Assignment 2: Microservices

Luuk van Sloun: 20141483

Install Guide

This project consists of multiple Docker containers and databases which all are handled by Docker-Compose. The following versions were used to create this project:

Docker: version 19.03.5
Docker-Compose: version 1.25.0

Please make sure both are installed before running this project. Instructions on acquiring these can be found using the following links:

https://docs.docker.com/install/linux/docker-ce/ubuntu/
https://docs.docker.com/compose/install/

Along with Docker and Docker-Compose, Python3 should be installed, since the entire project is built using this version of Python.
Everything is handled by one .yml-file, which contains the entire setup of all the containers and databases. To create these containers along with their images, simply run the following commands:

chmod +x ./install.sh
./install.sh

This will then setup all the dependencies for each of the containers and will also create an “online” instance of the website, which is run on localhost. Each service has its own requirements.txt file to ensure all necessary dependencies are installed correctly.

To make things easy, a *shutdown.sh* is included to stop the website. As with the installer, simply run:

chmod +x ./ shutdown.sh
./shutdown.sh

Architecture & Design Choices

This project runs on 4 services and 3 databases. The databases are not to be seen as a completely separate service, as these are linked to only one service.

The first service is the *delijnapi* service. This service contains all the necessary API-calls to the *De Lijn Kern Open Data Services* to extract data such as provinces, lines, stops, etc.
Due to the enormous amounts of data De Lijn returns in some calls, a database is set up to benefit the loading times on the website. There’s only one table present in this database, which consists of all the stops in Flanders. Since there are approximately 36000 stops, the best option would be to load them all during the setup of the website itself.

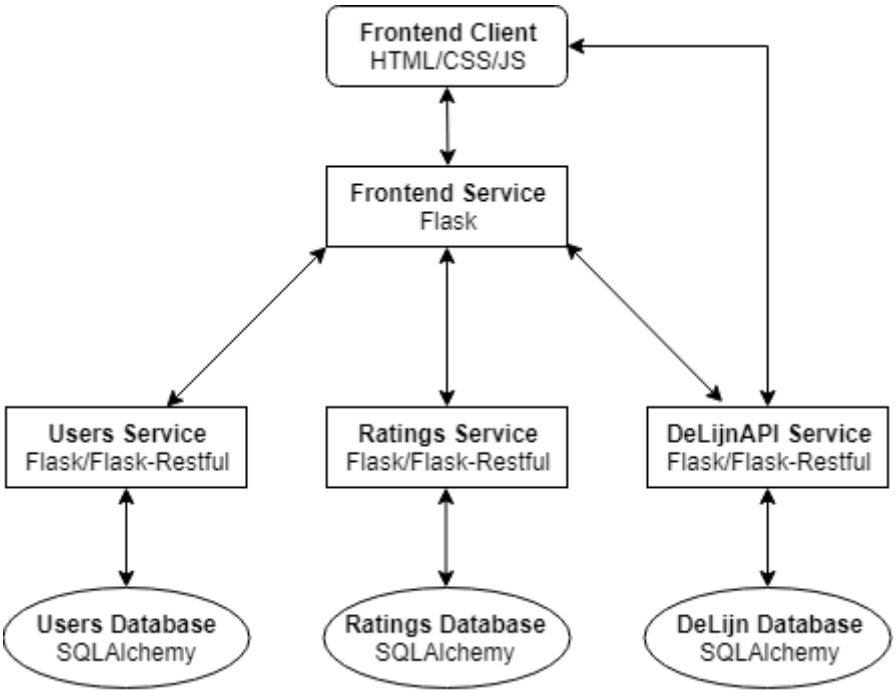
The second service is the *users* service. This service is used to keep track of all registered users, which are stored in a separate database, along with other methods such as verification of the combination of a username and a password. This database also contains just the one table.

The third service is the *ratings* service. As with the previous two services, this one has a ‘personal’ database with two tables, one for the vehicle (tram/bus) ratings and one for the stop ratings. I chose to keep these separate to keep the tables clean and have an easy separation between the two, since handling vehicle ratings does not involve any interaction with the De Lijn API whereas the stop ratings depend on it.

The final service is the *frontend* service. This service is simply a Flask app which handles the calls from Jinja and Javascript. The frontend also is the service which contacts all other services. All calls to the other services are done from within the Flask app, except for the calls needed to provide dynamic forms on the website, which use Javascript to directly communicate with the *delijnapi* service. It would have been cleaner to keep all calls within the Flask app, but this provides a very user-friendly interaction, making this the better way to go. HTML, CSS and the previously mentioned Javascript were used to create the frontend webui.

With the idea of microservices in mind, I kept all the services as separate as possible. Every service has its own database, to create the possibility with which a container might be replaced by another, without restructuring the entire project. If one was to replace the User service with a more elaborate one, some calls in the frontend client might have to be altered to fit the need of the new service, but no other services would have to be changed.

The following diagram shows the basic form of interactions between the services and their most important dependencies on which they're built.



Services

All services run on the internal port of 5000, which can be found in the *docker-compose-dev.yml* file. However, all services run on their own external port, ranging from 5000 to 5003. All services where created using Python3 and Flask. All but the Frontend service are built as a restful API using Flask-Restful. Other dependencies include Flask-SQLAlchemy for database creation and interactions, Passlib for encrypting the passwords provided by the users and Flask-CLI for creating commands which can be easily executed during the initial setup.

- Frontend Service:

  Runs on http://localhost:5000 and will present the website when this address is opened in a browser of your choice. Even though there's no use in calling the frontend, the internal address is http://frontend:5000.

- Users Service:

  Runs on http://localhost:5001 externally, but for internal contact between containers, http://users:5000 is the address needed to communicate with it.

  This service is built as an API and contains the following requests:

  - registerUser POST
  - verifyUser GET
  - getAllUsers GET

- Ratings Service:

  Runs on http://localhost:5002 externally, but for internal contact between containers, http://ratings:5000 is the address needed to communicate with it. This service contains the following options for requests:

  - addVehicle POST
  - removeVehicle POST
  - getAllVehicles GET
  - addVehicleRating POST
  - addStopRating POST
  - getUserRatings GET
  - getVehicleRatings GET
  - getStopRatings GET

- DeLijnAPI Service:

  Runs on http://localhost:5003 externally, but for internal contact between containers, http://delijnapi:5000 is the address needed to communicate with it. This service contains the following options for requests:

  - getProvinces GET
  - getLinesForProvince GET
  - getLineDirections GET
  - getStopsForLine GET
  - getAllTowns GET
  - getStopsForTown GET
  - getAllStops GET

Usage

As previously mentioned, the website runs on http://localhost:5000. The website is built around the idea of user-friendliness. No login-service is available, which means the user has to enter its username and password for every action for which you must have been registered. Public options are the requests of ratings for users, vehicles and stops. Simply enter a username, vehicle id or stop id and the ratings, if available, will be shown as well as the averages for the vehicles and stops.

To start off, an account can be created using the Register form. When registered, all services on the website become available for use.

When rating a vehicle, only the already created vehicles can be rated. If you wish to rate a vehicle which is not yet available, please add it yourself. Once you've done so, you are the only one who can delete this vehicle. The only other restriction on this is that you are at most the only one who has rated the vehicle. Once other users have rated the vehicle, it is no longer removable.

All ratable stops are already created for you using the De Lijn API services. When adding a rating for a stop there are three options to choose from:

- Select Stop by Line
    - Select from the stops after specifying the province, line and line direction (by selecting the destination).
- Select Stop by Town
    - Select from the stops after specifying the town in the dropdown menu.
- Select from all Stops
    - Even though it is possible to choose from all stops, please take into account that there are almost 2500 towns in Flanders and approximately 36000 stops, making the loading time for this page much slower than with the other rating options.

If you wish to check out other ratings for vehicles, stops or even users, using the search bar on the Ratings page, you can search for either one of them and if available, the ratings will be shown in the search results.