ChargeHub Berlin Project Documentation

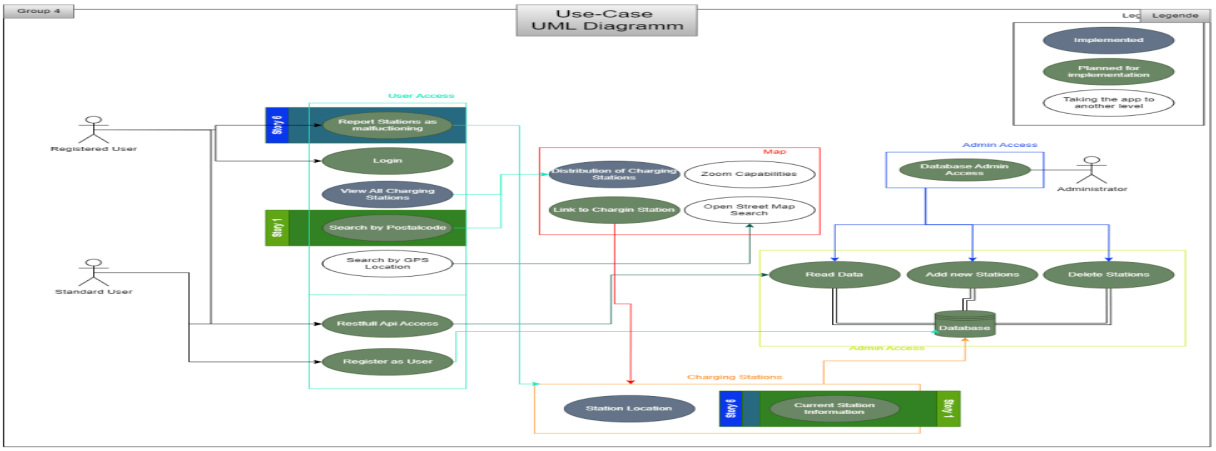
Data Science (M.Sc.) // Advanced Software Engineering // Project Documentation
Prof. Dr. S. Ipek-Ugay

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 electric vehicle (EV) users in Berlin easily find nearby charging stations and check their availability. The app also allows users to report malfunctions at stations, ensuring smooth operations and timely maintenance.

Two use-cases[snap shot from miro]



Use Case 1: Charging Station Search

- **Problem**: Users struggle to find nearby charging stations.
- **Objective**: Provide real-time station availability by postal code.
- **Goal**: Enable efficient trip planning.

Use Case 2: Malfunction Reporting

- **Problem**: Malfunctions disrupt charging services.
- **Objective**: Allow users to report issues directly.
- **Goal**: Ensure timely maintenance and reliability.

Technology Stack:

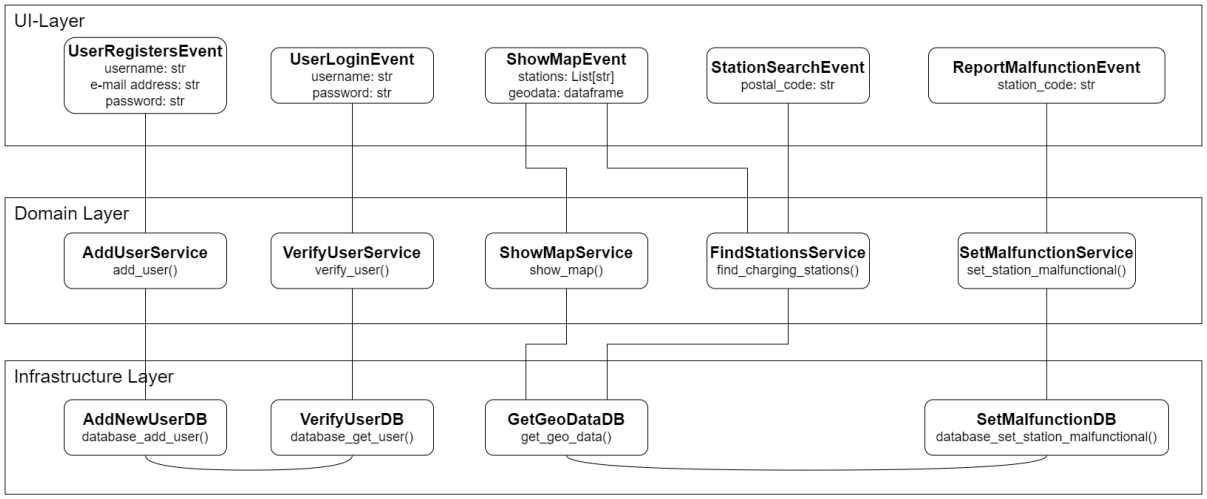
- Communication Tools: Zoom Meetings and Groupchat
- Modeling Tools: Miro, draw.io, Google Docs
- Languages: Python, TypeScript, CSS, HTML, Dockerfile, Bash
- IDE: Pycharm and Visual Code
- Frontend Technology: Angular
- Backend Technology: Rest-API using Flask
- Database: Postgresql
- Version Control: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

Domain Modeling and Event Structure

We chose our two use cases, because they fit well together. At first a user searches for charging stations in their area. Additionally they are informed about the current status of the charging station, while they themself can notify others, that the are currently using the station.

After considering the possible actions a user can take, we derived five events that can occur. They are displayed in the following image in the UI-Layer

Domain Event Flow



In the middle is the startup event displayed. All available charging stations are shown in the ui. Then new users can register and already registered users can login. The last two events then manage the charging stations. Users can search for all available charging stations in a given postal code; they are then displayed in the UI map. Next to each station is a button, where logged in users can change the operation status of the charging station.

Further uml-diagrams and screenshots can be found in the appendix or our git repository under documentation/

Project Development Documentation

At first we divided the team into two groups: one focused on the frontend and the other on the backend. Since not everyone had experience with writing tests, we made sure to assign testing responsibilities mainly to team members with a background in that area. In our first meeting, we introduced ourselves and based on our individual skill sets and backgrounds, we assigned tasks accordingly. The team then proceeded to work on their respective parts, with ongoing coordination between the frontend and backend teams.

b] Were you always able to adhere to the TDD principle during development?

During development we tried to adhere to the Test-Driven-Development (TDD) principles. One major problem was the usage of a Large Language Model (LLM). When we wanted to have tests generated, the model provided the resulting class as well. That would not have been a problem if the results would have been without errors. This led to a lacking, more superficial execution of TTD.

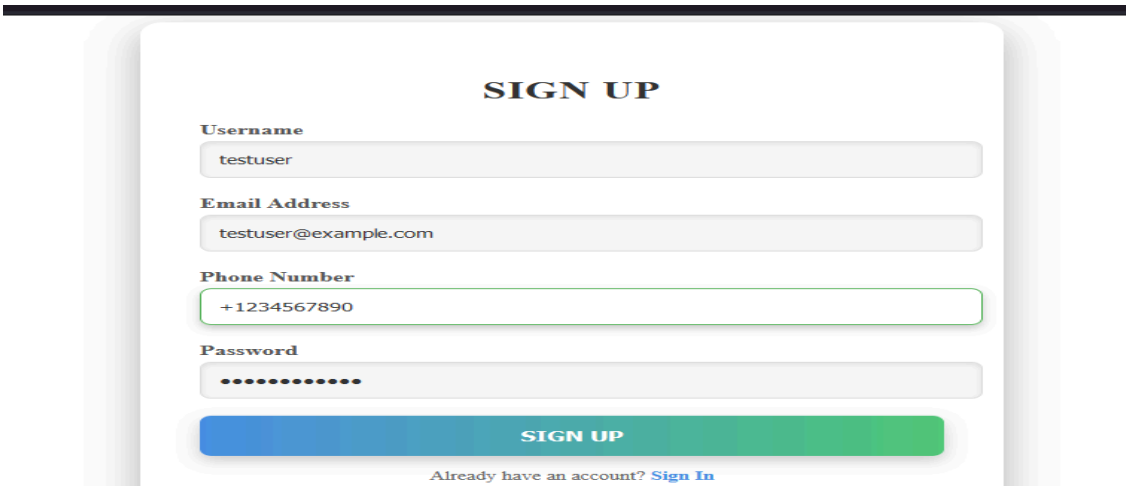
But utilizing a LLM we managed a test coverage of approximately 93%. See the appendix for the complete analysis. To measure this we used the open-source tool coverage. To ensure a most detailed coverage we made sure to have well defined exception handling. In the test we then covered both, the positive as well as the negative cases.

For the tests we used an InMemory database. For development we opted for a Postgresql database. The application can be started in either mode, using a specific flag. To ensure, that the database is exists, we also created a docker-compose.yaml, that starts the backend with a Postgresql database. This ensures that the rest-api runs even without further needed setup.

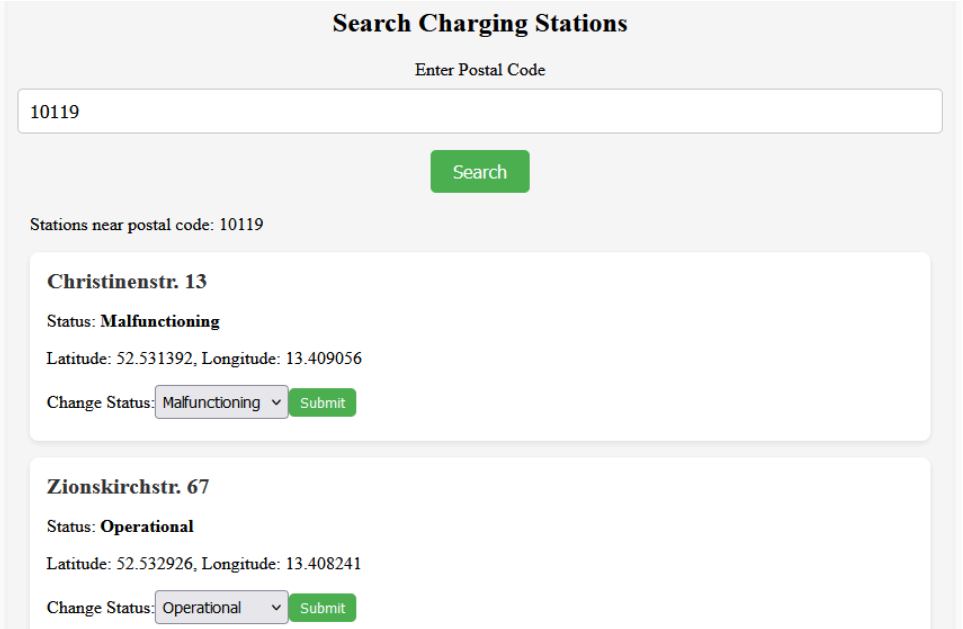
On startup all the necessary data is loaded into the database; this includes all postal codes of Berlin, all charging stations with reference to their postal codes, and a basic user. The rest of the backend is splitted into three parts: events, domain and infrastructure.
Then the frontend can interact with the backend using a rest-api. These are defined in the events. Each event is linked to a service, which is a subpart of the domain. The event formats the in- and output of the data. The services then implement the logic of operations. They call upon the infrastructure, in our case the database operations, to manage the request. A good example is here the search by postal code.
The event simply states if the given input is correct, and returns the result.
The service on the other hand calls first the infrastructure to find if the postal code is valid. Then it calls the infrastructure to find all charging stations if the postal code is valid.

The Frontend consists of an angular framework with four parts

1. **Sign-Up and Sign-In Forms:**
   These forms ensure that only authenticated users can sign up and sign in. They include validation checks to confirm user credentials before granting access.



2. **Search Bar:**
   The search bar allows users to enter a postal code to search for nearby charging stations. After entering the postal code, the user can check the availability of stations in that area.
3. **Malfunction Report Button/Submit button**:
   If a user encounters a malfunctioning charging station, they can click the Submit button to report the issue. A dropdown will appear, allowing the user to select the station's current status: Malfunctioning, Operational, or Used. Once the user selects the appropriate status and clicks Submit, the status of the charging station will be updated accordingly in the system.



4. **Map**:
   Each charging station is displayed on a map of berlin.

Technical Challenges During Development Phase

The team had multiple challenges to overcome during the development process.

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 it. This had another positive effect. We did not have to manage possibly occurring merge conflicts, 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 it could be tested by the frontend team, we hit an obstacle. 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

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

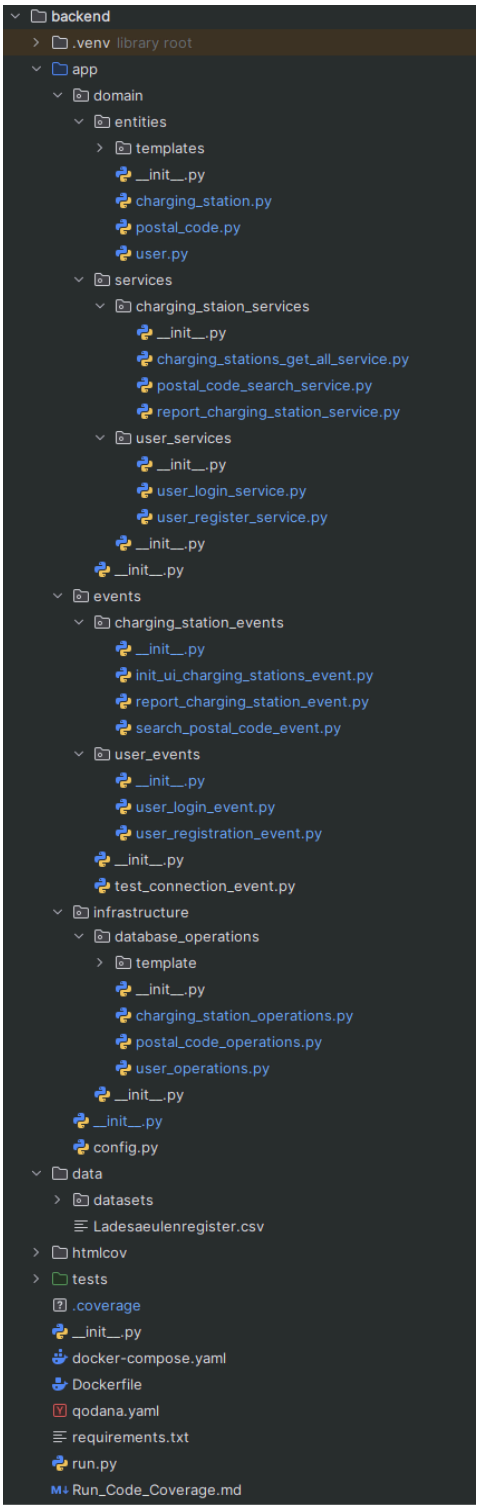
After finishing the first unit-test iteration, we considered, if there is anything missing. We consulted a LLM to give us hints on what we might have not yet tested. For a seamless integration we not only provided the code for the classes (entities, like user) and the tests, but also the whole code structure.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

All the further images can be found in the git repository

Project Structure - blue highlighted python files contain code.



Coverage Report

File	statements	missing	excluded	coverage
app__init__.py	102	19	0	81%
app\config.py	20	0	0	100%
app\domain__init__.py	0	0	0	100%
app\domain\entities__init__.py	0	0	0	100%
app\domain\entities\charging_station.py	92	13	0	86%
app\domain\entities\postal_code.py	27	2	0	93%
app\domain\entities\templates__init__.py	0	0	0	100%
app\domain\entities\templates\base.py	5	0	0	100%
app\domain\entities\user.py	41	1	0	98%
app\domain\services__init__.py	0	0	0	100%
app\domain\services\charging_staion_services__init__.py	0	0	0	100%
app\domain\services\charging_staion_services\charging_stations_get_all_service.py	10	0	0	100%
app\domain\services\charging_staion_services\postal_code_search_service.py	14	0	0	100%
app\domain\services\charging_staion_services\report_charging_station_service.py	16	0	0	100%
app\domain\services\user_services__init__.py	0	0	0	100%
app\domain\services\user_services\user_login_service.py	13	0	0	100%
app\domain\services\user_services\user_register_service.py	20	0	0	100%
app\events__init__.py	0	0	0	100%
app\events\charging_station_events__init__.py	5	0	0	100%
app\events\charging_station_events\init_ui_charging_stations_event.py	11	0	0	100%
app\events\charging_station_events\report_charging_station_event.py	19	0	0	100%
app\events\charging_station_events\search_postal_code_event.py	23	5	0	78%
app\events\test_connection_event.py	5	0	0	100%
app\events\user_events__init__.py	5	0	0	100%
app\events\user_events\user_login_event.py	22	2	0	91%
app\events\user_events\user_registration_event.py	27	20	0	26%
app\infrastructure__init__.py	0	0	0	100%
app\infrastructure\database_operations__init__.py	0	0	0	100%
app\infrastructure\database_operations\charging_station_operations.py	26	3	0	88%
app\infrastructure\database_operations\postal_code_operations.py	17	0	0	100%
app\infrastructure\database_operations\template__init__.py	0	0	0	100%
app\infrastructure\database_operations\template\template_operations.py	12	0	0	100%
app\infrastructure\database_operations\user_operations.py	16	0	0	100%
tests__init__.py	0	0	0	100%
tests\test_database_operations__init__.py	0	0	0	100%
tests\test_database_operations\test_charging_station_operations.py	44	1	0	98%
tests\test_database_operations\test_postal_code_operations.py	30	1	0	97%
tests\test_database_operations\test_user_operations.py	51	1	0	98%
tests\test_entities__init__.py	0	0	0	100%
tests\test_entities\test_charging_station.py	57	1	0	98%
tests\test_entities\test_postal_codes.py	52	1	0	98%
tests\test_entities\test_user.py	53	1	0	98%
tests\test_events__init__.py	0	0	0	100%
tests\test_events\test_charging_station_events__init__.py	0	0	0	100%
tests\test_events\test_charging_station_events\test_init_ui_charging_stations_event.py	31	1	0	97%
tests\test_events\test_charging_station_events\test_report_charging_station_event.py	51	1	0	98%
tests\test_events\test_charging_station_events\test_search_postal_code_event.py	39	1	0	97%
tests\test_events\test_test_connection_event.py	13	1	0	92%
tests\test_events\test_user_events__init__.py	0	0	0	100%
tests\test_events\test_user_events\test_user_login_event.py	43	1	0	98%
tests\test_events\test_user_events\test_user_registration_event.py	41	1	0	98%
tests\test_services__init__.py	0	0	0	100%
tests\test_services\test_charging_station_services__init__.py	0	0	0	100%
tests\test_services\test_charging_station_services\test_get_all_charging_stations.py	32	1	0	97%
tests\test_services\test_charging_station_services\test_postal_code_search_service.py	33	1	0	97%
tests\test_services\test_charging_station_services\test_report_charging_station_service.py	41	1	0	98%
tests\test_services\test_user_services__init__.py	0	0	0	100%
tests\test_services\test_user_services\test_user_login_service.py	40	1	0	98%
tests\test_services\test_user_services\test_user_register_service.py	48	1	0	98%
Total	**1247**	**82**	**0**	**93%**

coverage.py v7.6.10, created at 2025-01-26 20:50 +0100

UML - Complete Use Case 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

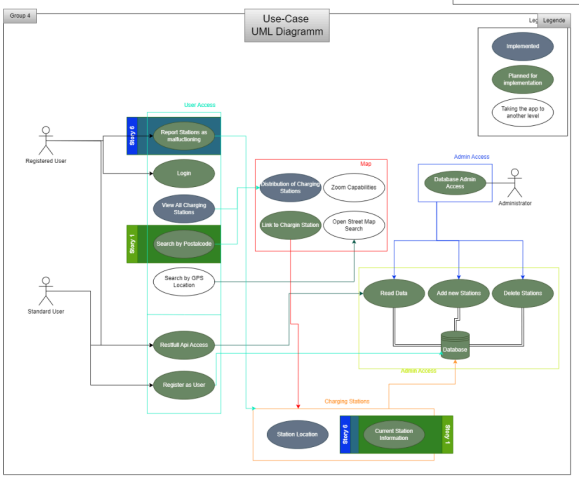
- Register new users using a username, password and e-mail address-
- Delete users either by admin or the logged in user itself

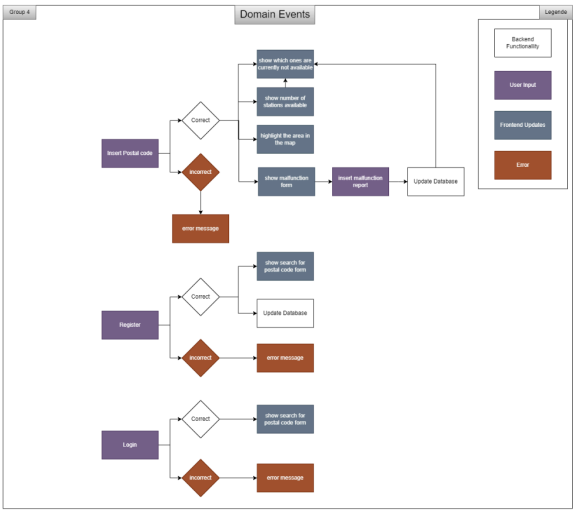
Chargin Station Management

- 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





Interaction Sequence Diagram