EasyRide Report

Contents

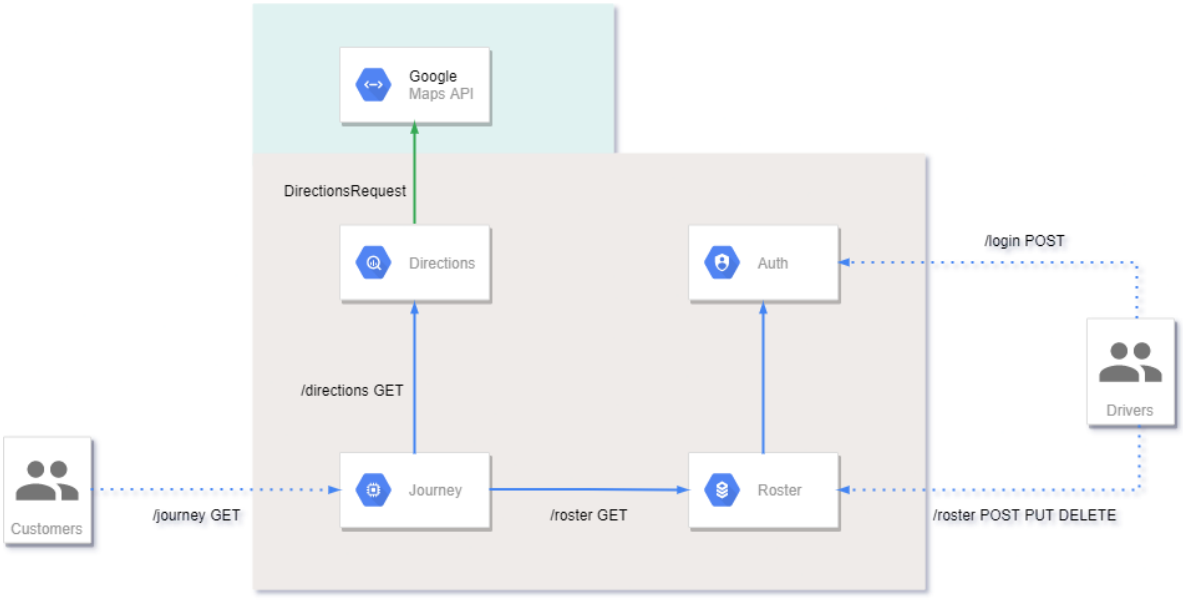
The structure of this submission is as follows:

```
easy-ride
|   docker-compose.yml
|   README.md
|__ API Specification - Full documentation of the APIs of each service
|
|__ Auth - Auth Microservice
|__ Directions - Directions Microservice
|__ Journey - Journey Microservice
|__ Roster - Roster Microservice
|
|__ Documents
    |___ PDF - PDF representations of documents
    |___ Markdown - Markdown representations of documents
```

Note: Read the README in the root directory for further information including test accounts and `docker-compose` instructions.

Architecture

Below, an architecture diagram shows the Microservices identified and the flow of data between them.



The microservices that make up the EasyRide platform are as follows:

- Auth
  - Handles the creation, delivery, and validation of JWT tokens
- Directions
  - Interfaces with the Google Maps API to find the distance of a route
- Journey

    - Provides information about a route including the cost and best driver.
- Roster
    - Handles the store of drivers including adding to roster, removing from roster, and updating price/km

RESTful Operation

The microservices communicate over HTTP using a RESTful API. Full API documentation is available in the `API Specification` folder in the root directory. The API has been described in `.yaml` format using the `OpenAPI` specification. This has then been exported to a browsable API viewer using the `ReDoc` converter.

RESTful Operations have been mapped to HTTP methods as follows:

RESTful Method	**HTTP Method**
READ	GET
CREATE	POST
UPDATE	PUT
DELETE	DELETE

Status codes are used with error messages to convey the success or failure of an operation:

Status Code	**Meaning**
200	Successful Operation
400	Bad Request
401	Unauthorized (likely invalid JWT)
500	Unexpected internal error

Authorisation has been implemented using JWT. Sending a correct username and password to the Auth service will generate a JWT that is active for 5 minutes. Token refreshing has not been implemented. This is something that could be achieved by the frontend re-calling the login endpoint when receiving a `401` response.

Testing

Unit Tests

The project has unit tests for the `Auth` and `Roster` modules. The project has separate Dockerfiles (denoted as `Dockerfile.test`) in each service, as well as as separate `docker-compose.test.yml` file to build the whole application. This can be used with the `docker-compose -f docker-compose.test.yml build` followed by `docker-compose -f docker-compose.test.yml up` to build the application.

Testing with CURL

Below is a subset of the CURL commands used to test the application. A full list of commands can be found in `full curl commands` in the `Documents` directory.

Login

```
curl -X POST -d username=sebvet -d password=astonmartin
http://localhost:8000/login -v

Status 200:
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InNlYnZldCIsIm5hb
WUiOiIiLCJleHAiOjE2MTU1NTgxMzZ9.dDhv7JpV1HmexRgQMFSH9YJH47nkckgFRWJLSIobdco"}
```

Unsuccessful Login

```
Unsuccessful Login:
curl -X POST -d username=notauser -d password=notapassword
http://localhost:8000/login -v

Status 401:
{"error": "Incorrect credentials provided"}
```

Join Roster

```
curl -X POST -H "Content-Type: application/json" --data "
{\"token\":\"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InNlYnZldCIsIm
5hbWUiOiIiLCJleHAiOjE2MTU1NTg2MTN9.6XKKiACOaL3Dny3OOaD64HL8OU9V34xceaOFjmiR-
dU\",\"rate\":5}" http://localhost:8001/roster -v

Status 200:
{"username":"sebvet","name":"Sebastian Vettel","rate":5}
```

Get Journey

```
curl -X GET http://localhost:8003/journey/Exeter/Crediton

Status 200:
{"start_point":"Exeter","end_point":"Crediton","total_distance":14007,"a_road_di
stance":13403,"best_driver":{"username":"sebvet","name":"Sebastian
Vettel","rate":5},"cost":280}
```