Noroff
School of technology
and digital media

Technical Report

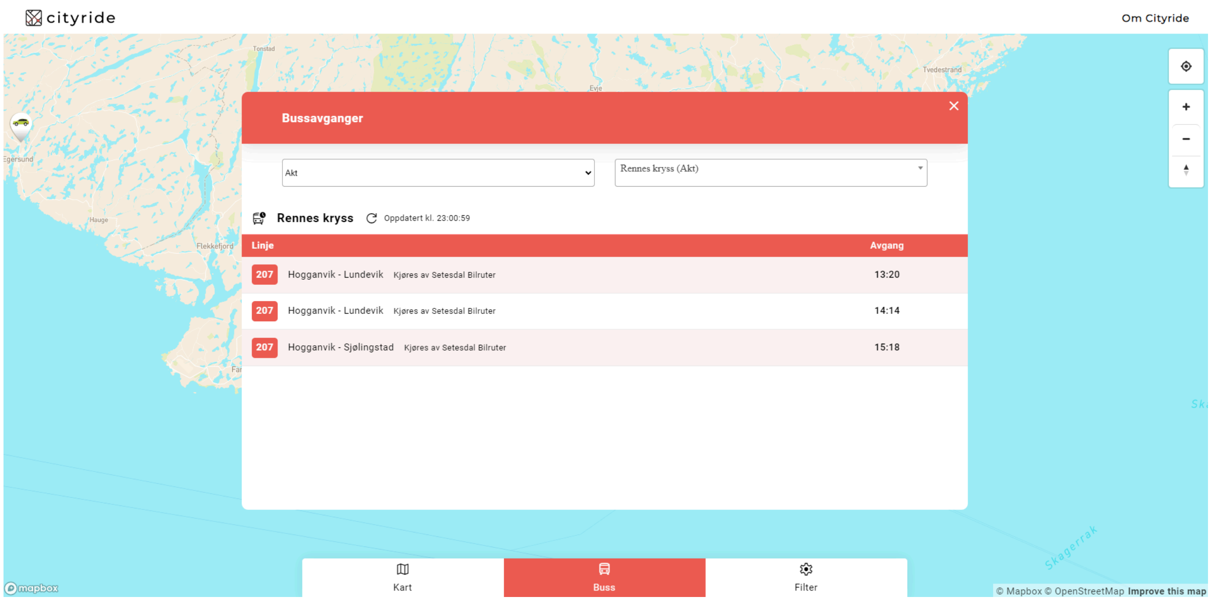
Exam 2 - The final

Tore André Rosander

Table of Contents

1. Introduction

During the frontend studies i have been developing a web service that display available escooters, citybikes and other mobility services.Cityride is written in vanilla JS with an autonomous PHP backend that import data from Entur's mobility apis and store the data as geoJSON for use with mapbox.



Bussavganger

Akt | Rennes kryss (Akt)

Rennes kryss Oppdatert kl. 23:00:59

Linje	Avgang
207 Hogganvik - Lundevik Kjøres av Setesdal Bilruter	13:20
207 Hogganvik - Lundevik Kjøres av Setesdal Bilruter	14:14
207 Hogganvik - Sjølingstad Kjøres av Setesdal Bilruter	15:18

Cityride refactor
For my project exam 2 I decided to refactor cityride from vanilla JS to react with the new name Movai.no and implement AI functionality.

The main requirements for this project will be the following:
Refactor the map so the refactor keep the same functionality and logic
Refactor the departure board
Add a travel planner
Find an innovative selling point for the application

In general the project is quite large with several moving parts and extensive use of external data. I have tried to split up functionality into separate components, where I could and where it made sense without nesting too deep.

I wish we had more time with react during our studies, and learned more advanced debugging, as I had major issues with maintaining map layers active state. We tend to only focus on the same limited set of API calls.
I also encountered issues with useEffect, props and state sharing within components and child components in relation to the mapbox react wrapper.

During the brainstorming phase I decided to have a look at different travel planner search interfaces when I noticed that they are all quite similar. So I decided to go a different route and train a couple of AI models to handle the search and user input. This way I will be able to implement different types of user interfaces in future versions.
The travel search in this version is able to handle searches that include destination and arrival stop, time and date.

For better accessibility I added a text-to-speech function that will play the search result out loud.

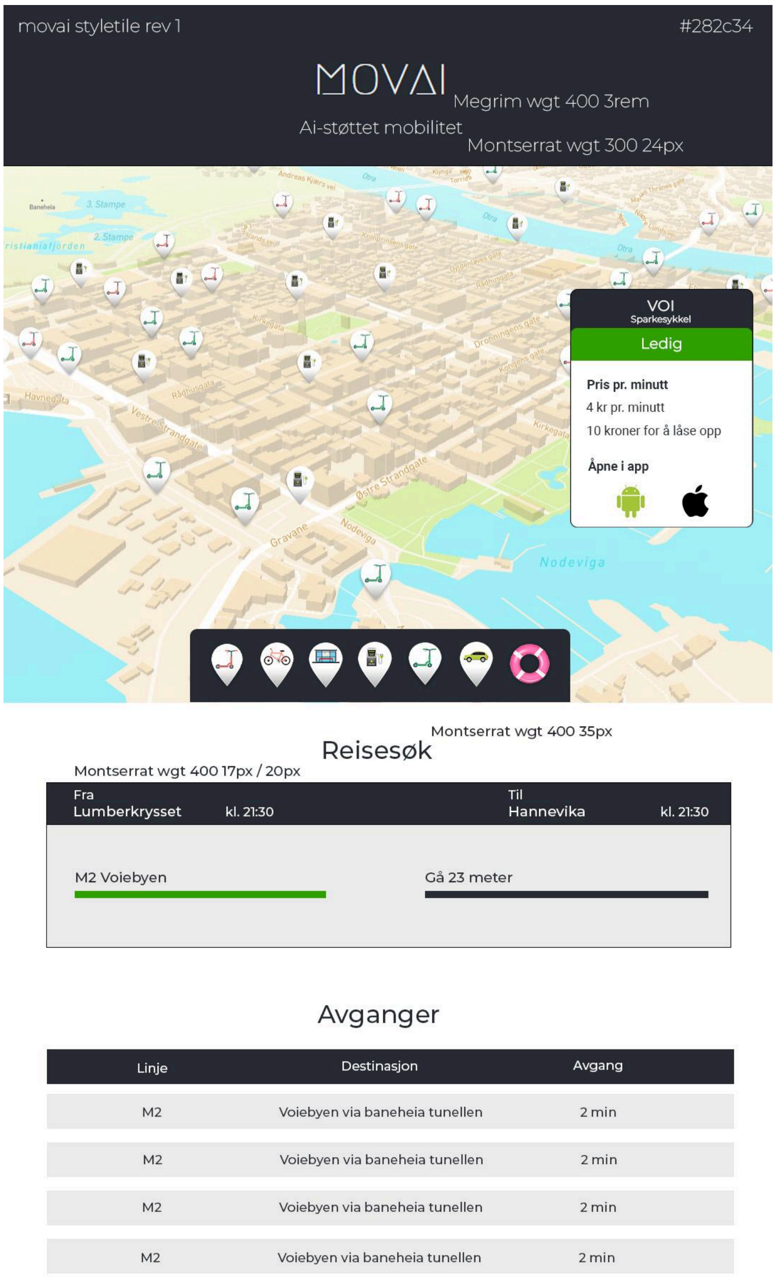
All in all I feel that I have delivered a project that should fit within the scope needed to judge my final exam. During the exam period I was made aware that there were several descriptions of the tasks available so I got confused regarding the scope.
I was not aware that we had to plan the “real world client” task together with a teacher. So I panicked and at the last minute (I started implementing login 17 minutes before deadline) I added a dashboard for an admin to log in using JWT tokens. The endpoint for the user handling is the same wordpress backend as I have previously used in deliveries.

There is a known bug where you don't get redirected to the dashboard after successful login, there was no time to fix.

Login details
Login URL: https://movai.no/dashboard
Username: Administrasjonen
password: speilegg1

2. Design

The design is simple and minimalistic. I Found some nice solutions to the display of trip details. And I am very happy with the result.



Avganger

Linje	Destinasjon	Avgang
M2	Voiebyen via baneheia tunellen	2 min
M2	Voiebyen via baneheia tunellen	2 min
M2	Voiebyen via baneheia tunellen	2 min
M2	Voiebyen via baneheia tunellen	2 min

3. Functionality

Ai assisted travel search
The travel planner search is executed by a real working AI model that I have trained from scratch to understand Norwegian travel enquiries.

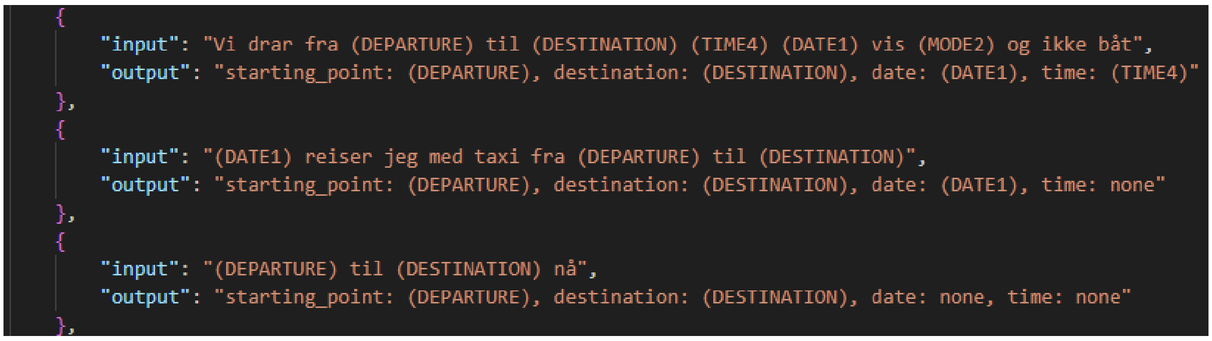
Example search phrases:

"Fra Kristiansand til Oslo i morgen kl 13"
"Fra Oslo til Drammen i morgen"

The Ai part is not really part of the exam scope, but I will spend a few paragraphs to explain how it was put together.

Instead of using a regular API I trained an AI to execute the search query sent from the react frontend. This enables the use of free text input from the frontend.

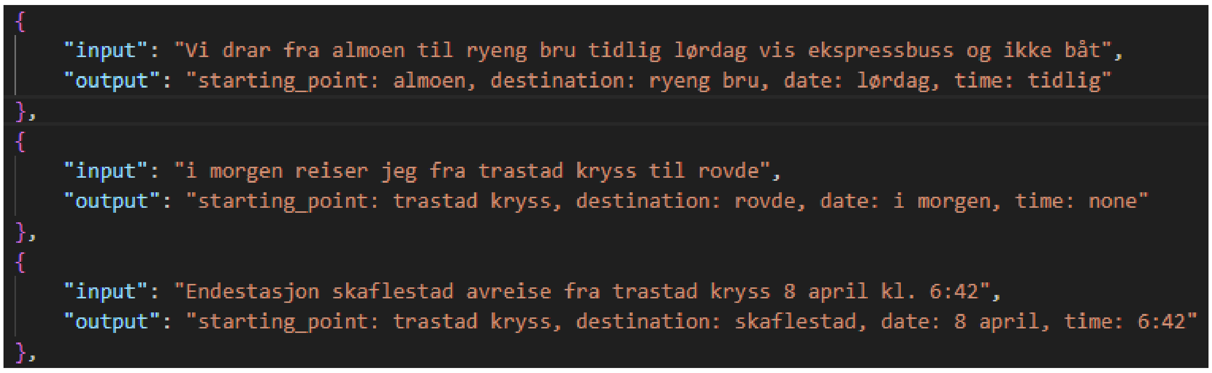
I started by creating a dataset containing around 1000 entries with placeholders.



```
    {
        "input": "Vi drar fra (DEPARTURE) til (DESTINATION) (TIME4) (DATE1) vis (MODE2) og ikke båt",
        "output": "starting_point: (DEPARTURE), destination: (DESTINATION), date: (DATE1), time: (TIME4)"
    },
    {
        "input": "(DATE1) reiser jeg med taxi fra (DEPARTURE) til (DESTINATION)",
        "output": "starting_point: (DEPARTURE), destination: (DESTINATION), date: (DATE1), time: none"
    },
    {
        "input": "(DEPARTURE) til (DESTINATION) nå",
        "output": "starting_point: (DEPARTURE), destination: (DESTINATION), date: none, time: none"
    },
```

I created ~1000 manual entries using placeholders

Then I created a python script that loaded all stop places in norway (150k), then picked 12 random entries for each stop place and replaced the placeholders with different data. Resulting in 1.8 million unique entries in the final dataset.



```
{
    "input": "Vi drar fra almoen til ryeng bru tidlig lørdag vis ekspressbuss og ikke båt",
    "output": "starting_point: almoen, destination: ryeng bru, date: lørdag, time: tidlig"
},
{
    "input": "i morgen reiser jeg fra trastad kryss til rovde",
    "output": "starting_point: trastad kryss, destination: rovde, date: i morgen, time: none"
},
{
    "input": "Endestasjon skaflestad avreise fra trastad kryss 8 april kl. 6:42",
    "output": "starting_point: trastad kryss, destination: skaflestad, date: 8 april, time: 6:42"
},
```

The dataset after the placeholder replacement

The use of AI enables the input of free text instead of entering every travel search detail one by one in a form. This will in later versions enable speech-to-text search that will increase accessibility for certain user groups.

When a user enters the travel search query, the request is sent to a PHP endpoint that acts as a reverse proxy and forwards the request to a python flask API that run on the same server and has the AI model loaded. The AI model will then extract the destination and arrival stop, time and date.

The AI model will try one of three ways to translate the destination and arrival stop to actual QuayIDs. If the destination or arrival stop is a location (city, area) it will use a geocoder to get the closest stop. It will also format time and data or replace with default values if the user did not enter any (if the user just search “from oslo to kristiansand " it will assume that the date and time is now. When all the data is cleaned and ready the AI will execute the travel search against Enturs journeyPlanner API.

The results are returned to the react frontend and displayed using the TravelPlanner component and child components. In cases where the AI misinterprets something or if you need to make small changes an additional “modify form” is loaded when the results are returned so you can make changes. The form input values will represent the actual data the AI used to search, so if the AI misinterprets the date you can make changes in the modified form.

Map

The core part of cityride was the map that displayed different data types.
I stumbled upon several major challenges during the refactor and the hardest was regarding the mapbox react wrapper.

There is no official support for Mapbox in react but it is possible to use the vanilla js library and do everything the old “JS way”, in react. I considered that it would not be sufficient to handle the map part “outside of react” without state management or use of components in the exam.

I was able to find a react wrapper for Mapbox, however there was an issue with the newer versions that disabled the use of symbols as map markers.
In Mapbox you can choose between multiple ways of showing markers on the map, each with its own pros and cons.
Since Movai might display several hundred markers on the screen at once we need to display the markers as symbols to reduce performance impact.

An older version of the React wrapper did include the symbol functionality, with the drawback that you will get a warning when installing this project. When installing node modules you must add "-- forced" since there is a mismatch between versions.
I have done extensive testing and did not find any issues that will impact the movai application because of this.

Managing state when using the local storage for saving the users filter selection was challenging and I had to rewrite the code several times. But after some trail and error, several restarts and a final cleanup the state is now stable across pageviews and user interactions.

In addition i had major issues with the bus stop popup in the map that display departure data for each stop. I tried several methods for fetching the data. useEffect did not work, after days of troubleshooting it turns out the react wrapper was to blame. It prevented the component to rerender after initial load, this effect can be seen when editing the BuStopPopup where the changes in the code is not reflected when the development server updates the DOM. As a workaround i am making the API call directly from the MapContainer and passing the data as a prop. Its a bit messy but it solves the issue.

The map shows the following:
Available electric scooters from Voi, Bolt and Ryde.
Available car sharing cars from Hertz, Otto and Hyre.
Citybike stations with available bikes from Kolumbus, Brakar, Oslo and Trondheim.
Bus stops with departure data for AKT, Brakar, Ruter, ATB and NFK.
Charging stations for electric cars.
Water temperatures.

Departure board
The departure board is ment to be used to check the next departures from a stop, either by an end user's device or on an information screen or smart screen.
The departure board uses the Entur geocode API to get the stop suggestions based on the user input with a 400ms debounce to reduce API calling.
The departure board also features two different themes, the ability to set the number of departures and a load more button that fetches more departures from the API.
The departures is updated every 30 seconds.

Dashboard
The project did not have any natural login function or listing of products and services as per the exam requirements. So I created a dashboard that is meant to be a situation dashboard for bus operators.
In the example the operator is AKT and I have loaded AKT specific data for showing deviation messages, delays and news updates.
The dashboard will load graphs showing available units, a map that shows bus delays as a heatmap and single markers if you zoom out. The map will animate slowly if left alone for a while.

The news messages consist of a mix of deviation messages from Enturs deviation API and a python rss feed importer. The data is exported as json and then sorted and displayed on the frontend. The messages are updated every minute, but this functionality is turned off during exam evaluation to reduce the moving parts, the data is still loaded from the same json data.

4. Summary

I had to prioritize hard when I stumbled upon the different issues in this project. But the end result is on a whole other level than the original city ride application. The design part could be stronger, but i knew i had to make up some of the design during the coding process.

The most successful part of this project would be the "new" user interface for making travel searches, and I look forward to implement more functionality in the coming weeks and months.