ChargeHubBerlin Project Documentation

Team Details

- **GitHub Repository**: https://github.com/lukedrichard/berlingeoheatmap_project1
- **Streamlit App URL:** https://berlingeoheatmapproject1-irinkaymxkgindrkfbvtdw.streamlit.app/
- **Group Members**:
  1. Luke Richard - luri1537@bht-berlin.de
  2. Saad Tozibar Rahman - sato7894@bht-berlin.de
  3. Sivasankar Subramanian - sisu9000@bht-berlin.de
  4. Muhammad Abdullah Khan - mukh7058@bht-berlin.de

Introduction to the Project and Use Case

The **ChargeHubBerlin** project aims to provide a way for users to search for EV charging stations in Berlin using postal codes. The project uses **Domain-Driven Design (DDD)** and **Test-Driven Development (TDD)** to ensure the application is well-organized and robust.

Use Case: Search by Postal Code

The core use case enables users to:

- Enter a postal code in Berlin.
- Retrieve a list of charging stations in the specified area.
- Display the results interactively in a web-based interface using Streamlit.

Objective: Validate user input, fetch relevant data, and display it interactively with maps.

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

The project is structured according to **Domain-Driven Design (DDD)** principles, ensuring clear separation of responsibilities. Below is the folder structure with its roles:

1. Charging Folder

The charging folder contains all the core logic and operations related to charging station functionality.

Application Layer

- **Services Folder**: Contains service classes that perform operations like searching for charging stations (station_search_service.py).

Domain Layer

This layer handles the actual functionality and consists of:

- **Entities**: Define core objects like ChargingStation (e.g., postal code, latitude, longitude).
- **Events**: Capture events like StationSearchPerformed for logging and tracking searches.
- **Value Objects**: Contain objects like PostalCode that validate and handle user inputs.

2. Infrastructure Layer

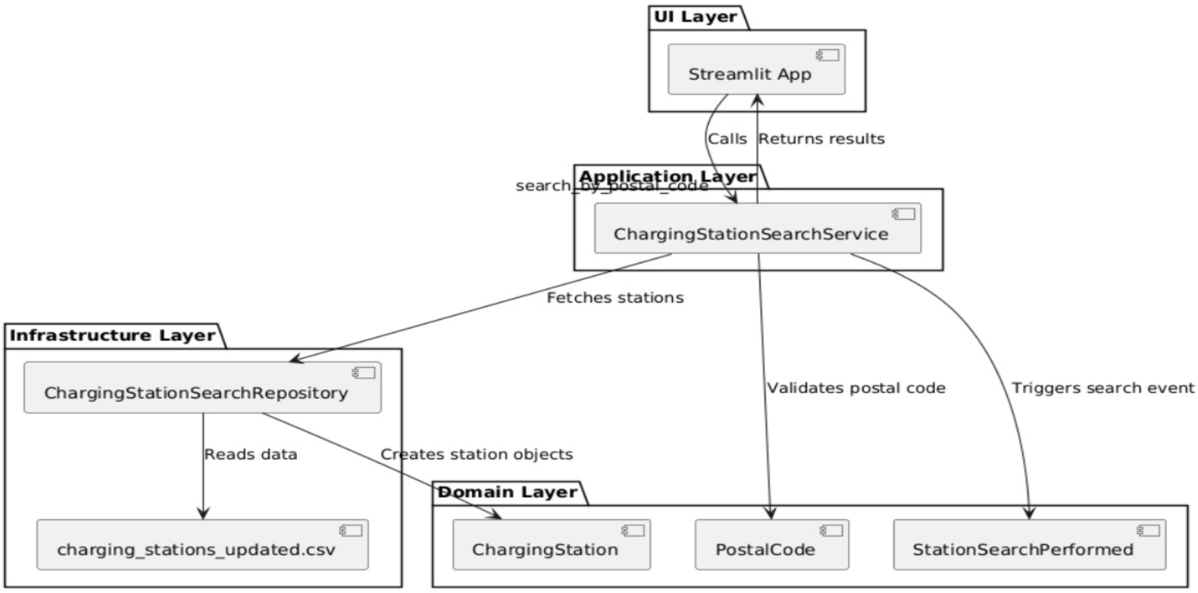
This layer contains the datasets (CSV files) used for fetching data, such as:

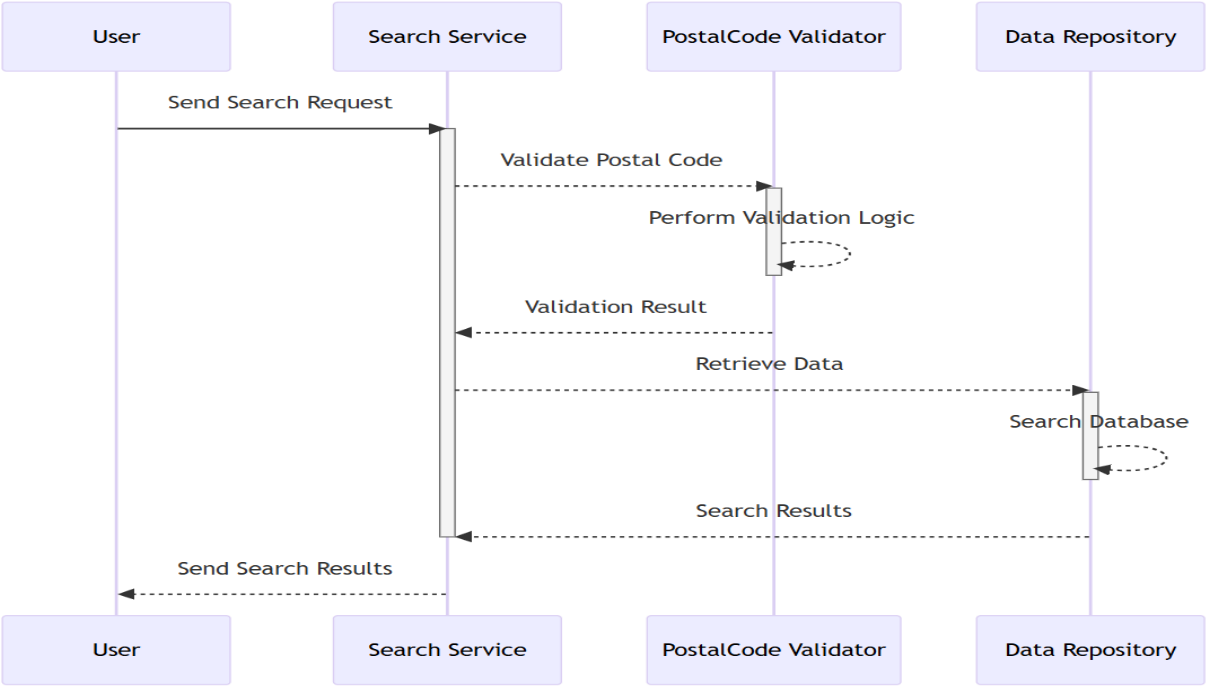
- charging_stations_updated.csv: Information about charging stations.
- plz_einwohner.csv: Postal codes and population data.
- geodata_berlin_plz.csv: Geospatial data for mapping postal codes.

Domain Event Flow

Here's how the system works for the "Search by Postal Code" use case:

1. **User Input**: The user enters a postal code in Streamlit.
2. **Validation**: The input is validated using the PostalCode value object (e.g., only Berlin postal codes are allowed).
3. **Data Fetching**: The repository fetches charging station data matching the postal code.
4. **Event Handling**: A StationSearchPerformed event is triggered to log the search.
5. **Output**: Results are displayed interactively in the Streamlit interface.





TDD Implementation

Development Workflow

1. **Red Phase**: Wrote failing tests for postal code validation, data retrieval, and user input edge cases.
2. **Green Phase**: Implemented functionality in small, iterative steps to pass the tests:
    - Added postal code validation logic.
    - Integrated the repository to fetch data from CSV files.
    - Built services to connect the backend and the user interface.

3. **Refactor Phase**: Optimized code structure and added comments for better readability.

Test Cases

- **Happy Path**: A valid postal code like 10115 retrieves charging stations successfully.
- **Edge Case**: Empty or invalid inputs like 99999 raise an exception.
- **Error Scenarios**: Postal codes outside Berlin (e.g., 20159) raise InvalidPostalCodeException.

Test Coverage

- Achieved approximately 80% test coverage by focusing on all critical parts of the project.

UI and Streamlit Integration

Streamlit is used for the user interface. Users can:

1. Enter a postal code to find nearby charging stations.
2. View a heatmap of charging stations and population distribution in Berlin.

UI Interaction Flow

- **Input**: Text input field for entering postal codes.
- **Validation**: Displays error messages for invalid inputs.
- **Results**: Charging station locations are shown on an interactive map.

Integration of Datasets

Steps Taken

1. Loaded and preprocessed CSV files using Pandas and Geopandas.
2. Cleaned data (e.g., replaced commas in latitude/longitude values).
3. Linked datasets to postal code geometries for visualization on maps.

Challenges and Solutions

1. **Organizing the Project with DDD Principles**
   Challenge: Structuring the project into domain, application, and infrastructure layers was initially difficult.
   Solution: Followed DDD guidelines and separated concerns for better organization.
2. **Basic Error Handling**
   Challenge: Managing invalid user inputs or edge cases during postal code searches.
   Solution: Implemented robust validation logic in the PostalCode value object and used exceptions like InvalidPostalCodeException.

3. **Integrating Streamlit**
   Challenge: Embedding backend services into a user-friendly interface was complex.
   Solution: Used modular service classes like ChargingStationSearchService to ensure seamless backend-frontend integration.

4. **Formatting Data for Mapping in Streamlit**
   Challenge: Preparing data to work with Streamlit's map visualization tools.
   Solution: Used Geopandas to process postal code geometries and ensure data compatibility for maps.

Project Completion

Milestones Achieved

- Implemented "Search by Postal Code" functionality using DDD and TDD.
- Integrated CSV datasets for real-world data.
- Built an interactive Streamlit interface for users.

Pending Tasks

- Adding further UI enhancements for heatmap layers.
- Extending features to support additional use cases.

Lessons Learned

- **Clear Code Organization**: Using DDD principles made the codebase easier to manage.
- **TDD Benefits**: Writing tests first ensured robust and reliable functionality.
- **Collaboration**: Clear division of tasks improved teamwork and efficiency.