# ChargeHubBerlin Project Documentation

## Team Details

- **GitHub Repository**: ChargeHubBerlin GitHub
- **Streamlit App URL**: ChargeHubBerlin Streamlit App
- **Group Members**:
    1. Luke Richard - luri1537@bht-berlin.de (Matriculation No.: 106100)
    2. Saad Tozibar Rahman - sato7894@bht-berlin.de (Matriculation No.: 106618)
    3. Sivasankar Subramanian - sisu9000@bht-berlin.de (Matriculation No.: 105998)
    4. Muhammad Abdullah Khan - mukh7058@bht-berlin.de (Matriculation No.: 106111)

---

## Introduction to the Project and Use Cases

The **ChargeHubBerlin** project is designed to provide users with an interactive interface to search for electric vehicle (EV) charging stations in Berlin and report station malfunctions. The project employs **Domain-Driven Design (DDD)** and **Test-Driven Development (TDD)** principles to ensure an organized and robust implementation.

### Use Case 1: Search by Postal Code

- Users can:
    - Enter a postal code in Berlin.
    - Retrieve a list of charging stations in the specified area.
    - View the results on an interactive map.

### Use Case 2: Report Malfunction

- Users can:
    - Report a malfunction at a specific charging station by entering details such as postal code, station ID, and comments.
    - View a list of all reported malfunctions, sorted by date.
    - Filter malfunction reports by postal code, also sorted by date.

---

## Technology Stack

- **Programming Language**: Python
- **Frameworks/Libraries**: Pytest, Pandas, Streamlit, Folium, Geopandas
- **Database**: CSV-based InMemory database for simplicity
- **Frontend Tool**: Streamlit (for user interface)
- **Development Environment**: VS Code

---

## Project Architecture

### Folder Structure and Responsibilities

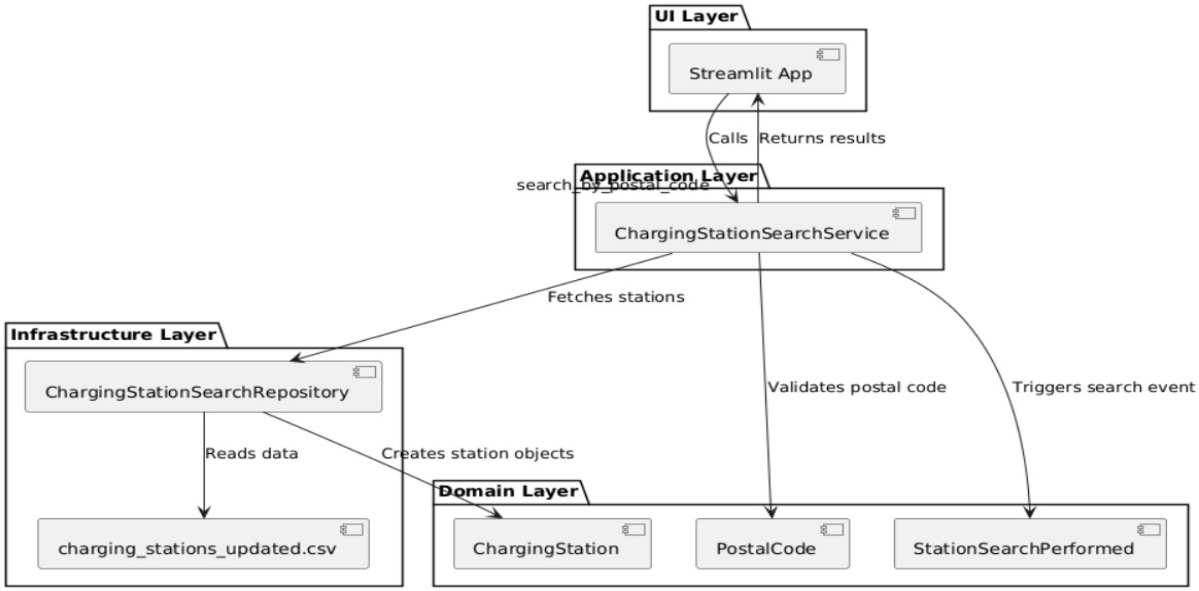
1. **Charging Folder**: Contains all core functionality.
    - **Application Layer**: Handles high-level operations.
        - station_search_service.py: Implements the "Search by Postal Code" use case.
        - malfunction_report_service.py: Implements the "Report Malfunction" use case.
    - **Domain Layer**: Core functionality.
        - **Entities**:
            - charging_station.py: Represents a charging station.
            - malfunction_report_entities.py: Represents malfunction reports.
        - **Events**:
            - station_search_performed.py: Logs searches.
            - malfunction_report_events.py: Logs malfunction reports.
        - **Value Objects**:
            - postal_code.py: Validates and manages postal code logic.
            - malfunction_report_value_objects.py: Handles report-specific logic.
    - **Infrastructure Layer**:
        - **Repositories**:
            - charging_station_search_repository.py: Fetches station data.
            - malfunction_report_repository.py: Handles malfunction report data.
        - **Datasets**: Includes CSV files (charging_stations_updated.csv, malfunction_reports.csv, etc.).
2. **Shared Folder**: Contains reusable utilities like methods.py, HelperTools.py.
3. **Tests Folder**: Implements test cases for both use cases (station_search.py, malfunction_report.py).

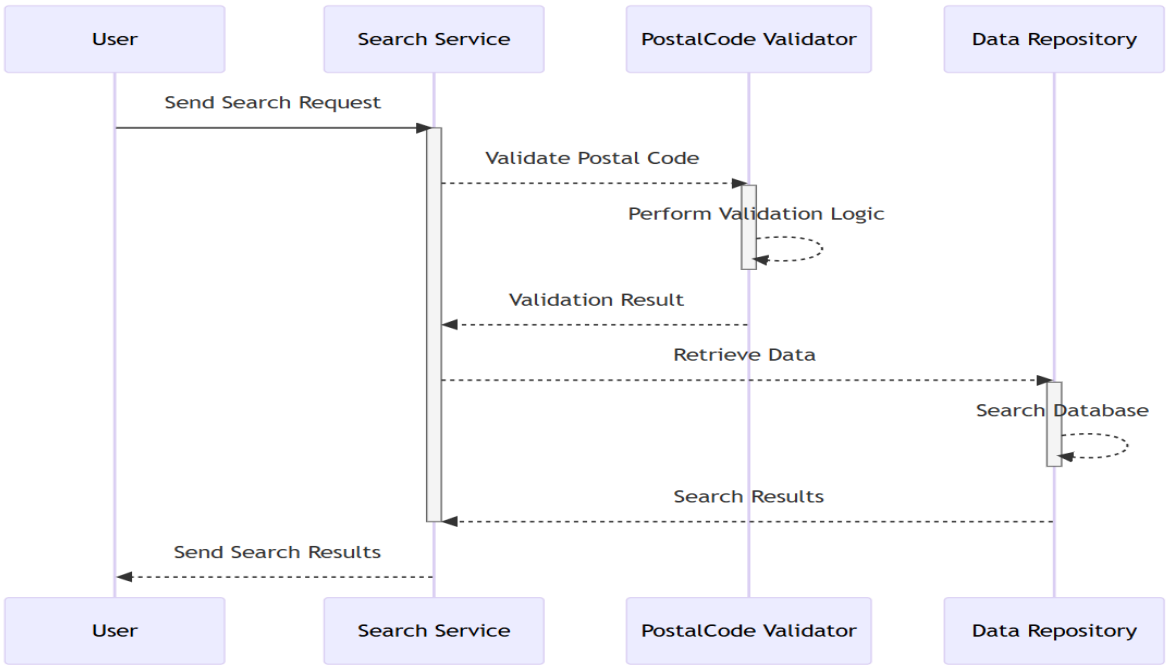
---

## Domain Event Flow

### Search by Postal Code

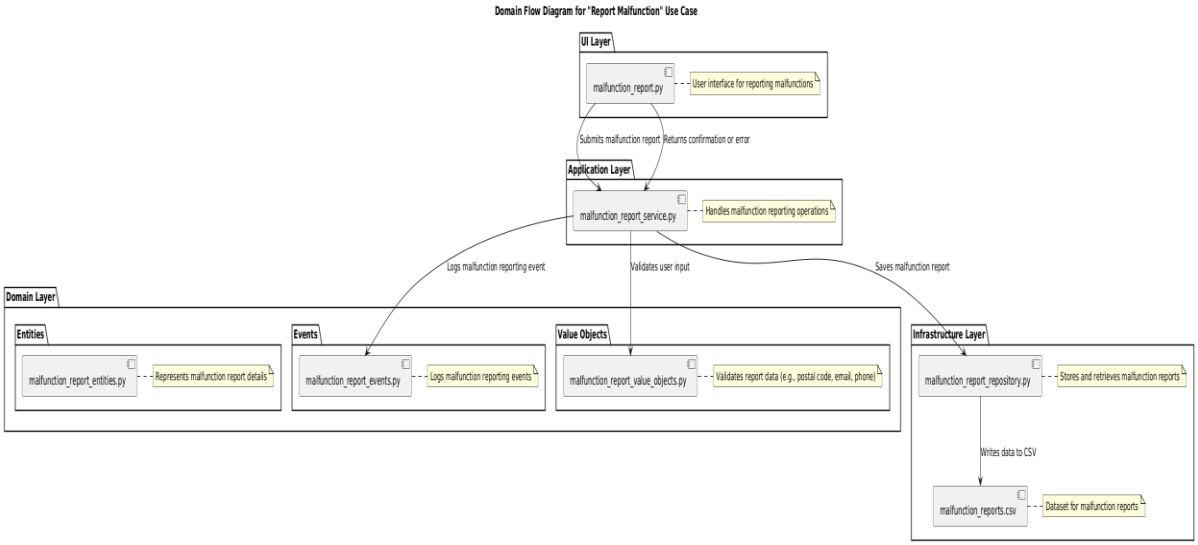
1. **Input**: User enters a postal code in Streamlit.
2. **Validation**: Input is validated using PostalCode.
3. **Data Fetching**: Repository retrieves matching stations.
4. **Event Logging**: StationSearchPerformed logs the search event.
5. **Output**: Results are displayed interactively.



### Report Malfunction

1. **Input**: User submits a malfunction report in Streamlit.
2. **Validation**: User input (postal code, station ID, comments, email, and phone number) is validated.
3. **Data Handling**: Report is saved in malfunction_reports.csv through the repository.
4. **Event Logging**: MalfunctionReportLogged logs the report event.
5. **Output**: Reports can be viewed or filtered.




---

## TDD Implementation

### Development Workflow

1. **Red Phase**: Defined test cases for edge scenarios like invalid emails, phone numbers, and incomplete malfunction reports.
2. **Green Phase**: Implemented functionalities step by step to pass the tests:
    - Added validation for postal codes, email, and phone numbers.
    - Integrated malfunction reporting features.
    - Ensured seamless interaction between UI and backend.
3. **Refactor Phase**: Improved code structure and added inline documentation.

### Test Cases

### Search by Postal Code:

- Valid postal code like 10115 retrieves stations.
- Invalid postal code (e.g., 99999) raises an exception.

### Report Malfunction:

- test_report_malfunction_for_valid_data: Ensures valid data creates a report successfully.
- test_cannot_report_malfunction_for_invalid_email: Invalid email addresses are rejected.

- test_cannot_report_malfunction_for_invalid_phone_number: Invalid phone numbers are rejected.
- test_cannot_report_malfunction_without_description: Reports without a description are rejected.
- test_get_reports_by_postal_code: Retrieves all reports for a specific postal code.
- test_get_all_reports: Retrieves all malfunction reports from the repository.

### Test Coverage

- Achieved ~85% test coverage by thoroughly testing both use cases.

---

## UI and Streamlit Integration

### Features

1. **Search by Postal Code**: Users can search for stations by entering a postal code.
2. **Report Malfunction**: Users can report issues via a dedicated interface.
3. **Map Visualization**: Displays charging station locations and availability.

### Interaction Flow

- **Search**: Input postal code → Validate → Fetch results → Display on the map.
- **Report Malfunction**: Input details → Validate → Save report → Display confirmation.

---

## Integration of Datasets

1. Preprocessed data using Pandas/Geopandas.
2. Ensured compatibility of CSV data for seamless visualization.
3. Added a new dataset (malfunction_reports.csv) for managing malfunction reports.

---

## Challenges and Solutions

1. **Implementing DDD Principles**:
    - **Challenge**: Structuring the project consistently.
    - **Solution**: Strictly followed DDD guidelines with clear separation of layers.
2. **Error Handling**:
    - **Challenge**: Validating and managing user inputs.
    - **Solution**: Added exceptions (InvalidPostalCodeException, invalid email/phone handling).
3. **Streamlit Integration**:
    - **Challenge**: Handling interactions in a user-friendly manner.
    - **Solution**: Added modular layers for scalability and efficiency.

---

## Project Completion

### Milestones Achieved

- Implemented "Search by Postal Code" and "Report Malfunction" use cases.
- Integrated datasets and ensured proper validation.
- Built an interactive Streamlit interface.

### Pending Tasks

- Enhancing map layers for malfunction visualization.

---

## Lessons Learned

- Clear separation of concerns using DDD principles simplifies maintenance.
- Writing tests first ensures high-quality, reliable code.
- Collaboration and iterative feedback improve overall development.