ChargeHub Berlin Project Documentation

Data Science (M.Sc.) // Advanced Software Engineering // Project Documentation
Prof. Dr. S. Ipek-Ugay

Team 4
https://github.com/ifti23/chargeHub-Berlin
Group member[4]:
106034, Raphael Bergner
106769, Roma Khalil Bhurgri
106099, Muhammad Iftikhar,
106070, Tharun Johny Mekala

Introduction to the Topic and Your Two Use Cases

ChargeHub Berlin is a web application designed to help people in Berlin locate charging stations for their electric cars. It provides information about nearby charging stations, including their availability and operational status.

Use case 1: Charging Station Search by Postal Code

> Users can search for charging stations within their postal code area and view which ones are currently available. This feature helps users plan their charging needs more efficiently

Use case 2: Malfunction Report

> Users can report malfunctions or issues at charging stations, ensuring that operators are informed and other users are alerted.

Technology Stack:

- Communication Tools: Zoom Meetings and Groupchat
- Modeling Tools: Miro, draw.io
- Languages: Python, TypeScript, CSS, HTML, Dockerfile, Bash
- IDE: Pycharm and Visual Code
- Frontend Technology: Angular
- Backend Technology: Rest-API using Flask
- Database: Postgresql
- Version Control: GitHub
- Style Guide Ensurance: Git-Hooks using black and isort

Project Development Documentation

Test-Driven Development

We decided to use **Angular** as our frontend solution. Since not all of the members are equally well adversed in the respective languages, we splitted our team in two expert teams.

1. **Frontend Team:** Responsible for designing and implementing the user interface, along with defining the expected REST API routes. This team also defined partial test cases to validate the API responses based on their expected functionality.

2. **Backend Team:** Tasked with developing the backend using Flask and ensuring it adhered to the specifications outlined by the frontend team. They then implemented the REST API endpoints and finalized the current form of the unit tests.

Effective collaboration between team members was ensured through regular Zoom meetings, lasting approximately 30 minutes to 1 hour, held at least once a week. During these meetings, we regularly shared our progress using screen-sharing. Occasionally, the respective teams held individual sessions for pair-programming or to find solutions to critical problems. Meeting times were coordinated using a group chat, where notifications about codebase updates were also published.

Resulting Status:
The backend is implemented using **Flask**, a Python web framework, and is integrated with a **PostgreSQL** database. Both components are orchestrated using a single **Docker Compose** file, ensuring a seamless deployment process and environment consistency. For the unit-tests we opted for an in-memory solution, to not interfere with the in production data.

The database defines three core database tables:

- **User Table:**
  Stores user-related information, username, password, e-mail and optionally the phone number
- **Postal Codes Table:**
  Contains data for postal code areas in Berlin. The information is inserted into the databse on startup of the docker-compose
- **Charging Stations Table:**
  Maintains details of individual charging stations, such as location, availability, and operational status. On startup all possible stations are inserted into the database with a functional status

The data is currently simply pasted from the first project. The same way we loaded the data there into the streamlit-app, we now load it into the database. Here we will work on a solution that is more suited for a sql-database. For example is the geographical data currently stored as a plain string, which is not the optimal. Other than this functionality we have not recycled any code from the first project so far.

The backend currently provides these essential services to support the frontend functionality:

- **User Registration and Login:**
  API routes for securely registering new users and logging in existing ones.
- **Postal Codes Data Retrieval:**
  Routes to fetch all available postal code areas or specific data for a given postal code.
- **Charging Station Queries:**
  Allows users to retrieve detailed information about charging stations in their specified postal code area. This function is only available if the user is logged in.

The web-application is implemented using **Angular**, which currently includes:

- A **Registration/Login Page** for user authentication.
- A **Serachbar** to search for specific postal codes.
- A **Map Interface** for displaying charging station locations and availability based on user queries.

The current status of the implemented unit-tests has been validated using the python tool **coverage**. We are currently on 94% test-coverage.

Additional Tools

During the development phase we used multiple sources to implement the code. From the website Stack Overflow, over the documentation pages of the respective tool, to Large Language Models (LLM). In this section examples of the useage of LLMs will be highlighted.

Use-Case 1 – missing imports

During the development we used the tool **coverage** from python to check the coverage of our unit-tests. These tests run using the IDE Pycharm. Using the console command from the tool resulted in import errors. There were no solutions stated on either the documentation pages, for **unittests** or **coverage**. Consulting with a LLM resulted in multiple possible solutions. We then found a working console command, that additionally sets the environment variable PYTHON_PATH.

Use-Case 2 – additional tests

After finishing the first unit-test iteration, we considered, if there is anything missing. We consulted a LLM to give us hints on what we might have not yet tested. For a seamless integration we not only provided the code for the classes (entities, like user) and the tests, but also the whole code structure. The model then added some useful tests and additions to the classes.

Use-Case 3 – less experienced languages

To ensure, that our codebase is **PEP8** conform, we implemented a Git-Hook. This hook runs the python third party python tools black and isort. Since no one in our team is well versed in the bash language, we asked a LLM to implement this rather simple hook. After not seeing any problems, we used the tool. It worked without a problem

Project Completion

Milestones

Here is a list of milestones we wish to achive. The green highlighted points have been achived so far

- basic angular frontend desing
- build the frontend map
- build the frontend registration and login screen
- build the frontend the search for postal codes
- first working backend rest-api routes
- first working backend database
- established connection between database and rest-api
- established connection between rest-api and frontend
- achive 80% test-coverage
- implement all necessary routes
  - implement the routes for the 1. Use-Case
  - implement the routes for the 2. Use-Case
- achive more than 90% test-coverage in all files
- build a docker-compose for the whole setup
- demonstrate a working cycle by registering two users, one marking a station as malfunctioning

Next Phase

In the next phase of the project we will focus on finalizing the integration of all tools. That includes the implementation of a functioning docker-compose file. Each member will run respective system test, to make sure we all are happy with the results.

Another important part is to optimize the database usage. The current Idea is to create an additional table for all the coordinates, and a table that mappes each postal code to their coordinates. This makes sense, since the coordinates represent edges. That means each coordinate can have multiple postal code areas linked to. To not break the current status of the project we will use branches in GitHub.

Technical Challenges During Development Phase

The team had multiple challenges to overcome during the development process. Here a sample is provided

Challenge 1 – experience

At the beginning of the project, we had our first meeting. There, we discussed our current expertise. We found out, that each member had a unique background, and experiences in different fields. After we decided based on the expertise which member should work in the backend and who in the frontend, we realised, that the members in the frontend are not well versed using streamlit.

Solution:

We let the frontend use a different technology than python, they are more used to. This had another positive effect. We did not have to manage possibly occuring merge conflictes, since the backend and frontend team are working in two different folders.

Challenge 2 – integration

After the development of the backend was finished so far, that in could be tested by the frontend team, we hit an obstecal. The frontend team could not run the Flask application. Even after a session of screen-share we could not figure out why.

Solution:

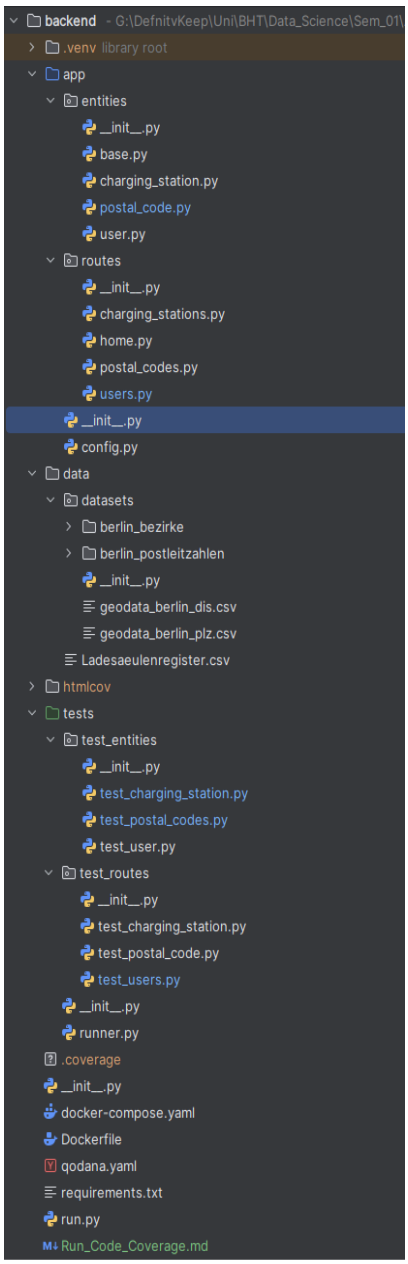
Instead of spending more time on this problem, we created a Dockerfile along with a docker-compose.yaml file. Using this technology ensured that we could efficiently run the application in a consistent and reproducible environment.

Appendix

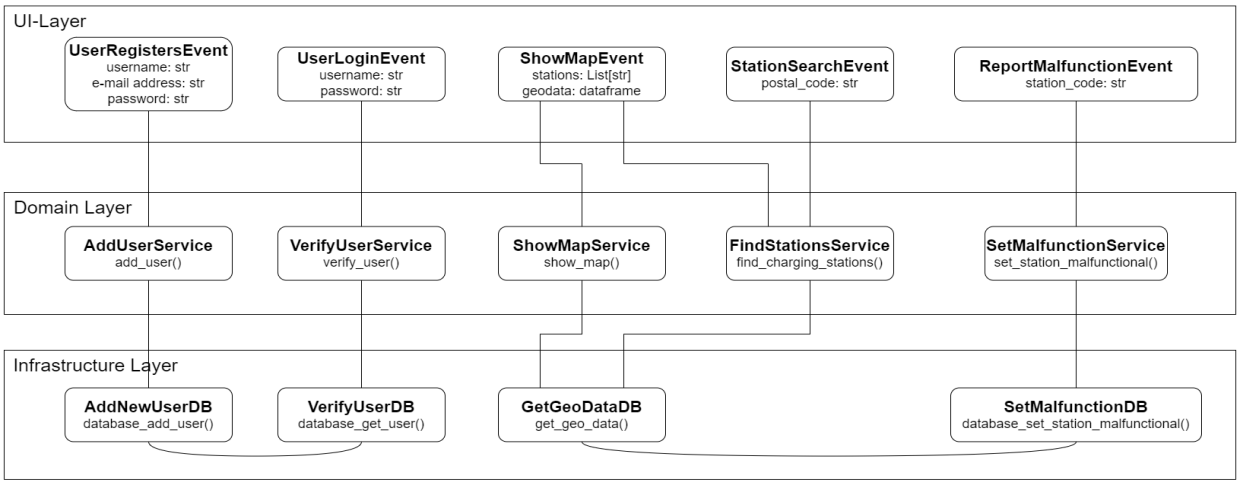
Test-coverage

File ▲	statements	missing	excluded	coverage
app__init__.py	100	19	0	81%
app\config.py	22	0	0	100%
app\entities__init__.py	0	0	0	100%
app\entities\base.py	5	0	0	100%
app\entities\charging_station.py	31	0	0	100%
app\entities\postal_code.py	27	1	0	96%
app\entities\user.py	33	0	0	100%
app\routes__init__.py	0	0	0	100%
app\routes\charging_stations.py	37	7	0	81%
app\routes\home.py	5	1	0	80%
app\routes\postal_codes.py	12	0	0	100%
app\routes\users.py	32	2	0	94%
tests__init__.py	0	0	0	100%
tests\test_entities__init__.py	0	0	0	100%
tests\test_entities\test_charging_station.py	41	1	0	98%
tests\test_entities\test_postal_codes.py	54	1	0	98%
tests\test_entities\test_user.py	53	1	0	98%
tests\test_routes__init__.py	2	0	0	100%
tests\test_routes\test_charging_station.py	58	1	0	98%
tests\test_routes\test_postal_code.py	45	1	0	98%
tests\test_routes\test_users.py	61	1	0	98%
Total	618	36	0	94%

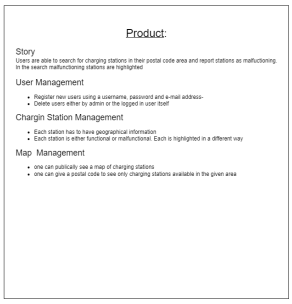
Project-Setup:



Event-Flow Diagram:



Event-Flow Diagram:



Product:

Story
Users are able to search for charging stations in their postal code area and report stations as malfuctioning. In the search malfunctioning stations are highlighted

User Management
- Register new users using a username, password and e-mail address.
- Delete users either by admin or the logged in user itself

Chargin Station Management
- Each station has to have geographical information
- Each station is either functional or malfunctional. Each is highlighted in a different way

Map Management
- one can publically see a map of charging stations
- one can give a postal code to see only charging stations available in the given area