README

Quickstart

Wir verwenden Docker, um die Umgebung für unsere API bereitzustellen. Voraussetzung dafür ist das Installieren von Docker, welches je nach Betriebssystem unterscheidlich gemacht werden muss. Die Dokumentation findet sich hier: https://docs.docker.com/install/.

Ist Docker installiert und gestartet, können in einen Commandline Fenster im Projektverzeichnis folgende Befehle ausgeführt werden:

```
shell> docker build -t wdb_mc1_web -f ./dockerfile .
shell> docker-compose up -d
```

Webserver

Für das Aufsetzen des Webservers muss eine Datenbank installiert werden. Wir verwenden MariaDB Server, welches von hier bezogen werden kann: https://mariadb.org/download/. Nach abgeschlossener Installation kann die Datenbank folgendermassen aufgesetzt werden:

```
shell> mysql

MariaDB [(none)]> CREATE DATABASE wdb;
MariaDB [(none)]> CREATE USER 'api'@'%' IDENTIFIED BY 'some_password';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON wdb.* TO 'api'@'%';
MariaDB [(none)]> FLUSH PRIVILEGES;
MariaDB [(none)]> (CTRL + C)

shell> mysql wdb < ./src/sql/structure.sql</pre>
```

Die Datenbank "wdb" ist nun aufgesetzt und kann mit Daten befüllt werden.

Als Nächstes müssen die Zugangsdaten der Datenbank dem Webserver mitgeteilt werden. Dazu die Datei 'credentials.json.sample' nach 'credentials.json' kopieren und die Felder "host, port, user, password, db" auf die Zugangsdaten des Datenbankservers anpassen.

Als Nächstes muss in den Docker Container gewechselt werden. Im Container muss folgender Befehl ausgeführt werden, um die Python Abhängigkeiten zu installieren und die API zu starten:

```
shell> pipenv run python ./src/main.py
```

Die API ist nun unter http://localhost:5000 verfügbar.

Test

Um die API zu testen muss man in den Docker Container wechseln und dort in der Shell folgenden Befehl ausführen:

```
shell> pipenv run python src/test.py
```

Waren alle Tests erfolgreich, wird folgende Meldung ausgegeben:

```
Ran (n) tests in (n)s
```

Branching System

In diesem Projekt arbeiten wir mit dem "Gitflow Workflow". Dabei wird pro Feature lokal ein Feature-Branch erstellt, nach Beedigung lokal mit dem "dev" Branch gemerged und nach "origin/dev" gepusht. Für die Auslieferung der Software müssen gemäss Kapitel "Test" alle Tests erflogreich durchlaufen werden. Erst dann darf der Fortschritt in den "main" Branch gemerged werden. Als Tag wird der Versionscode im Branch "main" gekenntzeichnet.

Weitere Informationen: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Übersicht Calls

User

Method	Route	Description
GET	/user	Get all users
GET	/user/ <user_id></user_id>	Get a specific user
GET	/user/ <user_id>/ratings</user_id>	Get all ratings of a specific user
GET	/user/ <user_id>/ratings/movies</user_id>	Get all movies which a specific user has rated
POST	/user	Create a user
DELETE	/user/ <user_id></user_id>	Delete a user
PUT	/user/ <user_id></user_id>	Update a user

Movie

Method Route Description	
--------------------------	--

Method	Route	Description
GET	/movie	Get all movies. Possible params: ?limit_n= only returns top n movies by vote average. ?title= only returns movies with titles containing the given string
GET	/movie/ <movie_id></movie_id>	Get a specific movie
POST	/movie	Create a movie
DELETE	/movie/ <movie_id></movie_id>	Delete a movie
PUT	/movie/ <movie_id></movie_id>	Update a movie

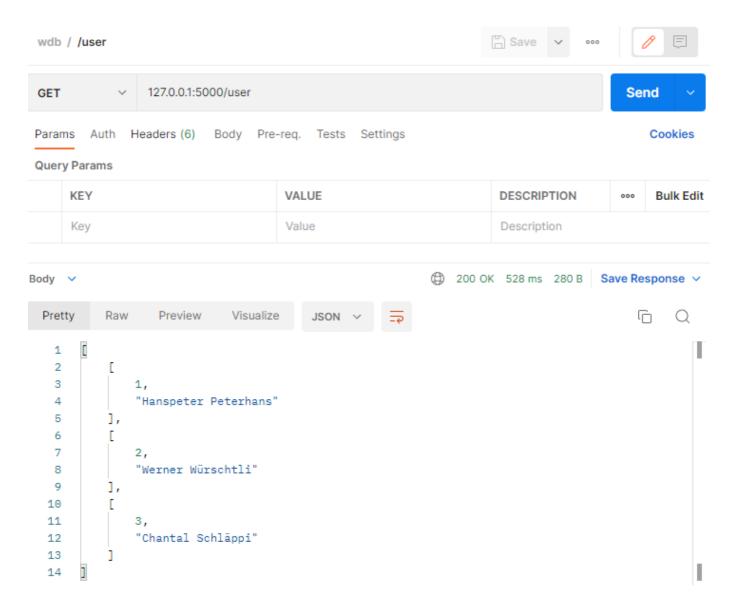
Ratings

Method	Route	Description
POST	/rating	Create a rating
PUT	/rating/ <rating_id></rating_id>	Update a rating
DELETE	/rating/ <rating id=""></rating>	Delete a rating

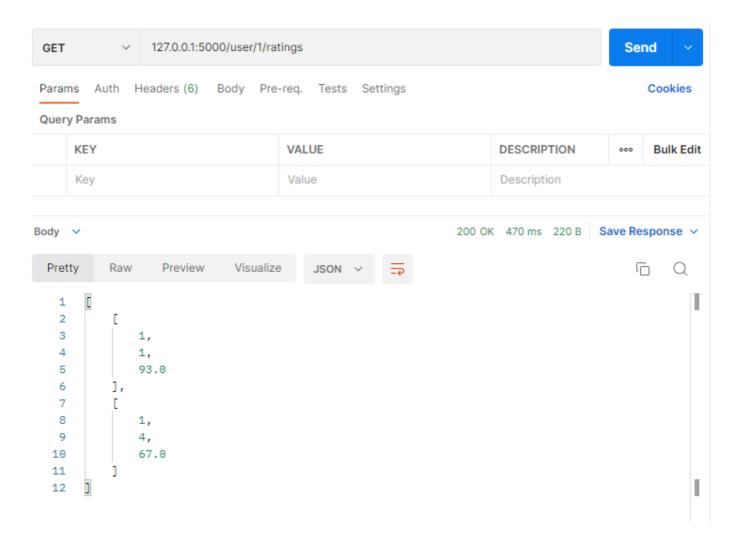
Sample Requests

Wurde die API gemäss Quickstart, Webserver gestartet, können Abfragen gemäss eingestellter URL im Connection File gemacht werden. Die Software Postman bietet sich dazu an. Als Beispiel wurden folgende Abfragen getestet:

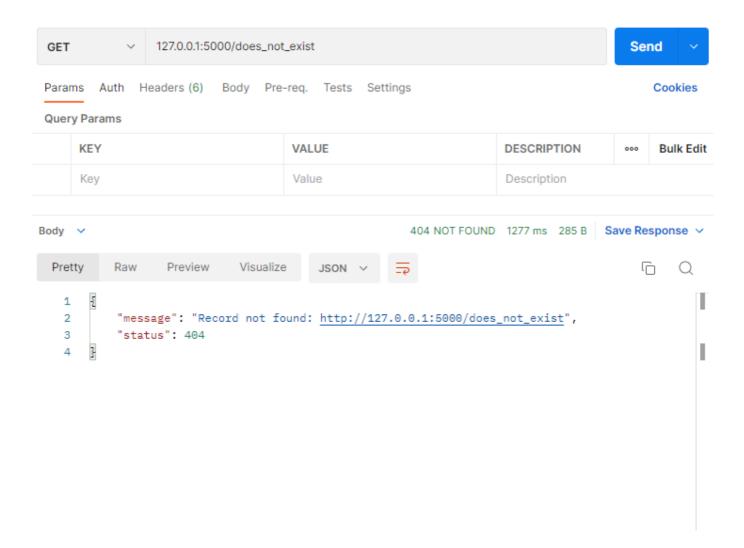
GET /users



GET /ratings



GET /does_not_exist



Report

Sie finden unseren Bericht in ./docs/Report.pdf.