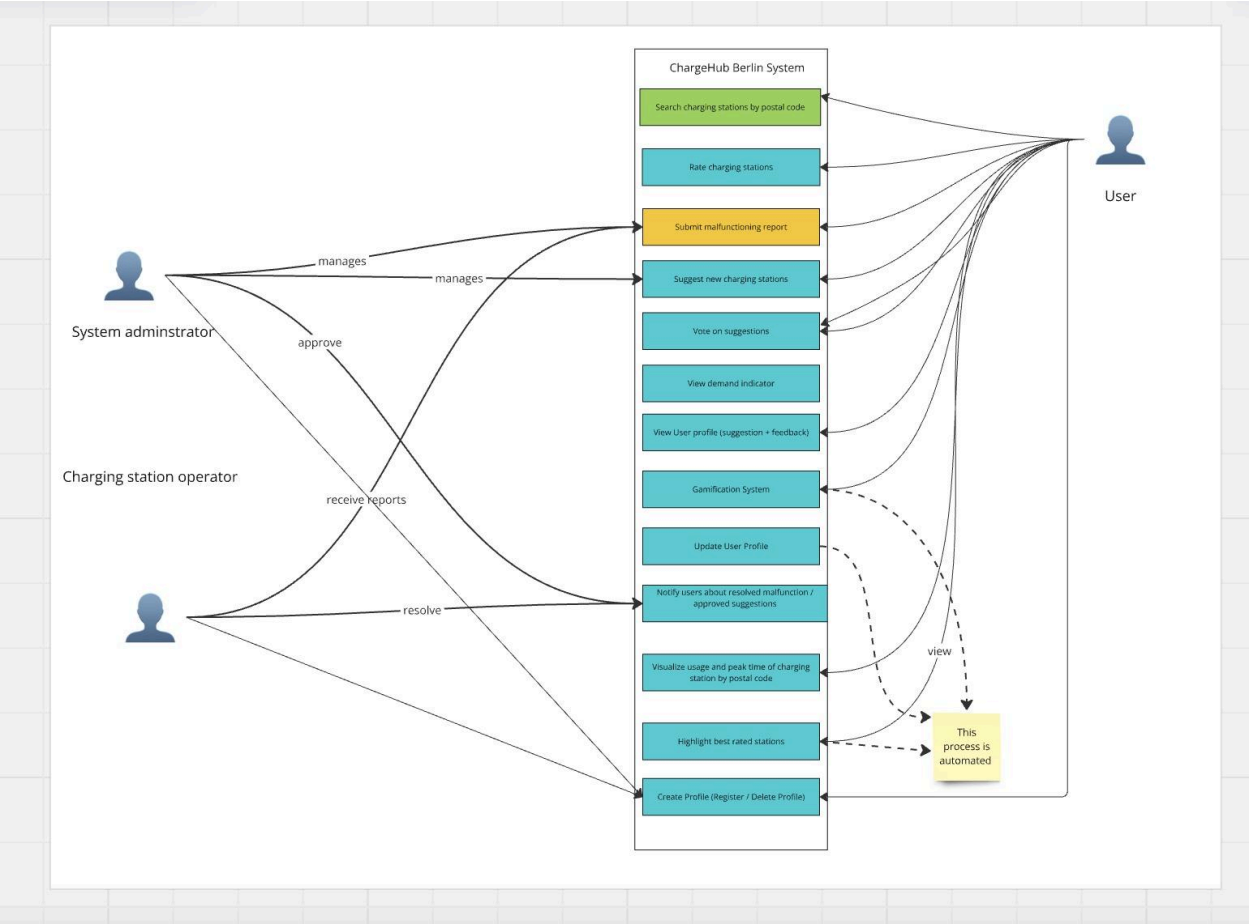
Group 14
Advanced Software Technik
Bou Kamel Nancy, Ibragim Rana Muhammad, Kamal Md Mustafa, Mishra Chiraag

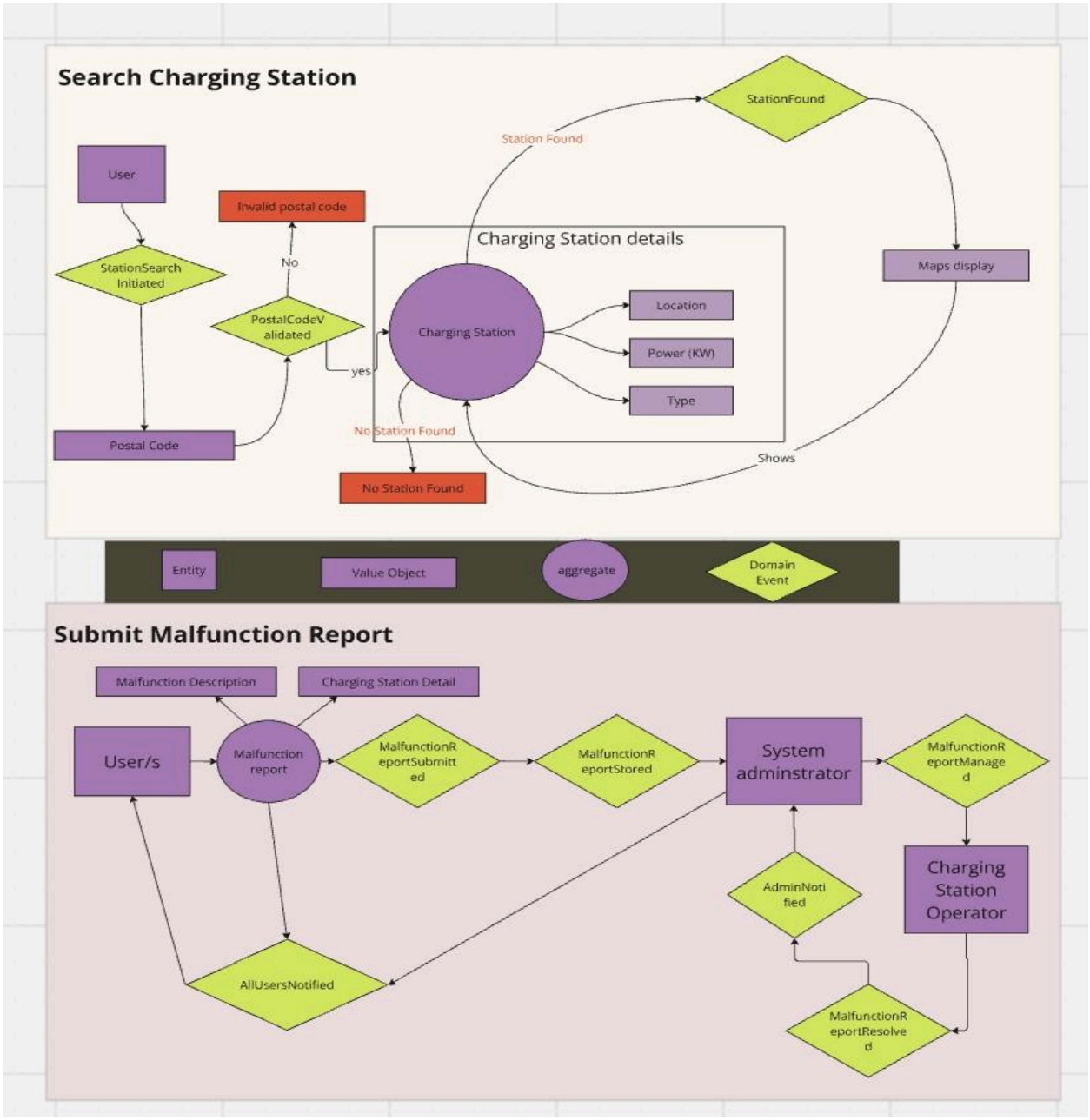
---

- Use case selected: search charging station by postal code
- All UML/user case diagrams/DDD/event storming/ domain event flow/ component integration sequence diagram can be found in the uml_ttd folder
- 2 User cases will be done: search station by postal code, report malfunction

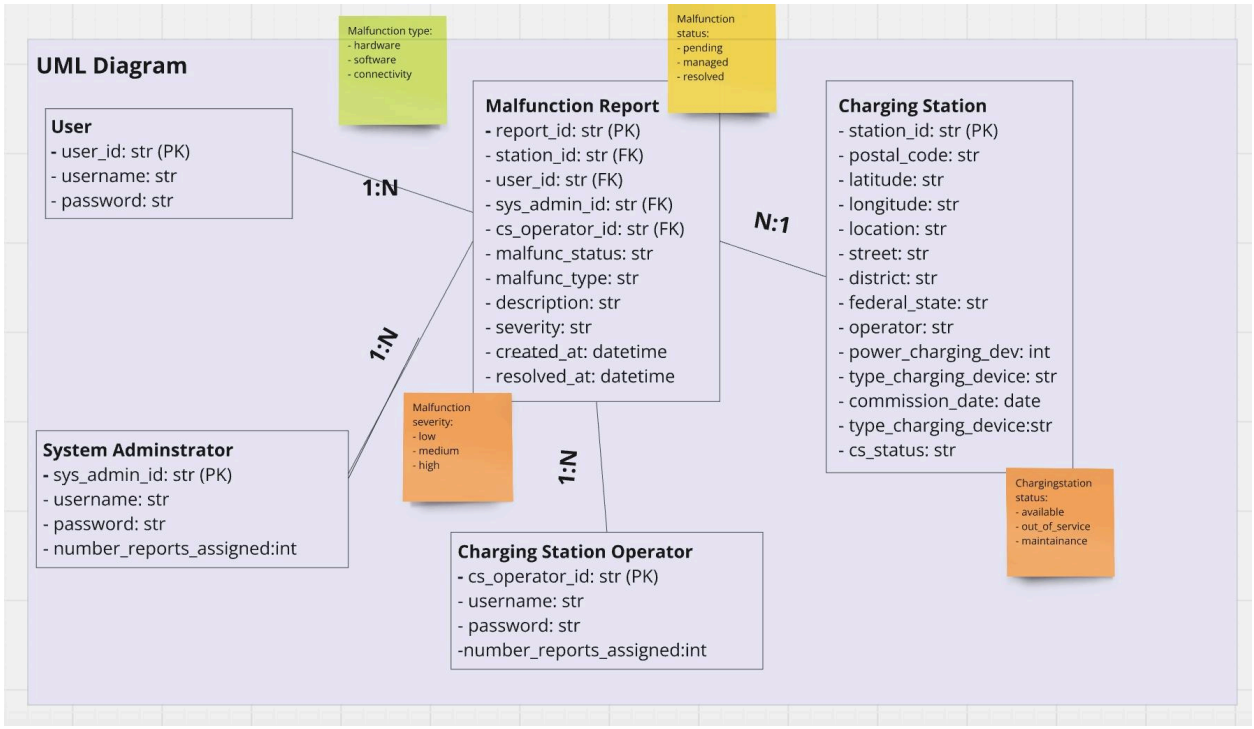
User Case Diagram



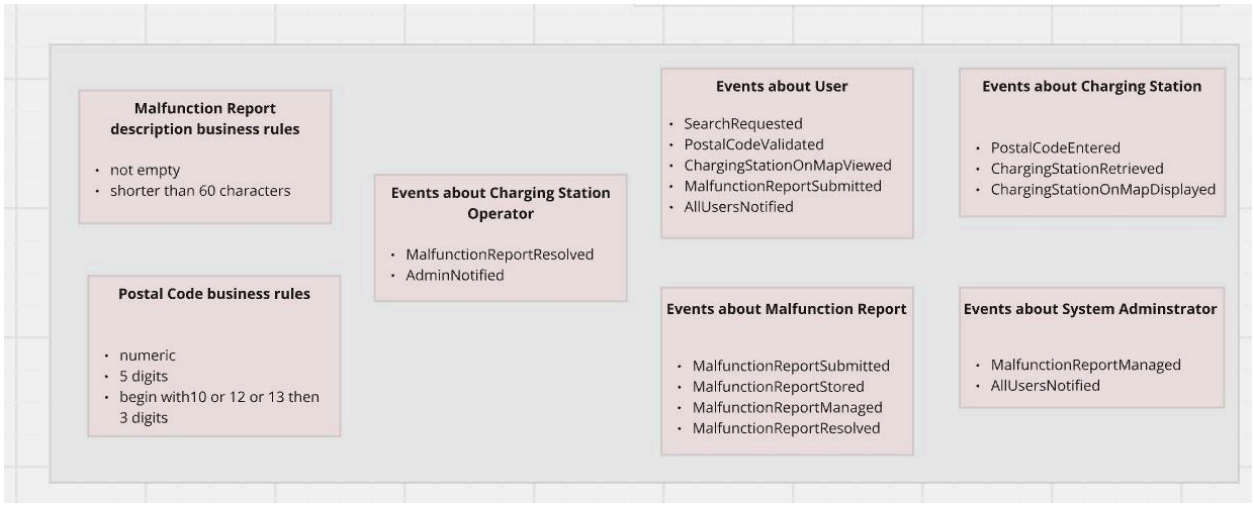
Domain Driven Design 2 Cases:


Search Charging Station
StationFound
Station Found
User
Invalid postal code
Charging Station details
Maps display
StationSearch Initiated
No
PostalCodeV alidated
Charging Station
Location
Power (KW)
Type
yes
Postal Code
No Station Found
Shows
No Station Found
Entity
Value Object
aggregate
Domain Event
Submit Malfunction Report
Malfunction Description
Charging Station Detail
User/s
Malfunction report
MalfunctionR eportSubmitt ed
MalfunctionR eportStored
System adminstrator
MalfunctionR eportManage d
AdminNoti fied
Charging Station Operator
AllUsersNotified
MalfunctionR eportResolve d

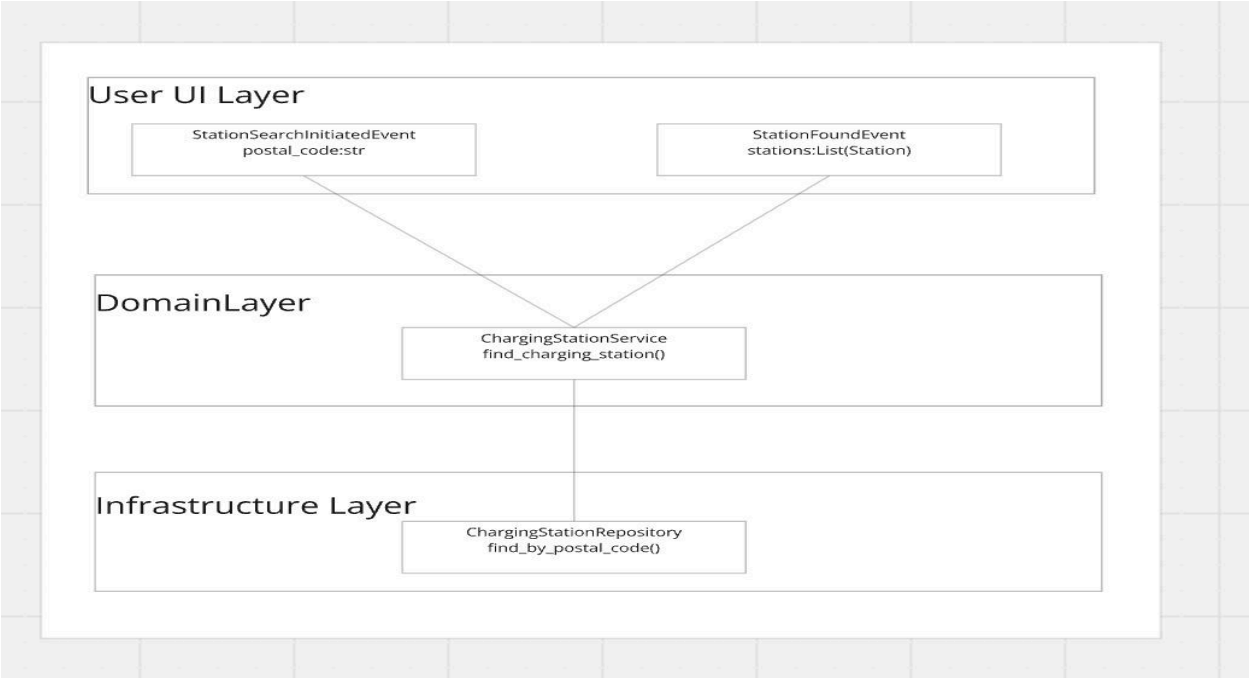
UML Diagram



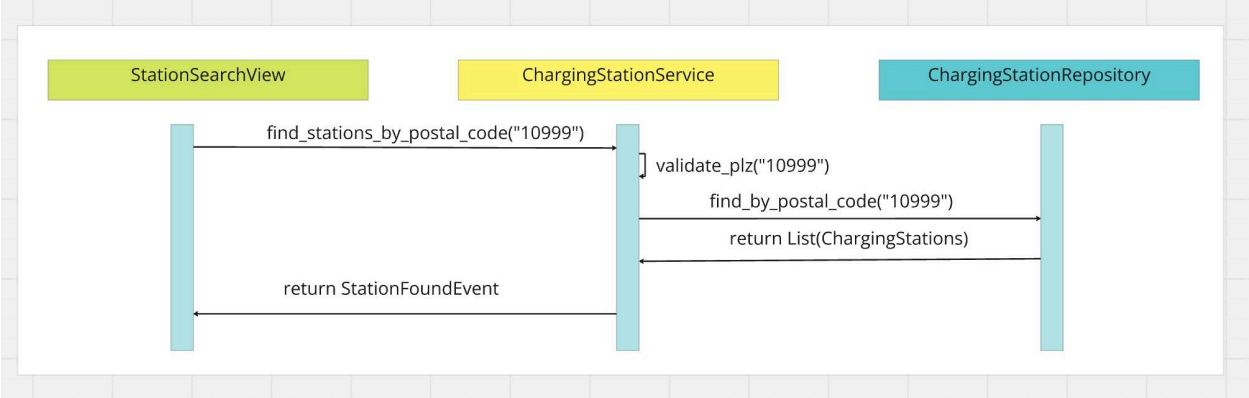
Event Storming:



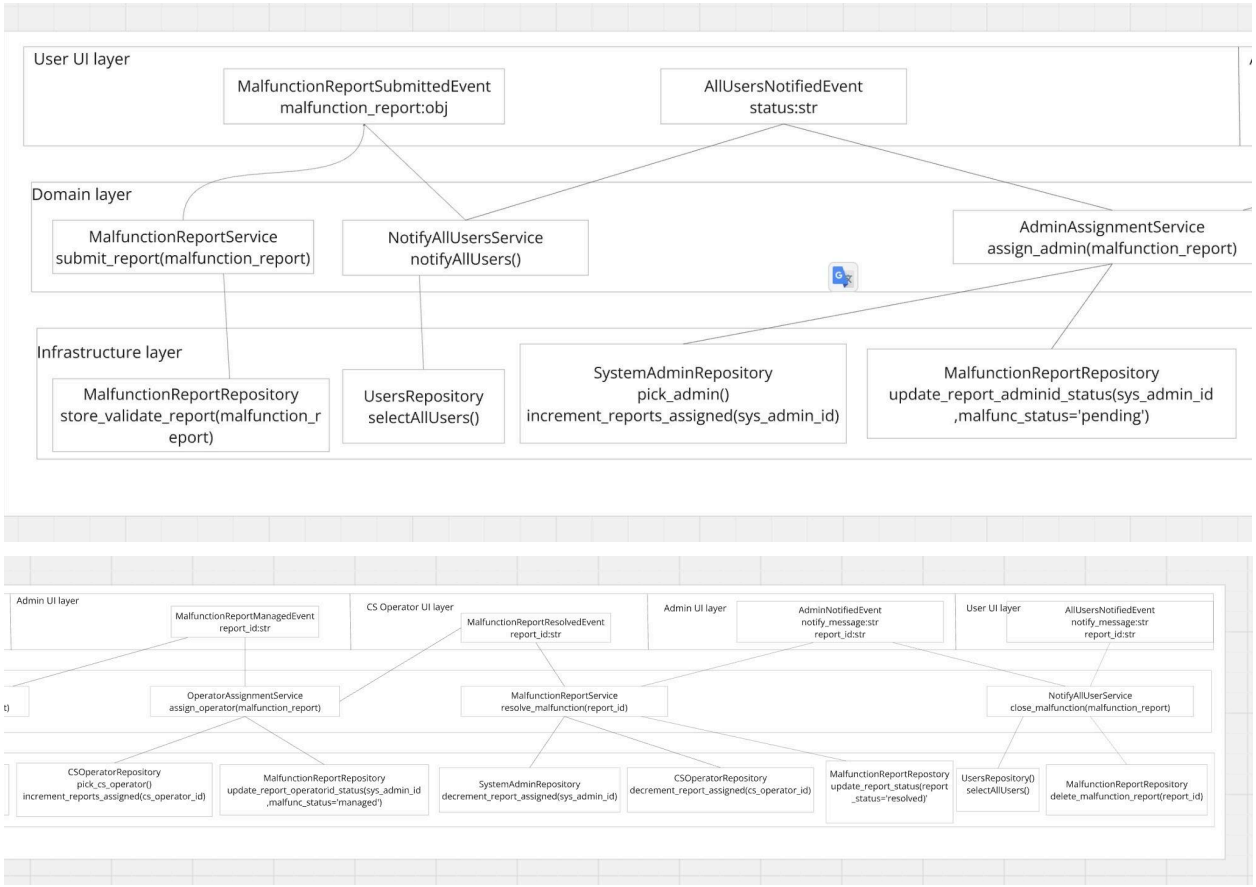
Domain event flow search charging station by postal code



Component Interaction sequence diagram of search station by postal code

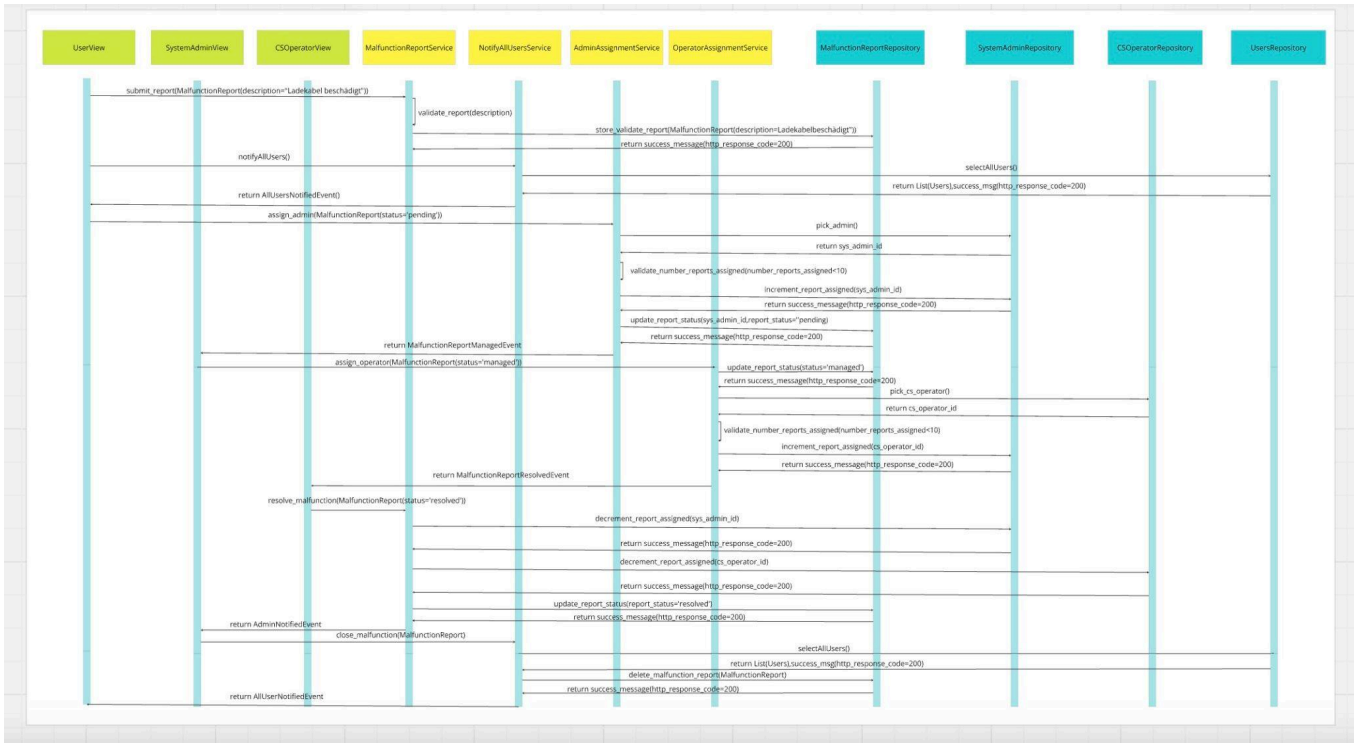


Domain event flow of report malfunction



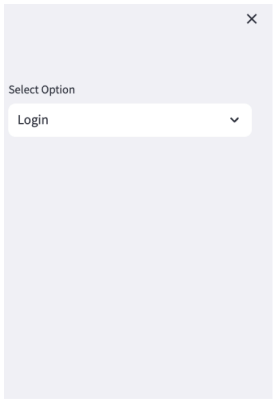
Note: These 2 images are 1 image (done like this so one can see it clear)

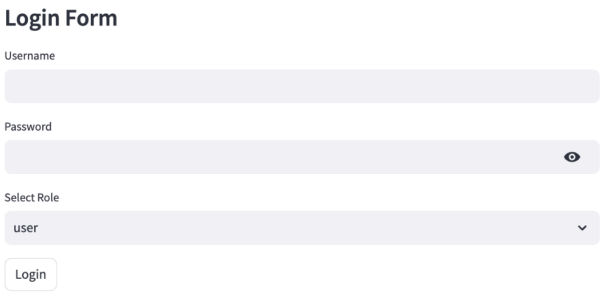
Component interaction sequence diagram of malfunction report



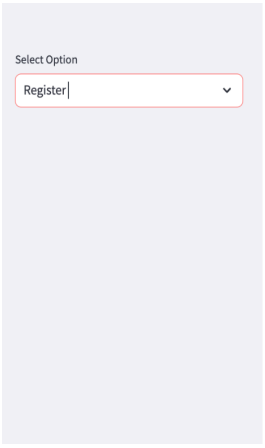
Interfaces Created:

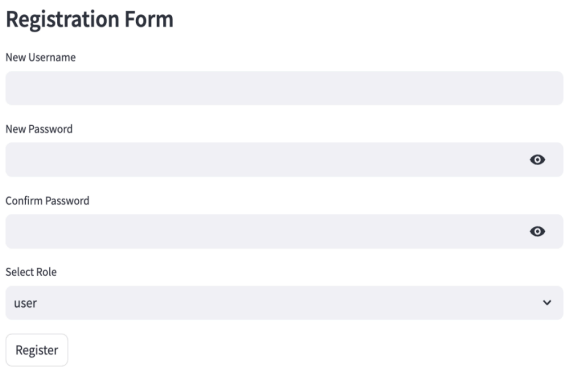
1) Login form
   User, Admin, Charging station operator can login





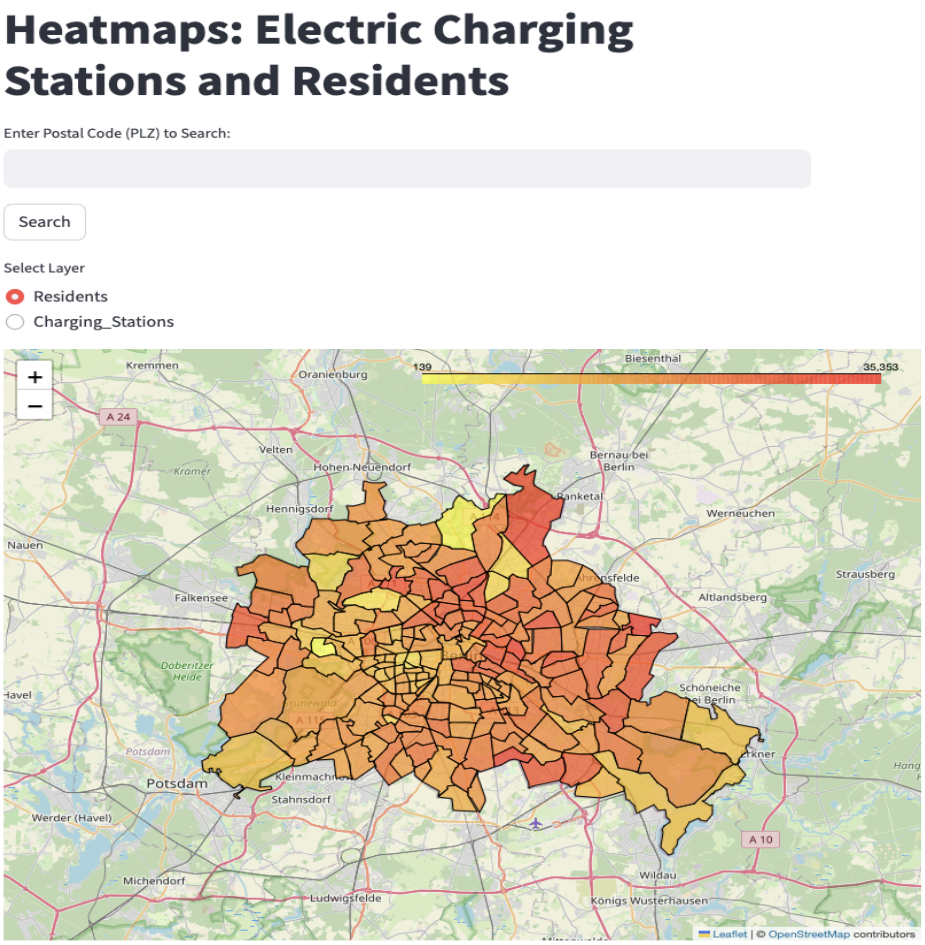
2) Registration Form:
   User, Admin, Charging station operator can register





Username should be unique. The password should be at least 8 characters, has at least 1 number and 1 special character, and has at least 1 capital letter. If the password is not satisfying these conditions an error will appear

3) Search station by postal code interface



4) Upon postal code enter and search button clicked
When a user, admin, or charging station operator enter a code (example: 1559) the pins will be plotted in the map for charging stations in this area. Also, the map is zoomed automatically to the surrounding area. I created a color code for the pin according to the power charging device such that low power station (power<=50) are highlighted by green color, medium power charging station (power <= 150) are highlighted by yellow color, high power charging station (power <= 500) are highlighted by orange color, ultra high power charging station (power>500) are highlighted by red color. Moreover, when the user click on the pin a popup will show up including all the informations of charging station as shown in the picture below.

Heatmaps: Electric Charging Stations and Residents

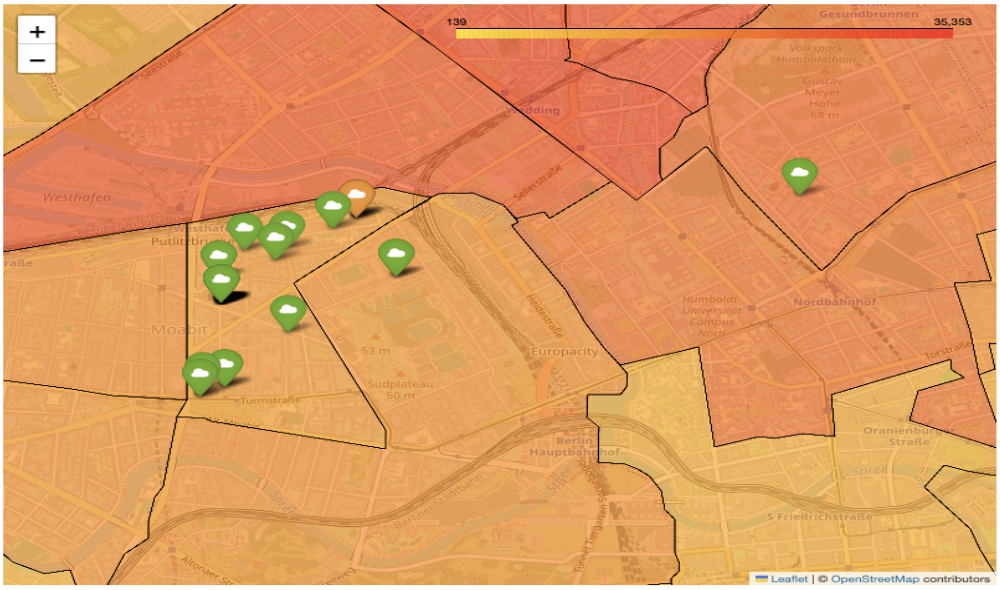
Enter Postal Code (PLZ) to Search:

10559

Search

Select Layer

- (•) Residents
- ( ) Charging_Stations



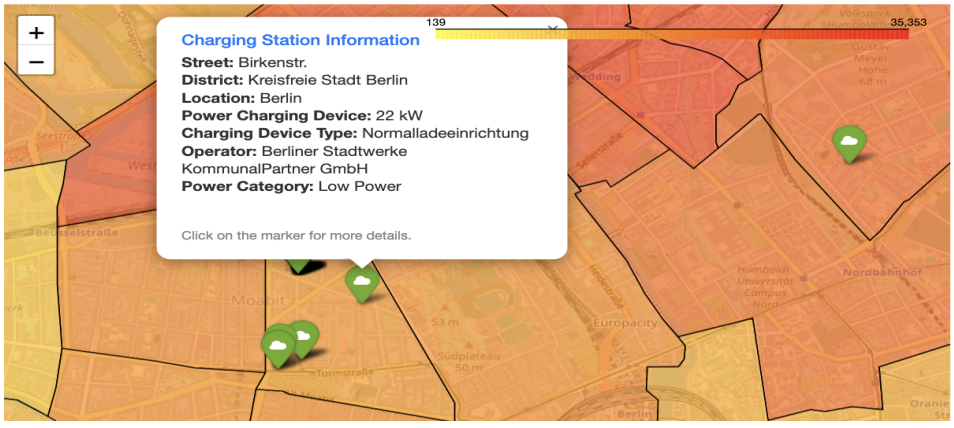
Enter Postal Code (PLZ) to Search:

10559

Search

Select Layer

- (•) Residents
- ( ) Charging_Stations



In case the postal code is invalid(not staring with 10,11, 12, 13 or 14 and greater than 6 digits) an error message will be displayed and the map is zoomed to the default location as shown in picture below

Heatmaps: Electric Charging Stations and Residents

Enter Postal Code (PLZ) to Search:

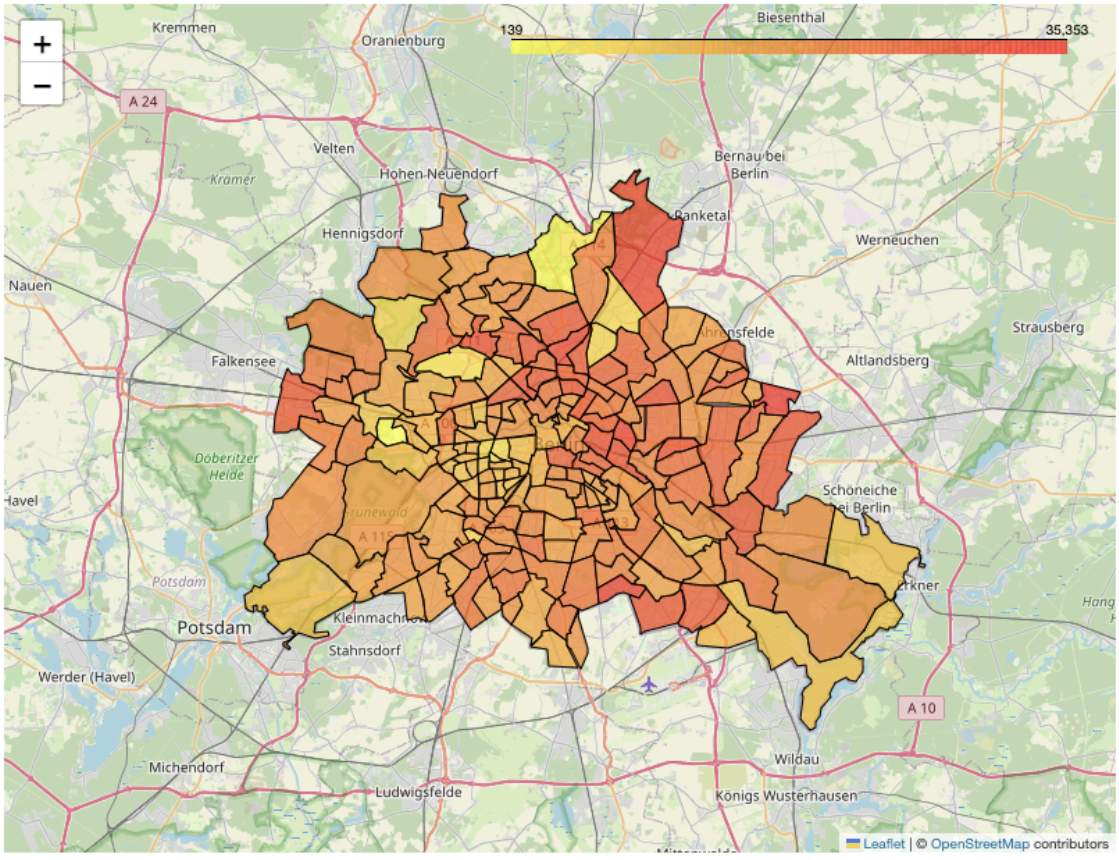
105591

Search

Select Layer

- (•) Residents
- ( ) Charging_Stations

Invalid postal code: 105591



Heatmaps: Electric Charging Stations and Residents

Enter Postal Code (PLZ) to Search:

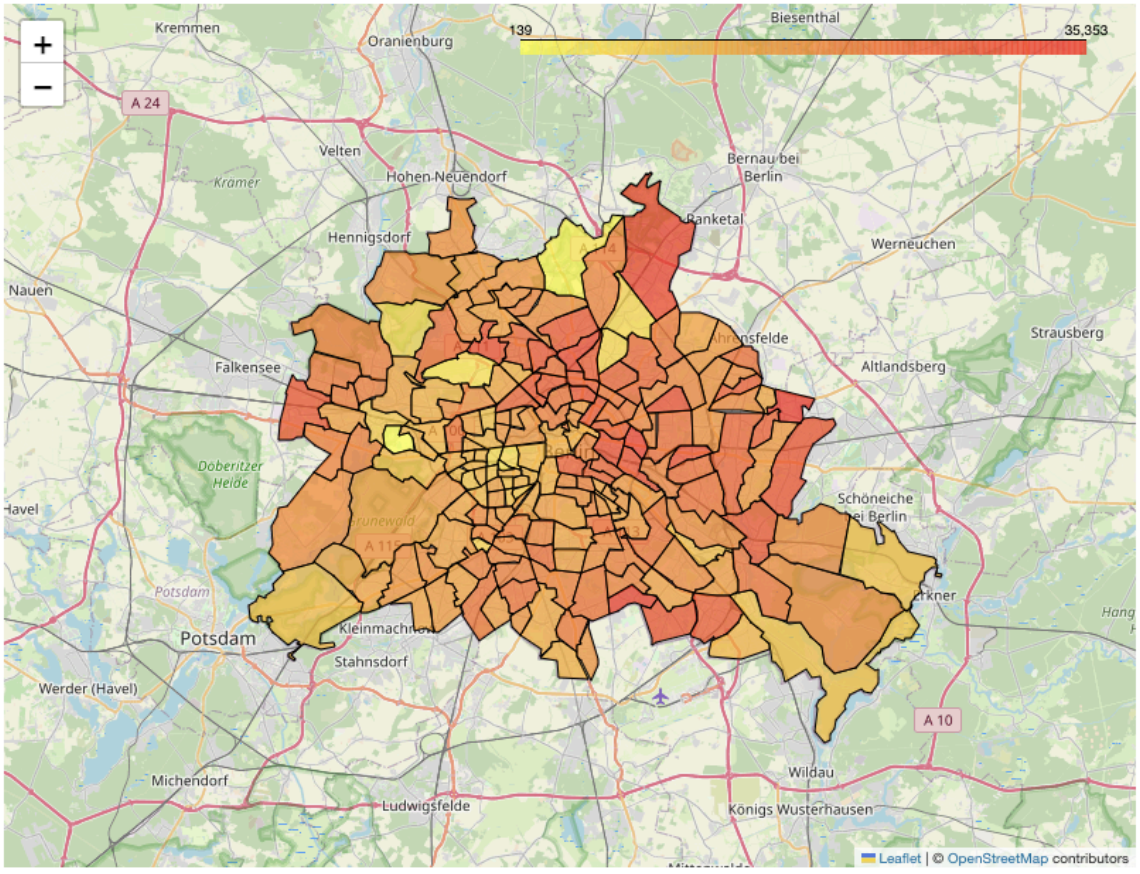
trfd

Search

Select Layer

- (•) Residents
- ( ) Charging_Stations

PostalCode must contain only numeric characters, got: trfd



Infrastructure:

- Sqlalchemy database is implemented with charging station, user, admin, charging station operator database tables.

Test Cases:

1) Test User:

1. test_create_user

- Purpose: This test case is designed to verify the functionality of creating a new user in the database.
- Setup: A User object with a unique username and password is created.
- Expected Outcome:
    - The newly created user should have the correct username and password as provided.
    - The user should be successfully stored in the database.

---

2. test_get_user_by_id

- Purpose: This test case checks whether a user can be retrieved from the database using their user_id.
- Setup: A user is first created using the create_user_in_db helper function.
- Expected Outcome:
    - The user is fetched by their user_id.
    - The retrieved user should match the one created earlier in terms of username and password.

---

3. test_get_user_by_username

- Purpose: This test case verifies the functionality of fetching a user from the database using their username.
- Setup: A user is created in the database.
- Expected Outcome:
    - The user is fetched by their username.
    - The retrieved user should match the one created earlier, both in terms of username and password.

---

4. test_create_user_with_duplicate_username

- Purpose: This test case is designed to check whether the database enforces the uniqueness constraint on the username field when trying to create a user with an already existing username.
- Setup: A user is created with a unique username. Then, an attempt is made to create another user with the same username.
- Expected Outcome:
    - The database should raise an IntegrityError due to the violation of the UNIQUE constraint on the username field.
    - The test checks if this error is properly caught and handled.

Failures

- test_create_user_with_duplicate_username Failure:

- Issue: The test failed due to an IntegrityError being raised because the username must be unique.
- Cause: The test expected a ValueError, but the error raised was an IntegrityError, which is the actual error type raised by SQLAlchemy due to a constraint violation on the username column.

```
    def do_execute(self, cursor, statement, parameters, context=None):
        cursor.execute(statement, parameters)
        sqlalchemy.exc.IntegrityError: (sqlite3.IntegrityError) UNIQUE constrain
 failed: users.username
        [SQL: INSERT INTO users (username, password) VALUES (?, ?)]
        [parameters: ('duplicate_user', 'AnotherPassword2@')]
        (Background on this error at: https://sqlalche.me/e/20/gkpj)

Applications/anaconda3/lib/python3.12/site-packages/sqlalchemy/engine/default.p
:924: IntegrityError
=========================== short test summary info ============================
AILED test_user.py::test_create_user_with_duplicate_username - sqlalchemy.exc.I
tegrityError: (sqlite3.IntegrityError) UNIQUE constraint f...
======================== 1 failed, 3 passed in 0.79s ===========================
base) nancyboukamel@Nancys-Air test %
```

2) Test Admin

1. test_create_admin

- Purpose: Test the creation of a new admin in the database.
- Description:
  - Inserts a new admin with specified username, password, and number_reports_assigned.
  - Verifies that the admin is successfully created and that the attributes match the input values.
- Expected Outcome:
  - The admin's username, password, and number_reports_assigned fields match the values provided during creation.

---

2. test_get_admin_by_id

- Purpose: Test fetching an admin from the database using their sys_admin_id.
- Description:
  - Creates a new admin in the database.
  - Fetches the admin using their unique sys_admin_id.
  - Verifies that the fetched admin's attributes match the input values.
- Expected Outcome:
  - The admin is successfully retrieved by their sys_admin_id.
  - The username, password, and number_reports_assigned values are correct.

---

3. test_get_admin_by_username

- Purpose: Test fetching an admin from the database using their username.
- Description:

    - Creates a new admin in the database.
    - Fetches the admin using their unique username.
    - Verifies that the fetched admin's attributes match the input values.
- Expected Outcome:
    - The admin is successfully retrieved by their username.
    - The username, password, and number_reports_assigned values are correct.

---

4. test_create_admin_with_duplicate_username

- Purpose: Ensure that the username field in the Admin table is unique.
- Description:
    - Creates a new admin with a specific username.
    - Attempts to create another admin with the same username.
    - Expects the database to raise an IntegrityError due to the violation of the UNIQUE constraint on the username column.
- Expected Outcome:
    - The database enforces the UNIQUE constraint on the username field.
    - An IntegrityError is raised when attempting to insert a duplicate username.

Testing Strategy

- Fixture: The db_session fixture sets up an in-memory SQLite database for isolated testing. This ensures that changes made during tests do not persist beyond the test scope.
- Assertions:
    - Attribute values (e.g., username, password, number_reports_assigned) are validated against input data.
    - Exceptions like IntegrityError are caught and verified when testing constraints.
- Error Handling: Tests are designed to gracefully handle and assert expected errors, such as attempting to insert duplicate data.

```
    def do_execute(self, cursor, statement, parameters, context=None):
>       cursor.execute(statement, parameters)
E       sqlalchemy.exc.IntegrityError: (sqlite3.IntegrityError) UNIQUE constrain
t failed: admins.username
E       [SQL: INSERT INTO admins (username, password, number_reports_assigned) V
ALUES (?, ?, ?)]
E       [parameters: ('duplicate_admin', 'AnotherAdminPassword@', 2)]
E       (Background on this error at: https://sqlalche.me/e/20/gkpj)

/Applications/anaconda3/lib/python3.12/site-packages/sqlalchemy/engine/default.p
y:924: IntegrityError
=========================== short test summary info ============================
FAILED test_admin.py::test_create_admin_with_duplicate_username - sqlalchemy.exc
.IntegrityError: (sqlite3.IntegrityError) UNIQUE constraint f...
========================= 1 failed, 3 passed in 0.69s ==========================
(base) nancyboukamel@Nancys-Air test %
```

3) Test CSOperator

1. test_create_cs_operator

- Purpose:
  Verifies that a new CSOperator can be created successfully in the database.
- Steps:
  1. Create a new CSOperator instance with a unique username, password, and number_reports_assigned.
  2. Insert it into the database.
  3. Validate that the CSOperator's details match the input values.
- Expected Outcome:
  The CSOperator is successfully created, and all attributes (username, password, number_reports_assigned) are stored as expected.

---

2. test_get_cs_operator_by_id

- Purpose:
  Tests retrieval of a CSOperator by its unique ID (cs_operator_id).
- Steps:
  1. Create and insert a CSOperator into the database.
  2. Retrieve the CSOperator using its cs_operator_id.
  3. Compare the retrieved CSOperator’s attributes to the original input values.
- Expected Outcome:
  The correct CSOperator is retrieved with matching attributes.

---

3. test_get_cs_operator_by_username

- Purpose:
  Tests retrieval of a CSOperator by its unique username.
- Steps:
  1. Create and insert a CSOperator with a specific username into the database.
  2. Retrieve the CSOperator using the username.
  3. Validate that the retrieved CSOperator matches the original data.
- Expected Outcome:
  The CSOperator with the specified username is successfully retrieved, and its attributes are correct.

---

4. test_create_cs_operator_with_duplicate_username

- Purpose:
  Verifies that creating a CSOperator with a duplicate username violates the database's UNIQUE constraint.
- Steps:
  1. Create and insert a CSOperator with a unique username.
  2. Attempt to create a second CSOperator with the same username.

3. Capture the expected exception (IntegrityError).

- Expected Outcome:
  The test raises an IntegrityError, ensuring that duplicate usernames are not allowed.

```
============================== warnings summary ===============================
../src/sqldatabase/hub/csoperator.py:6
  /Users/nancyboukamel/Desktop/BHT/software_engineering/berlingeoheatmap_project
1/src/sqldatabase/hub/csoperator.py:6: SAWarning: On class 'CSOperator', Column
object 'password' named directly multiple times, only one will be used: number_r
eports_assigned, password. Consider using orm.synonym instead
    class CSOperator(Base):

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
============================ short test summary info ============================
FAILED test_csoperator.py::test_create_cs_operator_with_duplicate_username - sql
alchemy.exc.IntegrityError: (sqlite3.IntegrityError) UNIQUE constraint f...
==================== 1 failed, 3 passed, 1 warning in 0.75s ====================
```

4) Test Password

1. test_valid_password

- Purpose:
  Verifies that a valid password satisfying all requirements can be successfully created.
- Steps:
  1. Pass a valid password (Valid1Password!) to the Password class.
  2. Assert that the password's value is stored correctly.
- Expected Outcome:
  The password is accepted without errors.

---

2. test_password_too_short

- Purpose:
  Ensures that passwords shorter than 8 characters are rejected.
- Steps:
  1. Pass a password shorter than 8 characters (e.g., Short1!).
  2. Capture the expected exception with the message:
     "Password must be at least 8 characters long."
- Expected Outcome:
  A TypeError is raised, and the short password is not accepted.

---

3. test_password_missing_uppercase

- Purpose:
  Ensures passwords without at least one uppercase letter are invalid.

- Steps:
    1. Pass a password missing uppercase letters (e.g., nouppercase1!).
    2. Capture the expected exception with the message:
       "Password must contain at least one uppercase letter (A-Z)."
- Expected Outcome:
  A TypeError is raised, and the password is rejected.

---

4. test_password_missing_lowercase

- Purpose:
  Ensures passwords without at least one lowercase letter are invalid.
- Steps:
    1. Pass a password missing lowercase letters (e.g., NOLOWERCASE1!).
    2. Capture the expected exception with the message:
       "Password must contain at least one lowercase letter (a-z)."
- Expected Outcome:
  A TypeError is raised, and the password is rejected.

---

5. test_password_missing_digit

- Purpose:
  Ensures passwords without at least one numeric digit are invalid.
- Steps:
    1. Pass a password missing a digit (e.g., NoDigitHere!).
    2. Capture the expected exception with the message:
       "Password must contain at least one numeric digit (0-9)."
- Expected Outcome:
  A TypeError is raised, and the password is rejected.

---

6. test_password_missing_special_character

- Purpose:
  Ensures passwords without at least one special character are invalid.
- Steps:
    1. Pass a password missing special characters (e.g., NoSpecialChar1).
    2. Capture the expected exception with the message:
       "Password must contain at least one special character (e.g., !@#$%^&*)."
- Expected Outcome:
  A TypeError is raised, and the password is rejected.

---

7. test_password_with_special_characters

- Purpose:
  Ensures that passwords meeting all requirements (including special characters) are valid.
- Steps:
  1. Pass a valid password (Valid1Password@) to the Password class.
  2. Assert that the password's value is stored correctly.
- Expected Outcome:
  The password is accepted without errors.

---

8. test_password_not_string

- Purpose:
  Ensures non-string password inputs are rejected.
- Steps:
  1. Pass a non-string value (e.g., 12345678, an integer) to the Password class.
  2. Capture the expected exception with the message:
     "Password value must be a string, got int."
- Expected Outcome:
  A ValueError is raised, and the password is rejected.

---

9. test_password_empty

- Purpose:
  Ensures that an empty password is rejected.
- Steps:
  1. Pass an empty string ("") to the Password class.
  2. Capture the expected exception with the message:
     "Password must be at least 8 characters long."
- Expected Outcome:
  A TypeError is raised, and the password is rejected.

Key Observations:

1. Validation Rules:
   - The password must meet all of the following criteria:
     - Minimum of 8 characters.
     - At least one uppercase letter (A-Z).
     - At least one lowercase letter (a-z).
     - At least one digit (0-9).
     - At least one special character (e.g., !@#$%^&*).
   - Non-string values are explicitly disallowed.

2. Error Messages:
   Meaningful and specific error messages are provided for different validation failures.

3. Test Coverage:
   The test suite robustly handles valid inputs, missing criteria, and invalid types.

```
========================== short test summary info ==========================
FAILED test_password.py::test_password_too_short - ValueError: Password must be
at least 8 characters long.
FAILED test_password.py::test_password_missing_uppercase - ValueError: Password
must contain at least one uppercase letter (A-Z).
FAILED test_password.py::test_password_missing_lowercase - ValueError: Password
must contain at least one lowercase letter (a-z).
FAILED test_password.py::test_password_missing_digit - ValueError: Password must
 contain at least one numeric digit (0-9).
FAILED test_password.py::test_password_missing_special_character - ValueError: P
assword must contain at least one special character (e.g., !@#...
FAILED test_password.py::test_password_not_string - TypeError: Password value mu
st be a string, got int
FAILED test_password.py::test_password_empty - ValueError: Password must be at l
east 8 characters long.
========================= 7 failed, 2 passed in 0.10s =========================
```

5) Test Charging station

Happy Path Tests

1. test_valid_charging_station_creation

- Purpose: Verifies successful creation of a charging station with valid input.
- Steps:
  1. Create a valid ChargingStation object.
  2. Assert that all attributes are correctly assigned.
- Expected Outcome: The object is created without any errors.

Edge Case Tests

2. test_min_max_power_charging_dev

- Purpose: Validates the boundary values for power_charging_dev.
- Steps:
  1. Create a station with minimum power (1).
  2. Create a station with maximum power (1200).
  3. Assert that both are accepted without errors.
- Expected Outcome: Stations with power_charging_dev in the valid range are created successfully.

3. test_edge_case_power_charging_device

- Purpose: Ensures edge cases for power_charging_dev are handled.
- Steps:
    1. Create a station with the lowest valid power value (1).
    2. Create a station with the highest valid power value (1200).
    3. Assert that the values are correctly assigned.
- Expected Outcome: No errors for valid edge values.

Error Scenario Tests

4. test_invalid_station_id

- Purpose: Ensures station_id must be a non-empty string.
- Steps:
    1. Create a station with an empty station_id.
    2. Assert that a ValueError is raised with the message:
       "Station ID must be a non-empty string."
- Expected Outcome: A ValueError is raised for invalid station_id.

5. test_invalid_power_charging_dev

- Purpose: Ensures power_charging_dev must be a positive integer.
- Steps:
    1. Create stations with 0 or negative power values.
    2. Assert that a ValueError is raised with the message:
       "Power must be a positive integer."
- Expected Outcome: A ValueError is raised for invalid power values.

6. test_invalid_status

- Purpose: Ensures cs_status must have a valid value.
- Steps:
    1. Create a station with an invalid status value (e.g., "Unknown Status").
    2. Assert that a ValueError is raised with the message:
       "Invalid status. Must be 'Active', 'Out Of Service', or 'Under Maintenance'."
- Expected Outcome: A ValueError is raised for invalid statuses.

7. test_empty_fields

- Purpose: Ensures that required fields cannot be empty.
- Steps:
    1. Create a station with empty strings for all required fields.
    2. Assert that a ValueError is raised.
- Expected Outcome: A ValueError is raised for empty required fields.

8. test_commission_date_format

- Purpose: Ensures the commission_date must follow a valid format.
- Steps:

1. Create a station with a valid date format ("11.10.2020").
2. Create a station with an invalid date format ("2020-11-10").
3. Assert that a ValueError is raised for invalid formats.

- Expected Outcome: Valid formats are accepted, and invalid formats raise a ValueError.

Domain Rule Tests

9. test_domain_rule_valid_status

- Purpose: Ensures that only valid statuses are allowed.
- Steps:
  1. Create stations with valid statuses ("Active", "Out Of Service", "Under Maintenance").
  2. Assert that the statuses are correctly assigned.
- Expected Outcome: Valid statuses are assigned without errors.

Key Observations

1. Validation Rules: The tests ensure compliance with:
   - Required fields being non-empty.
   - Correct data types and ranges for fields like power_charging_dev.
   - Adherence to domain-specific rules (e.g., valid statuses).
2. Error Handling: Meaningful error messages are raised for invalid inputs.

```
========================== short test summary info ===========================
FAILED test_charging_station.py::test_invalid_station_id - Failed: DID NOT RAISE
 <class 'ValueError'>
FAILED test_charging_station.py::test_invalid_power_charging_dev - Failed: DID N
OT RAISE <class 'ValueError'>
FAILED test_charging_station.py::test_invalid_status - Failed: DID NOT RAISE <cl
ass 'ValueError'>
FAILED test_charging_station.py::test_empty_fields - Failed: DID NOT RAISE <clas
s 'ValueError'>
FAILED test_charging_station.py::test_commission_date_format - Failed: DID NOT R
AISE <class 'ValueError'>
========================= 5 failed, 4 passed in 0.09s ========================
```

6) Test charging station Database

1. test_create_charging_station

- Purpose:
  - To verify that a ChargingStation record can be successfully created and persisted in the database.
- Scenario:
  - A valid ChargingStation object is provided, and the test asserts that all fields are stored correctly in the database.
- Key Assertions:

    - Each attribute of the created ChargingStation matches the input data.

---

2. test_get_charging_station_by_station_id

- Purpose:
    - To ensure a ChargingStation can be fetched from the database using its station_id.
- Scenario:
    - A station is added to the database, and the test retrieves it by its unique station_id.
- Key Assertions:
    - All retrieved fields match the data of the created station.
    - The station is not None.

---

3. test_get_charging_station_by_postal_code

- Purpose:
    - To verify that a ChargingStation can be fetched using its postal_code.
- Scenario:
    - A station is added to the database, and the test retrieves it by postal_code.
- Key Assertions:
    - The retrieved station is not None.
    - All fields match the data used during creation.

---

4. test_add_invalid_charging_station

- Purpose:
    - To validate that invalid ChargingStation objects are not accepted by the database.
- Scenarios:
    - Missing Required Field: Adding a station with a None station_id (violates primary key constraint).
    - Duplicate station_id: Attempting to add a station with a station_id that already exists in the database.
    - Invalid Data Type: Providing a non-integer value for power_charging_dev.
- Key Assertions:
    - IntegrityError or DataError is raised in each scenario.

---

5. test_get_charging_station_by_postal_code_not_found

- Purpose:
    - To confirm that querying a non-existent postal_code returns None.
- Scenario:
    - The test attempts to fetch a station by a postal_code not present in the database.
- Key Assertions:
    - The result is None.

Known Issues

- Test Failures:
  - Some tests fail due to PendingRollbackError. This error occurs when a session is in an invalid state after an exception is raised, and the session is not properly rolled back.
- Potential Fix:
  - Add explicit session.rollback() in the db_session fixture or in the test cases after an exception is caught.

```
========================== short test summary info ===========================
FAILED test_chargingstationdb.py::test_add_invalid_charging_station - sqlalchemy
.exc.PendingRollbackError: This Session's transaction has been ro...
FAILED test_chargingstationdb.py::test_get_charging_station_by_postal_code_not_f
ound - sqlalchemy.exc.PendingRollbackError: This Session's transaction has been
ro...
==================== 2 failed, 3 passed, 2 warnings in 0.54s ====================
```

7) Test Postal code

1. test_valid_postal_code

- Purpose:
  - Validate that a properly formatted postal code can be created and is marked as valid.
- Scenario:
  - A valid postal code string (e.g., "10115") is passed.
- Key Assertions:
  - The PostalCode object is created successfully.
  - The is_valid() method returns True.

---

2. test_postal_code_invalid_format

- Purpose:
  - Ensure that postal codes with invalid formats (e.g., incorrect length or prefix) are rejected.
- Scenarios:
  - A postal code with an invalid prefix ("15123") is passed.
  - A postal code with too few digits ("1000") is passed.
  - A postal code with too many digits ("101151") is passed.
- Key Assertions:
  - A ValueError is raised with an appropriate error message.

---

3. test_postal_code_non_numeric

- Purpose:
  - Ensure that postal codes containing non-numeric characters are rejected.

- Scenarios:
    - A postal code with an alphanumeric character ("10A15") is passed.
    - A completely non-numeric string ("ABCDE") is passed.
- Key Assertions:
    - A ValueError is raised with an appropriate error message indicating the issue.

---

4. test_postal_code_not_a_string

- Purpose:
    - Ensure that non-string inputs (e.g., integers, lists) are rejected when creating a PostalCode object.
- Scenarios:
    - An integer (10115) is passed instead of a string.
    - A list (e.g., ["10115"]) is passed instead of a string.
- Key Assertions:
    - A TypeError is raised with an appropriate error message specifying the expected input type.

---

5. test_edge_case_postal_codes

- Purpose:
    - Validate that edge case postal codes on the boundary of the valid range are accepted.
- Scenarios:
    - "10000" (minimum valid postal code) and "14999" (maximum valid postal code) are passed.
- Key Assertions:
    - Both postal codes are marked as valid.
    - The value attribute matches the input.

---

6. test_postal_code_minimum_digits

- Purpose:
    - Confirm that the exact minimum valid postal code ("10000") is accepted.
- Scenario:
    - "10000" is passed.
- Key Assertions:
    - The postal code is valid.
    - The value attribute matches the input.

---

7. test_postal_code_maximum_digits

- Purpose:
    - Confirm that the exact maximum valid postal code ("14999") is accepted.
- Scenario:
    - "14999" is passed.

- Key Assertions:
    - The postal code is valid.
    - The value attribute matches the input.

---

8. test_postal_code_case_insensitive_pattern

- Purpose:
    - Ensure that the regular expression pattern used for validation does not fail for valid postal codes.
- Scenario:
    - A valid postal code ("10115") is passed.
- Key Assertions:
    - The postal code is valid.

---

9. test_postal_code_with_whitespace

- Purpose:
    - Validate that postal codes with leading, trailing, or internal whitespace are rejected.
- Scenarios:
    - A postal code with trailing whitespace ("10115 ") is passed.
    - A postal code with leading whitespace (" 10115") is passed.
- Key Assertions:
    - A ValueError is raised for each scenario.

---

10. test_empty_postal_code

- Purpose:
    - Ensure that an empty string cannot be used as a postal code.
- Scenario:
    - An empty string ("") is passed.
- Key Assertions:
    - A ValueError is raised indicating that the postal code is invalid.

---

11. test_international_postal_code

- Purpose:
    - Ensure that non-German postal codes (e.g., U.S. ZIP codes) are rejected.
- Scenario:
    - A U.S. ZIP code ("90210") is passed.
- Key Assertions:
    - A ValueError is raised indicating that the postal code is invalid.

Known Issues

- Regex Matching Errors:
    - Some test cases fail because the regex pattern does not handle specific invalid cases correctly (e.g., whitespace or empty strings).
- Improper Exception Handling:
    - Inconsistent use of TypeError and ValueError. Align exception types for clarity.

```
FAILED test_postal_code.py::test_postal_code_invalid_format - ValueError: Invali
d postal code: 15123
FAILED test_postal_code.py::test_postal_code_non_numeric - ValueError: PostalCod
e must contain only numeric characters, got: 10A15
FAILED test_postal_code.py::test_postal_code_not_a_string - TypeError: PostalCod
e value must be a string, got int
FAILED test_postal_code.py::test_postal_code_with_whitespace - AssertionError: R
egex pattern did not match.
FAILED test_postal_code.py::test_empty_postal_code - AssertionError: Regex patte
rn did not match.
FAILED test_postal_code.py::test_international_postal_code - ValueError: Invalid
 postal code: 90210
========================= 6 failed, 5 passed in 0.10s =========================
```

8) Test map

1. test_charging_stations_data_display

- Purpose:
  To ensure that when valid charging station data is available, the map displays the correct markers for each station.
- Scenario:
    - Two charging stations are added to the database for a specific postal code.
    - A map is generated, and markers are added for the stations.
- Key Assertions:
    - The correct number of markers (2) is added to the marker cluster on the map.

---

2. test_color_categorization_for_power

- Purpose:
  To verify that charging stations are color-coded based on their power ratings.
- Scenario:
    - Three charging stations with low, medium, and high power levels are added to the database.
    - Markers are added to the map with color-coded icons (green for low power, yellow for medium power, and red for high power).
- Key Assertions:
    - Markers are successfully added to the cluster.
    - The get_marker_color() function correctly assigns colors based on power levels.

---

3. test_map_zoom_behavior

- Purpose:
  To test whether the map adjusts its zoom level appropriately based on the location of charging stations (e.g., postal code or specific region).
- Scenario:
  - A charging station is added with a postal code and coordinates.
  - A map is generated with a zoom level set to the station's location.
- Key Assertions:
  - The map bounds are correctly set to zoom into the region of interest.

---

4. test_empty_or_null_data_handling

- Purpose:
  To ensure that the application gracefully handles charging stations with missing or null values (e.g., power_charging_dev is None).
- Scenario:
  - A charging station with None as its power value is added to the database.
  - The test verifies that no station with null data exists in the final result.
- Key Assertions:
  - Charging stations with null or invalid data are not processed or displayed.

Known Issue

- Test Failure in test_empty_or_null_data_handling:
  - The test currently fails because the database allows null values for power_charging_dev.
  - Suggested Fix: Add validation at the database or application layer to reject null values before processing.