Advanced Software Engineering Documentation

Shadi Farzankia, Farel Arden, Rene Heldmaier, Hameez Ahmad

14 January 2025

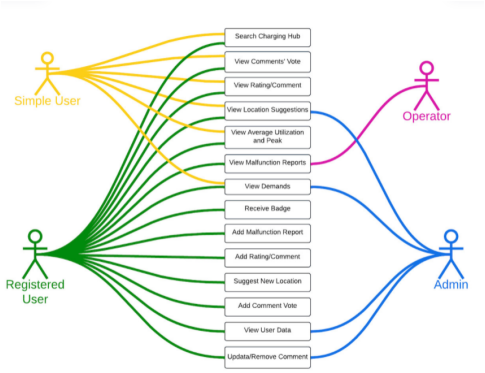
1 Introduction to the Topic and Use Cases

This documentation is for the final project of Advanced Software Engineering, titled "ChargeHub Berlin." The project aims to provide a comprehensive platform for managing and visualizing charging stations across Berlin. Key features include:

This project leverages modern software engineering practices, including Domain-Driven Design (DDD) and Test-Driven Development (TDD).

1.1 Test Cases

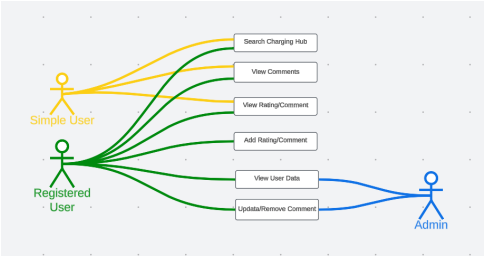
Figure 1: Use Case Diagram - Initialized Idea



This use case diagram depicts the functionalities available to different user roles: Simple Users (yellow), Registered Users (green), Operators (pink), and Admins (blue). Simple Users can access basic information like charging hub locations, comments, and utilization data. Registered Users have extended privileges, including adding feedback and location suggestions. Operators monitor utilization, malfunctions, and location suggestions. Admins possess the highest level of access, managing user data and system content.

To streamline development, our team prioritized the two most critical use cases from the diagram. This focused approach aims to deliver core value to users by enabling them to add and view ratings and comments for each charging station. These features are essential for building a collaborative community and improving service quality.

Figure 2: Use Case Diagram - Realized



This use case diagram illustrates the different functionalities available to three types of users: Simple User, Registered User, and Admin. Each user type is represented by a stick figure with a corresponding label and color: yellow for Simple User, green for Registered User, and blue for Admin.

- **Simple User (Yellow)**: Can search for charging hubs, view comments, and view ratings/comments.
- **Registered User (Green)**: Has additional capabilities such as adding ratings/comments, viewing user data, and updating/removing comments, in addition to the functionalities available to Simple Users.
- **Admin (Blue)**: Focuses on managing user data and updating/removing comments, ensuring the integrity and accuracy of the information within the system.

1.2 Domain Logic and Objectives

After focusing on the Use Case Diagram, we conclude 3 objectives that we are going to implement in this web application:

- **User Log in and Register:** To ensure authentic and valuable feedback, users have the option to log in to their accounts. Registered users can easily add reviews, track their charging history, and personalize their experience.
- **Add Rating/Comment:** This functionality allows registered users to contribute their feedback on each charging station. Users can provide ratings and write comments about their experiences, which helps other users make informed decisions and encourages continuous improvement of the charging stations.
- **View Rating/Comment:** Both simple and registered users can access this feature to see the feedback provided by others. By viewing ratings and comments, users can get insights into the quality and reliability of the charging stations, helping them choose the best location for their needs.

1.3 Key Features

The project offers key features that enhance the user experience and provide valuable insights:

1. **User Authentication:** Enables secure user registration and login, ensuring data privacy and personalized experiences.
2. **Interactive Map:** Displays a map of Berlin with visualized electric charging stations and resident density.
3. **Charging Station Search:** Allows users to easily search for charging stations on the map based on their location or specific criteria.
4. **User-Generated Content:** Facilitates the addition and viewing of user comments and ratings for each charging station, fostering community engagement and sharing of experiences.
5. **Data Integration:** Combines geospatial data, user-generated content, and user authentication for a comprehensive and dynamic user experience.

1.4 Technology Stack

The following technologies were used in the development of the **ChargeHub Berlin** project:

- **Programming Language**: Python
- **Web Framework**: Flask (for backend development)
- **Frontend Technologies**: HTML, CSS, JavaScript, htmx
- **Data Visualization**: Folium (for interactive maps)
- **Data Interchange Format**: JSON (for data storage and communication)
- **Version Control**: Git (with GitHub/GitLab for repository management)
- **IDE**: Visual Studio Code (or any other IDE used by the team)
- **Testing Framework**: pytest (for unit and integration testing), coverage
- **Dependency Management**: pip (for Python package management)

2 Project Development Documentation

The Project Development Documentation section provides a comprehensive overview of the technical and methodological approaches used during the development of the ChargeHub Berlin project. It covers the Domain Modeling and Event Structure, detailing the application's design based on Domain-Driven Design (DDD) principles, including domain event flows and bounded contexts. The Test-Driven Development (TDD) section explains how the team implemented TDD, ensuring high test coverage and describing the backend architecture and UI components. Additionally, it outlines the Integration of Explored Datasets, highlighting how existing datasets were incorporated and tested, and the LLM Integration, showcasing how Large Language Models were utilized for tasks like code optimization or test generation. This section serves as a detailed record of the development process, reflecting the team's technical decisions, challenges, and solutions.

2.1 Domain Modeling and Event Structure (DDD)

The Domain Modeling and Event Structure section outlines the core design of the ChargeHub Berlin application using Domain-Driven Design (DDD) principles. This approach ensures that the system is organized around the business domain, with clear boundaries and well-defined interactions.

2.1.1 Domain Event Flows

The **Domain Event Flows** represent the sequence of events triggered by user interactions within the system. For the **ChargeHub Berlin** project, the following domain events are identified:

- **User Authentication**:
  - **Event**: `accountCreated.py`
    Triggered when a new user registers.
  - **Event**: `accountLoggedIn.py`
    Triggered when a user logs in.
- **Charging Station Search**:
  - **Event**: `chargingStationSearched.py`
    Triggered when a user searches for charging stations.
  - **Event**: `chargingStationSelected.py`
    Triggered when a user selects a specific charging station.
- **Review Management**:

  - **Event**: `reviewAdded.py`
    Triggered when a user adds a review for a charging station.

- **User Profile Updates**:
  - **Event**: `userUpdated.py`
    Triggered when a user updates their profile information.

These events are encapsulated within the `domains/events` folder, ensuring a clear separation of concerns and adherence to DDD principles.

2.1.2 Bounded Contexts

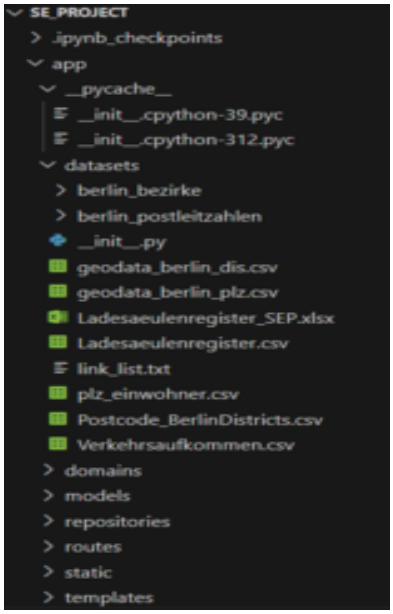
The **Bounded Contexts** define the boundaries within which a particular domain model applies. For the **ChargeHub Berlin** project, the following bounded contexts are identified:

- **User Management**:
  - **Entities**: `User.py` (in `domains/entities`).
  - **Repositories**: `user_repository.py` (in `repositories`).
  - **Events**: `accountCreated.py`, `accountLoggedIn.py`, `userUpdated.py` (in `domains/events`).
- **Charging Station Management**:
  - **Entities**: `ChargingStation.py` (in `domains/entities`).
  - **Repositories**: `map_repository.py` (in `repositories`).
  - **Events**: `chargingStationSearched.py`, `chargingStationSelected.py` (in `domains/events`).
- **Review Management**:
  - **Entities**: `Review.py` (in `domains/value_objects`).
  - **Repositories**: `review_repository.py` (in `repositories`).
  - **Events**: `reviewAdded.py` (in `domains/events`).

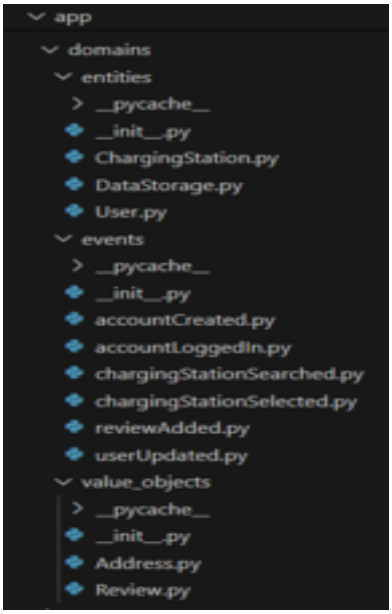
These bounded contexts ensure that each domain model is isolated and can evolve independently.

2.1.3 Project Structure: IDE Screenshots

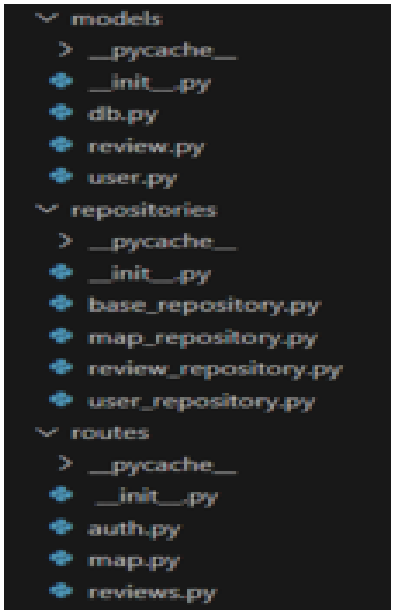
The project structure is organized into **domain**, **infrastructure**, and **application** layers, adhering to DDD principles. Below is a description of the structure, along with references to the provided screenshots:



(a) Project Dataset

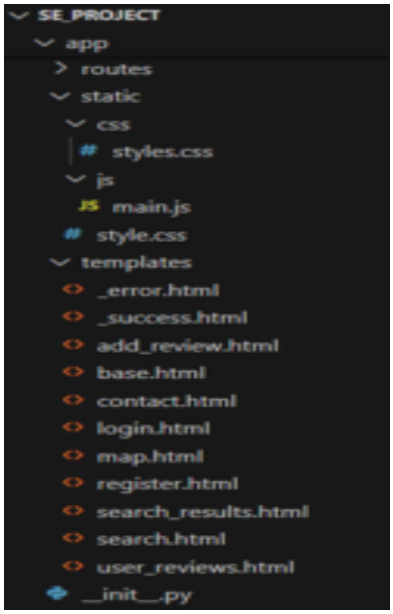


(b) Project Domain (Entities, Events, and Value Objects)

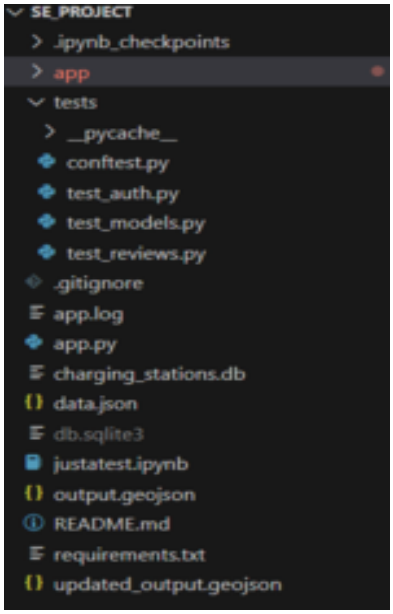


(c) Project Models, Repositories, and Routes

Figure 3: Project Structure (First Row)



(a) Project Front-End Structure



(b) Other Project Documents

Figure 4: Project Structure (Second Row)

2.1.4 Domain Layer

- **Location**: `app/domains`
- **Contents**:
  - **Entities**: Core business objects like `User.py`, `ChargingStation.py`, and `DataStorage.py`.
  - **Events**: Domain events such as `accountCreated.py`, `chargingStationSearched.py`, etc.
  - **Value Objects**: Immutable objects like `Address.py` and `Review.py`.
- **Screenshot Reference**: `Figure 3.(b)` shows the `domains` folder structure.

2.1.5 Infrastructure Layer

- **Location**: `app/models` and `app/repositories`
- **Contents**:
  - **Models**: Database models like `user.py` and `review.py`.
  - **Repositories**: Data access logic like `user_repository.py`, `map_repository.py`, and `review_repository.py`.
- **Screenshot Reference**: `Figure 3.(c)` shows the `models` and `repositories` folders.

2.1.6 Application Layer

- **Location**: `app/routes`
- **Contents**:
  - **Routes**: Handlers for HTTP requests, such as `auth.py`, `map.py`, and `reviews.py`.
- **Screenshot Reference**: `Figure 3.(c)` shows the `routes` folder.

2.1.7 Static and Templates

- **Location**: `app/static` and `app/templates`
- **Contents**:
  - **Static Files**: CSS (`styles.css`) and JavaScript (`main.js`) for frontend styling and functionality.
  - **Templates**: HTML files like `base.html`, `map.html`, and `search.html` for rendering the frontend.
- **Screenshot Reference**: `Figure 4.(a)` shows the `static` and `templates` folders.

2.1.8 Tests

- **Location**: `tests`
- **Contents**:
  - **Test Files**: Unit and integration tests for different components, such as `test_auth.py`, `test_reviews.py`, etc.
- **Screenshot Reference**: `Figure 4.(b)` shows the `tests` folder.

2.1.9 Datasets

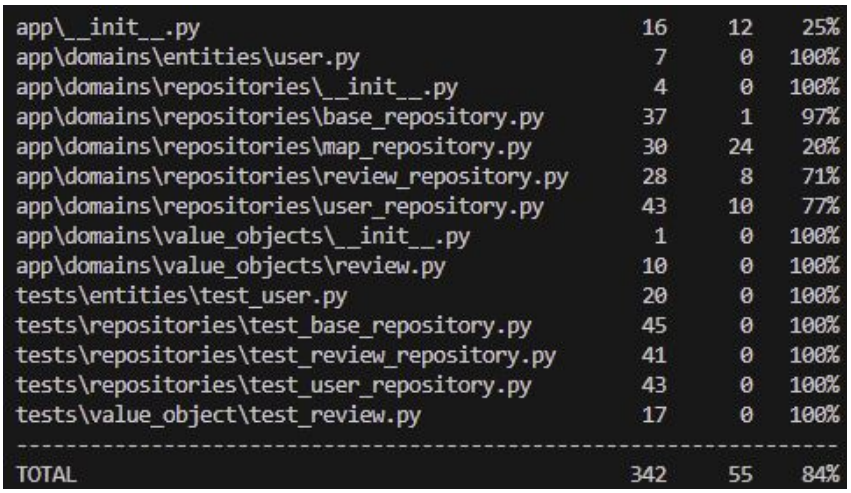
- **Location**: `datasets`
- **Contents**:
  - **Data Files**: CSV, Excel, and GeoJSON files used in the project, such as `Ladesaeulenregister.csv` and `output.geojson`.
- **Screenshot Reference**: `Figure 3.(a)` shows the `datasets` folder.

2.2 Test-Driven Development (TDD)

In this project, we adopted the Test-Driven Development (TDD) methodology to ensure the robustness and reliability of the system. The process generally begins by writing test cases first before implementing the corresponding functionality. However, we were not able to fully adhere to it in all cases. In our case, this happened primarily when time constraints and evolving requirements made it necessary to focus on rapidly implementing features, such as those related to the review entity, before their exact specifications were fully defined. To implement the tests, we used the **pytest** framework, a powerful and flexible testing tool for Python. Pytest provides features like easy test discovery, a simple syntax for writing tests, and detailed reporting.

2.2.1 Test Coverage

To achieve comprehensive testing and ensure at least 80% test coverage for our project, we implemented a well-structured approach by categorizing test files based on their respective scopes. These categories included repositories, routes, events, entities, and value object models. This systematic organization allowed us to independently and thoroughly validate each component of the application, ensuring reliability and maintainability. To measure the overall test coverage, we utilized the **coverage** library, which integrates well with pytest and analyzes the extent of our codebase covered by tests. Impressively, we achieved an 84% test coverage, as illustrated in the image below:



app__init__.py	16	12	25%
app\domains\entities\user.py	7	0	100%
app\domains\repositories__init__.py	4	0	100%
app\domains\repositories\base_repository.py	37	1	97%
app\domains\repositories\map_repository.py	30	24	20%
app\domains\repositories\review_repository.py	28	8	71%
app\domains\repositories\user_repository.py	43	10	77%
app\domains\value_objects__init__.py	1	0	100%
app\domains\value_objects\review.py	10	0	100%
tests\entities\test_user.py	20	0	100%
tests\repositories\test_base_repository.py	45	0	100%
tests\repositories\test_review_repository.py	41	0	100%
tests\repositories\test_user_repository.py	43	0	100%
tests\value_object\test_review.py	17	0	100%
TOTAL	342	55	84%

Figure 5: Text Coverage Result

2.2.2 Backend Architecure

- **Technologies:** For the backend implementation, we utilized the following two technologies:
  - **flask:** We chose Flask because it is a lightweight and flexible web framework that allows for rapid development and easy integration with other tools and libraries. Flask's simplicity and modular design made it an excellent choice for implementing our project's backend logic and managing routes.
  - **flask Caching:** The flask_caching module has been used in the code to improve the performance and efficiency of the Flask application by caching frequently accessed data and responses.
  - **SQLite:** We opted for SQLite as our database solution because it is a lightweight, serverless, and self-contained database engine. SQLite's ease of setup and low resource requirements made it ideal for our project, enabling quick data storage and retrieval without the need for a complex database server configuration.
- **Data Models:** The data model in our project consists of the following components:
  - **User:** The users table represents the users of our application. Each user has a unique identifier and authentication details.

Column Name	Data Type	Description	Constraints
`id`	INT	Unique identifier for the user.	Primary Key, Autoincrement
`username`	TEXT	Unique username for the user.	Unique, Not Null
`password_hash`	TEXT	Hashed password for authentication.	Not Null

  - **Review:** The reviews table stores feedback submitted by users for specific charging stations.

Table 1: Reviews Table Description

Column Name	Data Type	Description	Constraints
`id`	INT	Unique identifier for the review.	Primary Key, Autoincrement
`user_id`	INT	Identifier of the creator	Foreign Key, Not Null
`username`	TEXT	Username of the creator	Not Null
`station_id`	INT	Identifier of the charging station	Foreign Key , Not Null
`rating`	REAL	Rating provided by the user	Not Null
`comment`	TEXT	User's comment	Not Null

2.2.3 UI Components

In this section, we will describe the user interaction flow and technical details about the implementation of our UI:

- **User Intraction Flow:**
  - **Homepage:**
    - Users are welcomed with an interactive map powered by Leaflet.js, showcasing all available charging stations across Berlin.
    - Map markers represent charging stations, and clicking on a marker displays station details like name, address, and rating.
    - Registered users can log in to access additional features like adding reviews.
  - **Authentication:**
    - Users can register and log in using the authentication pages.
    - Logged-in users have personalized access to features like adding reviews or viewing their own review history.
  - **Review Functionality:**
    - Registered users can add reviews and ratings for charging stations via a dedicated review page.
    - All users (both registered and simple users) can view the reviews and average ratings of charging stations directly from the map.
  - **User Reviews Page:**
    - Logged-in users can view a summary of all their reviews/ratings
- **Technical Details:**
  - **Front-end:**
    - **HTML & CSS:** Provides the structure and style for pages.
    - **JavaScript:** Enhances interactivity, such as dynamic map updates and user interactions with charging stations.
    - **HTMX:** Ensures smooth partial page updates, reducing full-page reloads for an enhanced user experience.
    - **Leaflet.js:** Used for rendering the map and handling geospatial data visualization.
    - **Font Awesome:** Adds aesthetic icons, such as those for social media and other elements.
    - The UI is optimized for desktop and mobile views
- **App Screenshots:** These images provide a visual representation of the user interface and the features available on each page. They serve to demonstrate the functionality and design of the application in detail.

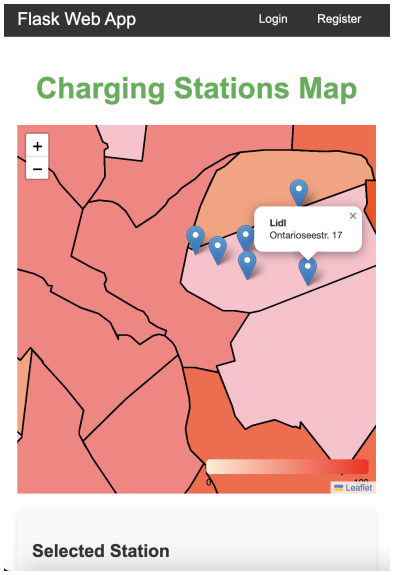


Figure 6: Chargin Hub Selection

Figure 7: Registering Page

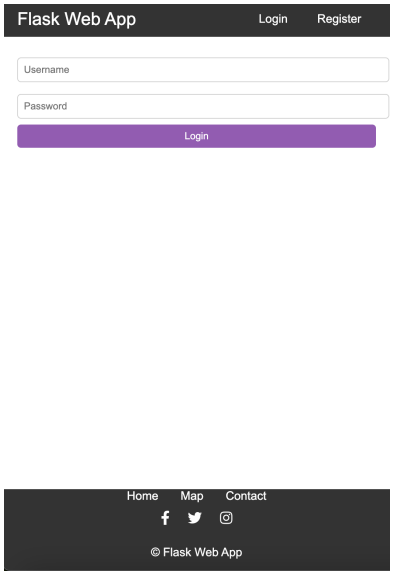


Figure 8: Login Page

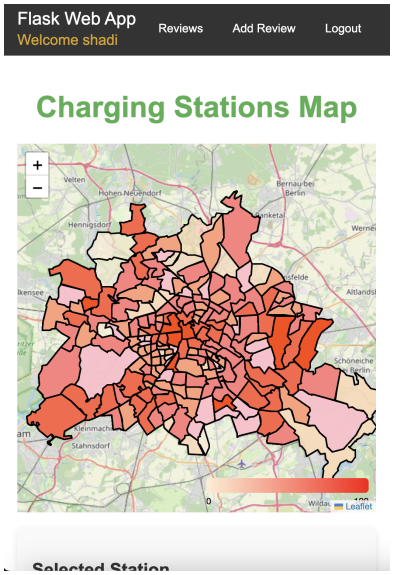


Figure 9: User Logged in

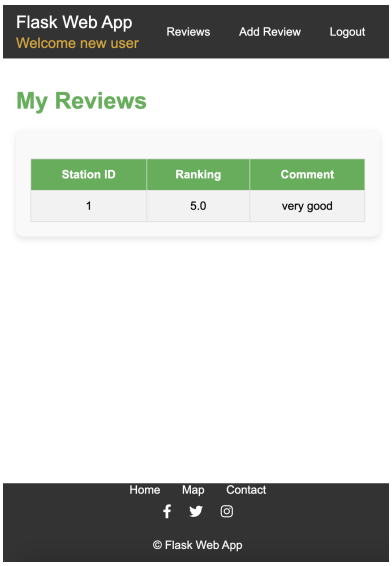


Figure 10: Review List

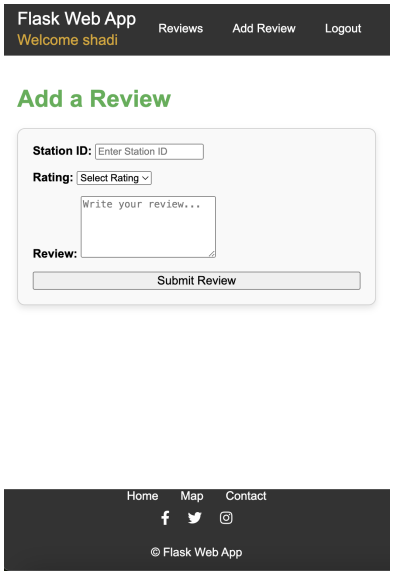


Figure 11: Add Reivew

2.3 Integration of Explored Datasets

2.3.1 Approach

To integrate datasets from the first part of the module, we utilized `Ladesaeulenregister_SEP.xlsx` and `geodata_berlin_dis.csv` to map out polygons for the interactive map. These datasets provided the necessary geospatial data to display charging stations across Berlin's districts accurately. The `Ladesaeulenregister_SEP.xlsx` dataset contained detailed information about charging stations, while `geodata_berlin_dis.csv` provided the geographical boundaries of Berlin's districts. By combining these datasets, we were able to create an interactive map that visually represents the distribution of charging stations in Berlin.

2.3.2 Accessing Existing Methods

Since we transitioned from **Streamlit** to **Flask** for this project, we did not directly reuse existing methods from the previous project. Instead, we rebuilt the functionality from scratch to align with the new framework. However, we leveraged some libraries from the first project, such as **Folium**, for creating

the interactive map. This allowed us to maintain consistency in the visualization of geospatial data while adapting to the new architecture.

2.3.3 Formatting Issues

During the integration process, we encountered several data formatting challenges. For example, some data formats did not adhere to the expected structure, which caused issues when displaying charging stations on the map. To resolve these challenges, we cleaned and preprocessed the datasets to ensure compatibility with the map visualization. While **Test-Driven Development (TDD)** played a role in identifying and resolving some of these issues, its implementation was not as frequent as initially planned. Nevertheless, TDD helped us catch formatting inconsistencies early in the development process. LLMs were also important for the development of this program as they helped us finding some solutions.

2.3.4 DDD Paradigm

To ensure the project structure adhered to **Domain-Driven Design (DDD)** principles, we used **Git** for version control and maintained a shared repository. Tasks were divided evenly

2.4 LLM Integration

2.4.1 LLM Usage

Large Language Models (LLMs) were primarily used for **code generation**, **test case creation**, and **code optimization**. In situations where the team encountered challenges in implementing specific functionalities or optimizing existing code, LLMs provided valuable assistance by generating code snippets and suggesting improvements. Additionally, LLMs played a key role in helping the team understand and implement **test-driven development (TDD)**, as they provided insights into creating effective test cases.

2.4.2 Tools Used

The following tools were used for LLM integration:

- **DeepSeek**: For generating code snippets and optimizing existing code.
- **ChatGPT**: For brainstorming test ideas and understanding testing concepts.
- **Gemini**: For generating LaTeX code and improving documentation formatting.

2.4.3 Examples

- **Test Case Generation**:
  - Before using LLMs, the team had limited experience with testing. With the help of LLMs, we gained a clear understanding of testing principles and were able to generate meaningful test cases. For example, LLMs provided ideas for test cases related to user authentication and charging station search functionality, which were then implemented in the project.
- **Code Optimization**:
  - In one instance, the team struggled to optimize a function for filtering charging stations by district. By using DeepSeek, we received a more efficient implementation that improved the performance of the feature.

3 Technical Challenges During Development

Challenge 1: Lack of Isolated Functionalities for Testing

Initially, we faced difficulties in testing our code because the functionalities were not isolated, making it hard to write unit tests. To address this, we reorganized our codebase according to **Domain-Driven Design (DDD)** principles. By breaking down the application into bounded contexts (e.g., user management, charging station management), we were able to isolate functionalities and write more effective tests. This approach significantly improved our test coverage and code reliability.

Challenge 2: Merge Conflicts in Git

During the development process, we encountered frequent **merge conflicts** when integrating changes from different Git branches. To resolve this, we adopted a more structured workflow:

- Regularly pulling the latest changes from the main branch.
- Using Git's version control features to review and revert to previous file versions when conflicts arose.
- Holding team discussions to align on coding standards and reduce inconsistencies.

These practices minimized merge conflicts and improved collaboration.

Challenge 3: Transition from Streamlit to Flask

We decided to rebuild the application using **Flask** instead of Streamlit, as Flask is more versatile and widely used. However, this transition required us to recreate key features, such as the interactive map and charging station pinpoints, from scratch. To overcome this challenge:

- We performed extensive **data cleaning** on the initial datasets to ensure compatibility with Flask.
- Leveraged libraries like **Folium** to rebuild the map functionality.
- Iteratively tested and refined the implementation until it met our requirements.

Challenge 4: Team Coordination and Coding Styles

As this was our first time working as a team, we faced challenges in coordinating schedules and aligning coding styles due to differing work and study commitments. To address this:

- We held regular **team meetings** to discuss progress, resolve issues, and align on goals.
- Established clear coding standards and documentation practices to ensure consistency.
- Supported each other by collaboratively solving problems and sharing knowledge.

These efforts improved teamwork and ensured smoother project execution.

4 Project Completion

4.1 Milestones Achieved

We successfully delivered a functional program that aligns with the initial project goals, despite some deviations from the original UML diagram. The final application adheres to **Test-Driven Development (TDD)** and **Domain-Driven Design (DDD)** principles, ensuring a robust and maintainable codebase. Key achievements include:

- Completing the core functionalities, such as user authentication, charging station search, and review management.
- Delivering a well-documented project, including both the program and the documentation.
- Demonstrating effective teamwork, where each member contributed despite busy schedules, and supported one another to overcome challenges.

4.2 Technical Innovations and Lessons Learned

Throughout the project, we gained valuable insights and learned important lessons:

- **Team Collaboration**:
  - Working with new team members was challenging at first, but building strong connections and open communication helped us overcome initial hurdles. This experience taught us the importance of teamwork and adaptability in achieving shared goals.

- **TDD and DDD Principles**:
  - We learned the significance of **Test-Driven Development (TDD)** and **Domain-Driven Design (DDD)** in creating reliable and scalable software. Although we did not apply these principles as frequently as intended, we recognized their value in minimizing bugs and ensuring code quality. Testing should not be an afterthought but a **core priority** that forms the foundation of the development process.
- **Technical Growth**:
  - Transitioning from **Streamlit** to **Flask** required us to rebuild key features, such as the interactive map, from scratch. This challenge deepened our understanding of web frameworks and geospatial data visualization.
  - Integrating **Large Language Models (LLMs)** for code generation and test case creation introduced us to new tools and techniques, enhancing our problem-solving skills.
- **Project Management**:
  - Balancing work, studies, and project deadlines taught us the importance of time management and clear task delegation. Regular team meetings and collaborative problem-solving were essential to our success.

GitHub Repository

The source code and documentation for the **ChargeHub Berlin** project are available in the following GitHub repository: **ChargeHub Berlin GitHub Repository**.