software.
- The development time frame was shorter due to having to write our thesis in the same time span as developing the system.
- The tech stack we decided to go with would challenge us and surely bring unsuspecting obstacles, making the development process highly unpredictable at times.
- The small size of our development team with only three members.
- Previous experience with different software development methodologies.

Keeping the above-mentioned points in mind it was clear to us that we needed to employ an agile methodology to deliver on this product. Seeing as we were not limited by very rigid hardware, such as in embedded development as well as Headit being very open to technological discussions. Not to mention we were working with some technologies and frameworks we did not have much previous experience with. We needed to be able to adapt to the situation as it was changing. Intermediate to large software projects change a lot during development as the needs of the stakeholder change. Therefore, having linear sequential steps, working non-iteratively such as the waterfall model would weaken our ability to deliver the best product we can.

Working in a way where previous steps must be completely done, and only if they are done do you move on to the next part, did not suit this project because of the ever changing flow of the development process. A model such as Scrum where we work with analysis, design, coding and testing which then leads to a discussion with Headit was a much cleaner and more open way to work and get to the best result possible.

3.1.2 Agile Development

As there are two major contenders when it comes to agile development, Kanban and Scrum. We looked at their differences and choose the optimal methodology both for our project and us as a team.

Kanban puts a high focus on visualizing tasks as well as limiting how much time is spent on each task. Tasks, presented as cards are visualized and organized on a Kanban board and flow through workflow stages. Whereas the most common ones are To Do, In Progress, In Review, Blocked and Done. The Kanban philosophy is “release when ready”, which entails not following a timeline with regards to task due dates such as Scrum’s sprint reviews. This provides a continuous flow during development which in turn brings both positive and negative aspects with it. On one hand it does not hold a team back if planning is too lenient with regards to time allocation towards tasks having to wait on scheduled meetings to progress. On the other hand it does not provide any inherent structure to the workflow which in turn can be straining on a team working on a project over a long period of time [3].

Scrum is a structured yet agile form of development methodology. Breaking up work into smaller increments and puts a high focus on a scheduled time frame of promised release of tasks. Scrum is built on empiricism and with focus on learning from previous experiences to improve future workflow. The entire project is divided into rigidly time framed sprints which usually lasts from one to four weeks with included scrum ceremonies such as sprint planning, sprint review and sprint retrospective. This way of working forces larger complex tasks to be split up into smaller ones. With clear roles these smaller tasks are prioritized and delegated between all developers. The role of product owner is responsible for prioritizing tasks with regards to the customers requirements. The role of scrum master focuses on keeping the team grounded in the scrum principles and uphold these. Whereas the role of the development team is collectively responsible for delivering work in the given time frame [4].

As aforementioned our final decision was using Scrum as our development methodology. We decided a more structured methodology would be necessary to work tirelessly with one project over a longer period of time, something we have not had much experience with during our studies so far. Also to keep communication within the team and between the team and other parties with regards to working remotely large periods of the semester.

3.2 Project Organization

3.2.1 Roles and Responsibilities

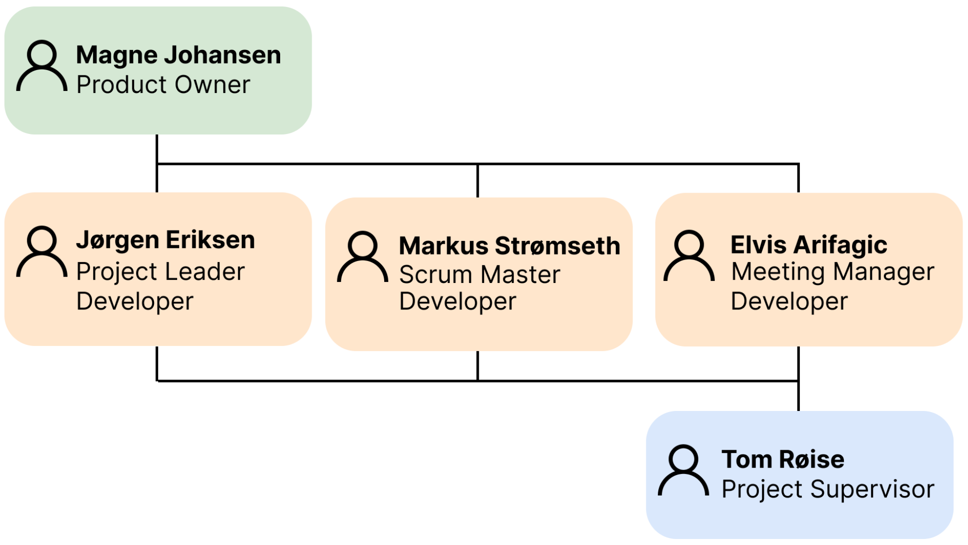


Figure 3.1: All parties and their roles in the project.

Product Owner: Represents Headit. During our weekly meetings the product owner would provide us with their needs and preferences for the end product, information regarding the overall completion of the relays as well as guidance regarding the development process.

Project Supervisor: Was responsible for overseeing the overall progress of both development and workflow. During our weekly meetings we would provide a status report and discuss our thought process regarding the development process. The project supervisor would be highly responsible for feedback regarding our final thesis.

Project Leader: Was the groups main representative and had veto in group decisions whenever any conflict were to arise. The team leader would have the main responsibility for assessing and maintaining progress.

Scrum Master: Was tasked with the planning, completion and assessment of sprints and thereby leading the biweekly sprint meetings. Scrum Master would also be responsible for quality assurance of issues.

Meeting Manager: Was tasked with taking meeting minutes and making sure that notes were taken for all meetings with the project supervisor, product owner and internal meetings.

3.3 Routines and Rules

3.3.1 Meetings and Schedules

As a part of our team's routine we planned to set up a minimum of two meetings per week. First meeting would be with the product owner every Tuesday morning. These meetings would comprise of sprint planning, sprint review and sprint retrospective biweekly. The second meeting would be with the project supervisor on Thursday mornings to discuss progress and possible tribulations.

With regards to internal meetings, especially daily stand ups we decided they would be unnecessary. There was a period during sprint 4 where we were all worked remotely and communication was halted. This was brought up as a concern during the sprint retrospective. We did not however take any measures as for the coming sprints we would meet daily for work at campus.

Monday	Tuesday	Wednesday	Thursday	Friday
Sprint planning Development	Development	Development	Status meeting Development	Development
Status meeting	Development	Development	Development	Development
Sprint review and retrospective Development	Development	Development	Status meeting Development	Development
Status meeting	Development	Development	Development	Development

Table 3.1: Our meeting schedule over a 4 week time period.

3.3.2 Group Decision Making

Group decisions with regards to the full development and project process will be decided based on democratic elections. With a final veto vote belonging to the

Team Leader as previously mentioned in Section 3.2.1.

3.4 Gantt Diagram

We devised a long term plan for the entire project with a total of 6 sprints and a period for writing the final thesis. For this we created a Gantt chart which is a type of bar chart which details a projects runtime and planned tasks. During this project the only noteworthy deviation from the Gantt chart was that sprint 4 was extended by one week.

Furthermore we planned for an initiating phase and some competence training during planning to get every member up to speed with .NET and the React and React Native frameworks as there were some discrepancies within the group. Adding this explicitly into the plan reassured us that the time would not have to be spent during the development process.

Roughly around the middle of the project period we planned for a user test session with Headit to get valuable feedback mostly on the frontend services of the system for further incorporation in later iterations.

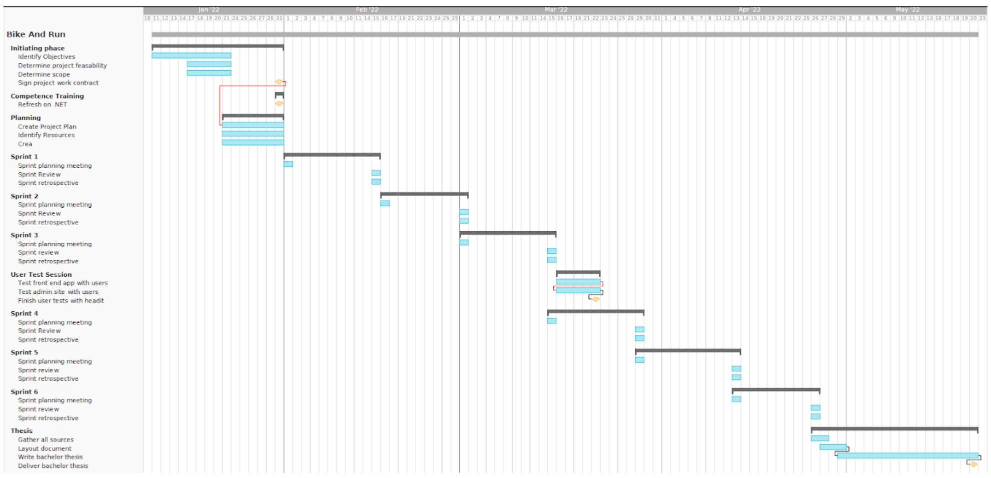


Figure 3.2: Shows the long term plan set forth early in the project.

Chapter 4

Development Process

In this part of the thesis, we will explore the software development process, looking more closely at our experience with the different facets of an agile scrum approach. We will look at the sprints and tools we used and how we interfaced with both our supervisor and our product owner.

4.1 Scrum Board

Trello is a tool that most of the team is familiar with so setting up the environment for our scrum board was an efficient process. We structured our scrum board with categories best representing our workflow shown in the figure below.

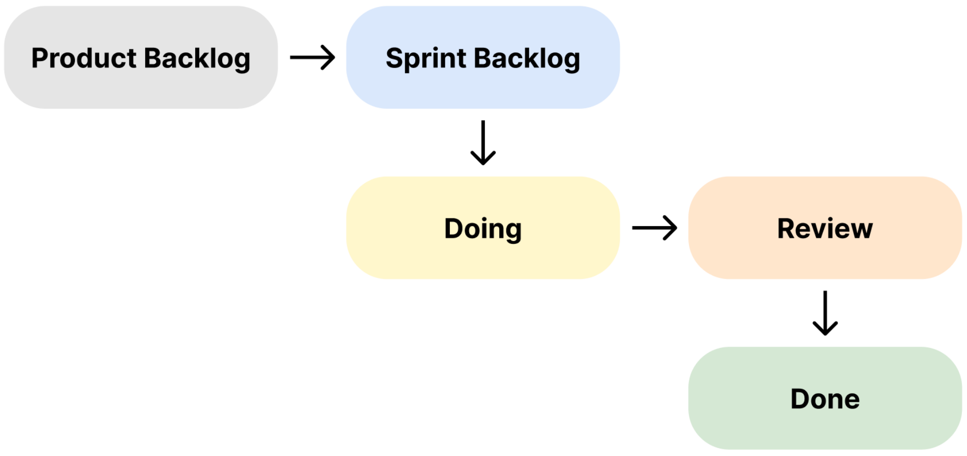
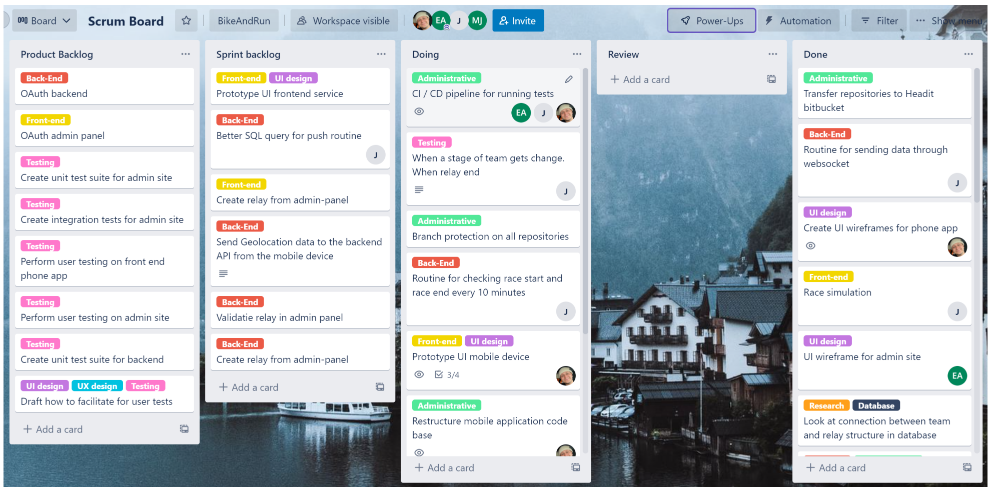


Figure 4.1: The structure of our Scrum Board

The product backlog represented all future tasks and was filled with issues in the planning period. The product backlog was expanded during development with occurring issues. The product owner had ownership of the product backlog. During sprint planning meetings, issues were moved from the product backlog to the sprint backlog based on a collective discussion internally in the team based on feedback from the product owner. Issues were then delegated to a team member and followed the life cycle based on a team member's workflow under the categories doing, review and done.

We highly prioritized having a review system to maintain quality assurance during development. With issues in the review section having to go through peer-reviews before they were considered done.

We decided to keep the categories for our workflow rather short, simple and standardized, as they best represented how we wanted to work. During the project, we never felt the need for any further categories to define the state of issues. We also experienced that the simplicity led to an easier overview of the board and thereby limiting the time spent in Trello as well as putting a focus on the review tabs size. This meant doing reviews of other team members' work became a fluent task and kept the review tab from overfilling.



Product Backlog
- Back-End — OAuth backend
- Front-end — OAuth admin panel
- Testing — Create unit test suite for admin site
- Testing — Create integration tests for admin site
- Testing — Perform user testing on front end phone app
- Testing — Perform user testing on admin site
- Testing — Create unit test suite for backend
- UI design, UX design, Testing — Draft how to facilitate for user tests
- + Add a card

Sprint backlog
- Front-end, UI design — Prototype UI frontend service
- Back-End — Better SQL query for push routine — J
- Front-end — Create relay from admin-panel
- Back-End — Send Geolocation data to the backend API from the mobile device
- Back-End — Validatie relay in admin panel
- Back-End — Create relay from admin-panel
- + Add a card

Doing
- Administrative — CI / CD pipeline for running tests — EA J
- Testing — When a stage of team gets change. When relay end — J
- Administrative — Branch protection on all repositories
- Back-End — Routine for checking race start and race end every 10 minutes — J
- Front-end, UI design — Prototype UI mobile device — 3/4
- Administrative — Restructure mobile application code base
- + Add a card

Review
- + Add a card

Done
- Administrative — Transfer repositories to Headit bitbucket
- Back-End — Routine for sending data through websocket — J
- UI design — Create UI wireframes for phone app
- Front-end — Race simulation — J
- UI design — UI wireframe for admin site — EA
- Research, Database — Look at connection between team and relay structure in database
- + Add a card

Figure 4.2: Screenshot from our Scrum Board during sprint 3.

In Trello, we created cards, which are the equivalent to something like Jira or Github issues on an issue board. On the card, itself was where we got more specific with what type of issue it was and added additional context to an issue. Our issues were categorized with the following labels shown in the table below.

Label Name	Usage amount
Documentation	5
Front-end	22
Research	4
Bug	3
UI design	6
Backend	16
UX design	3
Administrative	8
Report	10
Testing	8
Database	5

Table 4.1: Overview of all labels and their amount of usage.

For the Scrum board, we wanted to have a fine granularity concerning the categorization of labels. This made issues more descriptive and minimized time spent on reading through them to attain the needed understanding for tackling the task.

4.1.1 Issue Scoping

We started the project off by having all-encompassing issues concerning complex parts of the system which in turn made some issues too large to be finished in a single sprint and therefore flowing into the next sprint. We decided not to use any time estimation method, this is discussed in section 9.2.4. After attaining more experience with the scope of each task, issues were divided and scaled down to fit the schedule better. We did not experience that not applying any time estimation methods to issues halted our process but rather forced us to make concise issue headers with a more complementary description which in turn made issues clearer.

4.2 Sprint Planning

Sprint planning meetings were held before each sprint. This meeting was used to define the sprint goal and also an internal team discussion on which issues were of high priority and therefore chosen from the product backlog to be assigned to the sprint backlog. In the beginning, we defined our issues ourselves because we were not experienced with all the facets of Scrum, as we were not aware that the product owner was meant to create issues for us. This was corrected for sprint two and onwards where we brought the product owner on for those discussions where we created, moved and deleted various tasks as per his instruction.

4.2.1 Issue Delegation

During sprint planning meetings together with the product owner, we assigned issues to the sprint backlog. This was done based on each team member's preference following a pick-and-choose structure to ensure all team members were motivated for their tasks and could finish them for the sprint. Each team member naturally got an overarching responsibility of a certain area of the system which then in turn shifted the issue delegation more towards unspoken ownership of issues belonging to that part of the system.

4.3 Summary of Sprints

4.3.1 Sprint Length

At the beginning of the project period, we knew that our sprints needed to be two weeks long because of the size of the project. We wanted each time that a sprint ended to have a decent amount of work done and we wanted to decrease the bloat in documents related to meetings as well. Sprints, in the beginning, started on Thursdays, lasted for two weeks and ended on Thursdays. We found that to fit the schedule of the product owner which we wanted to be present, we needed to adjust it to start and end on Tuesdays.

4.3.2 High-Level Overview of Sprints

In this small section, we will give a brief overview of the contents of each sprint, the outcome of the sprint and how it aligned with the stated sprint goal for that sprint.

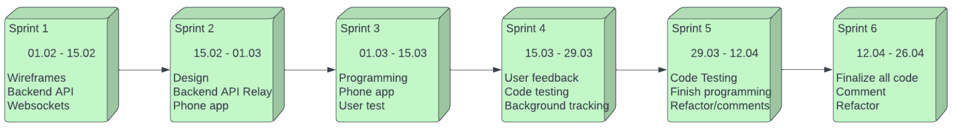


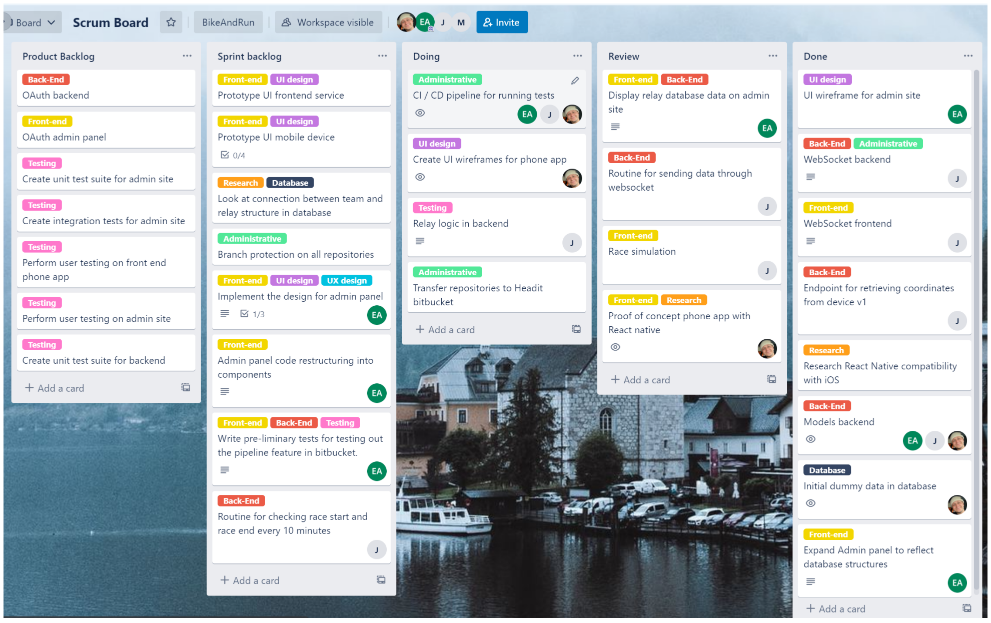
Figure 4.3: Diagram showing our Sprints and overview of what was done.

Sprint 1

In the first sprint, most of the work was done on wireframing the frontend for the administrator panel. This was done using Lucidchart and was mostly done as part of group brainstorming sessions. Another part was laying the foundation for the API. This was created using the patterns MVC [5] and Dependency Injection [6] as guides in setup. After this basic structure was set up we created the proof of concept communication of WebSockets between the API and the Map Display with a simple message sending routine.

Sprint 2

Defined the database design alongside our product owner. Created the basic GET and DELETE endpoints for all the basic data in the API. Implemented the design of the admin panel and its basic routing functionality alongside implementing some of the designs for the site. The admin panel also had the basic blocks added for viewing data present in the database. Created the phone application project and added the beginnings of background location tracking.



Product Backlog

- Back-End — OAuth backend
- Front-end — OAuth admin panel
- Testing — Create unit test suite for admin site
- Testing — Create integration tests for admin site
- Testing — Perform user testing on front end phone app
- Testing — Perform user testing on admin site
- Testing — Create unit test suite for backend

Sprint backlog

- Front-end, UI design — Prototype UI frontend service
- Front-end, UI design — Prototype UI mobile device (0/4)
- Research, Database — Look at connection between team and relay structure in database
- Administrative — Branch protection on all repositories
- Front-end, UI design, UX design — Implement the design for admin panel (1/3) EA
- Front-end — Admin panel code restructuring into components EA
- Front-end, Back-End, Testing — Write pre-liminary tests for testing out the pipeline feature in bitbucket. EA
- Back-End — Routine for checking race start and race end every 10 minutes J

Doing

- Administrative — CI / CD pipeline for running tests EA J
- UI design — Create UI wireframes for phone app
- Testing — Relay logic in backend J
- Administrative — Transfer repositories to Headit bitbucket

Review

- Front-end, Back-End — Display relay database data on admin site EA
- Back-End — Routine for sending data through websocket J
- Front-end — Race simulation J
- Front-end, Research — Proof of concept phone app with React native

Done

- UI design — UI wireframe for admin site EA
- Back-End, Administrative — WebSocket backend J
- Front-end — WebSocket frontend J
- Back-End — Endpoint for retrieving coordinates from device v1 J
- Research — Research React Native compatibility with iOS
- Back-End — Models backend EA J
- Database — Initial dummy data in database
- Front-end — Expand Admin panel to reflect database structures EA

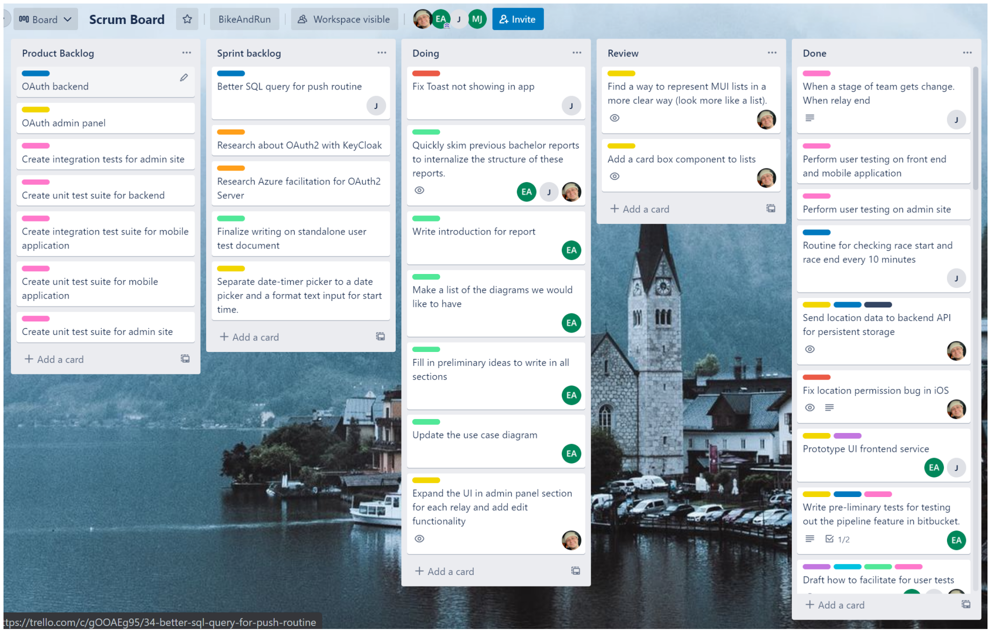
Figure 4.4: Screenshot from our Scrum Board during Sprint 2.

Sprint 3

For this sprint, we focused a lot on welding the separate pieces together. We worked on having the whole pipeline of communication be a coherent piece. The mobile application was able to track location in the background and sent it to the API. The API used WebSockets to send the location data to the frontend which in turn displayed the coordinate as a marker on a Google Maps display. A user test was also performed during this sprint where we visited Headit's offices and tested the admin panel and phone app on four users.

Sprint 4

The admin panel was redesigned in this sprint following the feedback received in the user test. The API had more endpoints implemented as different types of data were needed such as retrieving data about participants by relay id. Integration tests were also written for the API which included the usage of 2-4 components of the API together to test. Unit tests were also written for isolated functions that received some data such as checking if a participant is inside a radius.



Product Backlog
- OAuth backend
- OAuth admin panel
- Create integration tests for admin site
- Create unit test suite for backend
- Create integration test suite for mobile application
- Create unit test suite for mobile application
- Create unit test suite for admin site
- + Add a card

Sprint backlog
- Better SQL query for push routine
- Research about OAuth2 with KeyCloak
- Research Azure facilitation for OAuth2 Server
- Finalize writing on standalone user test document
- Separate date-timer picker to a date picker and a format text input for start time.
- + Add a card

Doing
- Fix Toast not showing in app
- Quickly skim previous bachelor reports to internalize the structure of these reports.
- Write introduction for report
- Make a list of the diagrams we would like to have
- Fill in preliminary ideas to write in all sections
- Update the use case diagram
- Expand the UI in admin panel section for each relay and add edit functionality
- + Add a card

Review
- Find a way to represent MUI lists in a more clear way (look more like a list).
- Add a card box component to lists
- + Add a card

Done
- When a stage of team gets change. When relay end
- Perform user testing on front end and mobile application
- Perform user testing on admin site
- Routine for checking race start and race end every 10 minutes
- Send location data to backend API for persistent storage
- Fix location permission bug in iOS
- Prototype UI frontend service
- Write pre-liminary tests for testing out the pipeline feature in bitbucket. 1/2
- Draft how to facilitate for user tests
- + Add a card

tps://trello.com/c/gOOAEg95/34-better-sql-query-for-push-routine

Figure 4.5: Screenshot from our Scrum Board during Sprint 4.

Sprint 5

In this sprint, we worked on finalizing all the features in discussion with the product owner. Past this point, no new big features would be implemented. Work on OAuth2 was started in this sprint with discussions being had with the product owner about the solution to use. To which we agreed on KeyCloak. We ran performance tests. The admin panel had a redesign to give it more colours and improve the UI even further.

Sprint 6

OAuth2 was finished on both the API and admin panel where we added a simple login page and a settings page to log out and view who you are logged in as. More tests for the controller endpoints were written for the API. Clean up of the codebases using both tools and manual cleaning was performed.

4.3.3 Release Increments

We decided early on that following the rigid structure of Scrum's release increments was not beneficial for our case. Rather we decided on having a continuous flow of releases after pull requests were reviewed to ensure all team members would be as up to date with the code base as possible at all times.

4.3.4 Sprint Review

Sprint reviews were done at each end of the two-week sprint cycle. This was a discussion type event that involved the product owner. We had discussions about what was completed during the sprint and we adapted the product backlog based on what we had discussed. In cases where we had demos to show something, we would use this time to demo the state of the system during these meetings.

During sprint three, the stakeholder invited another employee at Headit to observe the demo that we had, it was an open invitation for people to join these sessions.

4.3.5 Sprint Retrospective

In sprint retrospectives, we focused on the process itself of working on the project. We also had the product owner present for these as he had expressed interest in listening to us discuss these issues and hearing how we planned to solve them.

One example of this is how we at the end of sprint two felt disconnected. We discussed solutions for this and came to the conclusion that becoming more active on the scrum board and also being more descriptive of issues were the solution. We adapted our approach such that issues were more clear and more expressive. Adding specific issues in cards, updating them and following up on cards with comments were some of the specific actions we took.

Chapter 5

Technical Design

In this section, we will give a high-level overview of the technologies that we used to solve the task assigned to us by Headit as well as look at the systems architecture. Brief overviews of the modularity of the systems will be shown as well as descriptions of each technology.

5.1 System Architecture

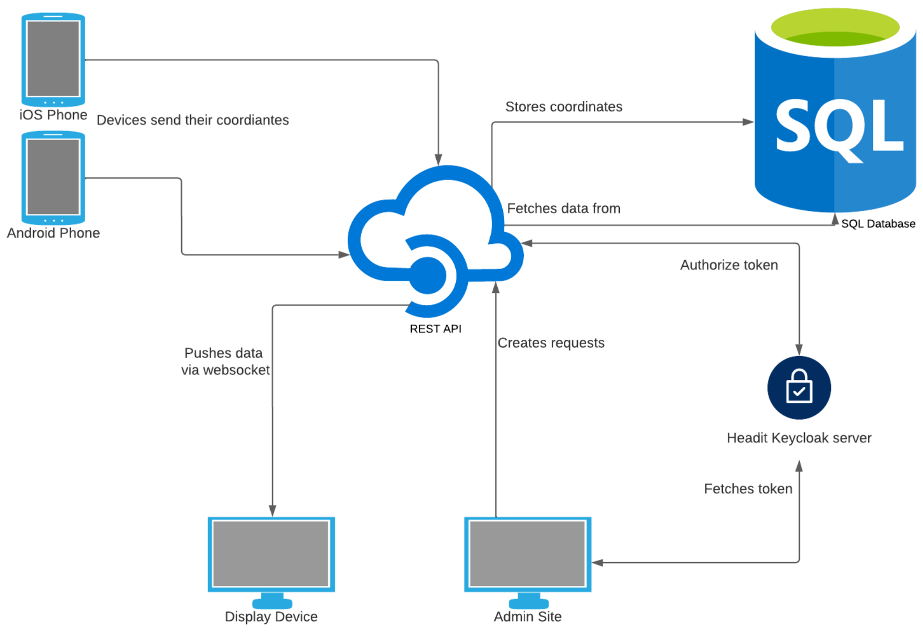


Figure 5.1: All of the main modules of the bike & run

The system is divided into four main parts which are the API, the mobile application for both Android and iOS devices, the Admin Panel and the Map Display.

5.1.1 Authorization

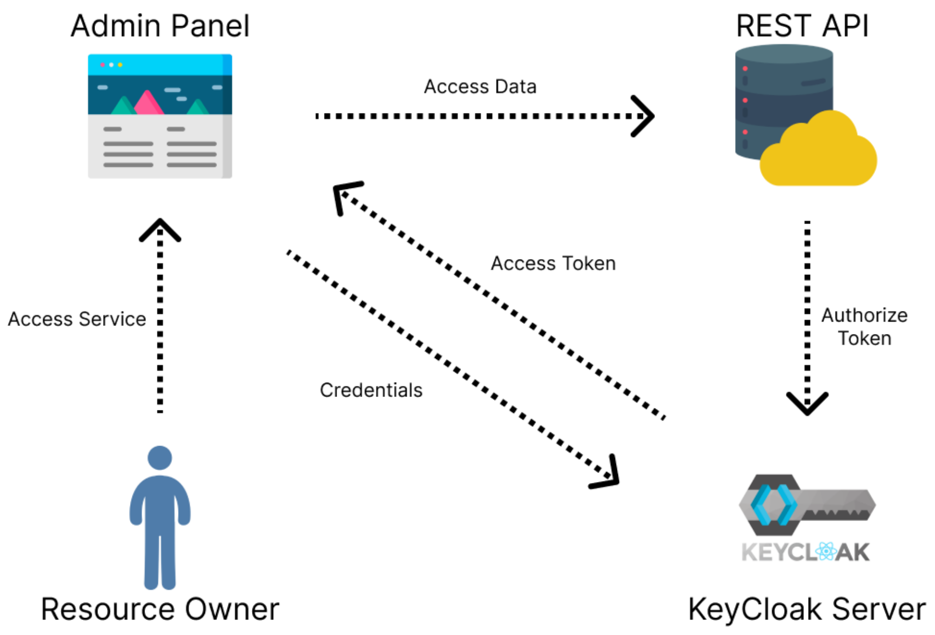


Figure 5.2: OAuth2 with Keycloak

The admin panel gets an access token from the Keycloak server in exchange for a username and password. The access token is used for all requests from the Admin Panel to the API, which the API authorizes through the Keycloak server.

The mobile application authorizes by an invite key which serves as a token when sending requests to the API.

5.2 Technology Overview

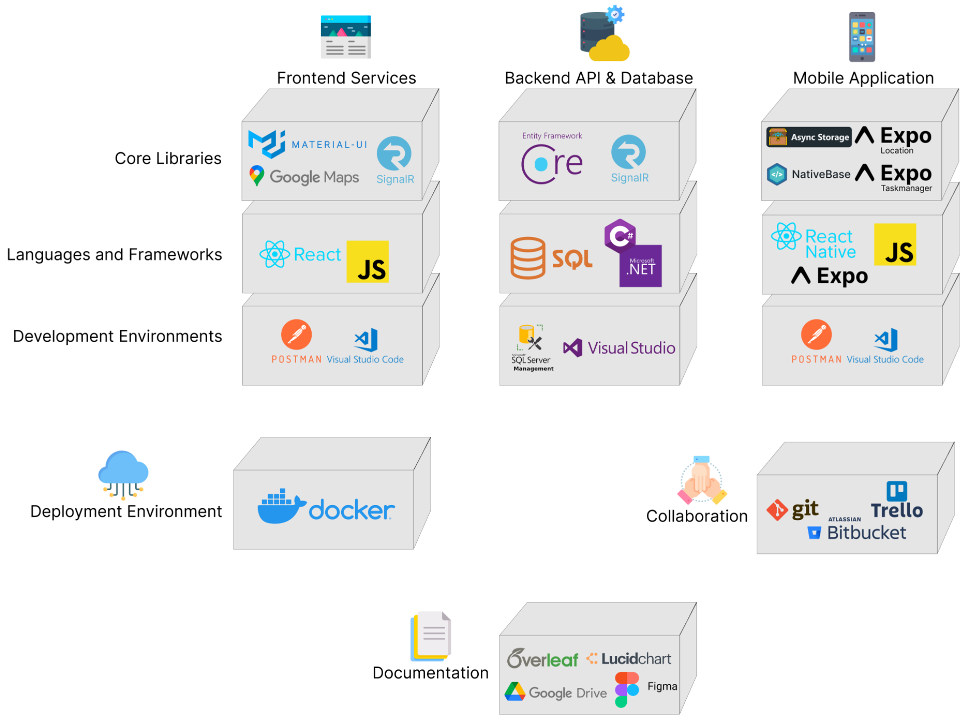


Figure 5.3: All the various technologies used by the team

5.3 Admin Panel and Map Display

For modern browsers we need three different types of tools to enable dynamic, descriptive, and aesthetically pleasing websites; those are HTML, CSS, and JavaScript. HTML and CSS are known as markup languages that purely describe how a document should look, while JavaScript is a programming language which enables the site to be dynamic and respond to user actions. Since its debut in 1996 JavaScript is used by 95 percent of websites which makes it the most popular of its kind [7].

From 2010 and onwards the web development space saw a large influx of new JavaScript frameworks and libraries such as Ember, Angular, React and Vue[8]. These were created with the idea to make developers' lives easier by reducing the complexity of managing state on websites. By not using a framework we would have had to deal intimately with the DOM and simple tasks would suddenly be much more difficult to execute. We have offloaded a lot of grunt work such that we can be freed up to focus on harder tasks.

5.3.1 Navigation Overview for the Admin Panel

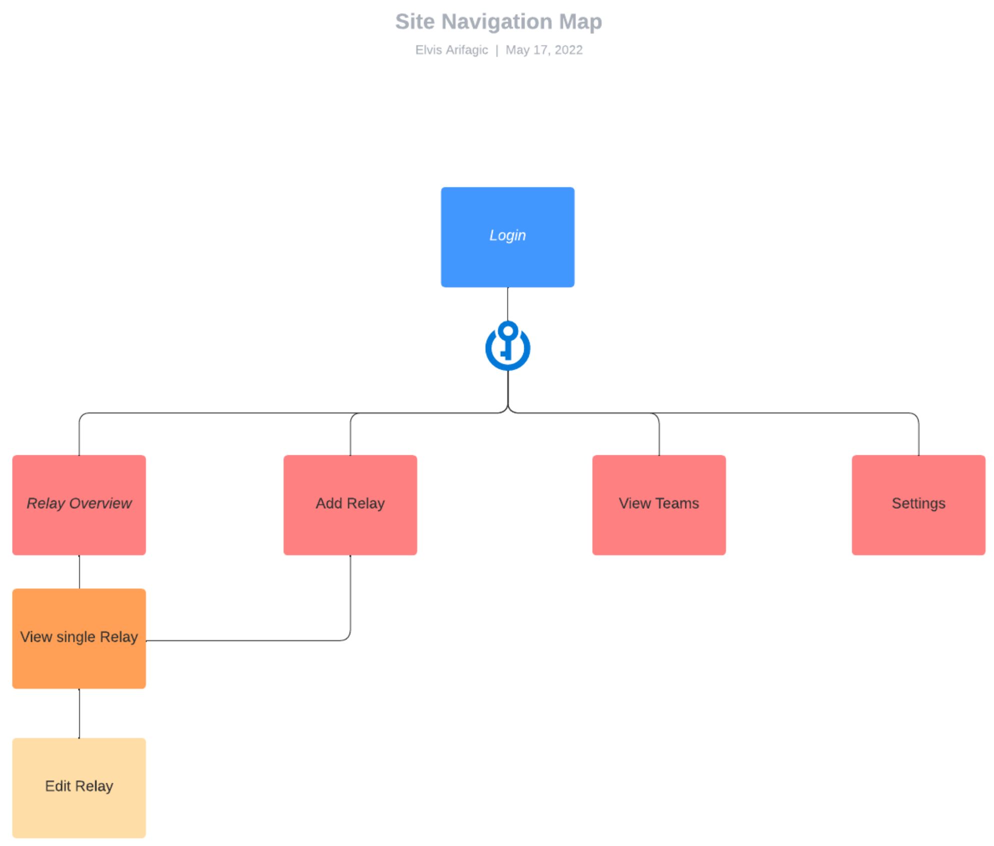


Figure 5.4: Navigational overview for the admin panel website

5.3.2 JavaScript Library

We wanted a library that was mature with good documentation and had a vast ecosystem of supporting libraries. It was also important to us that it would be easy to use, easy to test and performant.

The Team Leader has previous working experience with React powered websites. Going with React, in this case, meant that we could iterate swiftly, avoid common bugs (as the likelihood he would have seen them before is high), and rely on known conventions concerning system structure and what makes for quality React-based code. React also fulfils other criteria by having vast documentation and serving an easy-to-use template out of the box for a base React project.

5.3.3 Material UI

To help us with some stylized components such as buttons and lists we used a component UI library. These libraries come with stylized UI components out of the box that can be used in their applications straight away. We choose to use material UI because of its large library of components. As well as the fact that it follows Google's Material Design UI which is a system describing how one is to build high-quality UI experiences as defined by Google [9]. Component libraries are also good for consistency as they are created with this in mind by developers. This is expanded upon in Section 6.1.3

5.3.4 Map API

To provide a map layout for the frontend service for displaying the race we needed a Map API as creating our own from the ground was ruled out considering the scope and timeframe of the project. When choosing a provider for the map component, we looked into Mapbox, OpenStreetMaps, and the Google Maps API.

Mapbox was the one that was the most adaptive for customization, which makes it possible to shape it in the way that we want [10]. For the expected use Mapbox would not have posed any economical problem with 50,000 map loads each month for free [11]. The downside was that it was too extensive and unnecessarily advanced for the simplicity that we needed when displaying the relay on the map, which made us shy away from it as an option.

OpenStreetMaps (OSM) was, on the other hand, simple. It had the customization needed for the project and was also free. A downside was that excessive queries with the OSM API would result in getting blocked which could have happened quickly if multiple relays were running at the same time.

Ultimately we decided on using Google Maps API as it was simple and allowed for the necessary customization needed for the project. Google Maps API had a very detailed map and could convert geolocation to a street address and vice versa [12].

Since Google Maps is the most popular mapping app [13], it is fair to say that it has the most familiar map design, which makes the argument that it is the most friendly regarding user experience [14]. Google Maps API provides 28,500 map loads per month for free which also fits well within the expected use.

5.4 Map Display and SignalR

For handling WebSockets in the Map Display service, we used the package @microsoft/signalr for retrieving the data [15]. The reason we chose @microsoft/signalr was that we used SignalR for WebSockets in the API. The npm package is maintained by Microsoft and they are made to be used together. SignalR is also discussed in section 5.5.4

For the map display, we went with React to have the codebases be similar between the Admin Panel and the Map Display services. During cases of maintenance later on having continuity like this will scale well into the future. We used MUI for the components as well.

5.5 Backend

The API was known early in the process to be the centrepiece that would integrate all the different parts of the system. In discussions with Headit we arrived at going for .NET as our backend solution for several reasons. As per the project description, we wanted to have a solution that used WebSockets and Microsoft had as part of .NET an industry-standard library that was created for such cases. The framework allowed our application to be able to run on any platform from a Windows server to a docker container. The framework is very performant compared to Node.JS or Java Servlet or Python frameworks Django and Flask [16].

5.5.1 .NET Framework

The .NET framework is a software framework which at its core is run by the Common Language Runtime more commonly known as the CLR. The CLR is implemented as an application virtual machine, it is created when the applications start and are destroyed on exit. It is also sometimes called a process virtual machine. Some of the readers might be more familiar with the JVM. You can think of the CLR as the same thing. Giving you the same benefits.

Using techniques such as this you get a platform-independent programming environment and since programs are not run on the underlying hardware or operating system, programs can execute deterministically on any platform.

The framework also comes with the Framework Class Library (FCL) which is a library that is created by Microsoft supporting a host of different common types of libraries. FCL has libraries for creating UI, cryptography, web application development, database communications and network communications.

5.5.2 Backend Modules

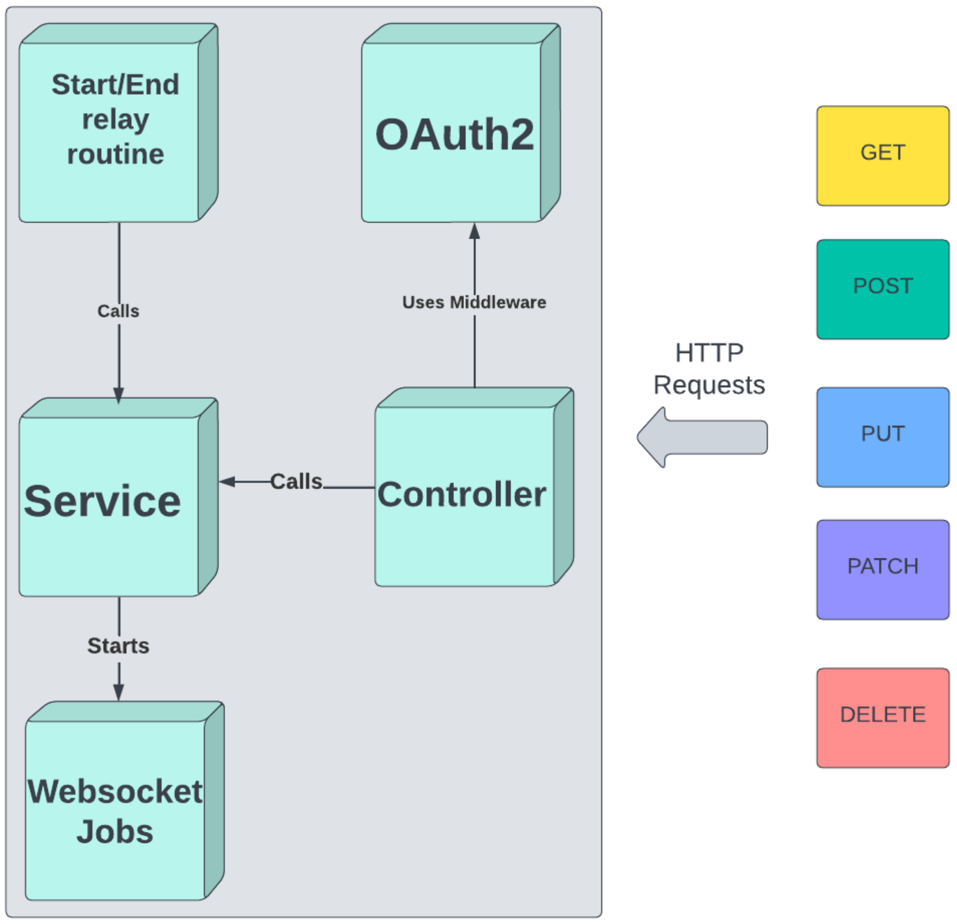


Figure 5.5: Modules that make up the backend REST API.

5.5.3 Entity Framework Core

EFC or Entity Framework Core is an object-relational mapper which allows for communication with a relational database using .NET objects. It reduces complexity because it strips away much of the manual data access code that is needed to communicate with SQL databases in C# for example. By using a tool such as this we can define our models in C# code and use EFC to generate the database for us based on attributes we put on the data. Using EFC is how we primarily communicated with the database through code in this project.

5.5.4 SignalR

This is a library that we used to create and use the WebSockets for bi-directional communication between the Map Display and the API. SignalR is created by Microsoft and is the standard for real-time .NET applications that need high-frequency updates. We chose this because of its already established integration with .NET and very good documentation and example projects that were available on the internet.

5.6 Mobile Application

In this section, we will go over different frameworks and how we chose to use React Native for the project. We will also discuss central libraries and tools that we used for the mobile application.

5.6.1 Choosing Mobile Framework

As per the project description, we were tasked with developing a mobile application supporting both iOS and Android. We could have made a native iOS and a native Android app, but that would have taken more time to develop than developing using a cross-platform framework, as it would have been two code bases instead of one. Also having one code base would be easier to maintain and make changes to. An advantage of developing native apps is that the app has better performance, and higher security [17]. But since the app did not require high performance, nor handle sensitive data that needed to be secure, it made more sense to choose a cross-platform framework.

Flutter is an open-source UI SDK created and supported by Google which uses the Dart language. We highly considered using Flutter as it has risen in popularity in the past years and considered the experience would benefit us going into the job market. What concerned us was that Headit would be unable to assist us with issues and it would create difficulties in maintenance [18].

Cordova uses HTML, CSS and Javascript, and wraps it in a container that renders the web app as a native app [19].

Ionic is an open-source SDK originally built on top of AngularJS and Apache Cordova for developing hybrid mobile applications. It supports coding in Angular, React and Vue [20].

5.6.2 React Native

React Native is a framework that would allow us to use React alongside the native platform capabilities of the devices. It is created and maintained by Facebook.

This framework was chosen to create the mobile application. It allowed for the codebases of the websites and the mobile app to have similar code so the jump from developing the different parts of the project would not be such a hard jump for the team. Another factor to choose React native was its popularity. It is much more popular compared to Ionic [21]

5.6.3 Expo

The mobile application was built with Expo as the development environment to avoid releasing the application on Google Play or App Store for testing the application on mobile devices. Expo provides a preset SDK with commonly used APIs like basic view components, images, camera access, notifications, device info, and more [22].

To better understand our experience with developing the mobile application we need to establish how the Expo CLI works and its two core systems for running the application in the Expo Go app on the local device. The Expo Development Server works to serve the Expo Manifest to the application and provides a communication layer between Expo CLI and the Expo Go application on your phone or simulator.

Expo Manifest and Metro Bundler

The Expo Manifest serves as the application's configuration description and as a pointer to the bundle URL on the local development server which contains our JavaScript code [23].

The Metro Bundler compiles all of our JavaScript code into a single file and translates any JavaScript code that we wrote which isn't compatible with our phone's JavaScript engine. JSX, for example, is not valid JavaScript; it is a language extension that makes working with React components more pleasant and it compiles down into plain function calls [24].

5.6.4 Background Location Tracking

We decided to use Expo's packages to enable background location tracking on a React Native Application. It featured all the necessary parts we needed and it supported both platforms iOS and Android.

5.6.5 Nativebase

Nativebase is a component library that is for React and React Native applications. It is a popular library that is often used as an accompanying library to Material UI. This made it a clear choice for us as well as having very good documentation and a large example set for which we could draw ideas on how to solve problems.

5.7 Database Design

In early meetings with the product owner, discussions were had about database design seeing as there were strict requirements related to the storage of coordinates. Mobile as an entity was decided not to be included because it brought no value to the diagram. We did not want to store Device or Hardware IDs to not violate any GDPR protections [25]. GDPR was also a concern that we had in mind when designing the rest of the tables. In regards to table quality, we checked for redundant data in the tables using normalization techniques. To verify correctness we tested queries against the tables to verify whether the data we wanted was retrievable.

We used an Azure SQL database for this project. This was only for development, Headit will use its database solution which the API can easily be connected to. The fundamental schema does not change.

5.7.1 Entity Relationship Diagram

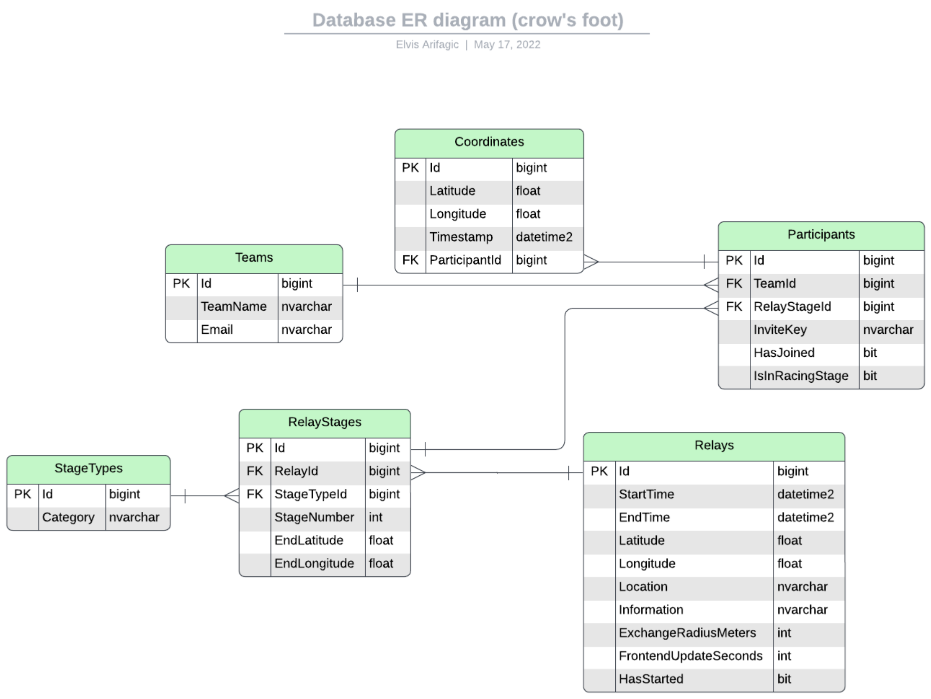


Figure 5.6: All of the entities in the bike & run database

Chapter 6

Graphical User Interface Design

In this part of the thesis, we will look at the various designs that were created for the mobile application, admin panel and map display. We will show examples from wireframes to the result and talk about responsive design, universal design and WCAG.

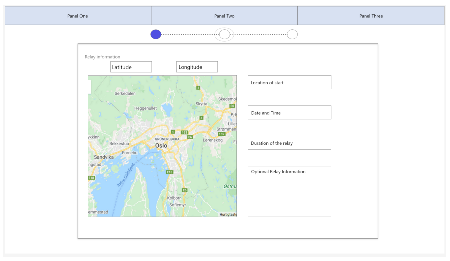
6.1 Admin Panel

The admin panel has the largest user interface and is responsible for the most critical task. Because of this, the admin panel is the system component that has the most amount of work related to design. When designing the admin panel, one of the big focal points was on reducing friction and displaying only what is needed. An administrator should have zero or close to zero visual clutter that could hamper their ability to perform their task. Any time an administrator is on this website, it would be because he has a task to complete, and the application must ensure that the process of doing that is as smooth and problem-free as possible.

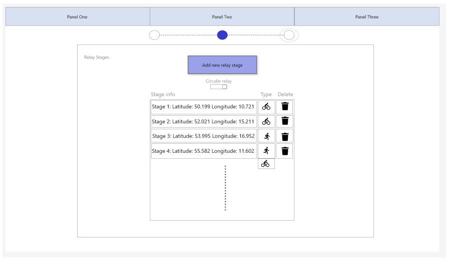
The React framework used to create this website is called Material UI and is one of the most popular frameworks for visual styling [26]. It is based on the Google design principles called "Material Design". It is used by large companies such as Spotify, Netflix, and Amazon [27]. This framework was chosen because of its maturity which in the UI framework spaces means that it comes with a lot of premade components such as Buttons, Lists, Textfields and Dialog boxes. Things that if you were to create and style on your own using HTML and CSS would take a lot of time to code to be at the same level as the premade ones.

6.1.1 Representing a Large Form

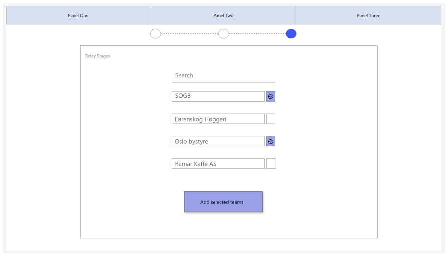
One of the more challenging design aspects of the admin panel was how to display all the information that was needed to create a relay. A problem that kept occurring was that using a single page meant that everything was linear, and the website kept getting longer and longer. This meant that all the information that the user was required to fill in was not visible on the page at the same time. You would have to scroll a considerable amount to view the next set of data inputs.



(a) First step


Panel One
Panel Two
Panel Three
Relay Stages
Add new relay stage
Circular relay
Stage info
Type
Delete
Stage 1: Latitude: 50.199 Longitude: 10.721
Stage 2: Latitude: 52.021 Longitude: 15.211
Stage 3: Latitude: 53.995 Longitude: 16.952
Stage 4: Latitude: 55.582 Longitude: 11.602

(b) Second step

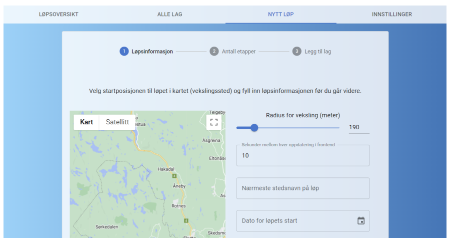


(c) Third step

Figure 6.1: Wireframe in Adobe XD for Creating/Editing a Relay

To solve this, the team used Adobe XD to design wireframes and came up with a design that uses a concept called a “stepper”, which is a form in which all the data is broken into logical steps and when you finish all the input on one page you can proceed to fill in the next portion. This solution allows you to break up this linear stream of data into logical parts.

Another way to solve this problem could have been to have everything on one page, and not centre the content of the page and instead use a wider part of the page. In our case, to reduce the scrolling you would be forced to use the entire width of the page. Problems arise when using such a method because the website in case of width or height change would look very different. This would thereby not conform with the concept of responsive design.

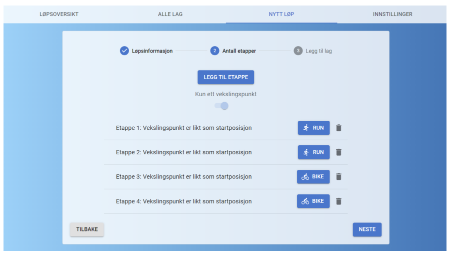


Velg startposisjonen til løpet i kartet (vekslingssted) og fyll inn løpsinformasjonen før du går videre.

Radius for veksling (meter): 190

Sekunder mellom hver oppdatering i frontend: 10

(a) First step



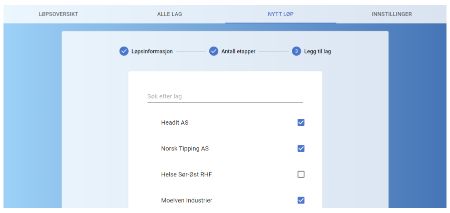
Etappe 1: Vekslingspunkt er likt som startposisjon RUN

Etappe 2: Vekslingspunkt er likt som startposisjon RUN

Etappe 3: Vekslingspunkt er likt som startposisjon BIKE

Etappe 4: Vekslingspunkt er likt som startposisjon BIKE

(b) Second step



Headit AS ☑

Norsk Tipping AS ☑

Helse Sør-Øst RHF ☐

Moelven Industrier ☑

(c) Third step

Figure 6.2: Screenshots of Creating/Editing a Relay

The above figure shows the final result, where you can see the similarity between the wireframe and the application.

6.1.2 Feedback

Don Normann talks about feedback as an indication given back to the user, so the user is aware they made an action. Thereby the user needs to receive feedback immediately. Without feedback, the user has no idea if the action was registered which leads to confusion and thereby results in the user repeating the action, even if the action was registered the first time [28, p. 23].

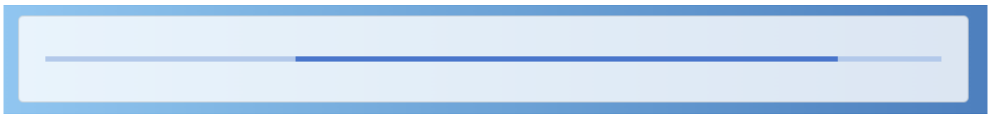


Figure 6.3: Loading bar

An example of where feedback was used in our application, is when a user opens the page for viewing all teams. It's then possible that the application will take some time to load all the data from the API if the user has a bad internet connection for example. To show the user that the page loads, the application renders a loading bar animation to give the user feedback that the application loads. The loading bar also gives the information that the application has not crashed, and makes the user impatient while the data is fetched [29].

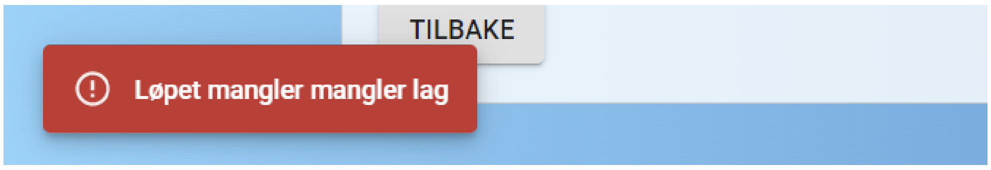


Figure 6.4: Error Message displayed if a relay has no team

The admin panel employs the same feedback method when the user submits a relay and the data is not valid as shown in figure 6.4 The error message is feedback provided after the user's action to submit the relay, which tells the user what went wrong. An alternative to not having feedback would be to make the submit button disabled if the relay data is invalid. We saw it as a worse solution as it would not give any information back to the user about what the problem was, which would make it frustrating for the user [28, p. 23-24].

6.1.3 Consistency

Consistency is important in design, as it makes the user learn new things quickly, thereby not needing to spend unnecessary time learning the system which can be a frustrating experience [30]. The system should be consistent both within the application and also across other applications.

Our way of providing consistency within the application was the process of adding and editing a relay. Since both adding and editing a relay operates with the same data, the process of managing the data is the same as well, as it would be confusing if this task would be different when editing compared to creating a relay. This is done without duplication of code, as it is expanded upon in Section 7.1.6.

6.1.4 Constraints

Constraints in design are what Don Norman explains as limiting the actions of the user. This provides an effect that would lead the user to achieve the desired action and avoid making any unwanted actions. Don Norman has divided constraints into four main categories: physical, cultural, semantic and logical constraints [28, p. 125-130]. We will only talk about physical and cultural constraints, as those are the most relevant constraints in the admin panel.

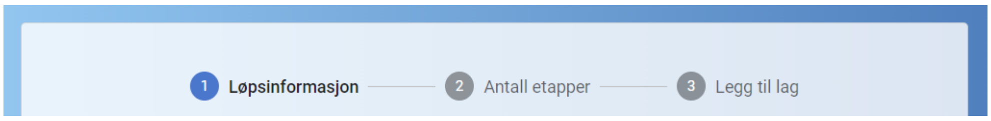


Figure 6.5: Admin Stepper

In Europe, people often go from left to right in actions, like when reading cartoons. But in Japan, for example, they go from right to left [31]. Because of this, culture constraint was implemented within the stepper, as it goes from left to right in the process of creating/editing relay.

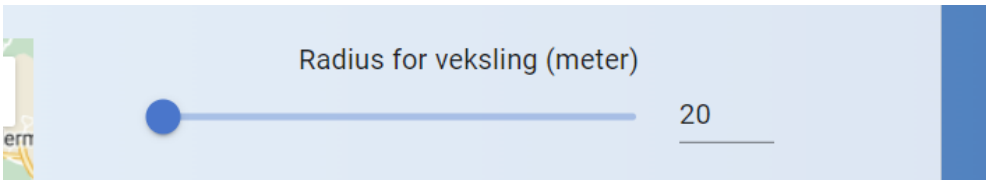
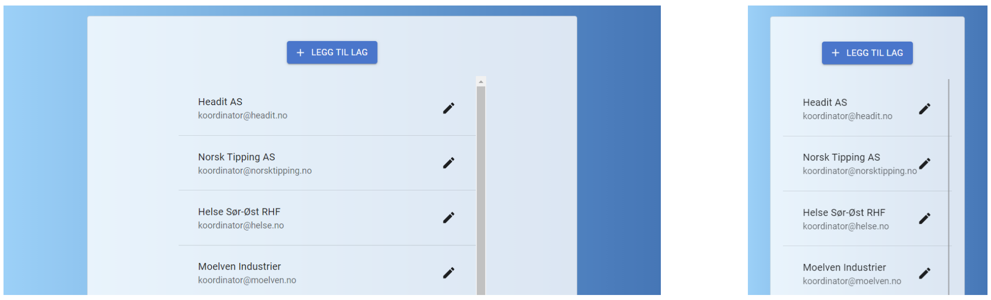


Figure 6.6: Slider Input

An example where physical constraints were used, is the slider for selecting radius related to stage switches. The minimum radius number was set to 20 meters and the maximum to 1000. The slider thereby imposes a restrain within these two values.

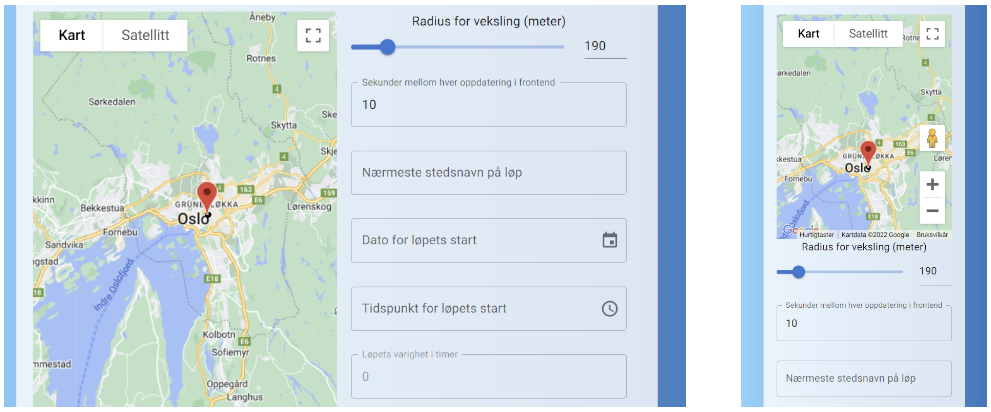
6.1.5 Responsive Design



(a) Desktop **(b)** iPhone 12 Pro

Figure 6.7: Browser width comparison in team view page

Responsive design was implemented on all pages. Since most pages have their elements aligned vertically, responsive design was implemented by scaling the width automatically based on the browser width.



Radius for veksling (meter) 190

Sekunder mellom hver oppdatering i frontend: 10

Nærmeste stedsnavn på løp

Dato for løpets start

Tidspunkt for løpets start

Løpets varighet i timer: 0

(a) Desktop **(b)** iPhone 12 Pro

Figure 6.8: Relay Form in Desktop and Mobile Comparison

In the first step of creating/editing a relay, the map and the input field list are aligned horizontally. Here we conform to the responsive design by having by making all elements align vertically when the browser width size is less than 780 pixels, while also scaling horizontally.

6.1.6 Universal Design

Universal Design is the design and composition of an environment so that it can be accessed, understood and used to the greatest extent possible by all people regardless of their age, size, ability or disability [32].

It was not prioritized to develop the system in compliance with the universal design principles, although we planned a development roadmap for what features to be implemented to raise the level of compliance. It follows the Web Content Accessibility Guidelines when implementing these changes [33].

For colour blindness or colour-deficient vision, the app could be simulated with the Colorblinding chrome extension [1] to find weaknesses in the colour contrasts.

The tab order works as one were to expect in the form for adding/editing relay, see Figure 6.2. The tab order starts from the top input fields, navigates down to the bottom, and also includes the next button and the back button. However, the tab order does not work on the navigation bar above the page content, which makes it impossible to navigate through pages. The map for placing a marker does already include support for tab navigation, but it does not include the possibility to set a marker with a keyboard. It either needs a mouse click input or to be manually coded with JavaScript [34]. All aforementioned points would therefore have to be redesigned.

[1]https://chrome.google.com/webstore/detail/colorblindly/floniaahmccleoclneebhhmnjgdfijgg

We would as a last feature implement text to speech so that people with visual impairment would be able to independently use the app without notable assistance.

6.1.7 Navigation

The website needed to be as responsive and fast as possible. That thinking influenced our design choices. Because of this, the app does not reload the page when navigating between pages or when being redirected within the app. Sometimes these things are unavoidable as is with logging in for example, as those requests that authorize and verify in the background will have to refresh to store session tokens. But having a navigation bar such as the one implemented allowed us to smoothly change between all the sections of the website quickly and easily. The user would easily be able to access any area of the website without the need to refresh the site.

The way we solved it was with a top locked navigation bar, which is a common way to solve the problem visually by showing all aspects of the website at the same time. It could have been solved with a sidebar, but that would have been a purely visual change, and the product owner noted that they were a fan of navigation bars so the sidebar was ruled out.

6.1.8 List of Teams and Relays

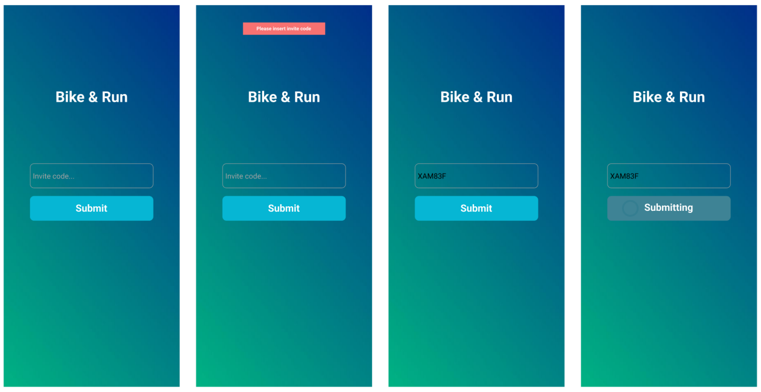
One of the downsides of the default list components in Material UI is that it's very unclear visually that lists are a list of items and that those items are clickable and give access to another action. The biggest issue is the lack of natural borders between items on the list. As previously mentioned this was discovered during testing. It was acknowledged and worked around to make it more clear.

6.2 Mobile Application

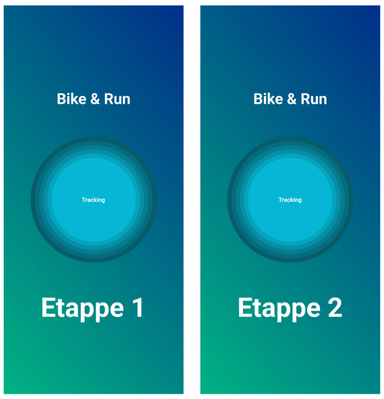
In this section, we will look at the GUI design evolution of the mobile application alongside looking at key design decisions made during the development.

6.2.1 Prototyping

During early discussions with the product owner, we quickly realised that the mobile application GUI would be rather small. Therefore it was decided to not design an early stage wireframe but rather a prototype. This was to save time as the wireframe would not bring more value than what was already known from internal discussions. The prototype was designed with Figma as the focus was on visualizing the user story and because Figma is quick, frictionless and cloud-based [35].



(a) Login Page



(b) Main Page

Figure 6.9: Mobile Application Prototype

6.2.2 Colors

For the application, it was decided to go for a light blue colour as a base colour. The background of the mobile app is a gradient that uses the base colour to a darker blue. Several factors made us go in this direction, one of which was that the MUI library which was used for the administrator panel has as its main colour theme blue. Another reason was to have a sense of coherency in the colour choice for the end-user. This was a qualitative choice to make the experience of working with the apps more appealing visually. The linear gradient background colour is the same on the mobile app as in the admin panel to give a sense of consistency. We applied the 60-30-10 rule to the GUI to accentuate call to action buttons and provide clarity with regards to the user experience [36].

6.2.3 Login Page and Tracking Page

The mobile app consists of two pages. The login page where the participants on a team input their invite key to log in, and the tracking page.

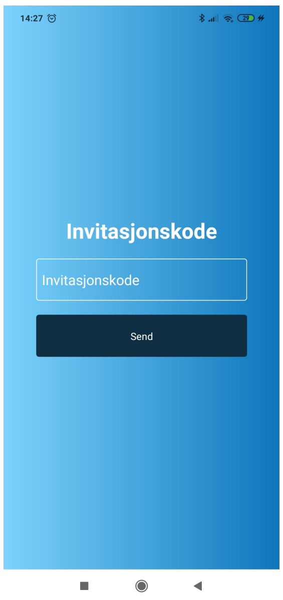


Figure 6.10: first figure

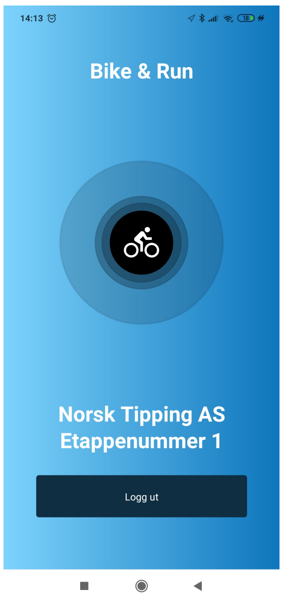


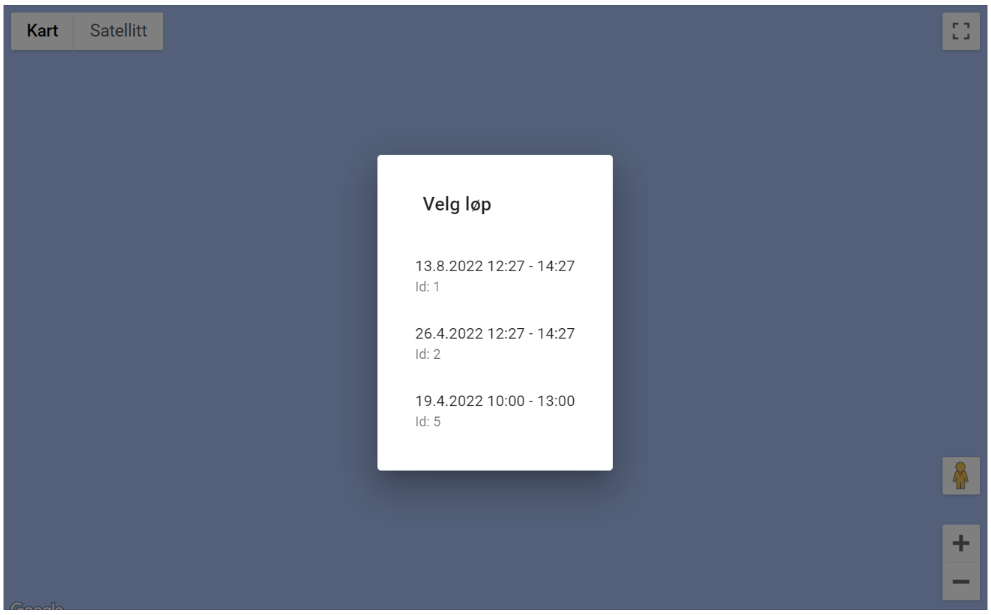
Figure 6.11: second figure

6.2.4 UI Clarity

A worry was that users might think the phone is not actively tracking them, that it would look frozen, as background tracking naturally does not indicate this on all mobile devices. On an Android phone, it will show a GPS icon in the top right corner if the phone is actively sending out GPS signals, which is not very noticeable, and on older models and other types of phones, it is not guaranteed to do this.
Therefore it was decided to implement animation on the screen at all times to show the users that the app is not frozen and that they are still sending their positions if they would check their phones during the relay. As a reference on how these looks see **??** which shows the beginning of the animation.

Another aspect that would clarify this was to change the icon of the stage type dynamically, if a participant is in a biking stage the phone app will display the biking symbol and for running stages, it will display a man running, this also added an extra level of clarity to the participants of the relay, see **??**.

6.3 Map Display



Velg løp

13.8.2022 12:27 - 14:27
Id: 1

26.4.2022 12:27 - 14:27
Id: 2

19.4.2022 10:00 - 13:00
Id: 5

Figure 6.12: Landing page of the map display

The first thing that occurs is that the user has to select a relay. As seen in the figure above. When this is selected and the relay has started, it would start displaying markers corresponding to the participant's coordinates in the chosen relay.

Since a focal point of the system was for it to be optimized for future workloads and expanded use, the system can handle multiple relays at once in case two or more relays would happen in the same time interval. Because of this, the landing page lets you select which relay to display on the landing page.

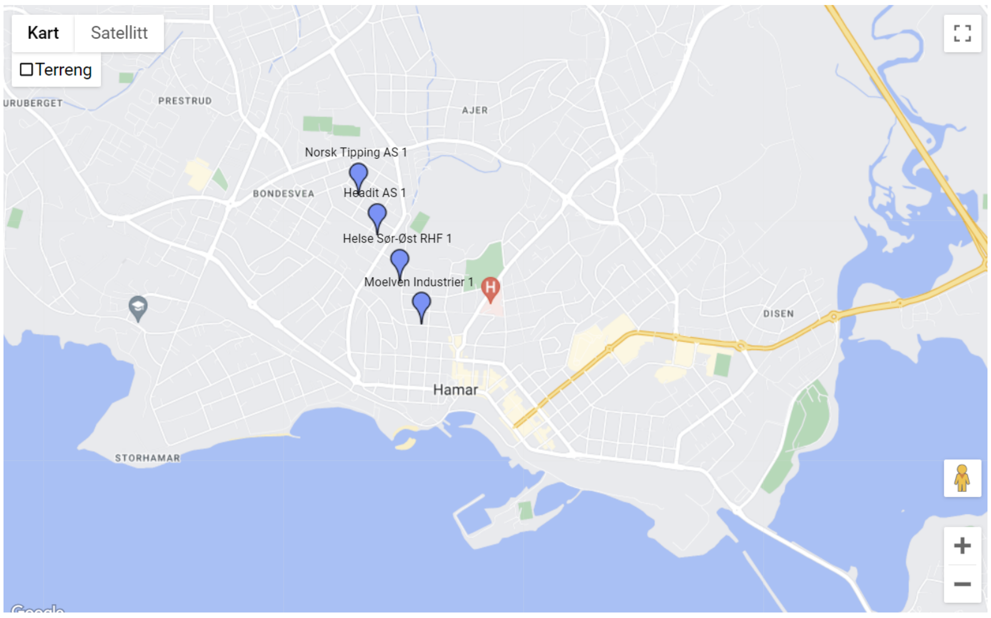


Figure 6.13: Map display of a relay

After choosing the relay, the map is centred at the relay start position and shows coordinates only for participants that are actively racing. The coordinate has a marker, with the name of the team above, with a number at the end representing which stage of the relay the team is at. The participants that are not in a racing stage will not have their position displayed on the map.

We also took care to remove extra information that was present on the Google Maps display. It used to have all street names, figures for restaurants and notable locations. This was changed to a lower detail level to not clutter the display with unnecessary information.

Chapter 7

Implementation

In this part of the thesis, we will display how we implemented solutions to achieve our result goals. We have chosen to focus on the most integral parts of our system. We will look into the admin panel, the API, the mobile application, the map display and the simulator.

7.1 Admin Panel

The admin panel is the biggest code base of all elements in our bachelor project. It has no compile warnings and is as the rest of the system structured for future expansions.

7.1.1 File Structure

```
src
├── components
│   ├── AddOrEditRelay
│   │   └── components
│   │       ├── AddTeams.js
│   │       └── BackNextButton
│   └── ...
├── containers
│   ├── EditRelay
│   ├── Login
│   ├── NewRelay
│   ├── RelayDetail
│   │   └── components
│   │       └── TeamList.js
│   ├── RelayOverview
│   ├── Settings
│   └── TeamsOverview
├── hooks
│   └── auth.js
├── utils
│   ├── apiRequests.js
│   ├── misc.js
│   └── validateRelay.js
└── index.js
```

We grouped the application code into 4 main folders: components, containers, hooks and utils, in the src folder.

Components contains global React components that are meant to be used anywhere in the application. There is one folder for each global component which in turn may have local components within.

Containers contain pages for the app, which is a React component that wraps the whole page with all its components. There is one folder for each page in the app, and each page can also have local components within.

Hooks contains our custom created React hooks.

Utils contains reusable javascript code, which is not React components.

```
RelayDetail
├── components
│   └── TeamList.js
└── index.js
```

This is an example of a React component. All containers and global components are stored in their folder with pascal case naming [37], and an index.js file. Local component files do not have their own folder and are named per their component name in pascal case. There is generally one file for each component unless it is a very small component, and no files extend 400 lines of code which we set as our max acceptable size of a file to contain good readability. If a file were to exceed 400 lines, it generally means that the file could be split into one or more components.

7.1.2 Routes

src/index.js

```
<AuthProvider>
  <Router>
    <Routes>
      <Route path="/login" element={<Login />} />
      <Route
        path="/"
        element={
          <AuthRoute>
            <Navbar />
            <Navigate to="/relay" />
          </AuthRoute>
        }
      />
      <Route
        exact
        path="/relay"
        element={
          <AuthRoute>
            <Navbar />
            <RelayOverview />
          </AuthRoute>
        }
      />
      ...
    </Routes>
  </Router>
</AuthProvider>
```

Routes were made with the react-router-dom package [1], where the Route component was used for each page. The component takes in three main props, the path which is the pathname to the page including URL parameters like id, elements which take in React component to be rendered on the page, and exact which is a boolean which means if the path is exactly what it says or if it also includes extensions of the path URL.

Pages that don't need authentication, which in our case only was the login page, just have the page component directly in the element prop. Pages that need authentication have their element wrapped in the AuthRouth component together with the NavBar component which is the navigation bar.

src/index.js

```
const AuthRoute = ({ children }) => {
  const auth = useAuth();
  if (!Boolean(auth.currentUser)) {
    return <Navigate to="/login" />;
  }
  return <>{children}</>;
};
```

The AuthRoute component checks if the user is logged in with the useAuth hook. If the user is logged in, then the elements that the AuthRoute has wrapped get rendered. And if the user is not logged in, it gets redirected to the login page.

[1]https://www.npmjs.com/package/react-router-dom

7.1.3 Authentication

As we saw in Section 7.1.2, the AuthRoute uses the useAuth hook to check if the user is logged in.

src/hooks/auth.js
```
const AuthContext = React.createContext();

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error("useAuth must be used within a AuthProvider");
  }
  return context;
};
```

The useAuth function itself is very simple as it just returns the data of the Auth-Context.

src/hooks/auth.js
```
export const AuthProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    // checks if user is logged in
    const keycloak = new Keycloak("/keycloak.json");
    keycloak.init({ onLoad: "check-sso" }).then((authenticated) =>
    ↪  {
      if (authenticated) {
        // if logged in
        window.accessToken = keycloak.token;
        setCurrentUser(keycloak);
        setIsLoading(false);
      } else {
        setIsLoading(false);
      }
    });
  }, []);
```

The provider for the context has a useEffect, so every time the Provider gets rendered it checks if the user is logged in. This is done with the keycloak-js package [2]. It first creates a Keycloak object with a JSON file that holds the information about the Keycloak server, like the server URL and client name for this app that is also registered on the server. Then it checks if the user is logged in with the init method, and if the callback for success (authenticated) exists, then the user is authenticated and it sets the keycloak object to the currentUser state and caches the token from Keycloak browser windows. The token is then used when making HTTP requests to the API, this is further discussed in section Section 7.1.4. Regardless of the result, the isLoading state is set to false.

src/hooks/auth.js

```
  if (isLoading) {
    return <>Laster...</>;
  }

  return (
    <AuthContext.Provider
      value={{
        currentUser,
        logout,
      }}
    >
      {children}
    </AuthContext.Provider>
  );
```

If the app is still in the process of checking if the user is logged in, it will just render “Laster...” text, but if the process is done, it will render the provider, including all its wrapping components.

src/containers/login/index.js

```
const loginButton = () => {
    const redirectUrl =
      window.location.protocol + "//" + window.location.host +
       ↪ "/relay";
    const keycloak = new Keycloak("/keycloak.json");
```

[2]https://www.npmjs.com/package/keycloak-js

```
    keycloak.init({
      onLoad: "login-required",
      redirectUri: redirectUrl,
    });
};
```

The login page just has a button for login which takes you to the Keycloak client login page, and then redirects to the /relay page if it gets successful.

7.1.4 API Requests

All API requests are located in apiRequests.js, so it is easy to make changes for all requests to the API and to have an overview of which request the app will do. There is one function for each request.

src/utils/apiRequests.js

```
const deleteTeamApi = async (id) => {
  try {
    const response = await axios.delete(APIURL + `Team/${id}`);
    if (response.status === 200) {
      return response.data;
    }
    throw new Error("Kunne ikke hente data fra API (men fikk
     ↪ kontakt)");
  } catch (error) {
    if (!error.status) {
      throw new Error("Nettverks problemer (ingen response fra
       ↪ API)");
    }
    throw new Error(error.response.data);
  }
};
```

This example is for deleting a team. If there is a problem with the request, it will be caught and then throw an error that will be displayed in a message box to the user, from where the deleteTeamApi function gets executed from. It will either be a standard network error message, or a custom message from the API, like the team id does not exist for an example.

src/utils/apiRequests.js

```javascript
const axios = require("axios").default;
axios.interceptors.request.use(
  (config) => {
    const token = window.accessToken ? window.accessToken : "";
    config.headers["Authorization"] = "Bearer " + token;
    return config;
  },
  (error) => {
    Promise.reject(error);
  }
);
```

All requests have a bearer token in the Authorization request header for authorization with the API, which is set in the axios configuration. The token is retrieved from Keycloak as mentioned in Section 7.1.3.

7.1.5 Global Components

The application has multiple global components. This was done to provide consistency in the app, avoid duplication of code, and for easy maintenance and expansion of the app. Some global components are only used in one place, and could therefore be a local component instead, but are made global as they could be used in multiple places if the application would be expanded. Some of the essential global components are ConfirmModal, MessageBox, Loader and CardBox.

ConfirmModal is the popup message which asks the user for confirmation. This is used when submitting a new relay to the API.

MessageBox is used to display popup messages. An example is the error messages in Figure 6.4.

Loader wraps the content and shows a loading bar until the data needed in the content has been loaded from the API.

CardBox wraps the content to be displayed in a card box UI. This component is used on every page except the login page.

7.1.6 Creating and Editing a Relay

AddOrEditRelay was the biggest React component created in the system.

```
AddOrEditRelay
├── components
│   ├── AddTeams.js
│   ├── BackNextButton.js
│   ├── CoordinatesDialog.js
│   ├── RelayInfo.js
│   └── RelayStages.js
└── index.js
```

The component has one parent file (index.js). This file has all the states related to the relay, and has a useEffect that converts the relayData prop from EditRelay into the right states and format.

For each step in the add/edit process, as seen in Figure 6.5, there is a component. RelayInfo for “løpsinformasjon”, RelayStages for “Antall etapper”, and AddTeams for “Legg til lag”.

src/components/AddOrEditRelay/index.js

```
  <Loader ...>
    <CardBox>
      <div ... >
        <Stepper ... >
          {steps.map((label) => {
            return (
              <Step key={label}>
                <StepLabel>{label}</StepLabel>
              </Step>
            );
          })}
        </Stepper>
      </div>
      {activeStep === 0 && (
        <RelayInfo ... />
      )}
      {activeStep === 1 && (
        <RelayStages ... />
      )}
      {activeStep === 2 && <AddTeams ... />}
      <BackNextButton .../>
    </CardBox>
    <ConfirmModal ... />
  </Loader>
  <MessageBox ... />
```

This is mainly how the index.js was set up. The steps are rendered from the Stepper component, and which step to be highlighted is controlled by activeStep state. The activeStep state also controls which component to be rendered as we can see in the inline rendering condition. Changing activeStep is done in the BackNextButton component, which is a back and next button at the bottom of the page.
It also uses global components like ConfirmModal, MessageBox, CardBox and Loader as mentioned in Section 7.1.5

7.2 API

We are following the design patterns mentioned in 4.3.2.

7.2.1 File Structure

```
 Controllers
 |__ Data
 |   |__ DTO
 |   |__ MODELS
 |__ Hubs
 |__ Jobs
 |__ Migrations
 |__ Scripts
 |__ Services
 |__ ViewModels
 |__ Program.cs
 |__ Startup.cs
```

- Controllers are where all endpoints are defined.
- DTO is for all Data Transfer Objects.
- Models is for all structures in the database.
- Hubs is for all SignalR hubs.
- Jobs is for all routine jobs.

- Migrations contains generated files for database schemas setup.
- Scripts contain reusable C# methods.
- Services contains business logic for data access.
- ViewModels contains data structures from and to UI.

7.2.2 Authentication and Security

Startup.cs

```
services.AddAuthentication(options =>
{
    options.DefaultScheme =
     ↪  JwtBearerDefaults.AuthenticationScheme;
    options.DefaultAuthenticateScheme =
     ↪  JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme =
     ↪  JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    options.Authority = KeycloakServerRealm;
    options.Audience = KeycloakClientId;
    options.TokenValidationParameters = new
     ↪  TokenValidationParameters
    {
        ValidateAudience = true,
        ValidateIssuer = true,
        ValidIssuer = KeycloakServerRealm,
        ValidateLifetime = false,
        RequireExpirationTime = false
    };

});
services.AddAuthorization();
```

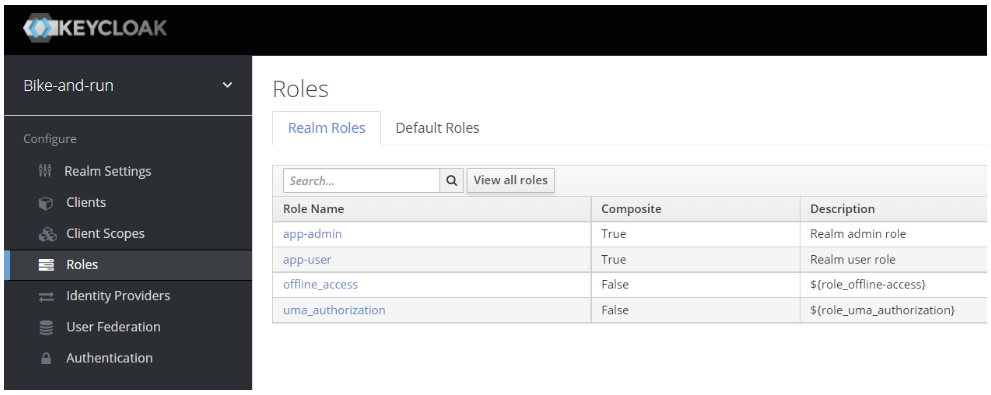
The setup for OAuth2 with Keycloak was mainly implemented in Startup.cs. The variables KeycloakServerReal and KeycloakClientId get their value from appsettings.json.

Controllers/TeamController.cs

```
[Authorize]
```

```
[HttpDelete("{id}")]
public IActionResult DeleteTeam(long id)
{
    try
    {
        _TeamService.DeleteTeam(id);
        return Ok();
    }
    catch (Exception ex)
    {
        _logger.LogWarning(100, ex.Message);
        return NotFound(ex.Message);
    }
}
```

The API was set up to authorize each endpoint that is only used by the admin panel with the authorize attribute. Since the admin panel is the only other module that uses OAuth2 with the API, and the admin panel only have one type of user, there is no role included in the authorization attribute for which user that should have access to the endpoint, as it is not needed.



Role Name	Composite	Description
app-admin	True	Realm admin role
app-user	True	Realm user role
offline_access	False	${role_offline-access}
uma_authorization	False	${role_uma_authorization}

Figure 7.1: Keycloak Roles list

However, even if there was only a need for one type of user, the team set up two types of users on the Keycloak server, app-admin and app-user, such as it can easily be implemented later by Headit if needed.

SQL injection

Entity framework is already secured against traditional SQL injections when using LINQ[38] to Entities for database queries. Since the API uses LINQ to Entities, and not raw SQL commands or Entity SQL, our application has a good SQL injection protection [39]. The API does use Entity SQL one place when sending coordinates in the push routine, but since the query in the push routine only gets query parameters directly from the database and not from user input it was safe to use Entity SQL here.

Hash and brute force

The mobile application only uses the assigned invite key to authorize itself. The invite key is not hashed, as the invite key would not be more sensitive than the hash. It is possible to brute force the system to find an invite key in a relay, by creating a script that tries every combination of numbers and characters, six characters long, against */api/Participant/authorize* or the */api/Coordinate* endpoint, and then log the invite keys that made a 200 response. To fix this, the API endpoints can be implemented with the *AllowXRequestsEveryXSecondsAttribute* attribute [40], and then log the invite keys from requests that received response 200.

7.2.3 WebSockets

The API uses WebSockets to send coordinates to the frontend. The benefit of sending through WebSockets compared to having the frontend retrieve coordinates with GET requests from the API was that WebSockets would allow for less load on the API. The WebSocket was implemented with SignalR library for .NET.

Jobs/PushRoutine.cs

```
public async void StartPushRoutine(int seconds, long relayId)
{
    var runLoop = true;
    // loops every n seconds
    while (runLoop)
    {
        // send relevant coordinates in relay to map display
        ...
        var delayTask = Task.Delay(1000 * seconds);
        await delayTask;
    }
}
```

When the API starts a relay, it will start a push routine for pushing positions to every map display connected through the WebSocket every *N* seconds. For each relay that starts, the API will have a unique WebSocket connection based on the relay's id. The while loop will end automatically when a relay has ended. If the API reboots while there is an active relay going on, it will restart the push routine automatically from Startup.cs.

7.2.4 Algorithm for Stage Switches

Since the API only sends coordinates for participants that are in the racing stage, the API needed to have an algorithm to determine when a stage switch has occurred between participants. The algorithm is executed when the API receives a coordinate from a participant since the stage switch is made based on coordinate data. An alternative solution could be to run the algorithm with a routine that checks for participant switches every 10 seconds for example. But as the algorithm was solely based on coordinates, the API will register a switch immediately when the coordinates come in with our solution, and it will never do a switch calculation if the coordinates for a relevant participant have not changed. The algorithm was first implemented with the following conditions to register a switch when a coordinate was received in the API. The algorithm runs conditions from top to bottom.

- If the owner of this coordinate is in the racing stage, and is not the participant for the last stage
- And the new coordinate is inside stage switch radius
- And if the owner is inside the stage switch radius, according to the last two coordinates
- And if the participant after the owner is outside the stage switch radius according to the last two coordinates

It was then noticed with this algorithm that it was impossible to handle switches in cases where the participant stops sending coordinates, like if the battery is empty or the app crashes for example. Because of this, the algorithm was changed to be executed when the next participant after the currently racing participant sends a coordinate. The new algorithm will register a stage switch if it fulfils the list of conditions. We will name the participant currently racing Alice and name the participant waiting for the next stage switch which in turn triggers the algorithm Bob.

- If Bob is not the first contestant and not currently running.
- And if the participant running in the stage before Bob is Alice.
- And Bob's new coordinate is outside the stage switch radius.
- And if both Bob and Alice have sent two or more coordinates.

    ⋆ Else if Alice has not sent any coordinates and the relay has been active for 15 minutes or more, as Alice might have problems with her phone or forgot to start the tracking on the app. Switch registered.

- And if the last two coordinates from Bob are outside the stage switch radius as well as if the last two coordinates from Alice are inside the radius.
    - ⋆ Else if ten or more minutes since the API has received a coordinate from Alice.

The algorithm checks each step before going further and receiving the data it needs from the database so that the API should not do any more calculations as soon as it is clear that it does not fill the conditions for a stage switch.

Code implementation

Service/CoordinateService.cs

```
// if not the first stage and not in racing stage
if (relayStage.StageNumber > 1 && !participant.IsInRacingStage)
{
```

The algorithm checks the first condition to see if the coordinate owner is not in the racing stage and is not racing the first relay stage.

Service/CoordinateService.cs

```
var team = _context.Teams.FirstOrDefault(t => t.Id ==
↪  participant.TeamId);
var allRelayStages = _context.RelayStages.Where(rs => rs.RelayId
↪  == relay.Id).ToList();
var prevRelayStage = allRelayStages.Find(rs => rs.StageNumber ==
↪  relayStage.StageNumber - 1);
var prevParticipant = _context.Participants.FirstOrDefault(p =>
↪  p.TeamId == team.Id && p.RelayStageId == prevRelayStage.Id);
// if participant in prev stage number is in race stage.
if (prevParticipant.IsInRacingStage)
{
```

Then it finds the participant that is in the relay stage before the owner, and then checks with the second condition to see if that participant is in the racing stage.

Service/CoordinateService.cs

```
var distance =
↪  CoordinateUtils.SimpleDistanceBetween(newCoordinate.Latitude,
↪  newCoordinate.Longitude, relay.Latitude, relay.Longitude);
```

```
double radius = relay.ExchangeRadiusMeters / 110000;
// if outside of radius
if (distance > radius)
{
```

It then finds the distance between the centre of the switch radius circle and the new coordinate and checks the third condition to see if the distance is outside of the radius. As you can see, it divides the ExchangeRadiusMeters which is the length of the stage switch radius in meters, from the relay by 110000, to get the meters to latitude/longitude length. This is not a completely accurate way of calculating length with latitude and longitude, as the length of latitude and longitude is different depending on how close the position is to the equator, and latitude and longitude has different length [41]. But with the small distance that operated with, dividing by 110000 works completely fine and saves the API from extra computations.

Service/CoordinateService.cs

```
var participantLastCoordinates =
↪  GetLastCoordinates(participant.Id, relay.Id);
var prevParticipantLastCoordinates =
↪  GetLastCoordinates(prevParticipant.Id, relay.Id);
// if participant and prev participant has two or more total
↪  coordinates registred
if (participantLastCoordinates.Count >= 2 &&
↪  prevParticipantLastCoordinates.Count >= 2)
{
```

It then gets the last two coordinates for both participants. This is used in the fourth condition to see if there are more than two registered for both participants.

Service/CoordinateService.cs

```
// if last two coordinates is inside the radius, and the last two
↪  coordinates of pref participant is inside the radius (this
↪  participant gets in racing, the prev not)
if (IsAllCoordinatesOutsideRadius(participantLastCoordinates,
↪  relay, radius) &&
↪  IsAllCoordinatesInsideRadius(prevParticipantLastCoordinates,
↪  relay, radius))
{
    // register relay stage change
participant.IsInRacingStage = true;
prevParticipant.IsInRacingStage = false;
_context.SaveChanges();
return;
}
```

Then for the fifth condition, it checks if the last two coordinates are inside the circle. The reason the algorithm checks for the last two coordinates and not just the last single coordinate is to have an extra fail-safe so it will not change the racing stage for two participants if some of the coordinates received are inaccurate.

Service/CoordinateService.cs

```
var minutesSincePrevPLastCoord = (dateNow -
 ↪ prevParticipantLastCoordinates[0].Timestamp).TotalMinutes;
// if over 10 minutes since prev participant har send coordinates
 ↪ (phone is dead for an example)
if (minutesSincePrevPLastCoord >= 10)
{
    // register relay stage change
    participant.IsInRacingStage = true;
    prevParticipant.IsInRacingStage = false;
    _context.SaveChanges();
    return;
}
```

If the fifth condition fails, it checks when the last time the previous participant sent in coordinates was, and if it is 10 minutes or more, it will make a switch, as the phone battery might be dead.

Service/CoordinateService.cs

```
var minutesSinceRelayStart = (dateNow -
 ↪ relay.StartTime).TotalMinutes;
// one or less coordinate from prev particpant and the relay has
 ↪ been active for atleast 15 minutes
if (prevParticipantLastCoordinates.Count < 2 &&
 ↪ minutesSinceRelayStart > 15)
{
    // register relay stage change on team
    participant.IsInRacingStage = true;
    prevParticipant.IsInRacingStage = false;
    _context.SaveChanges();
    return;
}
```

If the fourth condition fails, it will check if the previous participant has one or no coordinates. And if so, and it's been more than 15 minutes since the relay has started, it will make a stage switch. The reason it also checks for one coordinate and not just if there is none is just in case the app sends one coordinate before the app crashes.

Pythagorean formula versus Haversine

To calculate the distance between two coordinates, the API uses a simpler formula instead of haversine[42] which would take into account the curvature of the earth. The reason for not using haversine is that it is not needed for the small distances that are operated within the relay, and it also prevents the CPU from doing unnecessary calculations. The algorithm uses instead the Pythagorean formula on an equirectangular projection, commonly known as an equirectangular approximation[43]. This is the function *SimpleDistanceBetween*.

$$\sqrt{(x_2 + x_1)^2 + (y_2 + y_1)^2}$$

Compared to the haversine formula where angle θ is calculated from the points, (ϕ_1, ϕ_2) are the latitudes and (λ_1, λ_2) are the longitudes

$$havserine(\theta) = haversine(\phi_2 - \phi_1) + cos(\phi_1)cos(\phi_2)haversine(\lambda_2 - \lambda_1)$$

If the distances were far greater than 1000 meters, the API would need to use haversine to be more accurate. Haversine when implemented in code also makes heavy use of the trigonometric functions, which can be very computationally expensive[44]. The utils folder has the C# implementation of the haversine formula, in case Headit wants to use it.

7.3 Mobile Application

Since React Native follows the same structure as React, our Mobile Application follows the same file structure principles as the admin panel. For reference see Section 7.1.1.

7.3.1 Authentication

The system needed a solution to define what participant the device belonged to that was anonymous and appropriately secure. The solution that was implemented, was a uniquely generated invite key for each participant, similar to the pin code Kahoot uses [45], as no actual person can then be traced back to the participant by the data in the database.

utils/apiRequests.js

```
const authorizeInviteKeyToAPI = async (inviteKey) => {
  let config = {
    headers: {
      Authorization: inviteKey,
    },
  };

  try {
    const response = await axios.post(
      API_URL + "Participant/authorize",
      {},
      config
    );
    if (response.status === 200) {
      return response.data;
    }
    throw new Error("Could not get data from API");
  } catch (error) {
    if (error.response) {
      throw new Error(error.response.data);
    }
    throw new Error("No connection to server - Check your internet
     ↪  connection");
  }
};
```

As the user enters the invite key into the GUI and submits it. The app sends a request to the API using the standard HTTP protocol, to validate and authenticate the invite key. If there are no errors caught in this process, the app will proceed to store the invite key in the device's local storage and initiate location services.

7.3.2 Local Storage

utils/storage.js

```
const storeData = async (value) => {
  try {
    await AsyncStorage.setItem(storeKey, value);
  } catch (error) {
    throw new Error(error);
  }
};

const getData = async () => {
  try {
    const value = await AsyncStorage.getItem(storeKey);
    if (value !== null) {
      return value;
    }
  } catch (error) {
    throw new Error(error);
  }
};
```

We used AsyncStorage to store the invite key on the device's local storage to maintain persistence throughout the user's entire session, in case the user closes and reopens the app. If the app reopens, it will go right back to the tracking page and continue to send coordinates to the API if the invite key stored in the device's local storage is still valid.

7.3.3 Background Location Tracking

With the changes done to privacy regulations regarding accessing location in iOS 13 [46] and Android 11 [47] around the time of late 2019 and early 2020. A lot of libraries previously supporting cross-platform support for location services became obsolete. Because of this, the pool of available open-source libraries for React Native that enable location services was quite limited and not well tested. In contrast to other types of use case scenarios where there is a myriad of libraries. After some rigorous research, it was discovered that Expo delivers two libraries that could be used in conjunction to enable us to track mobile devices in the background on both Android and iOS.

App.js
```
const startBackgroundUpdate = async () => {
  // Don't track position if permission is not granted
  const test = await Location.enableNetworkProviderAsync();
  console.log(test);
  const foreground = await
   ↪  Location.requestForegroundPermissionsAsync();
  if (!foreground.granted) {
    throw new Error("Foreground location permission not
     ↪  granted");
  }
  const background = await
   ↪  Location.requestBackgroundPermissionsAsync();
  if (!background.granted) {
    throw new Error("Background location permission not
     ↪  granted");
  }
```

Expo Location [48] was therefore first used to get permission to access the location service. To ask for background permission, the app first needs to ask for foreground permission.

App.js
```
  await Location.startLocationUpdatesAsync(LOCATION_TASK_NAME, {
    // For better logs, we set the accuracy to the most
     ↪  sensitive option
    accuracy: Location.Accuracy.BestForNavigation,
    timeInterval: 5000,
    // Make sure to enable this notification if you want to
     ↪  consistently track in the background
    showsBackgroundLocationIndicator: true,
    foregroundService: {
      notificationTitle: "Location",
      notificationBody: "Location tracking in background",
      notificationColor: "#fff",
    },
  });
};
```

If the request for permissions is granted, the app will start running Expo Location's startLocationUpdateAsync method to register for receiving location updates asynchronously. The function passes a parameter to what background task will be receiving the location updates. As a default, the app is set with a time interval between location updates to 5 seconds, as it is deemed to be fitting based on discussions with the Product Owner.

App.js

```
const LOCATION_TASK_NAME = "LOCATION_TASK_NAME";

// Define the background task for location tracking
TaskManager.defineTask(LOCATION_TASK_NAME, async ({ data, error })
↪  => {
  if (error) {
    console.error(error);
    return;
  }
  if (data) {
    // Extract location coordinates from data
    const { locations } = data;
    const location = locations[0];
    if (location) {
      const coords = {
        latitude: location.coords.latitude,
        longitude: location.coords.longitude,
      };
      postLocationToAPI(coords);
    }
  }
});
```

The Expo TaskManager [49] library allows you to manage long-running tasks, in particular those tasks that can run while your app is in the background. The task is defined at the top level of the application that would receive the location updates even when the application is in the background.

App.js

```
  const stopBackgroundUpdate = async () => {
    const hasStarted = await
     ↪  Location.hasStartedLocationUpdatesAsync(
      LOCATION_TASK_NAME
    );
    if (hasStarted) {
      await Location.stopLocationUpdatesAsync(LOCATION_TASK_NAME);
      setToastMessage([...message]);
    }
  };
```

The task will then run on the device pushing location updates to the task regardless of the state of the application until it is purposefully stopped.

7.4 Map Display

The map display is the smallest module in the system. It compiles with no warning and follows the same file structure principles as the admin panel. For reference see Section 7.1.1.

7.4.1 WebSockets

Before the app connects to a WebSocket, the user has to select the relay, as seen in Figure 6.12.

containers/Landing/index.js

```
const createHubConnection = async (relayId) => {
    setSelectedRelayId(relayId);
    let relay = relays.find((o) => o.id === relayId);
    const centerPlaceholder = { lat: relay.latitude, lng:
     ↪  relay.longitude };
    setCenter(centerPlaceholder);

    const hubConnection = new HubConnectionBuilder()
      .withUrl(process.env.REACT_APP_WEB_SOCKET_URL)
      .build();
    try {
      await hubConnection.start();
      setHubConnection(hubConnection);
    } catch (e) {
      console.log("error", e);
      console.log(
        "tryng to connect again in " + secondsBeforeRetryConnect +
         ↪  " seconds..."
      );
      setTimeout(() => {
        createHubConnection(relayId);
      }, secondsBeforeRetryConnect * 1000);
    }
};
```

When the user chooses a relay, the createHubConnection gets executed and creates a WebSocket connection with the API. If the connection could not be made for some reason, like a drop in internet connection, the application will try again in a few seconds.

containers/Landing/index.js

```
useEffect(() => {
    if (hubConnection && selectedRelayId > 0) {
      hubConnection.on(
        "Coordinates-RelayId" + selectedRelayId,
        (coordinatesFromAPI) => {
            // event listner for reciving coordinates from
             ↪  websocket
            ...
        }
      );
    }
}, [hubConnection, selectedRelayId]);
```

When the WebSocket connection is set, it triggers the useEffect function, which then creates an event listener for the relay from the API. Each time the application receives coordinates, it is processed into a more convenient format, and then set to a React state used for displaying participants' positions.

7.5 Simulator

To test a relay test, a simulator app was built by the team to simulate the coordinates from a relay. This was created early in sprint 2 when it was realized that going outside and running around with phones connected would not be a good method to check whether certain backend logic implementations were sound. Therefore the simulator was developed early on to allow us to test the system and to show progress to the product owner. Also, it allows us to test the system against large relays without having to coordinate large masses of people.

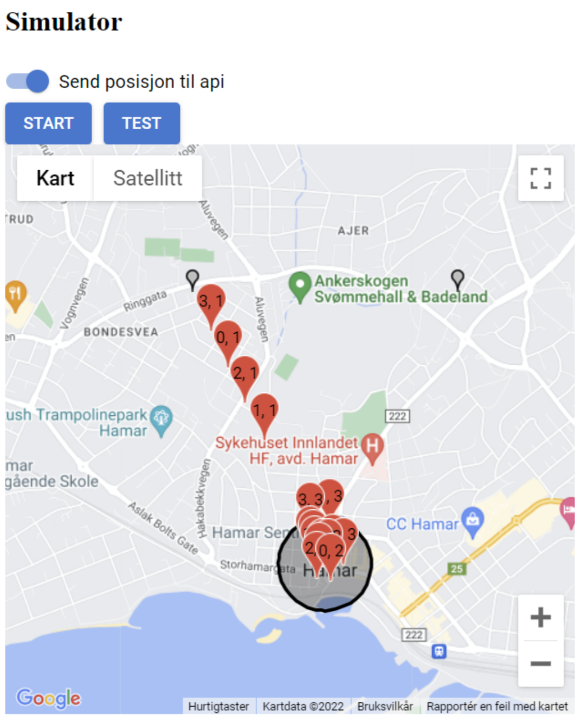

Simulator
Send posisjon til api
START
TEST
Kart
Satellitt
AJER
BONDESVEA
Ankerskogen Svømmehall & Badeland
ush Trampolinepark Hamar
Sykehuset Innlandet HF, avd. Hamar
CC Hamar
Hamar
3, 1
0, 1
2, 1
1, 1
3, 3
2, 0, 2
Google
Hurtigtaster Kartdata ©2022 Bruksvilkår Rapportér en feil med kartet

Figure 7.2: Screenshot of the simulator application operating with four teams

The simulator does not only send coordinates to the API, but it also visually represents all participants. Each mark is one participant, with the first number representing the team, and the second representing the relay stage for the participant.

The simulator is dynamic in the way that it gets automatically generated based on the relay data in the database. It receives relay start position, teams, participants with invite key, etc. and then runs a path in a triangle, relative to the start position. This path can easily be changed in the code, same with the randomized speed.

After having developed the simulator and seeing that given some start data for coordinates, and some points for it to move towards on the map, it was possible to simulate the state of a relay race as if it was real people. This allowed us to fine-tune our algorithm that is used for checking a relay stage switch for participants. The simulator is a piece of software that is also given to Headit for usage, should they seek to extend the system given. That way they can also benefit from the simulator and have the same flexibility as the group had.

7.6 Analysis of Recommended Display Medium

In this section, we will present our recommendation to Headit regarding a display medium. It was specified that this is just a recommendation and that budget should not be taken into account for the items. Headit wanted ideas for the following problem statements:

- Screen that is easy for the audience to see
- Unit to serve frontend
- Consider a lack of electricity on-site
- Hardware purchases have to be low cost

7.6.1 Raspberry PI

We believe this would be a good alternative to use for the unit for displaying the frontend. The latest Raspberry PI can be purchased for about 1000 NOK[50] so it's very cheap. Raspberry pi also had good support for accessories to be purchased along with it. Seeing as it's an exposed single-chip computer, the group would advise purchasing casing to secure it from the conditions such as rain and mud[51].

These types of computers can be powered with power banks which are portable small batteries meant to charge phones for example. Per the specification of the Raspberry PI 4 Model B has 5 V power outputs, so the power bank has to at least output that many volts.

One thing to consider is that power spikes can occur and that it is not for certain that the power bank will be supplying $5V \pm 0.25$ at all times. This is because the raspberry pi will be doing high CPU intensive tasks. This could in very rare cases cause the raspberry pi to short circuit.[52]

State	**Power Draw**
Idle	540 mA
High CPU Load	1010 mA
400 % CPU LOAD	1280 mA

Table 7.1: Table showing Raspberry PI power draw based on usage.

Assuming the middle case in the table we can calculate the expected uptime of the computer. We will assume the power bank to be the Anker 337 with 26800 *mAh*[53] for the calculation.

$$1.01\,A \times 5\ V = 5.05\ W$$

Now knowing the electrical power of the raspberry pi in watts we calculate the capacity of the power bank, multiplying it by the nominal cell voltage of the power bank.

$$26.8\,Ah \times 3.7\ V = 99.16\ Wh$$

With this, we now can calculate the run time assuming a continuous draw of power with now power spikes.

$$99.16\ Wh\ /\ 5.05\ W = 19.64\ hours$$

Airing on the side of caution we can assume even higher CPU load, WiFi usage and cases where more power is being drawn by more things connected to the Raspberry Pi. In this case, we can assume a large margin of 50 % reduction in lifetime, which brings the number down to around 10 hours uptime. A Bike & Run relay has historically only lasted for 3 hours total but assuming an error margin of 1-2 hours for unforeseen events, a 5-hour race would still be well within our calculation.

7.6.2 Display Medium

Considering the potentially unstable weather we advise Headit to purchase a TV screen with a mobile cart stand. To withstand the weather we believe that Headit should find a separate solution for this such as a tarp with poles fastened to the ground. We do not recommend going for a curved display seeing as those can be harder for the spectators to view on.[54]

We advise going with at least a 50-inch screen as a compromise between size and portability. As an example of a specific purchase, we recommend a modern TV screen[55] that is 4K with a ground stand that has sturdy fastening to the ground[56].

Power recommendation

To power larger things such as the display medium, we think that it will be best for Headit to use a silent generator to not cause too much noise pollution for spectators. Brand wise the group recommends going with the Honda brand of generators, specifically the EU2200i[57]. This is a mid-range generator that is very small and portable, efficient with 8.1 hours of usage on 3.78 litres of gasoline and good noise reduction. It operates on a range between 48-57 dB, which is in the same range as regular conversation[58]. It costs about 13 000 NOK MSRP.

Chapter 8

Testing

In this chapter, we will explore the testing that was done on users of the mobile application and the administrator panel. We will look at the software side of testing for the backend API and admin panel as well as how we used the simulator.

8.1 Software Testing

Many testing frameworks exist for many different types of languages, many different methods such as Test-Driven Development[59]. Often it can get very confusing knowing both what to test and how to test that which concerns us in the software.

This is something we are more familiar with as per our experience gained in the course PROG2052 Integration Project where we had a similar technology stack and went through the process of writing tests for .NET APIs and React frontends. For .NET we use the AAA pattern, arrange, act and assert. This makes the tests readable and predictable[60]. For admin panel testing we use the test runner Jest which comes with React[61].

The goal for us with testing was to catch bugs before Headit was to deploy the code. We wanted to give Headit code we knew was reliable and worked as we meant it to.

8.2 Administrator Panel Testing

When testing websites that use React as a framework you can write a variety of tests. It is generally agreed that the most useful tests to write are those called “End to End” tests[62]. These are tests where you are directly simulating user interaction with a specific part of the website. Therefore, you can simulate the state before interaction and then test what it ought to be post-interaction. An example of this might be a user pressing a button and some text is meant to appear.

Other types are unit tests where you test singular pieces of code logic that make up your program. These are less popular in React because React is concerned with user interaction for its testing.

8.2.1 Administrator Panel Unit Test

src/__test__/single_function_unit_tests.jss

```
// datasett uten error
...

// lag datasett med error
...

test("validateRelay passes for data-set 1", () => {
  var returnObject = {
    error: "valid",
    isValid: true,
  };
  expect(validateRelay(data_set_1)).toStrictEqual(returnObject);
});

test("validateRelay fails for data-set 2", () => {
  var returnObject = {
    error: "Ikke gyldig Latitude",
    isValid: false,
  };
  expect(validateRelay(data_set_2)).toStrictEqual(returnObject);
});
```

8.2.2 Administrator Panel End to End Test

src/__test__/mock_tests/index_test.js

```
describe("Test Case For App", () => {
  it("should render TextField", () => {
    const wrapper = shallow(<TeamList />);
    const textElement = wrapper.find("#Search-bar");
    expect(textElement).toHaveLength(1);
  });
});

it("accepts ListItemText props", () => {
  const wrapper = mount(
    <NestedDialog dialogTitle="Test Dialogue Title" listName="LIST
     ↪  NAME" />
  );
  expect(wrapper.props().dialogTitle).toEqual("Test Dialogue
   ↪  Title");
  expect(wrapper.props().listName).toEqual("LIST NAME");
});
```

8.3 Backend API Testing

In the case of testing the backend of our project, we must choose what type of testing setup we wanted to use. As per the official Microsoft documentation, there are two main choices for testing a .Net API when you are dealing with EF Core.

1. Running the tests against your production database system, same as the application.
2. Tests are run against a test double, which replaces the production system and mimics the database in memory during execution.

In recent years Microsoft has made it easier to test against the real production database by introducing such things as Class Fixtures[63] which eases the production of a test context that is shared among classes. But there still exists pains that make this a less desirable choice. A big problem is code isolation, how do we ensure that tests running in parallel (or in serial) to each other do not interfere with each other[64].

Read-only tests are simple and do not have to worry about isolation. But tests that are Write tests are more problematic and introduce scaffolding that needs to be created for them to run in an isolated environment. Those types of tests need to be wrapped in a transaction so that they can't interfere, and at the end of the test you roll back the transaction, no modification was made to the database, and you avoid that interference. But that is boilerplate and extra fill that takes time to write and maintain.

The method that we chose was to use the test double method because of the ease of implementation, maintainability, and the fact that we have experience using this method. To use this method you just install a package using the built-in NuGet package manager. To implement there are many resources, including Microsoft official documentation on how to set up this environment.

We went for using SQLite (in memory mode) as our database fake which has some crucial parts that differentiate it from other choices. This has better compatibility with production relational databases because SQLite itself is a relational database. This allows you to test foreign key constraints, run raw SQL queries and more closely map the database schema in memory. A different choice would use Microsoft's in-memory provider which is the same in principle except that it provides fewer types of queries to run as it's not relational, no transactions, no raw SQL and it's not optimized for performance at all.

8.3.1 Backend API Unit Test

Arrange

UnitTests.cs

```
[Fact]
public void CheckIfCoordinateIsOutsideRadius()
{
using var ctx = CreateContext();
double radius = 0.0021213203435596723;
coordinateService = new CoordinateService(ctx, hubContext);

Assert.NotNull(coordinateService);

// lager ett relay objekt.
...

// lager tre koordinater
...
```

Act

UnitTests.cs
```
var res = coordinateService.IsAllCoordinatesOutsideRadius(
    coordinateDTO, relay, radius);
```

Assert

UnitTests.cs
```
Assert.True(res);

DisposeContext();
```

8.3.2 Backend API Integration Test

This test checks when an admin user is editing a team, and whether the modification was correctly done in the database.

Arrange

TeamIntegrationTests.cs
```
[Fact]
public void EditSingleTeam()
{
    using var ctx = CreateContext();
    teamService = new TeamService(ctx);
    var controller = new TeamController(logger.Object,
     ↪  teamService);

    var newTeam = new TeamVM()
    {
        TeamName = "NewTeamName",
        Email = "NewEmail@gmail.com"
    };
    var id = 1;
```

Act

```
TeamIntegrationTests.cs
    var editedTeam = controller.EditTeam(id, newTeam);

    var editedTeamResult = editedTeam as OkResult;
    Assert.Equal(200, editedTeamResult.StatusCode);

    var editedTeamFromDatabase = ctx.Teams.FirstOrDefault(t =>
    ↪   t.Id == 1);
```

Assert

```
TeamIntegrationTests.cs
    Assert.NotNull(editedTeamFromDatabase);
    Assert.Equal("NewTeamName", editedTeamFromDatabase.TeamName);
    Assert.Equal("NewEmail@gmail.com",
    ↪   editedTeamFromDatabase.Email);

    DisposeContext();
```

8.3.3 API Stress Test

As a part of our API testing, we performed a stress test. We wanted to inflict the system extremely high load so we tested it by having 200 teams participate in a race. This comes to a total of 800 participants. We tested with this to be sure that in the future if Bike & Run grew to have maybe 60 or 70 teams it would be well within the margin of not crashing. For this, we used the simulator to view the relay and by monitoring the console of google chrome whether any POST requests were crashing.

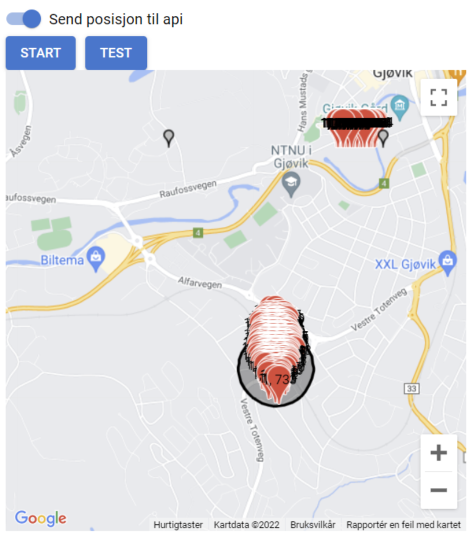


Figure 8.1: Simulator running and displaying 800 markers during stress test

What we found was that the web socket connection did not break and all the coordinates that we expected to see were present in the database. One thing to note is that the simulator display was lagging heavily during the test, considering it was not optimized for that type of displaying of markers. This was purely a rendering bottleneck in the simulator which is inconsequential.

8.4 User Testing

In this section, we will go over the user test that was performed at Headit on location with members of their team. We also refer the reader to the separate standalone document which has more specific details on the test and environment. We will mostly do a small recap, of what we learned and what we did post the test to improve.

Early in discussions with the product owner, both parties agreed that we needed to have at least one user test be performed on real people. Our software is going to directly interact with people who are of varying degrees of skill, and we wanted it to be resistant to obvious design flaws.

In sprint 3 we held the test on-site at Headit's offices and three people participated in the test. It was a task-based usability test primarily where testers would create a relay and then view the invite codes that were generated. The one test participant that was a UX designer participated in a free form discussion after her test with us after all testing was done. Testing user experience on admin panel and phone The first task that the tester had to do was to create a relay. They were given specific information about the relay, such as the location, start/end time and the number of relay stages and their types.

Results

The main takeaway from this section of testing was that most people needed more direction than we had previously thought. Only one tester remembered that they needed to place a location on the map before they proceed further in creating the relay. The reason stated for this seemed to be that there was no form of text, outline or guiding visual element that directed their attention to the map. We fixed this in the next iteration.

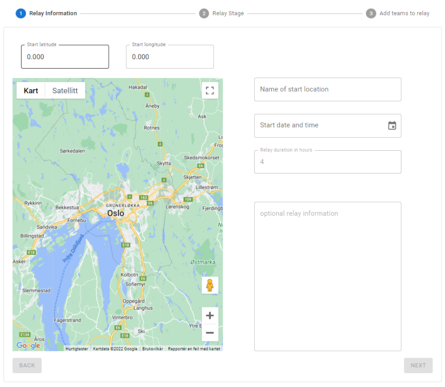


Figure 8.2: Design during user test

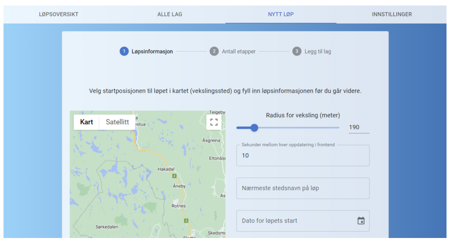

LØPSOVERSIKT
ALLE LAG
NYTT LØP
INNSTILLINGER
Løpsinformasjon
Antall etapper
Legg til lag
Velg startposisjonen til løpet i kartet (vekslingssted) og fyll inn løpsinformasjonen før du går videre.
Kart
Satellitt
Radius for veksling (meter)
190
Sekunder mellom hver oppdatering i frontend
10
Nærmeste stedsnavn på løp
Dato for løpets start

Figure 8.3: Design post implementation of feedback from user test

The other major problem in usability was our solution for selecting the date and time for the relay. MUI has an experimental component which is called a DateTimePicker. This is a combination of a standard calendar picker but linked in one with a clock, where the first click is for the hour and the second is for the minutes. This was a problem and only one of the testers managed to use it properly. We had forgotten for the test to change it to a 24 clock set up so it was confusing in that way but also in the way it was set up where users had to click to change the display to view the clock instead of the calendar. We had to help them to use it. We found out that a "sleek" solution that seems good on the surface because it combines the two things is not a good solution for those who are less experienced in using computers in general.
We solved this by separating the two elements. We also changed the clock to 24 hours.

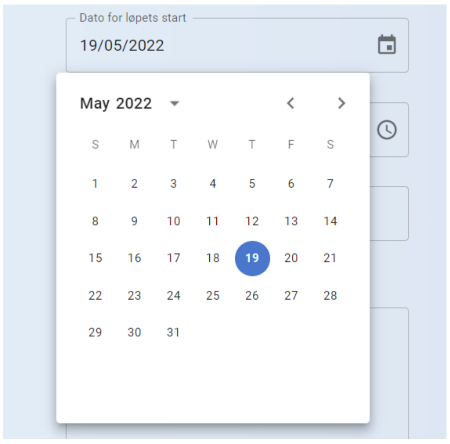


Figure 8.4: Design during user test

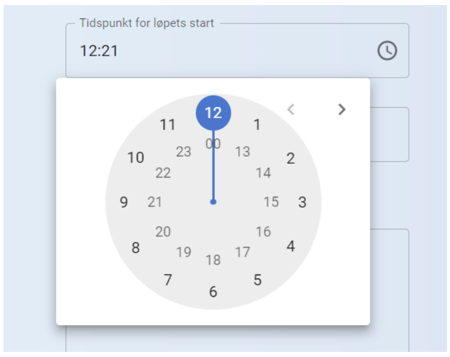


Figure 8.5: Design post implementation of feedback from user test

Summary

We think the value gained in doing the user test was very high and we wished that we could have found time to do another with the same participants but scheduling conflicts made this difficult. Even so, we feel that the feedback that was given by the users was very good and that we got a lot out of it. The team feels that the admin panel quality is now much higher than what it was during the test.

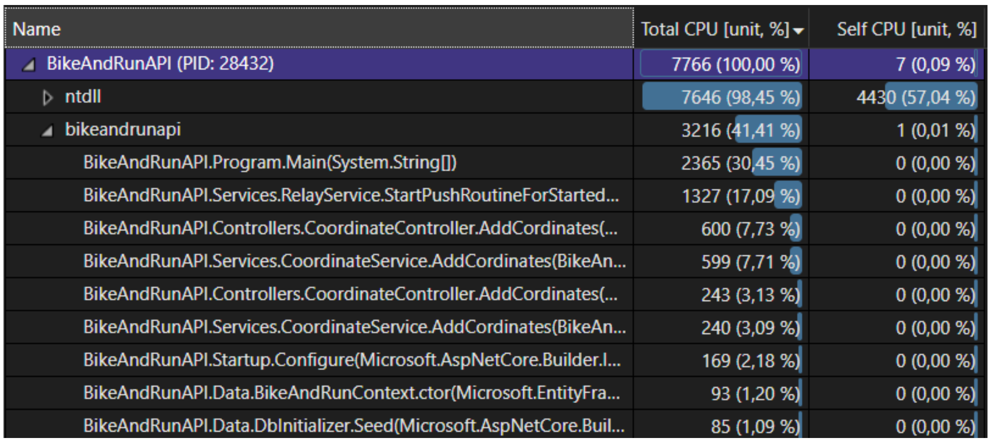
8.5 Simulator Testing

One of the most crucial extra pieces of work that we created was the simulator. This was created early in sprint 2 when it was clear that we realized that going outside and running around with phones connected would not be a good method to check whether certain backend logic implementations were sound. So we developed the simulator early on to allow us to show progress to the product owner and also to free us from having to coordinate large masses of people to test simple relay logic in the backend. See figure 7.2 for an image of simulator.

After having developed the simulator and seeing that given some start data for coordinates and some points it move towards on the map we could feasibly simulate the state of a relay race as if it was real people. This allowed us to fine-tune our algorithm that is used for checking a relay participant swap. The simulator is a piece of software that is also given to Headit for usage should they seek to extend the system we are giving them. That way they can also benefit from the simulator and have the same flexibility as we had.

8.6 Performance testing

During the latter stages of the project, we performed performance analysis and testing for the backend API. We used the built-in tools of visual studio to do this. We wanted to test the number of time queries to the database would take. Tests also were meant to uncover the API's performance and what it was doing during a relay race, to simulate these conditions we used the simulator.



Name	Total CPU [unit, %]	Self CPU [unit, %]
BikeAndRunAPI (PID: 28432)	7766 (100,00 %)	7 (0,09 %)
ntdll	7646 (98,45 %)	4430 (57,04 %)
bikeandrunapi	3216 (41,41 %)	1 (0,01 %)
BikeAndRunAPI.Program.Main(System.String[])	2365 (30,45 %)	0 (0,00 %)
BikeAndRunAPI.Services.RelayService.StartPushRoutineForStarted...	1327 (17,09 %)	0 (0,00 %)
BikeAndRunAPI.Controllers.CoordinateController.AddCordinates(...	600 (7,73 %)	0 (0,00 %)
BikeAndRunAPI.Services.CoordinateService.AddCordinates(BikeAn...	599 (7,71 %)	0 (0,00 %)
BikeAndRunAPI.Controllers.CoordinateController.AddCordinates(...	243 (3,13 %)	0 (0,00 %)
BikeAndRunAPI.Services.CoordinateService.AddCordinates(BikeAn...	240 (3,09 %)	0 (0,00 %)
BikeAndRunAPI.Startup.Configure(Microsoft.AspNetCore.Builder.I...	169 (2,18 %)	0 (0,00 %)
BikeAndRunAPI.Data.BikeAndRunContext.ctor(Microsoft.EntityFra...	93 (1,20 %)	0 (0,00 %)
BikeAndRunAPI.Data.DbInitializer.Seed(Microsoft.AspNetCore.Buil...	85 (1,09 %)	0 (0,00 %)

Figure 8.6: screenshot of performance profiler modular view

In this image, you can see the modular view of the performance session. What you can see is that what takes up the most time is the ntdll. This is a kernel related dynamic link library. The reason for its high CPU usage is not bad and it's doing what it's told to do by the computer. It is doing low-level operations and if you were to remove it, the computer would crash. It is present in the application because it's a dependency of the WIN32 API which is used by .NET.

Below that, you can see the bike and run API and we see that 30.45 % is being used by the main function which initializes the whole REST API, this is expected. From there we can see that it sits idly and is just sending coordinates to the database and the frontend display as it's receiving them in real-time. The rest of the CPU time is unrelated .NET functions.

In testing the database query performance we only profiled the ones we thought might be slow. Those were selected by the number of records they were thought to need to access for their data. Another factor was the amount of data they return. We did perform some of the relay query tests with the same data set for teams on a relay that was used for the stress test for the backend.

Function Call	Parameter	Time spent (ms)
GetParticipantsByRelayID	Relay with 200 teams	5442
GetParticipantsByRelayID	Relay with 4 teams	233
GetRelay	N.A	40
DeleteRelay	Relay with 200 teams	1598
PostRelay	5 TeamIds And 4 Stagetypes	167

Table 8.1: Table showing the performance of the queries.

Based on our programming experience and discussions with the product owner, we concluded that considering the size of the relay for the query with 200 teams, the result was satisfactory. Relays will probably never have more than 50 teams, and the results for those tests were very fast. The other tests were assurances that the queries would not take a very long time and having profiled it we are now sure of this.

Chapter 9

Discussion

In this section, we reflect on the entirety of the process and the result that was achieved. As well as looking at the technologies that were used.

9.1 Reflection on Technologies

In this section, we will discuss our experience with the frontend and backend technologies used during development as well as how development was for the cross-platform mobile application.

9.1.1 Backend

For the backend, we feel like we should have considered the repository pattern, which is recommended by Microsoft. The benefit would have been that less time would have been spent on testing. Most of the testing frameworks that are for mocking API calls work on the assumption that you have followed this pattern. The problem comes when you have to do the manual mocking yourself, as you are bound by the Dependency Injection to satisfy all of the parameter requirements. So when we want to test a controller we have to create a service and a logger object. With a framework like Moq we could mock the entire controller and skip that entirely. It would make for more readable/shorter tests and more maintainable tests as well.

It would also give a higher degree of separation in the code but this can be argued both ways seeing as we already have that to an extent using services the way we do and having 5 or more layers would be bordering on indirection, making simple changes tedious as one change would lead to a chain of accompanying changes elsewhere.

9.1.2 Mobile Application

We experienced a considerable amount of difficulty in developing the mobile application. It mostly came down to how Expo works together with the background location package. It was very unstable in the sense that it would start tracking before it was enabled, it would start tracking if you were to remove the code to enable tracking. There was something we never understood about how Expo runs the apps in some form container in a cloud environment that kept making problems for us.

In hindsight, we would not have chosen React Native because of the number of hours that were lost trying to make it do what should be basic things. About 1.5 weeks were spent debugging the background location tracking functionality. The problem was that the code seemed to behave very differently each time it was run, and together with the issues mentioned about Expo it was very hard to get the software stable enough to a point where Headit could ship it.

There were early signs that we should have switched to something like Flutter but the product owner early on made the argument for React Native as maintaining two code bases may have become a time-consuming process, as well as the concern of spending too much time raising our competence level within another framework, which then kept us going with React Native. In hindsight, the code base for the mobile application is so small that splitting it into two would pose no issue at all. You could make the argument that as it grows it gets more difficult but that is not the situation at hand currently, that would have to be re-evaluated at a later stage.

9.1.3 Frontend

9.2 Project Process

For this section, we will evaluate how well the project plan was followed, and our experience with using Scrum in the way we did. Looking at ourselves, we will analyze how the group worked together and reflect on how the process could have been done differently.

9.2.1 Project Plan Evaluation

Looking backwards at the whole run-through of the project we can confidently say that we followed our project plan rather rigorously. In our documented plan we laid out a clear pathway that the project was supposed to take, not everything was set in stone but we followed the key points such as the sprint plan and the meeting schedule. Following the plan in this strict way allowed us to finalize development at a comfortable distance away from the thesis due date. Having this extra time allowed for more time to look over the thesis and re-write sections as we saw fit. This also allowed us to get more feedback from our supervisor which we managed to do twice.

We were very pleased with how the user testing worked out because there seemed at some points that it might have been needed to be postponed but in just following the plan, things came into place because working with deadlines is a good motivating factor for the team.

9.2.2 Scrum

Scrum was the right choice for this project seeing as Headit was hands-off with some decisions meaning they were left up to us to decide the best path forward. Multiple times during the project a situation would change and the agility given by our software development methodology allowed us to adapt to those situations efficiently and prioritize the most pressing task.

Scrum has more components to it than we utilized. We now see that we did not use Scrum as we perhaps should have. One example is stand up meetings, these are something we should have done seeing as we did work every day. In the early, to mid-stages of the project, we felt a bit disconnected from what each other were doing, having a quick 15 minute stand up each morning would have helped make the team feel more connected.

In contrast to this, there are other parts of Scrum we are happy we did not do, an example would be some teams running Scrum won't allow for code to be released were our version of this was pushing code into the main branch until after the sprint review is completed. Following such a strict methodology was something we did not feel would suit the team because it would impact the feeling of progression and would perhaps complicate the GIT situation, which you don't want because those are not the hard problems you want to spend time-solving. They are pure obstructions that might take away several hours of more critical development time.

Looking back we can see that the way we used Scrum was a combination of things we like with Scrum as well as things we like about Kanban. An example of this was our usage of continuous flow. Issues would commonly go from one sprint to another and who was assigned to each task could change. We were happy with this approach as it allows for the better parts of both software development methodologies to be combined into one rather than following a rule set which you then run the risk of resenting.

9.2.3 Trello

The issue board was used by the team during the whole process but not to its full potential. A pain point for us early on was that the cards were not descriptive enough and were not updated enough. The update criticism was also levied by members feeling that the board was not being updated enough in general.

To fix this we had a meeting where we brought this up to the product owner and we had an open discussion on how we could fix this issue. This can be seen documented in this meeting log as an issue (# LINK). After implementing the changes suggested we did not experience this as an issue again during the rest of the project period which goes to show the solution was effective.

9.2.4 Sprints

We are content with how our sprints went, we feel like we did all that we could for each of the sprints and if some tasks were not finished, it was pushed over into the next sprint. This was our chosen adaptation to this issue that we arrived at after consulting the product owner.

Sprint ID	Allotted Time	Time Spent
1	180 hours	202 hours
2	180 hours	220 hours
3	180 hours	217 hours
4	180 hours	268 hours
5	180 hours	293 hours
6	180 hours	309 hours

Table 9.1: Time breakdown of sprints

One issue with the sprints was that we did not perform any time estimations on tasks. One way this error manifested was that tasks would take much longer than previously anticipated. Issues would be present on the Trello board for up to a week more than we assumed. Such was the case for the issues regarding background location tracking on the mobile application and queries related to WebSocket routines. However these were just carried into the next sprint without much issue, the product owner was okay with that.

9.2.5 Group Evaluation

Overall we feel that the group worked very well and we are pleased with how we handled tribulations along the way. There were issues regarding the Trello board and with open and honest communication in the group as well as reaching out for help those were solved swiftly. As per the project plan section 3.2 we have followed the rule set in regards to workload and group decisions strongly and we believe that this has benefited us greatly.

For the project period, there were times when we did work at school and then parts where we did work from home. This was mostly controlled by when Jørgen Eriksen had to travel to and from Fredrikstad. When we worked at school those sessions would start at 10 and would last 6 hours until 4 o'clock. We feel the most productive work sessions were the ones we had together as we would be able to get instant responses to questions. This was also a big motivational factor for us because we would have deep discussions about a design and we would go back and forth arguing about the best way to program something or how we should write something for the thesis.

9.2.6 Revision Control

As part of our project, Headit had decided that we should use private repositories in their BitBucket group. Initially, for the first two weeks, we hosted the code ourselves on GitHub but as per their wishes of keeping the code proprietary, we transferred the repositories over there.

At the start of the second sprint, we instated a rule that all the repositories would be subject to “branch protection”. This means that no one can push/merge a branch into the master branch without the merge being approved by at least one team member. This way only quality code would be merged in and also members would be forced to familiarize themselves with code they weren't directly writing.

Having branch protection means that we had branches that would be "feature branches", this was nice because it isolates all the new code in case something goes wrong. It also gave a clear overview of how things were added over time. The branches would follow a Git-Flow style pattern, except we would not follow the idea of having multiple “main” branches.

9.2.7 Time Allocation Breakdown

Accounting for developing the system as well as writing the thesis, the team spent 2165 and half hours in total. We are very satisfied with the number of hours invested as we feel it reflects in the end product. We are satisfied with the time allocation over the months with there being a steady increase towards the climax month of April. May was a month which mostly was spent writing the thesis which we in retrospect feel was a good decision as we experienced that the time allocated earlier to finish the development process allowed more time to solely work on the thesis.

Overall the modules which required the most time allocated were the Admin Panel, Backend and Mobile in that order. The feedback received from the product owner suggests that prioritizing these modules was a key point in delivering a fully functional end product.

If the team were to do something different, we would probably have allocated even more time to administrative tasks during development to even further increase our productivity and collaboration.

9.2.8 Critique of our Project Process

Formally defined code reviews would have been beneficial to the project. Having a clear expectancy of what a code review consists of, how often it should be done, and who should look at what code. Answers to these types of questions would have helped us keep code quality and documentation of the code a more present thing in our codebases. Though, we did program a large part of the code at school together so we were always talking during that about the code which means issues were solved there and then. But at certain points members of the team felt disconnected from code they weren't directly responsible for, which broader code reviews would have helped mitigate.

We did not do performance testing as often as we should have. At certain points, we used software inside of Visual Studio to test the performance of an API call or a query to the database. But we feel as though we did not do it often enough. One could argue that you optimize only if performance becomes a problem, but having at least a basic understanding that if you keep going down a certain road it will not lead you to a performance bottleneck would have been beneficial.

9.3 Product

In this section, we will analyse the product itself and the accomplishments surrounding it. We will look at how we fared in achieving the result goals, the ethical and societal effects of our product, the consequences of our design choices and lastly a critique of the end product.

9.3.1 Revisiting Result Goals

Following is a discussion on how we have met the result goals set during planning, presented in Section 1.6.1. The goals will be listed as a bulletin with the discussion following beneath.

- To develop a backend service that can receive, process, store and distribute coordinate data, race information and team information. The backend service must be expandable to allow support for different types of devices and further data such as pulse and heart rate.

We reached our goal of designing and developing an expandable backend service that handles all data mentioned in the requirements. The performance of the backend service and the quality of code and design are one of the vocal points of the finished product.

- To facilitate the deployment of the backend and frontend services as a docker image.

  By end of the project, both the backend and frontend services are ready for deployment by Headit. In hindsight, we should have set this up at the very begging of the development process to have it deployment-ready through all iterations of the applications.

- To develop a cross-platform mobile application with background location tracking. The mobile application must be identifiable by team and relay stage.

  One of the hallmarks of this project we hold to a high standard is the fact that we were able to reach the goal of developing a cross-platform mobile application with background location tracking based on our perceived level of difficulty regarding the matter.

- To develop a frontend service for displaying the relay race in near real-time.

  We have reached our goal of developing a frontend service for displaying the race. With the use of WebSockets, we are confident that our system should uphold the standard of a near real-time view of the race. Although we would have wanted to improve the overall UI for better a better user experience as this was under prioritized below other features.

- To develop a frontend service acting as an admin panel for managing the race information.

  We have reached our goal of developing a frontend service for managing relay races which have all the functionality needed for both facilitating a race as well as administrating it afterwards.

- To provide an analysis on what display medium would be best suited for the solution.

  As a part of this thesis in Section 7.6. we have provided an analysis for display mediums we would recommend for our system under the conditions of a relay race scenario, with regards to weather conditions and facilities surrounding the race.

9.3.2 Ethical and Societal Outcomes

The system will hopefully reinvigorate Bike & Run as an event after some years affected by the Covid-19 outbreak. By increasing recurring participants and attracting new ones. Helping Innlandet Fylke improve the overall public health. Thereby lowering the probability of lifestyle deceases like atherosclerosis, heart disease, stroke, obesity and type 2 diabetes [65]. Hopefully, the system can lead to improved public mental health as well by motivating outdoor exercise and socializing [66].

9.3.3 Consequences of Design Choices

Trails in the Admin Panel

A design decision we pondered on during the early stages of development was whether we should have functionality for specifying the trail of the race. After discussions with the product owner, we concluded not to design the application for it. As a consequence, the system is not able to display data such as positions on the leader board. We do not at the moment support giving feedback to users on time spent on a stage or their speed during certain parts of the stage although the data structure design can calculate the information.

9.3.4 Critique of Product

A point of criticism for our end product is that the system in the case of a crash, would not be able to maintain the state of the race if two-stage switches have happened during the time frame the system is down.

The reason for this is how we implemented stage switching as an automated process where it is based on participants entering and exiting a given parameter area. More on this in Section 7.2.4.

Therefore in the specific scenario of a crash right before a stage switch and considering an exceptionally fast runner which could finish a running stage in less than 10 minutes, there may be an issue with maintaining the state of the race.

Chapter 10

Conclusion

In this section, we summarize the project and process as well as give Headit some guidance on what can be done to further the quality of the product down the road.

10.1 Process

All in all, we are satisfied with the process and planning for the project. Having weekly meetings with both the product owner and project supervisor helped us complete things in time, following the Gantt chart at the same time using scrums agility to allow us to adapt to situations in real-time as they develop.

10.2 Product Result

We have completed all of the parts of the project that we set out to do. Headit now has a full system that it can further develop and maintain in the coming years. It is reliable due to our testing that is present both on the backend and frontend and with the user test performed we can be sure that users find the product usable and can achieve their goals in using it.

10.3 Beyond this Project

1. Add a table in the database between Relay and Participants to create a much easier query to fetch participants by relay id.
2. The display for relays should support a form of colour coding for the teams to increase visibility for spectators.
3. All the websites need to support universal design to some level and verify through WCAG principles that they satisfy basic website requirements.
4. Implement a mechanism to stop brute force attacks on invite code entries.
5. Extend the mobile application to support smaller displays. This would enable participants to use IoT devices such as smartwatches instead of their phones.

6. Secure the web socket connection by authenticating clients that attempt to connect to the socket.
7. Automate the process of sending out invite codes to the participants on a team.

10.4 Final words

Having completed this bachelor project, we have deepened our knowledge of complex software projects in both planning and execution. We have delivered Headit, a new piece of software that will help execute a public health offer beloved by those who participate.

The team has explored new frontiers of their ability and widened our base of knowledge for this such as full-stack development and designing systems of a large scale.

We are wholly satisfied with our effort and how the project turned out. This could not have been done without our support systems, our supervisor and the product owner at Headit. The team thanks them for their unwavering guidance and through their help we have written a thesis documenting this experience and developed a system that we are overjoyed to present to Headit.

Bibliography

[1] COPSCE. ‘Thesis-ntnu.’ (2022), [Online]. Available: https://github.com/COPCSE-NTNU/bachelor-thesis-NTNU (visited on 15/05/2022).

[2] GDPR-EU. ‘General data protection regulation.’ (2021), [Online]. Available: https://gdpr-info.eu/ (visited on 15/05/2022).

[3] M. Rehkopf. ‘Kanban vs. scrum: Which agile are you?’ (n.d.), [Online]. Available: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (visited on 20/04/2022).

[4] M. Rehkopf. ‘What is scrum?’ (n.d.), [Online]. Available: https://www.atlassian.com/agile/scrum (visited on 20/04/2022).

[5] R. Anderson. ‘Getting started with asp.net mvc 5.’ (2021), [Online]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started (visited on 22/04/2022).

[6] S. S. Kirk Larkin and B. Dahler. ‘Dependency injection in asp.net core.’ (2022), [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0 (visited on 12/05/2022).

[7] w3techs. ‘Usage statistics of javascript as client-side programming language on websites.’ (2022), [Online]. Available: https://w3techs.com/technologies/details/cp-javascript (visited on 10/05/2022).

[8] Jellyfish. ‘Web development evolution from the 2000s’ to 2020.’ (2020), [Online]. Available: https://jellyfish.tech/web-development-evolution-from-2000s-to-2020/ (visited on 19/04/2022).

[9] Interaction-Design. ‘What is material design?’ (2021), [Online]. Available: https://www.interaction-design.org/literature/topics/material-design (visited on 25/04/2022).

[10] Mapbox. ‘Api reference.’ (2022), [Online]. Available: https://docs.mapbox.com/mapbox-gl-js/api/ (visited on 25/04/2022).

[11] Mapbox. ‘Maps pricing.’ (2022), [Online]. Available: https://www.mapbox.com/pricing%5C#maps (visited on 25/04/2022).

[12] M. Palchuk. 'Mapbox vs google maps: What maps api is best for your app?' (2022), [Online]. Available: https://www.uptech.team/blog/mapbox-vs-google-maps-vs-openstreetmap (visited on 25/04/2022).

[13] Statista. 'Leading mapping apps in the united states in 2021, by downloads.' (2021), [Online]. Available: https://www.statista.com/statistics/865413/most-popular-us-mapping-apps-ranked-by-audience/ (visited on 25/04/2022).

[14] J. Codruiet. 'Enhance your experiences by embracing the familiar.' (2021), [Online]. Available: https://uxdesign.cc/enhance-your-experiences-by-embracing-the-familiar-54668995269e (visited on 25/04/2022).

[15] NPM. '@microsoft/signalr.' (2022), [Online]. Available: https://www.npmjs.com/package/@microsoft/signalr (visited on 27/04/2022).

[16] Techempower. 'Web framework benchmarks.' (2021), [Online]. Available: https://www.techempower.com/benchmarks/ (visited on 16/05/2022).

[17] ClearBridge. '5 key benefits of native mobile app development.' (2021), [Online]. Available: https://clearbridgemobile.com/benefits-of-native-mobile-app-development/ (visited on 16/05/2022).

[18] Google. 'Flutter.' (2022), [Online]. Available: https://en.wikipedia.org/wiki/Flutter_(software) (visited on 12/05/2022).

[19] Cordova. 'Cordova architecture.' (2021), [Online]. Available: https://cordova.apache.org/docs/en/latest/guide/overview/index.html%5C#architecture (visited on 14/05/2022).

[20] Drifty. 'Ionic.' (2022), [Online]. Available: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framewor) (visited on 12/05/2022).

[21] statista. 'Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021.' (2019), [Online]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (visited on 13/05/2022).

[22] J. Ide. 'Expo go — a new name for the expo client.' (2020), [Online]. Available: https://blog.expo.dev/expo-go-a-new-name-for-the-expo-client-4684a2709904 (visited on 16/05/2022).

[23] Expo. 'Configuration with app.json / app.config.js.' (n.d.), [Online]. Available: https://docs.expo.dev/workflow/configuration/ (visited on 16/05/2022).

[24] Expo. 'Customizing metro.' (n.d.), [Online]. Available: https://docs.expo.dev/guides/customizing-metro/ (visited on 16/05/2022).

[25] E. Union. 'Art. 4 gdpr definitions.' (2016), [Online]. Available: https://gdpr-info.eu/art-4-gdpr/ (visited on 23/04/2022).

[26] Brainhub. 'Top 10 react libraries every javascript professional should know in 2021.' (2021), [Online]. Available: https://brainhub.eu/library/top-react-libraries/ (visited on 11/05/2022).

[27] MUI. 'Mui: The react component library you always wanted.' (n.d.), [Online]. Available: https://mui.com/ (visited on 15/05/2022).

[28] D. A. Norman, *The Design of Everyday Things*. Basic Books, 2013.

[29] W. A. Staff. 'Loading bar design: Do's and don'ts you should know.' (2019), [Online]. Available: https://wpamelia.com/loading-bar/ (visited on 17/05/2022).

[30] U. World. 'Consistency – a key design principle.' (2018), [Online]. Available: https://uxdworld.com/2018/06/02/consistency-a-key-design-principle/ (visited on 08/05/2022).

[31] A. Osmond. 'How to read manga.' (2019), [Online]. Available: https://study.soas.ac.uk/how-to-read-manga/ (visited on 17/05/2022).

[32] N. D. Authority. 'What is universal design.' (n.d.), [Online]. Available: https://universaldesign.ie/what-is-universal-design/ (visited on 20/04/2022).

[33] W3C. 'Web content accessibility guidelines (wcag) 2.1.' (2018), [Online]. Available: https://www.w3.org/TR/WCAG21/ (visited on 20/04/2022).

[34] C. J. Shull. 'Improved accessibility in the maps javascript api.' (2021), [Online]. Available: https://cloud.google.com/blog/products/maps-platform/improved-accessibility-maps-javascript-api (visited on 02/05/2022).

[35] E. Shoemaker. '8 reasons why every product design team needs to use figma — or get left behind.' (2021), [Online]. Available: https://webuild.io/why-use-figma-for-digital-product-design (visited on 16/05/2022).

[36] A. Adelugba. 'How the 60-30-10 rule saved the day.' (2020), [Online]. Available: https://uxdesign.cc/how-the-60-30-10-rule-saved-the-day-934e1ee3fdd8 (visited on 20/04/2022).

[37] C. McKenzie. 'What is pascal case?' (2021), [Online]. Available: https://www.theserverside.com/definition/Pascal-case (visited on 15/05/2022).

[38] Microsoft. 'Linq.' (2022), [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ (visited on 15/05/2022).

[39] Microsoft. 'Security considerations (entity framework).' (2021), [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations?redirectedfrom=MSDN (visited on 07/05/2022).

[40] T. T. (Technical). 'Dotnet securities dos and don't for owasp top 10 security vulnerability.' (2021), [Online]. Available: https://triveniglobalsoft.com/dotnet-securities-dos-and-dont-for-owasp-top-10-security-vulnerability/ (visited on 07/05/2022).

[41] M. Rosenberg. 'The distance between degrees of latitude and longitude.' (2020), [Online]. Available: https://www.thoughtco.com/degree-of-latitude-and-longitude-distance-4070616 (visited on 04/05/2022).

[42] A. Upadhyay. 'Haversine formula – calculate geographic distance on earth.' (2022), [Online]. Available: https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/ (visited on 14/05/2022).

[43] M. T. Scripts. 'Calculate distance, bearing and more between latitude/longitude points.' (2022), [Online]. Available: https://www.movable-type.co.uk/scripts/latlong.html (visited on 12/05/2022).

[44] n.a. 'A simple benchmark of various math operations.' (2022), [Online]. Available: https://latkin.org/blog/2014/11/09/a-simple-benchmark-of-various-math-operations/ (visited on 15/05/2022).

[45] Kahoot. 'What is kahoot!?' (n.d.), [Online]. Available: https://kahoot.com/what-is-kahoot/ (visited on 11/05/2022).

[46] A. Chugh. 'Handling ios 13 location permissions.' (2019), [Online]. Available: https://betterprogramming.pub/handling-ios-13-location-permissions-5482abc77961 (visited on 14/05/2022).

[47] Android. 'Location updates in android 11.' (2022), [Online]. Available: https://developer.android.com/about/versions/11/privacy/location (visited on 14/05/2022).

[48] Expo. 'Location.' (n.d.), [Online]. Available: https://docs.expo.dev/versions/latest/sdk/location/ (visited on 02/05/2022).

[49] Expo. 'Taskmanager.' (n.d.), [Online]. Available: https://docs.expo.dev/versions/latest/sdk/task-manager/ (visited on 02/05/2022).

[50] Prisguiden. 'Raspberry pi 4 model b.' (2022), [Online]. Available: https://prisguiden.no/produkt/raspberry-pi-4-model-b-8gb-448032 (visited on 10/05/2022).

[51] Amazon. 'Argon neo raspberry pi 4 case.' (2022), [Online]. Available: https://www.amazon.co.uk/dp/B07WMG27T7?tag=georiot-trd-21&ascsubtag=tomshardware-no-1415542353981084700-20&geniuslink=true&th=1 (visited on 11/05/2022).

[52] J. Geerling. 'Power consumption benchmarks.' (2021), [Online]. Available: https://www.pidramble.com/wiki/benchmarks/power-consumption (visited on 15/05/2022).

[53] Anker. 'Anker 337 powerbank.' (2022), [Online]. Available: https://us.anker.com/products/a1277 (visited on 11/05/2022).

[54] I. Engineering. 'What happened to the curved tvs anyway?' (2022), [Online]. Available: https://interestingengineering.com/which-is-better-curved-or-flat-screen-tvs (visited on 12/05/2022).

[55] Amazon. 'Samsung 50 inch tv uhd 4k.' (2022), [Online]. Available: https://www.amazon.com/TAVR-Universal-Adjustable-Tempered-Storage/dp/B07MKK75K8/ref=sr_1_9?keywords=50+inch+tv+stand&qid=1652866064&sr=8-9 (visited on 11/05/2022).

[56] Amazon. 'Fitueyes tv stand for max 65 inch tvs.' (2022), [Online]. Available: https://www.amazon.com/FITUEYES-Upgrade-Universal-Adjustable-Shelves/dp/B08LK1X1SZ/ref=sr_1_15?keywords=tv%5C%2Bfloor%5C%2Bstand%5C&qid=1652866317%5C&sr=8-15%5C&th=1 (visited on 11/05/2022).

[57] Honda. 'Hond eu2200i.' (2022), [Online]. Available: https://powerequipment.honda.com/generators/models/eu2200i (visited on 11/05/2022).

[58] C. for Disease Control and Prevention. 'What noises cause hearing loss?' (2022), [Online]. Available: https://www.cdc.gov/nceh/hearing_loss/what_noises_cause_hearing_loss.html (visited on 10/05/2022).

[59] T. Hamilton. 'What is test driven development (tdd)? tutorial with example.' (2022), [Online]. Available: https://www.guru99.com/test-driven-development.html (visited on 30/04/2022).

[60] Microsoft. 'Write your tests.' (2022), [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022#write-your-tests (visited on 10/05/2022).

[61] Create-react-app. 'Running tests.' (2022), [Online]. Available: https://create-react-app.dev/docs/running-tests/#react-testing-library (visited on 06/05/2022).

[62] Reactjs. 'End-to-end tests.' (2022), [Online]. Available: https://reactjs.org/docs/testing-environments.html#end-to-end-tests-aka-e2e-tests (visited on 06/05/2022).

[63] xunit. 'Class fixtures.' (2021), [Online]. Available: https://xunit.net/docs/shared-context#class-fixture (visited on 11/05/2022).

[64] Microsoft. 'Involving the database (or not).' (2022), [Online]. Available: https://docs.microsoft.com/en-us/ef/core/testing/#involving-the-database-or-not (visited on 12/05/2022).

[65] A. S. Jackson, X. Sui, J. R. Hébert, T. S. Church and S. N. Blair, 'Role of Lifestyle and Aging on the Longitudinal Change in Cardiorespiratory Fitness,' *Archives of Internal Medicine*, vol. 169, no. 19, pp. 1781–1787, 2009. eprint: https://jamanetwork.com/journals/jamainternalmedicine/articlepdf/224845/ioi90078_1781_1787.pdf. [Online]. Available: https://doi.org/10.1001/archinternmed.2009.312.

[66] R. Walsh, ‘Lifestyle and mental health,’ *American Psychologist*, vol. 66, no. 7, pp. 579–592, 2011.

Appendix A

Project Description

Oppdragsgiver

Oppdragsgiver:	Headit AS
Kontaktperson:	Rune Kollstrøm
Adresse:	Løvstadvegen 7, 2312 Ottestad
E-post:	rune.kollstrom@headit.no

Kontaktperson oppgave: Magne Johansen, magne.johansen@headit.no

Bakgrunn

Headit AS utvikler unike fag- og innsiktsløsninger som hjelper deg til å jobbe enklere, og ta riktige beslutninger. Vi er et stort og innflytelsesrikt miljø med tverrfaglig kompetanse innen UX, data science, forretnings- og systemutvikling. Vi har hovedkontor på Hamar, og er i dag 33 ansatte.

Lytte, forstå og løse!

Headit har i flere år deltatt i en stafett som heter «Bike & Run», som består av to løpeetapper, og to sykkeletapper. I forbindelse med kommende års løp ønsker vi å etablere en løsning som gir informasjon om hvordan laget ligger an i konkurransen, og når det er tid for veksling.

Oppgaven – Bike & Run

Behovet for en slik løsning er følgende:

- **Mobil app** som leser av GPS-posisjon og rapporterer denne inn til et API. Enheten må kunne knyttes til et lag i stafetten, og etappene må kunne identifiseres. Vi ser for oss både en Android-versjon og en iPhone-versjon som blir tilgjengelig via Google Play og App Store.
  Et alternativ til mobilapp kan være en IoT enhet med GPS og lowband. Eventuelt tid til å finne et system for dette er utenfor scope for oppgaven. Hvis IoT device velges, så er det et krav at det er lav kostnad forbundet med kjøp, for ikke å hindre små arrangementer å kunne tilby tjenesten.
- **Backend-tjeneste**
  - API for å motta posisjoner
    - API-et skal være utvidbart til senere å kunne ta imot meldinger fra andre type device
    - API-et skal være utvidbart til å kunne ta inn andre målinger slik som hastighet, puls etc.
    - API-et skal lagre posisjoner for senere bruk i analyser og resultatservice
  - API for å avlevere posisjoner til frontend-tjeneste
    - API-et skal fortrinnsvis levere push-meldinger om posisjoner for deltakerne. Dette reduserer antall kall mot serveren.
  - API for å støtte admin applikasjonen (web applikasjon)
  - API-et skal distribueres som et dockerimage til driftsmiljø
  - API-et skal fortrinnsvis skrives i Java
- **Frontend-tjeneste**

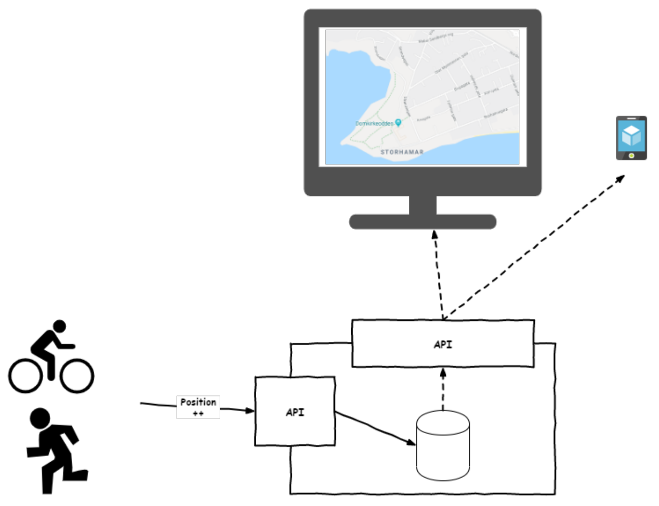
-
    - Fullscreen-kart som plotter posisjoner *near realtime* (push eller pull, valgfritt hvilken variant som velges)
        - Forslag til kartløsninger er Google Maps, Open Street View, eller lignende
        - Det skal være enkelt å se hvor deltakerne er (det kan være opptil 40 lag i konkurransen, med 4 løpere pr lag)
    - En admin-side hvor man kan registrere enkel informasjon som konkurranse, lag og enheter knyttet til laget
    - Frontend-tjenesten skal fortrinnsvis utvikles ved bruk av Angular eller React
- **Visning av tjenesten på løp**
    - Komme med forslag om egnet visningsmedium for løpet
        - Skjerm som er enkel for publikum å se
        - Enhet til å serve frontend (RaspberyPi, PC, eller lignende)
        - Ha i tankene at det ikke er strøm på området, og at det kan være skiftende vær (batteri, vantett skjerm, etc.)
        - Innkjøp av hardware o.l. må være lavkost-varer

Headit vil bistå prosjektet i gjennomføringen av oppgaven. Jevnlige møter og oppgavefordelinger avtales underveis.

Oppgaven er egnet for to til fire utviklere, noe som også vil bidra til en innføring i Scrum-metodikk og team-samarbeid, og vil gi innsikt innen følgende områder:

- Utvikling av app
- Database
- Java/Rest/Frontend
- Utviklingsverktøy
- GPS kommunikasjon
- Rådgivning av teknisk utstyr
- Sikkerhet i rest API-er
- GDPR (Personvernforordningen)

Skisse over løsningen



Appendix B

Project Agreement

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt:
Veileder ved NTNU: e-post og tlf.
Ekstern virksomhet: Ekstern virksomhet sin kontaktperson, e-post og tlf.:
Student: Jørgen Eriksen Fødselsdato: 10.07.1993
Ev. flere studenter[1] Elvis Arifagic 24.08.1999
Markus Strømseth 04.09.1996

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 11.01.2022
Sluttdato: 20.05.2022

Oppgavens arbeidstittel er: Bike & Run

[1] Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[2]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

[2] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

X	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene: Headit AS skal bruke systemet og drifte systemet ved endt prosjekt.

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: Dato:
Veileder ved NTNU: Dato:
Ekstern virksomhet: Dato:
Student: Markus Strømseth Dato: 28.01.2022
Ev. flere studenter Jorgen Eriksen
Elvis Arifagic

Appendix C

Project Plan

Project plan

Jørgen Eriksen
Markus Strømseth
Elvis Arifagic

26.01.2022

1. Goals and Constraints

1.1. Background

This bachelor's project was provided by Headit AS, and our contact person from the company is Magne Johansen. Headit is a IT- and software-company based in Hamar.

Headit have for the past several years participated in a relay race called Bike & Run. The race is hosted by Bedriftsidretten Innlandet as a low threshold public health event. This race consists of two running stages and two biking stages. In conjunction with this year's race, Headit wants to have established a software solution for informing participants and audience of how the race is progressing. With real-time updates on the runners and bikers to help with tracking teams and preparing for exchanges. The previous races have not had any digital aids to help facilitate the race and as Bedriftsidretten Innlandet is an independent membership organization with focus on public health for everyone the threshold for price range and equipment need can't be too high as this would be in direct conflict with their goal for the race.

Headit is interested in developing a holistic solution that would help facilitate the race from multiple perspectives. They want to be able to organize the relay and to be able to reflect on the data at a later date. All of this will provide a better public health offer. Being centered in this type of action would help Headit establish itself more solidly as a company that is forward thinking about societal issues such as public health.

1.2. Project goals

The goal for the project is to create a backend/API service, frontend service for displaying the race on a map, a frontend service for admin application, a mobile app or IoT for the participants, and to come up with a recommendation for a display medium.

As discussed with the project owner, the most important part is to get the backend finished to a deployment ready stage, so that will be our main goal and priority. Our end goal will be to finish all parts of the projects and not just the backend. The project is quite extensive so there is a possibility that we will not finish all parts, if that happens the code should be easy for the project owner to continue working on it. That means that the code for the unfinished parts should be easy to read and with comments, while following typical coding patterns and file structure within the frameworks.

1.2.1 Result goals

- API that receives coordinates from the team/contestants mobile devices.
- Cross platform mobile application that sends the coordinates to a server.
- The mobile device should report coordinates at **minimum 15 second intervals**.
- A front end page for viewing the race in real time
- An admin site for administering the relay race (adding teams, adding relays…)

1.2.2 Effect goals

- Quantitative goals

Reduce the time needed to setup the race by 50%
Increase the participation rate by 25%.
Create an offer for viewing the relay data post race.

- Qualitative goals

Increase the number of people participating in the relay.
Create a better environment of running the relay for the people managing it.

1.3. Constraints

1.3.1. Time Constraints

The finished project plan and signed work contract has to be delivered before 31st of January 2022.

Finished product along with the finished bachelor thesis needs to be delivered before 20th of May 2022.

1.3.2. Legal constraints

The overall software must conform to the General Data Protection Regulation (Personopplysningsloven, 2018) as some users will store personal information.

2. Scope

2.1. Subject Area

This project includes subject areas that we are familiar with from our study programme, but also some core subjects that are new. It includes concepts and more advanced subjects within the familiar subject areas that are new to us that we will have to research and learn while developing. In total the project will provide insight into the following subject areas:

- App development
- Database
- REST API
- Frontend development
- GPS communication
- Development tools
- Consulting of technical equipment
- Security in API
- Cloud technology
- Scrum
- GDPR

This project is related mostly to our previous courses PROG2005 Cloud Technologies, IDATG2204 - Data modeling and database systems, and PROG2053 - Web Technologies. Also the competence related to operations with coordinates and vectors in two dimensional from PROG2002 - Graphic Programming and BMA1020 - Mathematics for Programming will be relevant.

We will use the following technologies:

- HTML, CSS and Javascript
- React and React Native
- C# .NET 5 (Entity Framework Core)
- SQL
- Azure
- Git
- Docker

We will also use different libraries within the code. These will be managed with package managers like NPM for React and React Native, and NuGet for .NET. The decisions of technologies are made together with Headit where we discussed the different options. The decisions are based on their prefered technologies, some of our competency within the different technologies. Technologies decisions that might happen under development will also be discussed with product owners to ensure that they are satisfied with our choosing.

Relays such as this often happen in difficult conditions which means that it is something we have to consider when choosing technology. IOT devices could be a more attractive choice rather than having participants carry their expensive phones with them. They might fall off a bike, stumble running on a trail and break their phone. But considering the added overhead of an IOT device in terms of price, maintenance and lack of programming experience, phones seem to be the better choice.

2.2. Delimitation

For the project we are tasked with expanding the API in such a way that the API can at a later time be expanded to include data other than the one we will directly deal with. It is stated that we should only make it so that this can happen at a later date, but the team itself will not add those features.

All finished parts of the project should be finished for the deployment stage, but we will not handle the actual deployment.
The deployment of this service is to be done by Headit themselves. Our work is meant to come to the point where they are able to deploy it without any additional coding required.

As a minimum phones that are to be used together with this system have to be at least running iOS 11.0 and Android 5.0 (version 21) as per React Native documentation (Microsoft, 2021).

2.2.1. Hardware and Software Delimitations

We will develop server support for both iOS and Android mobile devices as GPS trackers. Other smart devices will not be supported although we will structure the REST API to be scalable in the sense that it would require minimal work for integrating support for other smart devices.

The final product migration to Headit will be handled by the Headit to their respective internal servers. The final product is not expected to exceed the deployment stage.

2.3. Task description

Our goal is to develop a backend/API service, a frontend service for displaying the race on a map, a frontend service for the admin panel, a mobile application or IoT application for the participants, and to come up with a recommendation for a display medium.

The backend/API service should be connected to a database that saves positions from the participants, both for real-time race and to save the race as a history. The API should be able to push participants' positions to multiple applications, support the admin application, and be ready to be distributed in a docker image. The API should be expandable for future work after we have delivered the project so it can be implemented with features from the participant device like speed and pulse, and also from other device sources.

*The frontend service for the admin panel sh*ould be able to manage races, with teams and number of participants per team. It should also display invite code for each position for each participant in each team.

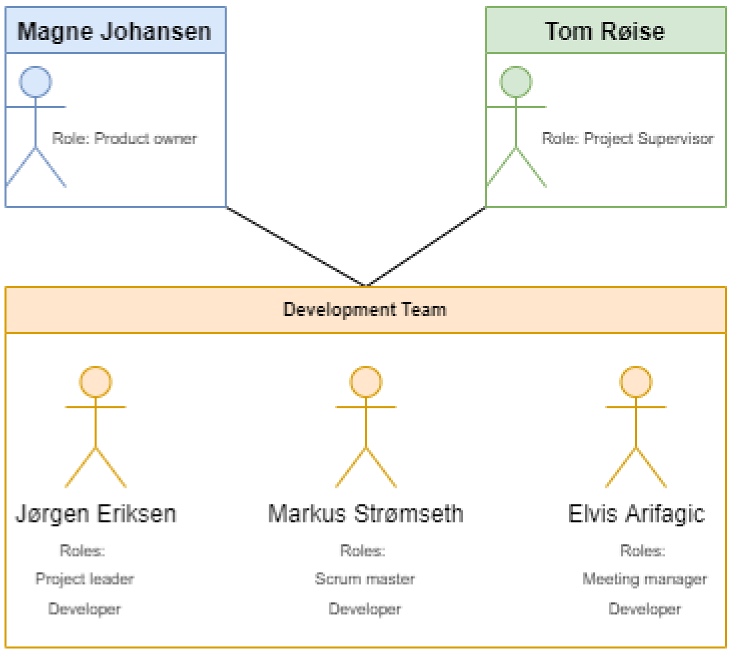
The frontend service for displaying the race should display each participant in the race. The race could have up to 40 teams with 4 participants each. The frontend should be able to be in fullscreen.

The mobile application or IoT application should read GPS positions to the participant and send it to the API. The device must be able to be linked to a team, and each stage of the race should be identifiable. If the solution is a mobile app it should be available on both Android and iPhone. And if the solution is an IoT device, it must have GPS and low band, while also being low cost.

The recommendation for a display medium is meant to run the frontend to display the race. The medium should have a display for the audience to see. Should serve the frontend with a Raspberry Pi, a PC or similar types of distributors. We have to take into account that there is no power on the site and the multiple weather conditions that can happen, so things such as battery and waterproof display etc are relevant. The solution should be low cost hardware.

3. Project Organization

3.1. Responsibilities and roles



Product owner is our communicational link to our employer for this project. Weekly meetings are required with this person and we will ask him questions directly relating to the assignment.

Project supervisor is responsible for making sure that progress is being made. Tom Røise will give us feedback and check in with us weekly to see what progress has been made in the project.

Team leader is the person with the veto power in the group should there arise a conflict in any domain. He is responsible for the overall leading of the project as a whole.

Scrum master is tasked with the running of the sprints and meetings related to the project. They are also responsible for usage of branch protection rules, issues and their quality.

Meeting manager takes meeting minutes and makes sure that notes are taken for all meetings with the project supervisor, product owner and internal meetings.

3.2. Routines and rules

3.2.1 Group decision making

Group decisions with regards to the full development and project process will be decided based on democratic elections. With conflicts or discussions the group leader Jørgen

Eriksen will have veto rights as to what outcome of choices will be decided. In general the group should practice respect towards one another and respect the decisions that are made.

3.2.2 Meetings and schedule

As a part of our team's routine we have set up a minimum of two meetings per week. First meeting is with the Headit contact every Tuesday morning and the second is with the supervisor on Thursday mornings.

Internal meetings in the group will be scheduled ahead of time and meeting summons will be sent out a minimum of two days ahead of the meeting. Other than that these meetings can happen at any day of the work week.

All group members are expected to attend every meeting unless absence is clarified with the rest of the group.

3.2.3 Workload and working hours

The group members are to be available during working hours 09:00 to 15:00 every day with room for flexibility. When a group member deviates from being available during working hours this should be communicated to the rest of the group.

Each member is expected to work more or less depending on the workload an average of 30 hours per week.

In cases where a team member is not contributing either consciously or not, there will be formal warnings given to the person as well as a copy of the same warning to Tom Røise. External help from the department at NTNU will be utilized should the project suffer greatly and possibly come to a stop by a problem of this type.

4. Planning, Followup and Reporting

4.1. Development process

4.1.1 Project characteristics

The time given for this project is from the 11th of January to the 20th of May. Considering that writing the thesis has to occur during this same timespan, time is a bit short for development of the software itself.

Headit has given us some semi-loose requirements specification seeing as for example the backend API was intended to be written in Java but we got that changed to rather be written in C# with the EF Core framework in .NET instead. Another example would be the authentication where the strategy for that might be in a state of flux for a while until it's completely decided.

Overall the team feels confident in the project success seeing as the flexibility given allows us to choose to work with technologies that we are familiar with such as React and .NET. The more arduous part of the project is it’s size, there are alot of things to keep in the air at the same time if one is to consider the report as well (meeting minutes, time logs, documentation, writing, documenting scrum usage).

Taking factors like this into consideration we are choosing to go with an agile scrum methodology for this project. There were considerations of using kanban instead but seeing the size and scope of the project the team feels as though we need to limit ourselves a bit more and have specific tasks that have end dates. We think a more rigid structure is going to help with discipline in the long run of the project.

Alternative to scrum would be Kanban which we heavily considered. Kanban is a more loose arrangement where tasks are put on a kanban board. The team is then free to choose whichever tasks they feel like working with as well as there being a flat hierarchy in the team. This was attractive to us because we had recently practiced scrum and felt it was a bit restrictive and it had the possibility of providing a developer with a task they did not enjoy working on.

Problems arise when considering kanban as the bachelor project is a much larger, more serious piece of work that we think requires discipline which we think we can achieve by limiting ourselves through scrum. Also having a hierarchical structure frees up developers to think more about their work tasks and not external things that might impact their work.
The aforementioned factors heavily influenced our decision to go with a scrum methodology for working on this project.

4.1.2 Utilizing SCRUM methodology

In our project Markus has been chosen as the scrum master for the duration of the entire project. The product owner is the contact person from HeadIt. We will have a retrospective look after two weeks. Sprints will start on Tuesdays and last two weeks until the next Tuesday where we will conduct sprint review, sprint retrospective and sprint planning for the next sprint, which will then in turn start that same day. During sprint planning we will derive a sprint backlog from the project backlog with tasks expected to be completed in the forthcoming sprint.
We will run a scrum board on Trello which is something we are more familiar with in the team. We will use labels and have a board division of

- Sprint backlog
- Backlog
- Doing
- Review
- Done

Labels have to be used by each team member for purposes of analyzing the data for thesis writing. Also to have a clearer view of what people are working on.

Issues will also use checkmark tasks which are tasks inside of an issue that go from 0 percent to 100 percent and help make issues clearer to outside viewers that aren't necessarily working on the same issue.

The Product owner from Headit will participate in the tuesday sprint review together with the team.

4.2. Plan for meetings and decision points

Monday	Tuesday	Wednesday	Thursday	Friday
Workday.	Sprint planning, sprint start. Status meeting with Magne.	Workday.	Workday. Status meeting with Tom.	Workday.
Workday	Sprint review, sprint retrospective with Magne.	Workday.	Workday. Status meeting with Tom.	Workday.

5. Quality Assurance

5.1. Documentation, Standards and Development Environment

For development of the front end services we will be using VS Code which has great integration for extensions which support front end development. Backend work with the APIs will be done through Visual Studio 2019.

For backend APIs in C# we will follow the convention of XML documentation as per the Microsoft documentation. Visual studio has integration for this feature which makes this a favorable option.

Documentation for frontend functions and React will follow a doxygen style of commenting. There are no common conventions for solving this so we will use whatever we think best represents our software and the quality.

The team has an expectation that team members commit frequently and create branches for features. Commit messages have to clearly state the work done specifically to this commit and avoid messages that are short and give no context to the reader.

We plan to use a Bitbucket instance of Git version control provided to us by Headit where we will have four repositories giving modularity and reducing clutter that would come with one repository. Currently it will be the backend, mobile application, frontend and admin panel.

We will take screenshots of the scrum board at start and end of each sprint to be included in the thesis.

5.2. Source code

The source code for this project is proprietary to Headit. Access will be granted on a contract basis. Tom Røise and others that need will be given access to the code.
All functions written in both frontend and backend need to have a comment describing what the function does. Other code should be commented in such a way where the “why” in the code is clearly conveyed to the reader.

For frontend services we will follow modern React practices which starkly contrast earlier ways to work with React. There are not any guidelines we will follow specifically for writing the javascript code itself as it’s a dynamic language and is very flexible in how things can be written with no performance cost between the alternatives.

In the backend which is written in C# we will follow the best practices laid out by Microsoft in their documentation. There is a comprehensive guide in how to write the most performant C# code which we will refer to periodically in development to use a guide when reviewing code.

5.3. Plan for Testing and Test Environment

For the backend we are planning to set up unit testing for .NET, which will test individual parts/components of the code to check if the result is expected, both with valid parameters and unvalid parameters. Methodology wise we will use AAA which is Arrange, Act and Assert. This is a way to set up unit tests to increase their readability and efficiency.

By using the create-react-app feature that is supported by facebook the projects come with Jest as a test runner and react-testing-library as a primary module for writing the tests themselves. Jest grants us access to the DOM via jsdom which enables the testing of the react components we create.

We plan on facilitating integration testing as soon as the core features of the full pipeline are developed. Midway in the development process, at the end of sprint 3 we will conduct full integration user tests with Headit employees or other potential users of the service.

5.4. Risk Analysis

For the risk analysis we will lay out a table describing the risks and their possible outcomes and then have a separate strategy section where we in detail describe how we would solve each of the risks. These mitigation strategies will help us should any of the risks arise in the actual bachelor project.

There are possibilities of other risks that could rank lower than the ones we have mentioned but we feel as though only the most critical ones where a clear strategy can be defined are worth mentioning.

5.1.1 Breakdown of project risks

Risk Number	**Risk Description**	**Risk severity**	**Risk Outcome**	**Risk mitigation**
R-1	Server load is too large.	2	With the load too high servers will stall and be unable to function properly.	Decrease the number of calls and do stress testing to figure out the minimum and maximum bounds of data flow.
R-2	Unable to perform proper testing of backend	3	Limits the amount of confidence in the code and makes maintenance more difficult.	Go through channels available to us to help us and be more disciplined about writing unit tests right after writing the code.
R-3	Authentication is lack luster on front end	3	Headit will be unable to use the software without modifications because login to admin site will expose information sensitive to GDPR regulations	Scope down from using OAuth2 and use a simpler solution while retaining safety.
R-4	Unable to create docker image of API	2	Headit will be unable to deploy the software in a manner that was expected. They will have to do it themselves or change strategy.	Perform early test deployments of a docker image locally to go through the process. Making sure to document the steps along the way.
R-5	Technological risk	2	Tools under use in the project perform updates breaking the current way the system functions. Necessitating modifications to function properly.	In most cases software can either be modified to not update automatically or we can choose to skip the update to avoid breaking changes.
R-6	Incomplete sprint work	3	Functionality will come at a slower pace than planned, increasing the risk for build up which can spiral out of control resulting in lacking features at the end of the project.	Assign tasks based on the strength of the team. Inform Headit about the status of each sprints asking for help deciding about more vital parts.

5.1.2 Risk mitigation strategies

Risk 1:
In cases where the load is too high we need to measure the throughput of the api firstly and determine the limit. One way to do this would be to stress test the system with high data flow. Running a profiler over the system in visual studio while it is working would let us know any hotspots (pieces of code that are doing a lot of work). Knowing the hotspots you could analyze further how one could optimize the code.

Risk 2:
Headit has some experience writing dot net code so we assume they also have the experience of writing tests so we will contact them for guidance and direction. In an extreme case where no unit tests are being written we should halt development of all other parts of the project and focus on all resources of figuring out the problems associated with the test writing. To mitigate this we should also rely on internet sources since dot net is a very widely used framework in the industry such that the likelihood that good information exists about how to do something is very high.'

Risk 3:
In a case where OAuth2 is not working, either because of our lack of experience with it or any other factor, it is possible to scope down and use something else. There are many other authentication protocols that exist and other ways to do authentication. If authentication though the backend does not work we will try a backend that is done through javascript instead. We can use express for the backend server and there are many packages available through npm package manager for authentication which make it very easy to do even with OAuth2.

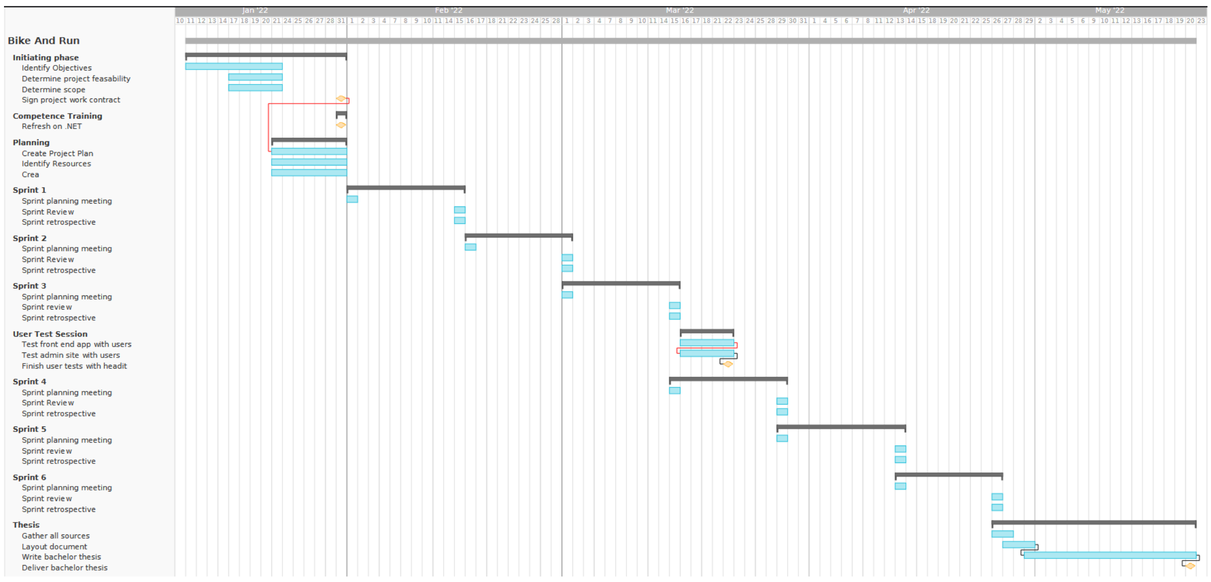
Risk 4:
We have done some docker work earlier which we can refer to. That deployment was also a cloud one so we could use that dockerfile to look how we might be able to setup something similar here. Docker has extensive documentation so another plan of action would be to carefully review the documentation and carefully assemble the correct dockerfile.
Headit also currently has some systems deployed with docker so we could also inquire our product owner for some advice on any eventual issues.

Risk 5:
Oftentimes tools have an option for auto updates. We will also practice an in-house rule of not updating the software we started the project with. So for example we will not upgrade to use .NET 6 instead we will stick with 5 which still is under long term support and is a safe choice for developing software with it.

Risk 6:
If we fall behind on our sprint work we will implement more deadlines making sure that progress is more easily viewed by the team hopefully boosting the work morale internally. In a rare case where work will not get done we will inform the product owner of the situation hopefully starting a conversation about which things should be re-focused to be more important than others.

6. Development Plan

6.1. Gantt Diagram



6.2. Activities, Milestones and Decision Points

6.2.1 Milestones

1. 31.01.2022: Competence development and Research.
2. 31.01.2022: Project plan and agreement between Team and Headit.
3. 15.02.2022: Sprint 1 complete.
4. 01.03.2022: Sprint 2 complete.
5. 15.03.2022: Sprint 3 complete. User test session.
6. 29.03.2022: Sprint 4 complete.
7. 12.04.2022: Sprint 5 complete.
8. 26.04.2022: Sprint 6 complete. Development ceases.
9. 20.05.2022: Hand in bachelor thesis.
10. TBD: Project presentation.

Bibliography

Personopplysningsloven (2018) *Lov om behandling av personopplysninger.* Available at: https://lovdata.no/dokument/NL/lov/2018-06-15-38/*#&#x2a (Accessed: 31. January 2022).

Facebook (2021) *React Native*. Available from: https://github.com/facebook/react-native/blob/main/README.md#-requirements (Accessed: 31. January 2022)

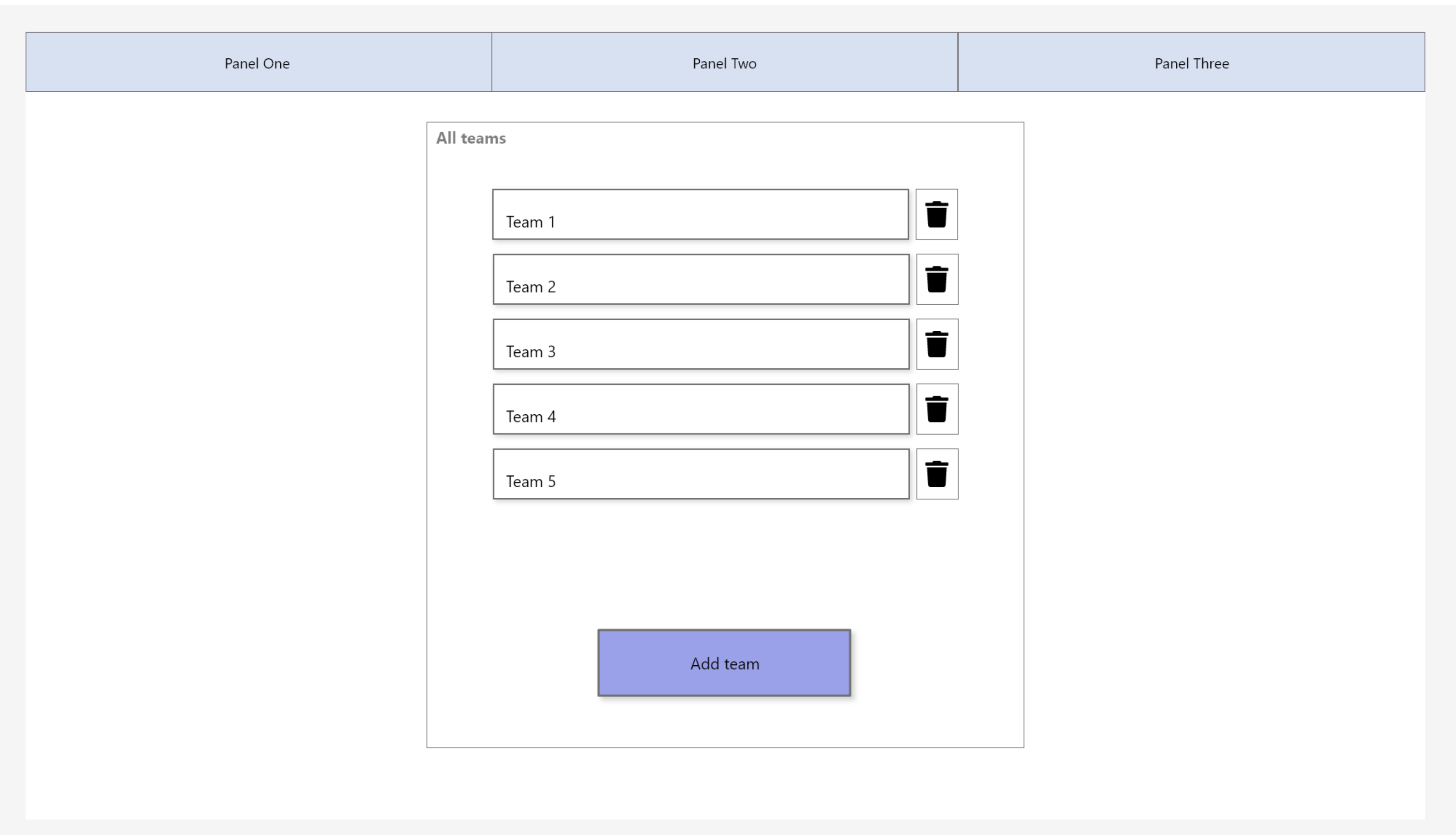
Risks (2021) 7 *Common project risks* Available from: https://asana.com/resources/project-risks (Accessed: 29.01.2022)

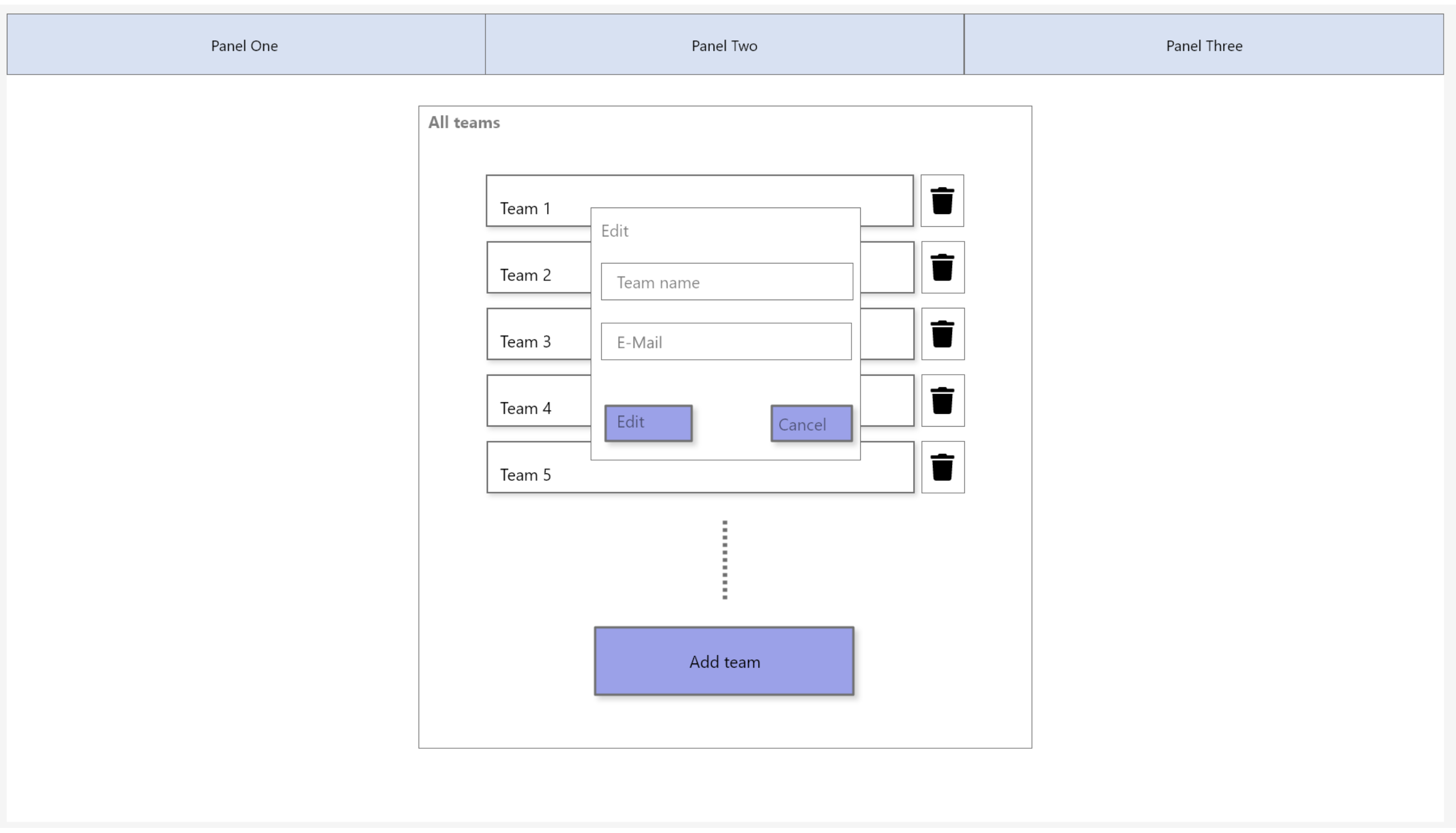
Risk analysis (2008) *Risk analysis and management* Available from: https://www.pmi.org/learning/library/risk-analysis-project-management-7070 (Accessed 30.01.2022)

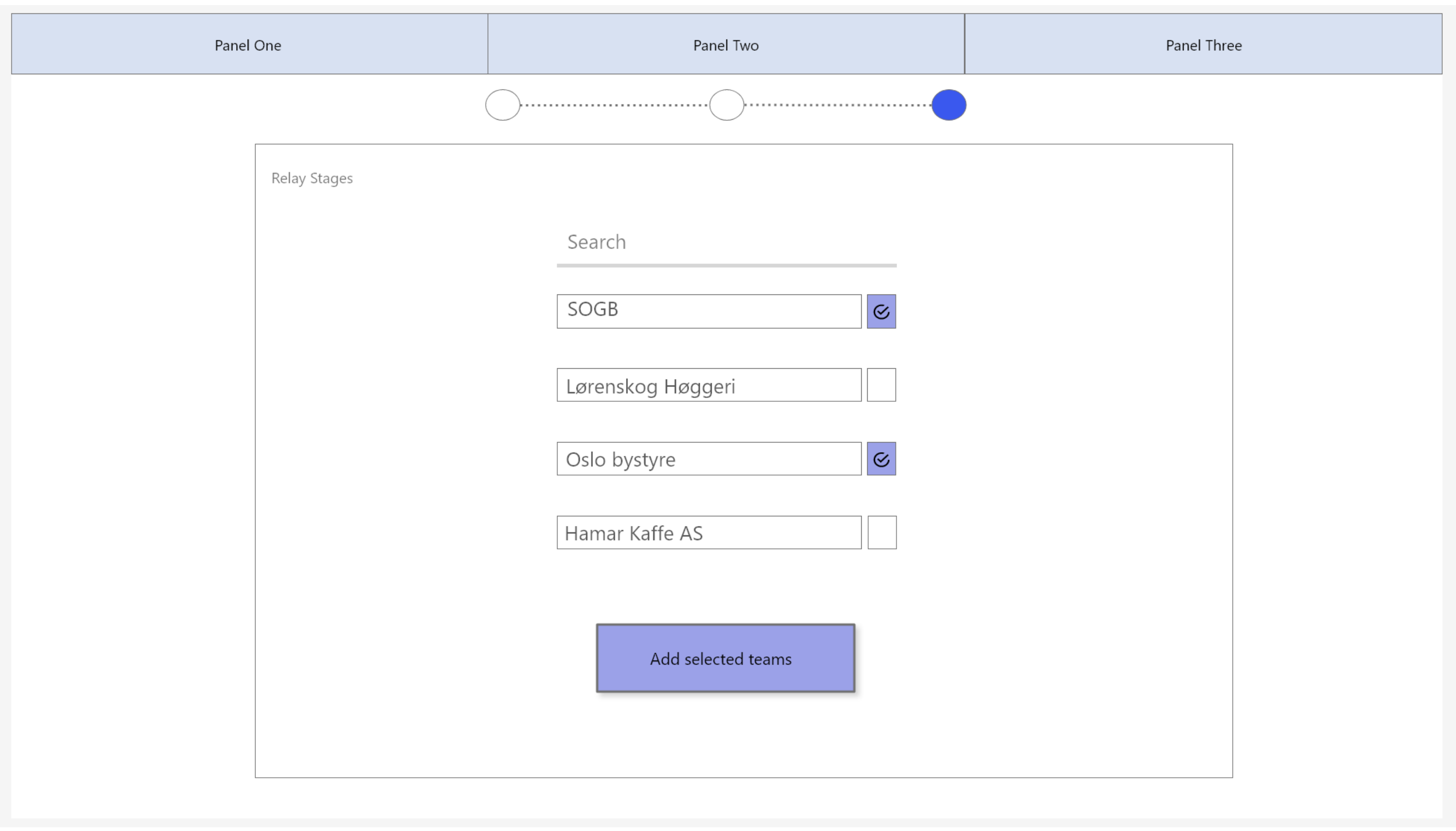
Hva er kanban (2019) *Kanban vs Scrum* Available from: https://www.prosjektbloggen.no/hva-er-kanban (Accessed 30.01.2022)

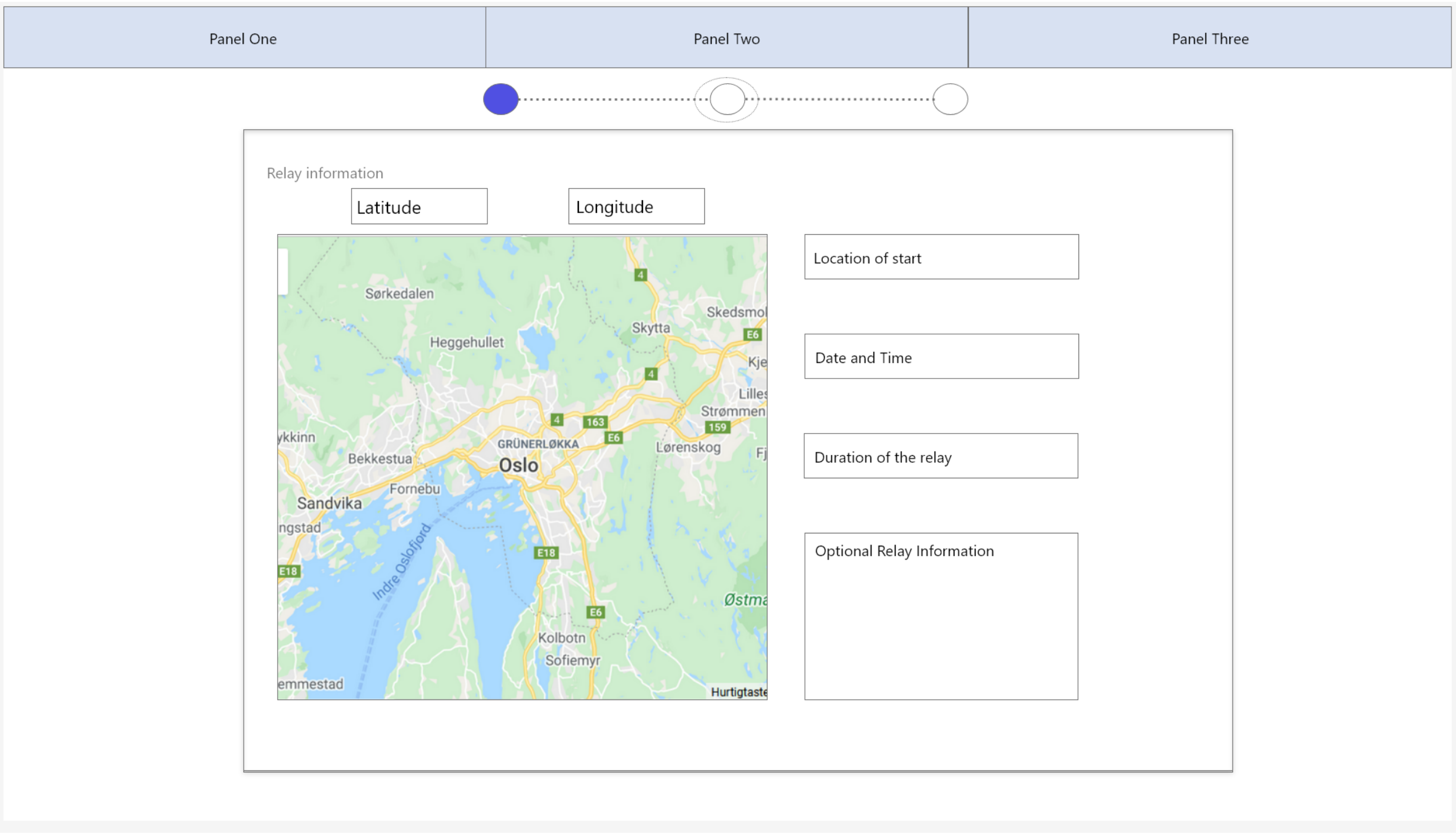
Appendix D

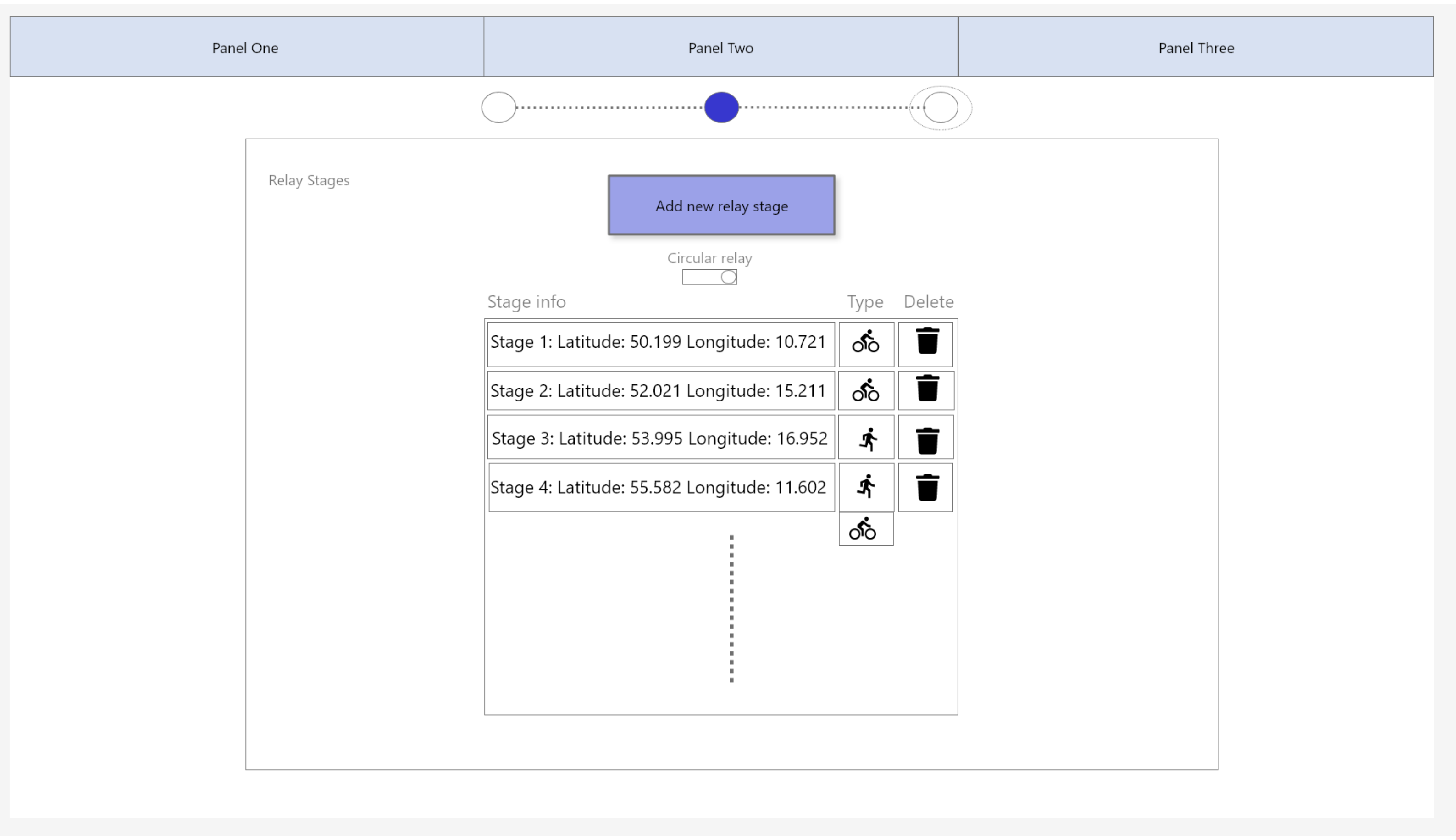
Admin Panel Wireframe


Panel One
Panel Two
Panel Three
All teams
Team 1
Team 2
Team 3
Team 4
Team 5
Add team


Panel One
Panel Two
Panel Three
All teams
Team 1
Team 2
Team 3
Team 4
Team 5
Edit
Team name
E-Mail
Edit
Cancel
Add team


Panel One
Panel Two
Panel Three
Relay Stages
Search
SOGB
Lørenskog Høggeri
Oslo bystyre
Hamar Kaffe AS
Add selected teams


Panel One
Panel Two
Panel Three
Relay information
Latitude
Longitude
Location of start
Date and Time
Duration of the relay
Optional Relay Information
Sørkedalen
Heggehullet
Skytta
Skedsmo
Strømmen
Lørenskog
GRÜNERLØKKA
Oslo
Bekkestua
Fornebu
Sandvika
Indre Oslofjord
Kolbotn
Sofiemyr



Panel One | Panel Two | Panel Three

Relay Stages

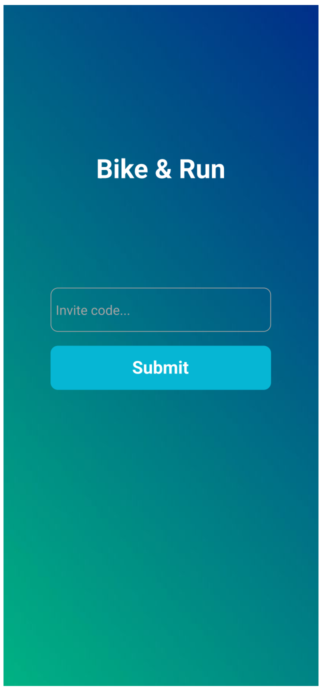
Add new relay stage

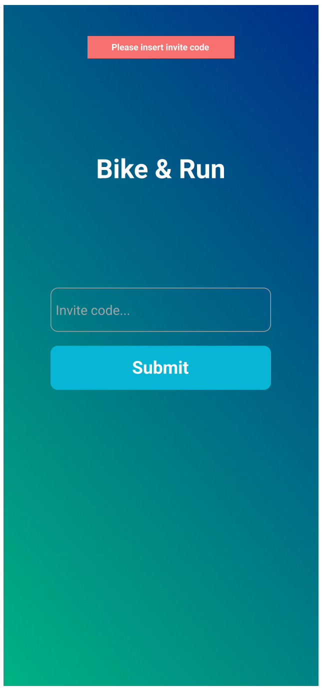
Circular relay

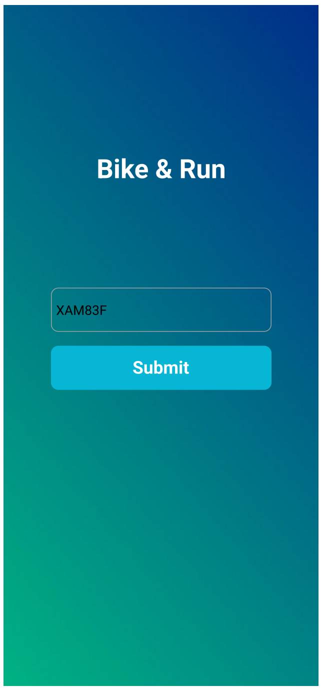
Stage info	Type	Delete
Stage 1: Latitude: 50.199 Longitude: 10.721	🚴	🗑
Stage 2: Latitude: 52.021 Longitude: 15.211	🚴	🗑
Stage 3: Latitude: 53.995 Longitude: 16.952	🏃	🗑
Stage 4: Latitude: 55.582 Longitude: 11.602	🏃	🗑
	🚴	

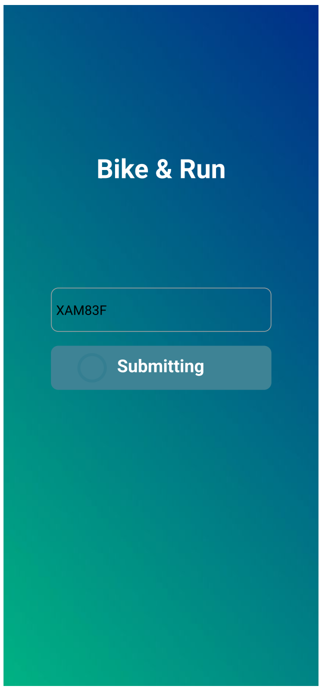
Appendix E

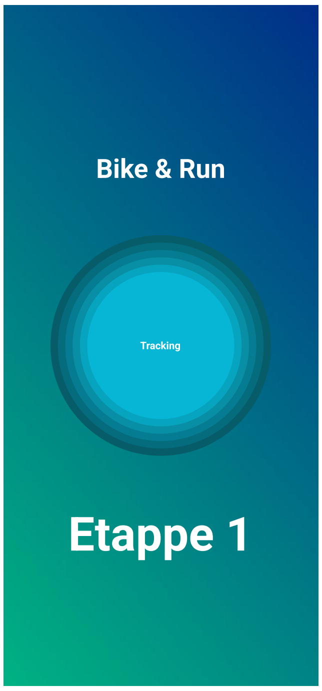
Mobile Application Prototype

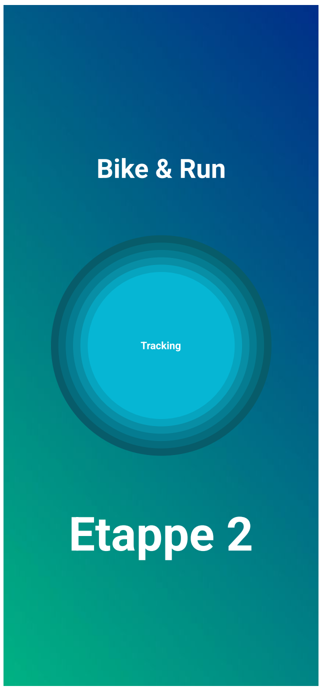

Bike & Run
Invite code...
Submit


Please insert invite code
Bike & Run
Invite code...
Submit


Bike & Run
XAM83F
Submit


Bike & Run
XAM83F
Submitting


Bike & Run
Tracking
Etappe 1


Bike & Run
Tracking
Etappe 2

Appendix F

Testing

BIKE & RUN

Bike & run user test

Jørgen Eriksen, Elvis Arifagic, Markus Strømseth
User test

Table of contents

Introduction

This document pertains to a user test performed as part of a bachelor’s project at NTNU Gjøvik. The task is given by Headit which is a software consulting company located in Hamar. This document is meant to be a supporting appendix document to the bachelor’s thesis. In this document we describe the process and findings that we found during our user test that we performed on location for Headit.

For this test we wanted to focus on the administrator panel and the mobile phone seeing as those two components of the Bike & Run system are the ones that are to be used by users directly. Other components are the API and frontend neither of which have any facilitates that enable user interaction.

In this document we will discuss the demographics of the participants, but we do not expose any information that can be used to track any personal information about the individual participants. This is to comply with the GDPR rules that govern the country that the test was performed in.

Demographics

For this test we are testing with 3 people. Below you will find a table with their basic information.

ID	Age	Gender	Occupation	Technological Experience
1	48	Male	Software developer	Very high
2	59	Male	Software Project Lead	Low
3	26	Female	UX-Designer	High

We will hereby refer to specific participants by their column ID value.

The demographics were chosen beforehand by the product owner that is working with us on the project. We did know who they were before we met them.

Having this factor in mind, the team requested specifically to have a UX-Designer participants to have a post-test discussion with them about the design.

We also wanted varying degrees of competency as one of the requirements for the pool of user tests. To figure out if the more modern design choices on the website would work with such a user, we had a participant that considered himself low in competency with technology in genera.

User tasks and Execution

The tests were performed at an unnamed office building room with a table and a television. Users were given one computer to perform their tasks on. That same computer was connected to a large TV using a HDMI cable and was used by the team to be able to view their actions. We did not want to make them feel rushed which could happen if we were sitting next to them.

For the mobile app users were asked to download the Expo Go app which is needed to run the application. They were then to scan a QR code that was one the screen of the laptop they had just used, and then they were given an invite code to enter by the team.

Jørgen Eriksen was responsible for verbally communicating the tasks to each of the participants and answering their questions should they have forgotten what they are supposed to do. Elvis Arifagic was taking notes of their actions, viewed on the TV. Notes about their difficulties and questions participants might have raised during the test itself. Markus Strømseth was tasked with IT help in case of any malfunction from the software side.

As part of the test's users were asked to do the following

- Lage et relay som starter ::::: ::: :: ::: : :

For this part we wanted to see how users would fare from the start of the website and then completing what is the most difficult task. The team wanted insight on how they would navigate to the tab, how they would input fields required for relay data and if they would remember to click the button to send the relay.

For the mobile application part users were given an invite code and they had to the input that invite code. The mobile app does not support more actions than that other than logging out and inputting a different invite code to the same or a different relay. We wanted also to find out whether users would try to do something else if the intention of the mobile app was confusing or not.

Findings

Tester ID	Findings
1	● Able to find location on google maps display just fine ● Managed to choose stagetype ● Sendt the relay to the database and viewed the invite codes ● Noted that the invitation code site gives not enough info ● Copy button was a little bit hidden for invite codes ● Missing a button for sending invite codes with e-mail ● Feels that the map is unclear ● Commented that the map should be closed and should be opened by clicking a button ● Wants to be able to search by adress
2	● Spent a lot of time clicking around trying to find the right pane to create the new relay ● Did not manage to add time ● Unable to efficiently navigate on google maps display ● Enjoyed the stepper. ● Commented that stagetype buttons needed to look more like buttons. ● Did not delete two stages as asked
3	● Feels as though the Textfields are slow and lagging. ● The minutes for time selection was not clear. ● Time is difficult to figure out how works with the clock. ● Did not modify the location ● Stage was unclear that it was a list. ● Did not add name of relay in the first stepper page. ● Wanted to be able to send a mail right away. ● Commented that all the lists are very unclear that they are clickable lists. ● Mentions she feels names should pop up in the after-creation screen.

Plans to address findings

Based on our findings on the user test we can see that the website is not as user friendly as we had thought it was. The clock that we have used is too complicated for users and we believe that just two input fields might be a better solution. Furthermore, we need to look at how we styled the lists. We need to make the edges more visible and more clearly communicate that each item in a list is clickable.

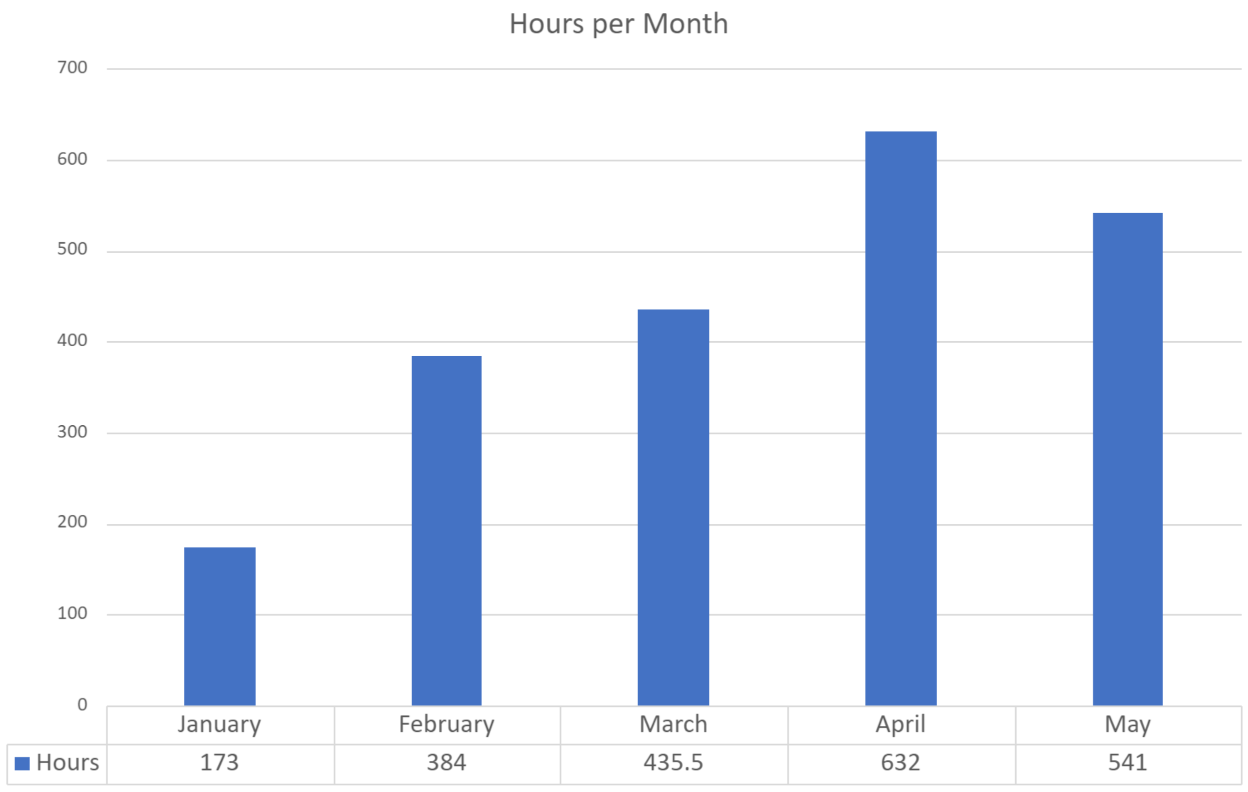
To see exactly how and which things we tackled we refer you to the bachelor project which can be found at NTNU Open website, the thesis is called “Bike & Run”. In there you will find a detailed testing section where you can view more information about the user test.

Other notices

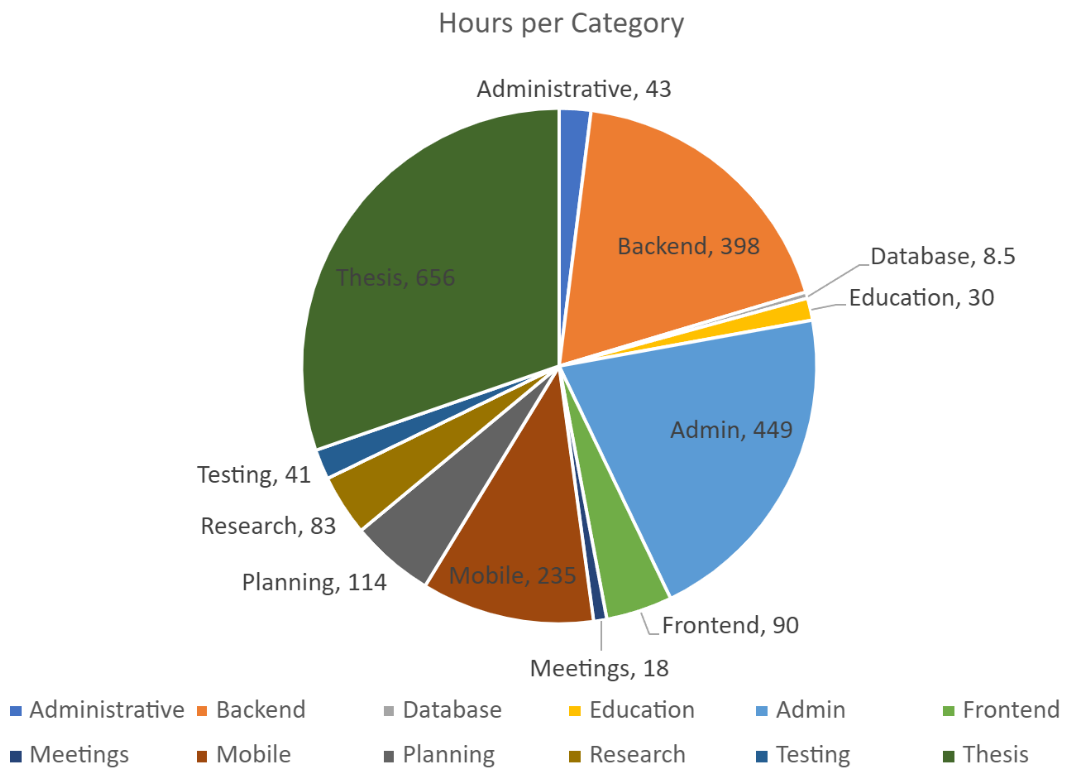
- User 3 was confused during testing and thought that the admin panel was a panel that was meant to be used by users. This was our mistake for not clarifying. We thought the name would give the context, but it was later clarified to her that it was not meant for general users but only admin users.
- After the official test was over, the room was open for anyone in the office building to come in and try the app. This was not official, and we were not paying deep attention to how people used the app, so we did not take notes of this but rather just had conversations with people about the product.

Appendix G

Time Allocation



	January	February	March	April	May
🟦 Hours	173	384	435.5	632	541



Appendix H

Meeting Logs

H.1 1st Meeting

Date: 11.01.2022

Duration: 45 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: First meeting with product owner.

Content: Short descriptions of what we did:

- Introductions.
- Discussing the freedom within the limits of the task.
- Talked about how we are planning to work.
- First couple of weeks are dedicated to working on project plan.

H.2 2nd Meeting

Date: 13.01.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: First meeting with project supervisor.

Content: Short descriptions of what we did:

- Discussion about SCRUM, how we should have meetings and what parts of scrum to use.
- Talking about the project plan and getting ideas on how to structure it.
- Need to sign the work agreement for the project physically.
- We need to set clear rules about expected work and where we work.

H.3 3rd Meeting

Date: 18.01.2022

Duration: 60 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Talking about development and the project plan deliverable.

Content: Short descriptions of what we did:

- Talking about .NET and compatibility with SQL databases.
- PO showed us a Visual Studio extensions for interfacing with Azure through Visual Studio called SQL server explorer.
- Talked about the expectations for WebSockets. How those are supposed to work.

H.4 4th Meeting

Date: 20.01.2022

Duration: 45 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Discussing the project plan that we gave him to look over.

Content: Short descriptions of what we did:

- Entire meeting was PS giving feedback and us discussing the feedback.
- Describing the issue, we had with the role of the product owner. PS cleared this up. Next meeting, we will talk to PO about this.

H.5 5th Meeting

Date: 25.01.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Talking about PO role in SCRUM and database design.

Content: Short descriptions of what we did:

- Show PO some of project setup of the admin panel and frontend.
- Discuss database design with PO to get a picture of the domain.
- PO shows us SQL server management studio 2019.
- Talk about how PO is to be included in SCRUM execution for the project.

H.6 6th Meeting

Date: 27.01.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Show PS the progress on project setup.

Content: Short descriptions of what we did:

- Discussed the project setup and progress.

H.7 7th Meeting

Date: 01.02.2022

Duration: 60 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Demo the WebSockets to PO and frontend app. Also show the prototype of admin panel.

Content: Short descriptions of what we did:

- Jørgen shows PO the progress in websockets by sending a simple message between backend and frontend.
- Discussion about how exactly “push” should work in our application.
- PO brings up a point that we need an architecture diagram.
- Another discussion was had about the database design, perhaps we need a table between relay and participant tables.
- Talking about how work should be divided in the group. PO thinks everybody should work a bit on everything.

H.8 8th Meeting

Date: 03.02.2022

Duration: 20 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Demo the systems so far to PS.

Content: Short descriptions of what we did:

- During the meeting all that occurred was showing the PS the state of the various applications that make up the system.
- Showed PS the WebSockets and frontend. The design and start of implementation of the admin panel.
- Discussed briefly about the phone app, talking about cross platform solutions we had thought about.

H.9 9th Meeting

Date: 08.02.2022

Duration: 45 minutes.

Location: Online via MS Teams.

Participants present: Elvis Arifagic, Jørgen Eriksen and Product Owner.

Absence: Markus Strømseth (Job Interview).

Agenda: Clear up relay specifics and transferring to bitbucket.

Content: Short descriptions of what we did:

- Explaining the algorithm, we came up with for stage changes in the relay. We plan to use a radius and then check the coordinates against that.
- Explaining that the race is started by a timer, not a person. It is defined in the relay data structure itself.
- Pipeline automated testing is possible with bitbucket.
- We need a max time in the API for the push websocket timer.

H.10 10th Meeting

Date: 10.02.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Update PS on new developments.

Content: Short descriptions of what we did:

- Discussing with PS how we are closer to a solution for how the relays should work logic wise. He agrees with us that radius-based swapping sounds like a good idea to pursue.
- PS talks about how we need to document our choices and perhaps we should investigate technical memos to solve this problem.

H.11 11th Meeting

Date: 15.02.2022

Duration: 45 minutes.

Location: Online via MS Teams.

Participants present: Elvis Arifagic, Jørgen Eriksen and Product Owner.

Absence: Markus Strømseth (Job Interview).

Agenda: Show demo to PO of frontend and simulator. Share access to Trello.

Content: Short descriptions of what we did:

- We discussed performance with PO and he explained that we need to perform stress tests of the system.
- To boost performance of the SQL queries we need to use SQL explain and indexing as boosting measures.
- We shared access to the Trello board to PO, we had previously just shown it at sprint reviews and retrospectives.
- Jørgen does a demo of frontend and simulator and PO explains that he feels the simulator is a very clever move to have made it.
- We show PO the new design for the admin panel and he says he thinks it's getting better.

H.12 12th Meeting

Date: 17.02.2022

Duration: 20 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Discuss Headit's involvement so far. Show demo to PS.

Content: Short descriptions of what we did:

- PS asked about how Headit is involved so far. We explain that they are very involved and very enthusiastic at the same time being hands off with some key decisions that they trust us to make.
- Show PS the demo of the frontend and simulator.
- PS again mentions that we need to document our choices because it will help us when we are writing the report.
- Short discussion about other groups without identifying any group.

H.13 13th Meeting

Date: 22.02.2022

Duration: 45 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Show wireframe for mobile app. Talk about testing and edge cases.

Content: Short descriptions of what we did:

- Markus shows to PO the wireframes he has created for the mobile application.
- PO asks about if we should perhaps show extra information as part of the race on the mobile app. We do not agree because the mobile app is single purpose.
- Talking about various libraries we could use for tracking, but none was decided just yet.
- Should the phone app have a "start tracking, end tracking" button? We don't think so.
- PO mentions to display a "tail" on people participating in the relay as to show some form of "speed" of the contestant.
- Remove some info on the google maps display that is not needed such as stores and all the street names.

H.14 14th Meeting

Date: 24.02.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Talk about the thesis and what is expected of us.

Content: Short descriptions of what we did:

- PS discussed what is expected for the thesis:
  - Between 60-70 pages perhaps, not a hard rule on this.
  - Follows a strict format.
  - Expected to be delivered as one pdf.
- PS explains that it is wise to start writing it soon, 1 month before delivery because it can get pretty large and daunting if one waits too long to start.

H.15 15th Meeting

Date: 01.03.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Talk about testing strategies.

Content: Short descriptions of what we did:

- We discuss with PO various testing strategies such as AAA for .NET.
- PO would like to see unit tests and integration tests for the backend.
- For the admin panel mange would like to see some tests but he does not specify which ones. He is not that familiar with this area, so he does not give specifics. It is possible for both end to end and unit tests.
- We agreed we don't need 100 percent test coverage but we should test what we "need" to test.

H.16 16th Meeting

Date: 03.03.2022

Duration: 20 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Talk about testing with PS and how to write about it in the report.

Content: Short descriptions of what we did:

- Talk to PS about what was discussed with PO two days ago.
- PS explains that stress tests are important as well and that we need to know the limits of the system so we can write about it.
- PS mentions that integrations tests are generally more attractive to have as testing how things worked together is where the tough bugs can be caught.
- Discussion about how the simulator fits into the testing framework and how we can structure writing about it in the thesis.

H.17 17th Meeting

Date: 08.03.2022

Duration: 45 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: New demo to PO and another employee of Headit.

Content: Short descriptions of what we did:

- We show PO a full demo of the simulator performing swapping on a relay race with many participants. Displaying that our algorithm works correctly.
- PO brought along another person to view the demo, after the demo we answer various questions about how the algorithm works and how the system works as of now.
- Guest leaves and we hold a retrospective with PO.
- Discussion about what was good with the sprint.
- PO and us decided that we needed to make the sprint coming up one week longer but we will decide as we get closer to it's end date.

H.18 18th Meeting

Date: 10.03.2022

Duration: 30 minutes.

Location: NTNU Gjøvik.

Participants present: Development Team and Project Supervisor.

Agenda: Show PS full demo.

Content: Short descriptions of what we did:

- We show PS the full demo same as was shown in the last meeting with PO.
- Answering PS questions about how the system works in its current state.
- PS asks security questions relating to what is stored regarding the phone devices. We explain that as of now nothing is stored about the devices themselves and there is no way to get that information in any one the information that is in flight during relay races.

H.19 19th Meeting

Date: 15.03.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Mobile app with IOS. Show backend state.

Content: Short descriptions of what we did:

- Explain that we are having difficulties with the IOS part of the cross platform mobile app. Expo Go is causing a lot of trouble for us and is slowing down development quite a lot.
- PO suggest to not have "checkpoints" as they would be nice to have but more important things are missing such as stable cross platform support.
- We discuss the idea of caching certain things temporarily in memory about relays to reduce load. But we can't prove that it is an issue yet so it's not top priority.
- PO suggest a person at Headit we can speak with to talk about IOS background tracking of position.
- Discuss some details about the user test. Done physically and there will be at least three people.

H.20 20th Meeting

Date: 17.03.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Discuss difficulties with PS. Get advice on what to do.

Content: Short descriptions of what we did:

- Talk to PS about difficulties in the cross platform part of the phone app. Iteration is slow and iOS is giving a lot of problems.
- PS suggests to reach out to people who might know what to do at NTNU as well as speak with Headit.

H.21 21st Meeting

Date: 22.03.2022

Duration: 6,5 hours.

Location: Headit offices at Hamar.

Participants present: Development Team and Headit Employees.

Agenda: Perform user tests on the frontend, mobile app and integrated system.

Content: Short descriptions of what we did:

- A detailed description on the user test is part of the appendix.

H.22 22nd Meeting

Date: 24.03.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Elvis Arifagic, Markus Strømseth and Project Supervisor.

Absence: Jørgen Eriksen.

Agenda: Detail the user test to PS.

Content: Short descriptions of what we did:

- Discuss all the details of the user test to PS.
- PS thinks it's a good idea to have a separate test document that we can refer to in the report.
- PS thinks the user test will help a lot to have done it for the project and to have done it physically with people on location.

H.23 23rd Meeting

Date: 29.03.2022

Duration: 60 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Discuss thesis writing and OAuth2.

Content: Short descriptions of what we did:

- The group has decided that Elvis will work exclusively on the report from now on to get it up to shape for the first delivery to PS. As part of the project we are delivering the report to get feedback up to maybe three times.
- Jørgen and Markus will keep working on the admin panel.
- Jørgen says that backend is mostly finished and it only needs to be polished.
- OAuth2 can be done via Headit's keycloak server and will reduce the amount of time we have to spend on it.
- Discussed the details of the user test with PO.

H.24 24th Meeting

Date: 07.04.2022

Duration: 40 minutes.

Location: NTNU Gjøvik.

Participants present: Development Team and Project Supervisor.

Agenda: Questions regarding the thesis and status of the admin panel.

Content: Short descriptions of what we did:

- We went through the new features of the admin panel.
- We showed PS the changes done to comply with the feedback from the user test.
- Asked questions regarding some structural parts of the thesis.
- We organized and decided on a date for delivering the draft of the thesis for feedback.

H.25 25th Meeting

Date: 20.04.2022

Duration: 40 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Showed the status of the new iteration of the admin panel and showed finished versions.

Content: Short descriptions of what we did:

- We went through all changes done to comply with the feedback from the user test.
- Discussed some parts where we were unsure what was the right solution.
- Showed the finished version of the admin panel and mobile application.
- Asked questions regarding KeyCloack. And organized setting up the environment for it.

H.26 26th Meeting

Date: 26.04.2022

Duration: 10 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Just a quick status report.

Content: Short descriptions of what we did:

- Not much was discussed. Just a quick check up on how things were going.

H.27 27th Meeting

Date: 27.04.2022

Duration: 20 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Organized and planned for getting further feedback on the thesis.

Content: Short descriptions of what we did:

- We decided on having a second feedback session on the new draft of the thesis.
- Asked questions regarding PS opinion on different matters regarding the thesis.

H.28 28th Meeting

Date: 12.05.2022

Duration: 30 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Product Owner.

Agenda: Final meeting.

Content: Short descriptions of what we did:

- We talked about the different aspects we were proud of and asked for PO's opinion with regards to having it in the thesis.
- Thanked each other for the cooperation and PO's exceptional job as the role of PO.

H.29 29th Meeting

Date: 13.05.2022

Duration: 60 minutes.

Location: Online via MS Teams.

Participants present: Development Team and Project Supervisor.

Agenda: Received feedback on the draft of the thesis.

Content: Short descriptions of what we did:

- Went through the entire draft and discussed his comments on the different chapters.

NTNU

Kunnskap for en bedre verden