Assignment 1: Web Services

Luuk van Sloun: 20141483

Contents & Install Guide

API source code	Client source code	Frontend sources	
api.py	client.py	./templates	./static

All source code was created with Python3. Used packages include Flask, Flask Restful, WTForms and others. All the required dependencies are listed in *install.sh*.

Installation of these dependencies and starting both the API and the Client can be done with:

```
chmod +x ./install.sh
chmod +x ./run.sh
./install.sh
./run.sh
```

It's recommended to do so in a Virtual Environment, as versions may vary across systems.

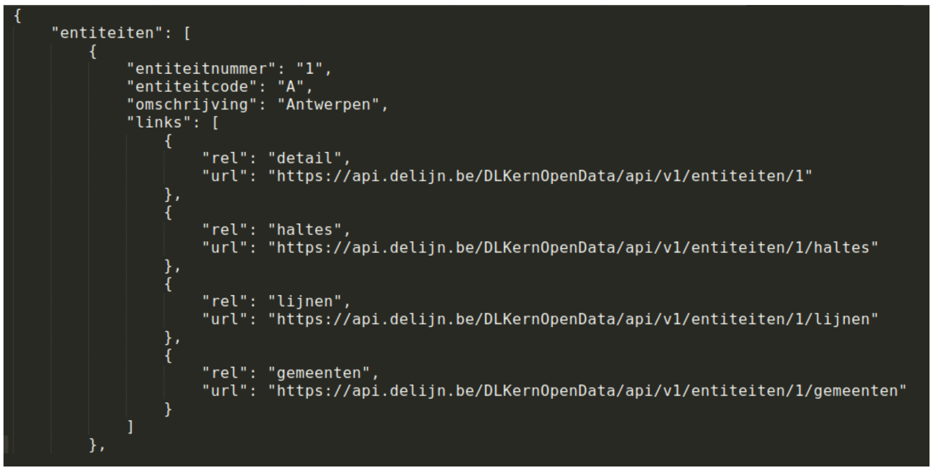
The used APIs in this project are:
- De Lijn Kern Open Data Services API
- HERE Weather API (Requesting weather for certain coordinates)
- HERE Routing API (Calculating routes according to waypoints)
- MapBox API (Drawing map, markers and routes)

API & Client

The API runs on http://localhost:3000, whereas the Frontend Client runs on http://localhost:5000. To prevent issues, please make sure no other programs are running on these locations.

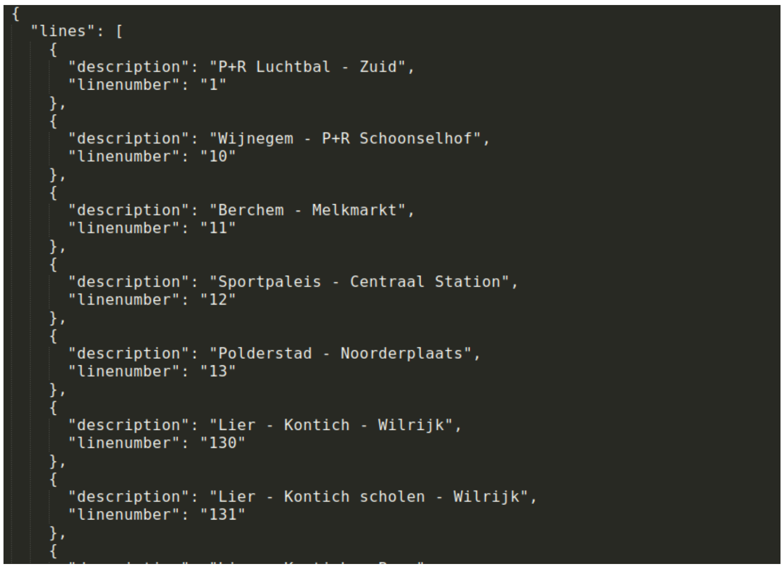
The following GET requests can be made using the API:

- getProvinces: /provinces
  → Returns all provinces/entities provided by De Lijn Kern Open Data Services



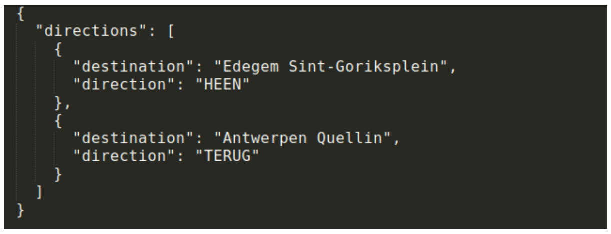
```
{
    "entiteiten": [
        {
            "entiteitnummer": "1",
            "entiteitcode": "A",
            "omschrijving": "Antwerpen",
            "links": [
                {
                    "rel": "detail",
                    "url": "https://api.delijn.be/DLKernOpenData/api/v1/entiteiten/1"
                },
                {
                    "rel": "haltes",
                    "url": "https://api.delijn.be/DLKernOpenData/api/v1/entiteiten/1/haltes"
                },
                {
                    "rel": "lijnen",
                    "url": "https://api.delijn.be/DLKernOpenData/api/v1/entiteiten/1/lijnen"
                },
                {
                    "rel": "gemeenten",
                    "url": "https://api.delijn.be/DLKernOpenData/api/v1/entiteiten/1/gemeenten"
                }
            ]
        },
```

- getLinesForProvince: /provinces/<provinceNumber>/lines
  → Returns all lines corresponding to the received province number



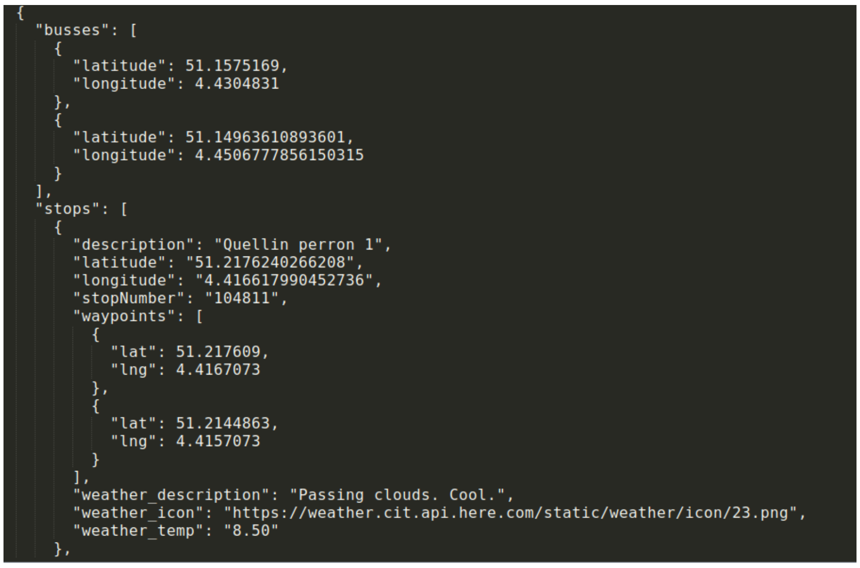
```
{
  "lines": [
    {
      "description": "P+R Luchtbal - Zuid",
      "linenumber": "1"
    },
    {
      "description": "Wijnegem - P+R Schoonselhof",
      "linenumber": "10"
    },
    {
      "description": "Berchem - Melkmarkt",
      "linenumber": "11"
    },
    {
      "description": "Sportpaleis - Centraal Station",
      "linenumber": "12"
    },
    {
      "description": "Polderstad - Noorderplaats",
      "linenumber": "13"
    },
    {
      "description": "Lier - Kontich - Wilrijk",
      "linenumber": "130"
    },
    {
      "description": "Lier - Kontich scholen - Wilrijk",
      "linenumber": "131"
    },
    {
```

- getLineDirections: /<provinceNumber>/lines/<lineNumber>/directions
  → Returns both to and from directions with their destination for a certain line in a province.



```
{
  "directions": [
    {
      "destination": "Edegem Sint-Goriksplein",
      "direction": "HEEN"
    },
    {
      "destination": "Antwerpen Quellin",
      "direction": "TERUG"
    }
  ]
}
```

- getLineRoute: /<provinceNumber>/lines/<lineNumber>/directions/<direction>
  → Returns all stops and current busses on the requested line and their info
  → Stops contain location and weather info
  → Busses contain line number and realtime location info
  → Stops contain a list of their waypoints and their successor's in the requested route.



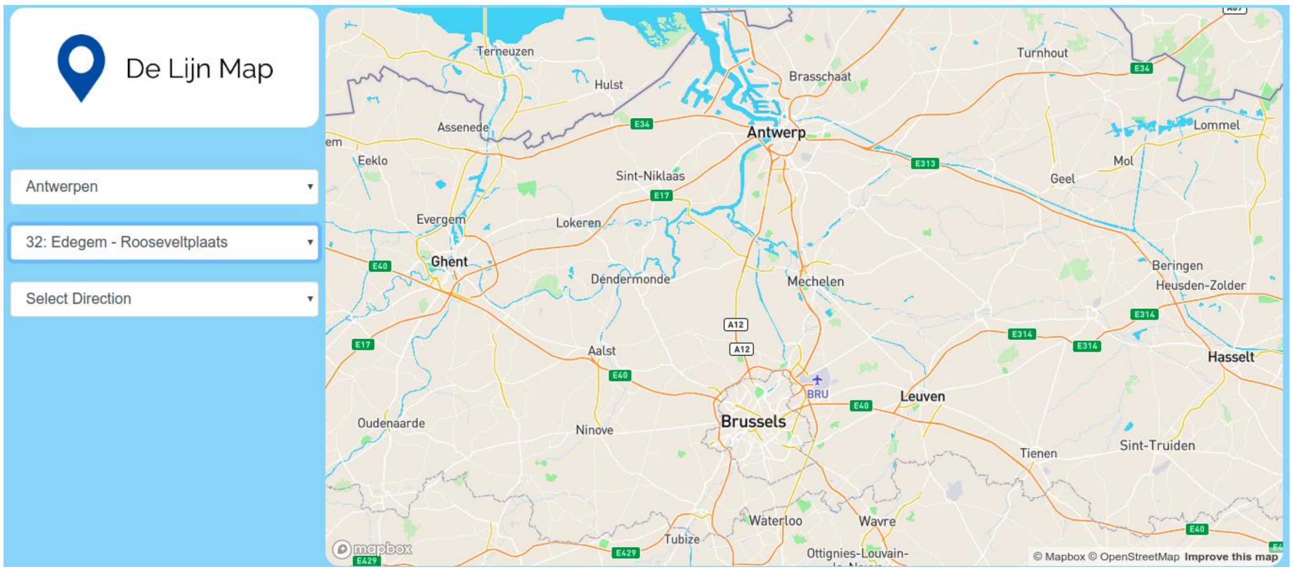
```
{
  "busses": [
    {
      "latitude": 51.1575169,
      "longitude": 4.4304831
    },
    {
      "latitude": 51.14963610893601,
      "longitude": 4.4506777856150315
    }
  ],
  "stops": [
    {
      "description": "Quellin perron 1",
      "latitude": "51.2176240266208",
      "longitude": "4.416617990452736",
      "stopNumber": "104811",
      "waypoints": [
        {
          "lat": 51.217609,
          "lng": 4.4167073
        },
        {
          "lat": 51.2144863,
          "lng": 4.4157073
        }
      ],
      "weather_description": "Passing clouds. Cool.",
      "weather_icon": "https://weather.cit.api.here.com/static/weather/icon/23.png",
      "weather_temp": "8.50"
    },
```

Usage

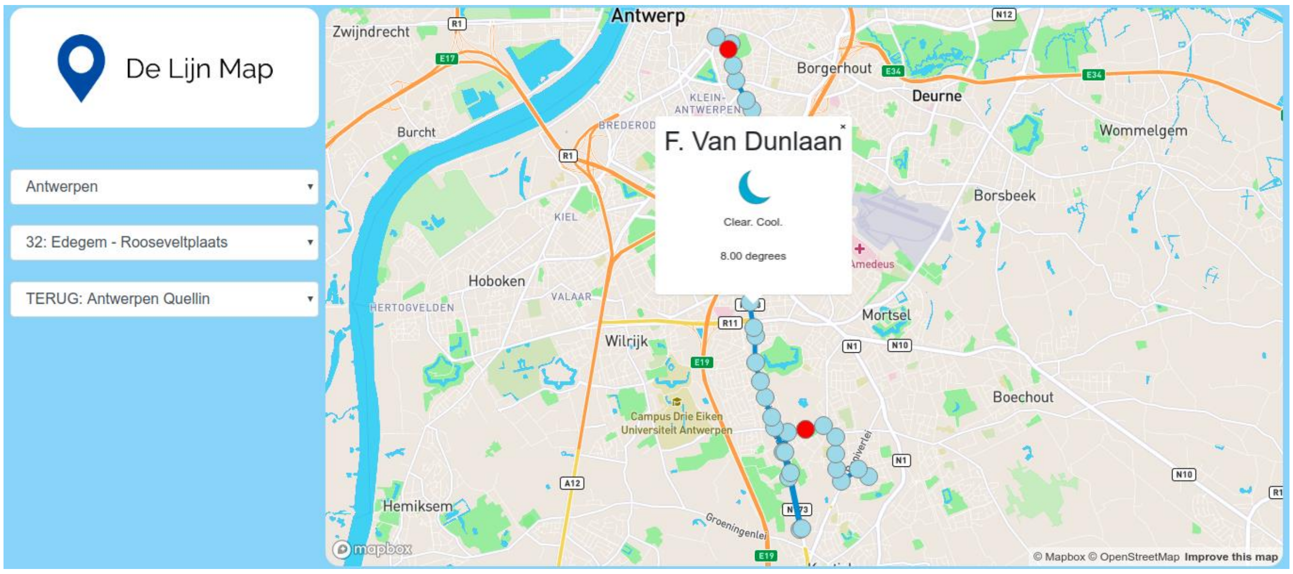
As previously mentioned, the API runs on http://localhost:3000, through which it can be communicated when using the right URLs. Programs designed to interact with such service, for instance Postman, can be used to interact with the API. Simply entering the variables into the URL will return the requested JSON on the webpage.

The web-app itself runs on http://localhost:5000. At first only a province select field will be shown alongside the map. The other select fields will be shown when a choice is provided for the other field(s).

When a line and its direction has been chosen it will appear on the map with all its stops. Current busses will be shown. The location of the busses when loading is real-time, but they aren’t updated throughout time. To do so, please refresh the page and request the line again.



The route will appear after a short period of waiting. The spinning pinwheel will indicate the loading process. Blue markers are stops and red markers are vehicles (busses, trams, etc). Clicking on a Stop Marker will present a popup containing the name of the stop and the current weather at that location.



Design Choices

I decided to split my project up into three sections: the API service, the Client service and the frontend files.

The API service is based on functions available in Flask Restful and provide a simple and compact way of creating a RESTful service. Basic features from the standard Flask package where used to provide the necessary functionalities.

The Client service is based on the Flask package (not the RESTful). I chose to implement a certain pass-through between frontend and backend to keep things organized and because of prior experience in setting up a web-service using this method. The Client receives requests from the frontend through Javascript and passes them on to the API. When the API responds, the Client simply sends it back to Javascript. This might seem as overkill, but it provides an easy way to debug both front- and backend in one place.

The Frontend was created using HTML, CSS, Javascript and the Bootstrap Library. All these were chosen due to prior experience with them. I kept the website fairly basic, since the map can get quite ‘busy’. That’s why I used as little Bootstrap as possible, to keep it as clean as I could.

As far as the used APIs are concerned: I first planned on using only the HERE API for all aspects to keep everything combined, including drawing the map and such, but after some research I stumbled upon MapBox, which had a more intuitive syntax, so I decided to go with that.