

Microservices - manual

Stijn Rosaer

November 27 2018

1 Requirements

This project is based on the python 3.6 standard. Please make sure that the correct version is available before installing the necessary packages and running the program.

All required packages and dependencies will be installed when compiling the docker images. To install and start the service run following command from the app folder: `docker-compose -f docker-compose-dev.yml up -d --build`. This will install all necessary images with the correct version of dependencies.

To create a full working website with everything online, please run `start.sh` in the app folder. It is important that docker is able to run without sudo rights.

When browsing to `http://localhost` you will reach the home page of the website.

There are multiple possibilities that allow to initialize databases, and seed them with data.

The API from delijn is used to fill the stops with their city and names. The lines on which they are is requested every time a detailed overview of a top is requested.

`docker-compose -f docker-compose-dev.yml` (followed by)

- `run users python manage.py recreate-db`
- `run users python manage.py seed-db`
- `run scores python manage.py recreate-db`
- `run scores python manage.py load-stops`
- `run scores python manage.py seed-stops`
- `run scores python manage.py seed-vehicles`

This will reate all databases and fill them with dummy data.

2 Containers

2.1 client

The client containers runs the react server that is responsible for the user interface that is used to display the website. The `app.py` contains every route that is possible and the home page. In the components' folder I placed all the other pages and elements that are used on that pages. Taking into account that some elements can be shared on multiple pages.

To make the request to the back-end, I used `axios` since this is easy to set up in react and allows easy response handling.

To create a good-looking website, I used `bulma`. This allows a quick creation of styled elements.

2.2 scores

The scores' container contains everything to make the service functional. In here, I placed all the functions that handle the requests to preform various operations. Using this container, it is possible to get an average score, add busses or trams, rate something or remove a vehicle that you added and no one else has rated.

You have to be logged-in to remove/add a vehicle or rate something.

2.3 scores-db

The scores-db container contains the postgresql database that holds all ratings and entities that can be rated.

stops

Column	Type
id	intiger
name	string
entity	string
city	string

stopscores

Column	Type
id	intiger
score	intiger
username	string

vehicles

Column	Type
id	intiger
type	string
username	string

vehiclescores

Column	Type
id	intiger
score	intiger
username	string

2.4 users

The users container handles all user and authentication request. This includes the registration, login and logout.

2.5 users-db

The users-db container contains the postgresql database that holds all user data.

<u>users</u>	
Column	Type
id	integer autoincrement
username	string
email	string
password	string (hashed)
loggedin	boolean

2.6 nginx

To make sure that all traffic is correctly routed and the user doesn't need to enter a port number in the url, I chose to use nginx in an extra container. This allows to define where each request should be forwarded to.

3 API calls

3.1 Users

Url: */users*
Method: GET
Parameters: /
Response: json formatted users

Url: */auth/register*
Method: GET
Parameters:

- username
- email
- password

Response: success object
Info: user will be added to the users table

Url: */auth/login*
Method: POST
Parameters:

- username
- password

Response: success object
Info: verification if username and password combination is correct

Url: */auth/logout*
Method: POST
Parameters:

- username

Response: success object
Info: set logout to true

3.2 Scores

Url: */scores/search/s/id/ < id >*

Method: GET

Parameters: stop id

Response: json list of stops with requested id

Url: */scores/search/s/line/ < line > / < entity >*

Method: GET

Parameters:

- line number
- province

Response: json list of stops on requested line and entity

Url: */scores/search/s/city/ < city >*

Method: GET

Parameters:

- city

Response: json list of stops in requested city

Url: */scores/search/v/id/ < id >*

Method: GET

Parameters:

- vehicle id

Response: json list of vehicles with requested id

Url: */scores/search/v/type/ < type >*

Method: GET

Parameters:

- vehicle type

Response: json list of vehicles of requested type

Url: */scores/vehicle/remove*

Method: POST

Parameters:

- vid (vehicle id)

Response: success object if vehicle is correctly removed

Url: */scores/vehicle*

Method: POST

Parameters:

- vid (vehicle id)
- score
- username

Response: success object with vehicle id if score is set

Url: */scores/stop*

Method: POST

Parameters:

- sid (stop id)
- score
- username

Response: success object with stop id if score is set

Url: */scores/stops/avg*

Method: GET

Parameters: /

Response: success object with list of top and worst stops and their average score

Url: */scores/vehicles/avg*

Method: GET

Parameters: /

Response: success object with list of top and worst vehicles and their average score

Url: */scores/vehicles/id/ < id >*

Method: GET

Parameters:

- vehicle id

Response: json object with all vehicle details of requested id including list of all ratings

Url: */scores/stops/id/ < id >*

Method: GET

Parameters:

- vehicle id

Response: json object with all stop details of requested id including list of all ratings

Url: */scores/addvehicle*

Method: POST

Parameters:

- id (vehicle id)
- type
- username

Response: success object if vehicle is correctly created and added to the database

4 Design choices

I make several choices when designing this system. For the back-end, I decided to divide the service in more logical containers than requested. I split the databases to make a clear separation between the functionality. This is also important for scalability and security. If the security of the scores' database would be compromised, they wouldn't have access to the user data. The passwords are also not stored in plain text but hashed using a `bcrypt` function.

The API's are splitted in the same way as the databases to keep an overview of what functionality can be found where. This way, it would also be possible to use the users' container for another application or even use it for multiple applications at the same time.

I decided to make use of `nginx` as service to route the traffic to the correct microservice. This makes it possible to define what request should end where. This prevents that the user of my service has to enter the port numbers for every request in the url.

Because everything is contained in `docker`, there are no issues with dependencies or versions of plugins. Everything is defined in the `docker` files and the requirements for every service. The `docker-compose-dev.yml` is a place where all the settings for every microservice is defined such as what port the services should be accessed on, what directory they can be found in, what `docker` file to use and the dependencies to other services. This makes it really easy to install everything at once.

5 Website

The website is made using react that is hosted in its own container and compiles the requested pages. To make the layout more attractive, I used bulma for styling and the build in components such as the stars for rating the stops and vehicles.

There are multiple pages that are explained below and images of them on the following pages. I provided multiple users that can be used.

- username: user1, password: 1111
- username: user2, password: 2222

5.1 Web pages

- Home
Landing page with a short introduction
- Stops
A page with two lists. One containing the top 50 rated stops and one containing the worst 50 stops
- Vehicles
A page with two lists. One containing the top 50 rated vehicles and one containing the worst 50 vehicles. If logged in, this page will also show a button to add a vehicle
- Add vehicle
A form that can be used to add a vehicle if logged in
- Search
Several pages to search for stops and vehicles based on different parameters
- Login / Logout / Register
These forms are used to log in, register of the button to logout

- Item specific information

This pages can be accessed by the info button on the lists. It will show all the possible information on a stop or vehicle including a list of all the rating of different people. If logged in, it is also possible to use the start to give a rating. When u are the creator of this vehicle and no one else has given a rating, u can remove the vehicle from the database.

Home

This website can be used to rate busses and trams as well as stops, provided by De Lijn.

Top rated vehicles

ID	Type	Score	Info
4	Bus	★★★★★	Info
3	Bus	★★★★★	Info
2	Bus	★★★★★	Info
6	Bus	★★★★★	Info

Worst rated vehicles

ID	Type	Score	Info
1	Bus	★★★★★	Info
5	Bus	★★★★★	Info
8	Bus	Not rated	Info
7	Bus	Not rated	Info

Top rated stops


ID	Name	Score	Info
101001	Zurenborg	★★★★★	Info
101000	A. Chantrainestraat	★★★★★	Info
590335	Dorp	Not rated	Info
590334	Dorp	Not rated	Info
590333	Hoge Dijken	Not rated	Info
590332	Hoge Dijken	Not rated	Info
590331	Dorpsstraat	Not rated	Info
590330	Dorpsstraat	Not rated	Info
590010	Jan Mioenstraat	Not rated	Info
590009	De Vierweg	Not rated	Info
590008	De Vierweg	Not rated	Info
590007	Scholen Westerlaan	Not rated	Info
590006	Scholen Westerlaan	Not rated	Info
559866	Oude Aalbeeksestraat	Not rated	Info
559715	Stadensteenweg	Not rated	Info

Worst rated stops

ID	Name	Score	Info
101074	Landbouwweg	Not rated	Info
101073	Landbouwweg	Not rated	Info
101072	Heiligstraat	Not rated	Info
101071	Heiligstraat	Not rated	Info
101070	Aartselaarstraat	Not rated	Info
101069	Irisweg	Not rated	Info
101068	Irisweg	Not rated	Info
101067	Werkplaatsen	Not rated	Info
101066	Werkplaatsen	Not rated	Info
101064	Hof Daman	Not rated	Info
101063	Tunnelweg	Not rated	Info
101062	Tunnelweg	Not rated	Info
101061	Industrieweg	Not rated	Info
101060	Industrieweg	Not rated	Info
101055	A. Van De Wielelei	Not rated	Info

Login

Submit



De Lijn rating system

Home

Stops

Vehicles

Search ▾

RegisterLog In

Stop id
Line
City
Vehicle id
Vehicle type

Top rated vehicles

ID	Type	Score	Info
4	Bus	★★★★★	Info
3	Bus	★★★★★	Info
2	Bus	★★★★★	Info
6	Bus	★★★★★	Info

Worst rated vehicles

ID	Type	Score	Info
1	Bus	★★★★★	Info
5	Bus	★★★★★	Info
8	Bus	Not rated	Info
7	Bus	Not rated	Info

Search

Search for stops on a line

Submit

ID	Name	Score	Info
102886	Meibloemstraat	Not rated	Info
102995	J. Notéstraat	Not rated	Info
108900	Ter Elststraat	Not rated	Info
104875	Rubenslei	Not rated	Info
102688	Lamorinièrestraat	Not rated	Info
107290	Kattenbroek	Not rated	Info
108905	Ter Elststraat	Not rated	Info
104050	Loosplaats	Not rated	Info

Stop info

Stop id: 101001

Stop name: Zurenborg

Province: Antwerpen

Enter a rating

★★★★★

Ratings

Username	Score
user2	★★★★★

Vehicle info

Vehicle id: 2

Vehicle type: Bus

Added by: user1

Enter a rating

★★★★★

Ratings

Username	Score
user1	★★★★★
user2	★★★★★

Vehicle info

[Remove](#)

Vehicle id: 1

Vehicle type: Bus

Added by: user1

Enter a rating

Ratings

Username	Score
user1	*****