iParkinStan

23.05.2018

Mikael Dahlgren
mdahlgre@kth.se
19850702-0272

Johannes Carlsten
jcarlste@kth.se
19851014-4051

Alexander Nordh
alnordh@kth.se
19911214-2634

Louise Gröndahl
lougro@kth.se
19950427-4482

Description

iParkinSTHLM is a webapp made for mobile devices. It will use Stockholm Stad's API together with Google Maps API to create an app which gives the user a quick and easy way to see which streets in the near vicinity where they are able to park. The user see's a map where available streets are marked in green.

Technology Overview

1. React-Native - https://facebook.github.io/react-native/
2. Google Maps API - https://developers.google.com/maps/documentation/javascript/tutorial
3. Stockholm Stad - Open Stockholm Parkering API - https://openparking.stockholm.se/Home/Parking

Project Structure

./App.js
This is the container for the whole app. All it does is render a view which contains <Map />, which is imported from ./Map.js

./Map.js
Here the Google Maps API and Trafikverkets API is used in order to draw lines (polylines) around the centered location.

./components/DeviceMarker.js
The component that displays the users current position

./components/GoogleSearch.js
The component uses Google Places to search for nearby locations and when selected, triggers a function that animates to that location.

./helper_functions/checkTime.js

This function takes start times and end times of parking features and compares with the current time. If the current time is inside the range, it will render the feature as a line on the map. It also looks at a time buffer, and if the parkingtime is close to ending, it displays the feature line in red instead.

./helper_functions/debounce.js

This is a general debounce function. It returns a function, that, as long as it continues to be invoked, will not be triggered. The function will be called after it stops being called for N milliseconds. (This is so that the app won't spam API calls whenever the map is being moved.)

./helper_functions/getParkingData.js

This function fetches JSON data from stockholm stad API.

Meteor JS - A huge mistake

Meteor looked so promising! On their website it reads "*Accomplish in 10 lines what would otherwise take 1000* [...]" and "*Use the same code whether you're developing for web, iOS, Android, or desktop* [...]".

And we did get an app in Meteor up and running, along with Google Maps integration. Then Meteor completely broke because of a Windows Update. Since half of our project members were on devices running windows on auto-update, we had to make a decision: Should we wait and hope for a fix, or should we scrap our current progress and start from scratch?
We had already found that Meteor maybe was promising more than it could deliver; at it's best it is simple to use but extremely buggy and unreliable. Also, time was running short. As such, we decided to scrap Meteor and start from scratch using something we're familiar with; react.

React-Native - A small mistake

The idea with React Native is to let you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components. Since we were all somewhat familiar with React, this felt like and obvious choice as a replacement for Meteor JS. We overestimated how similar it would be to react though, as we found it very different to work with.

Parkering API

Stockholm Stad provides a couple of different API's where we used their Parking API to get information about where in Stockholm you are able to park your car.
Their API's are free to use and you just send a request to get a hold of an api-key.
With the API you can get a ton of information about parking places in Stockholm.

The API follows the format: https://openparking.stockholm.se/LTF-Tolken/v1/{föreskrift}/{operation}?apiKey=YOUR API-KEY

It was relatively easy retrieving the data from the Parking - API because of the structure and the most relevant data for us where the location with coordinates latitude and longitude.

The operation that we considered most useful in a user purpose were the radius because that allowed us to only get the most relevant coordinates based on a radius from where the user are located.

One thing that wasn't that good about the API for us are the lack of regulations. They have a regulation for bus, truck, motorbike and for disabled drivers with special permissions. It would have been a lot easier if they also had a regulation for only cars. But, with the regulation we ended up using you get data about every single street where a vehicle can park and it is pretty easy though to just get data about where a car can park.

Google Maps API

We had used Google Maps API previously in an early Lab, but since we decided to use react-native, we had go another way about implementing Google Maps API. The most common way seems to be react-native-maps. We could get the map up and running pretty easily, however when we wanted to implement specific features, documentation was sparse and we mostly had to rely on stack-overflow threads for implementation of functions.

Reflection

We believe the way we planned our project was sensible. We made a plan for what functionalities to implement and broke them down into smaller problems. We then made a TO DO-document with these smaller problems that each in our group could tackle. We communicated via facebook messenger about which problem we were currently working on, and noted the completion of these in the TO-DO document. We feel like we all contributed equally to the project, and that we learned a lot from the experience. Because of our choices of dev-tools we ran into more problems than we would have liked. Had we done more preliminary research about which tools to use we could maybe have saved a lot of time and effort and had a better end result. We are however happy with what we accomplished during the circumstances, and believe that the idea of the app is solid and which to further develop the app in our spare time.