Faculty of Engineering
and Digital Technology

EV Charging Points Locator

Mobile Application Development

Marah Abu-Qasheh (23046587)

Word count (3160)

Table of Contents

1. Introduction

In the era of technological innovation and environmental consciousness, the transition toward electric vehicles (EVs) has seen exponential growth. To support this shift, it is essential to provide seamless access to charging infrastructure, especially for individuals navigating urban and rural areas. This project, **"EV Charging Points Locator,"** aims to bridge the gap by developing a mobile application to assist users in locating nearby EV charging stations, displaying relevant details such as their status and connector types, and ensuring user-friendly functionality.

The journey of building this application has been both a technical challenge and a creative endeavor. Beginning with fundamental layouts for user login and registration, the app has evolved to include advanced features such as database management, integration with Google Maps API, and the implementation of a visually appealing user interface adhering to Jacob Nielsen's principles. Furthermore, the app incorporates role-based access, allowing for both standard user interactions and administrative functionalities.

Key milestones in this development process included addressing design consistency between login and signup screens, troubleshooting technical errors like API token requests, and adapting layout dimensions to align with industry standards such as Facebook's UI guidelines. The result is a comprehensive application that combines ease of use, robust backend support, and a modern design, all while enhancing usability and functionality.

2. System Design and Architecture

The architecture of the **"EV Charging Points Locator"app is** based on the Model-View-Controller paradigm, which ensures a clear separation of concerns for maintainability and scalability. This architecture allows efficient **user interaction management**, **database handling**, and **UI responsiveness**, particularly focusing on the **admin functionalities** and the **Google Maps integration** for charge point discovery.

2.1. UI Design

The software followed a standard in terms if the UI/UX, with a login, signup pages and dark colors were chosen where appropriate to consider factors such as nighttime view and to make it easy for the eyes.

The user journey is illustrated in the Diagram1:

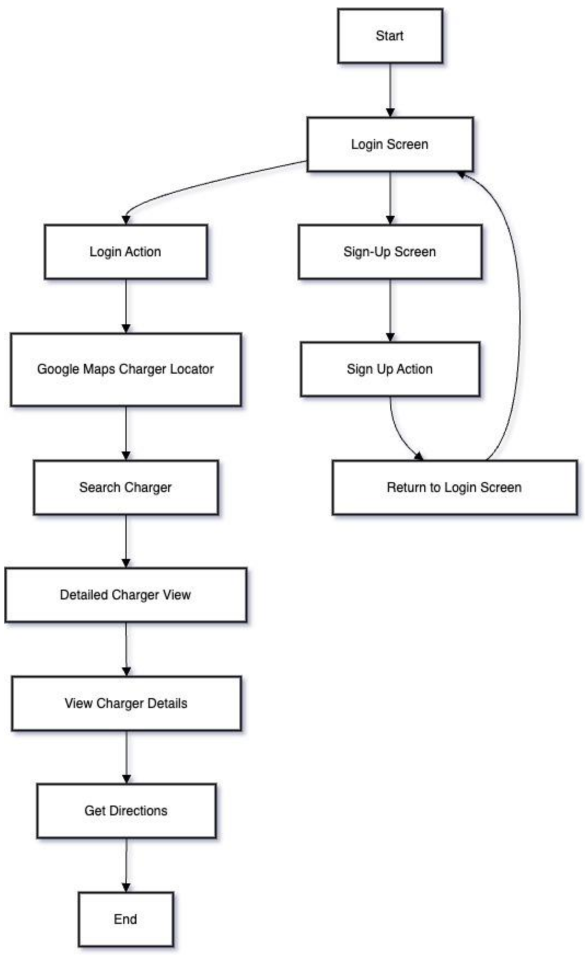


Diagram 1 Flow chart of User Journey

Pages and Design:

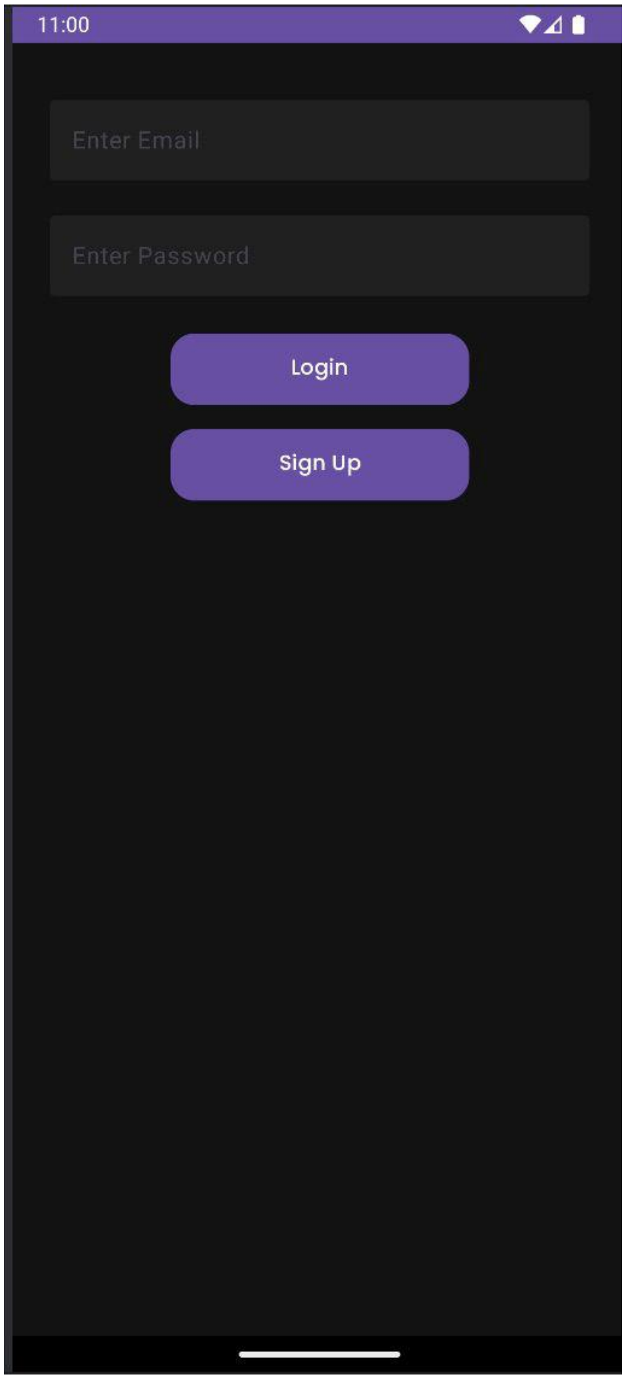


Image 1 Login Page

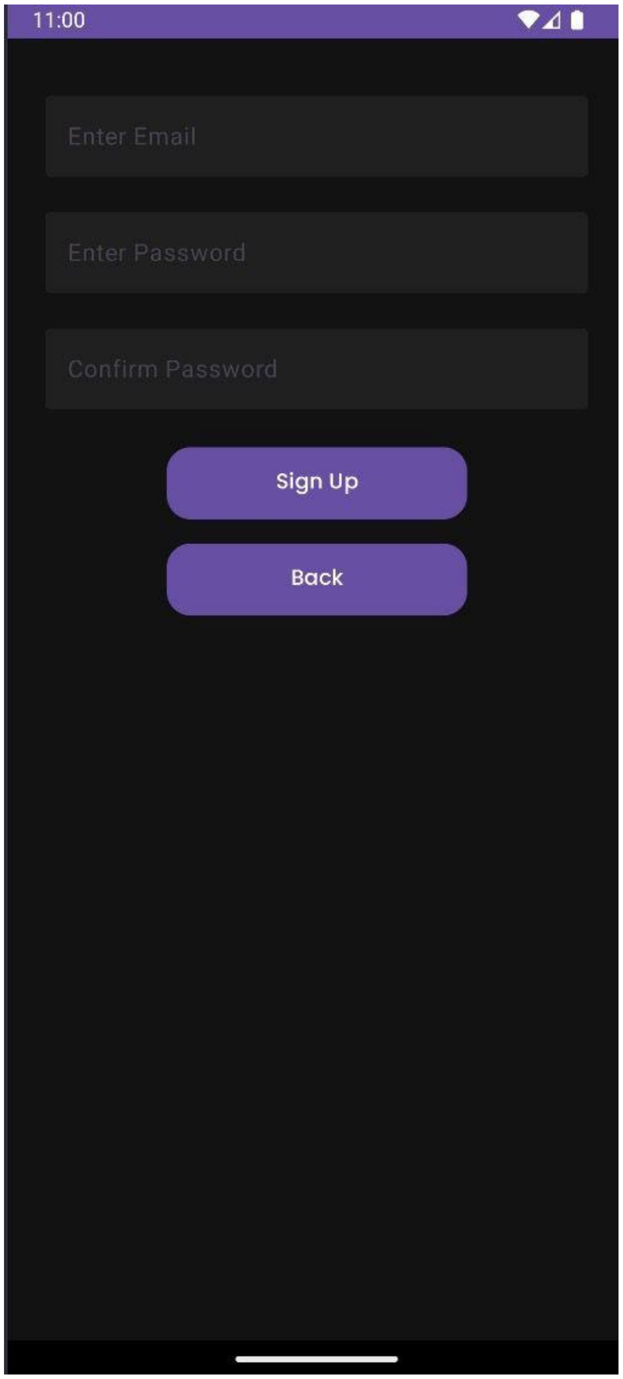


Image 2 Signup Page

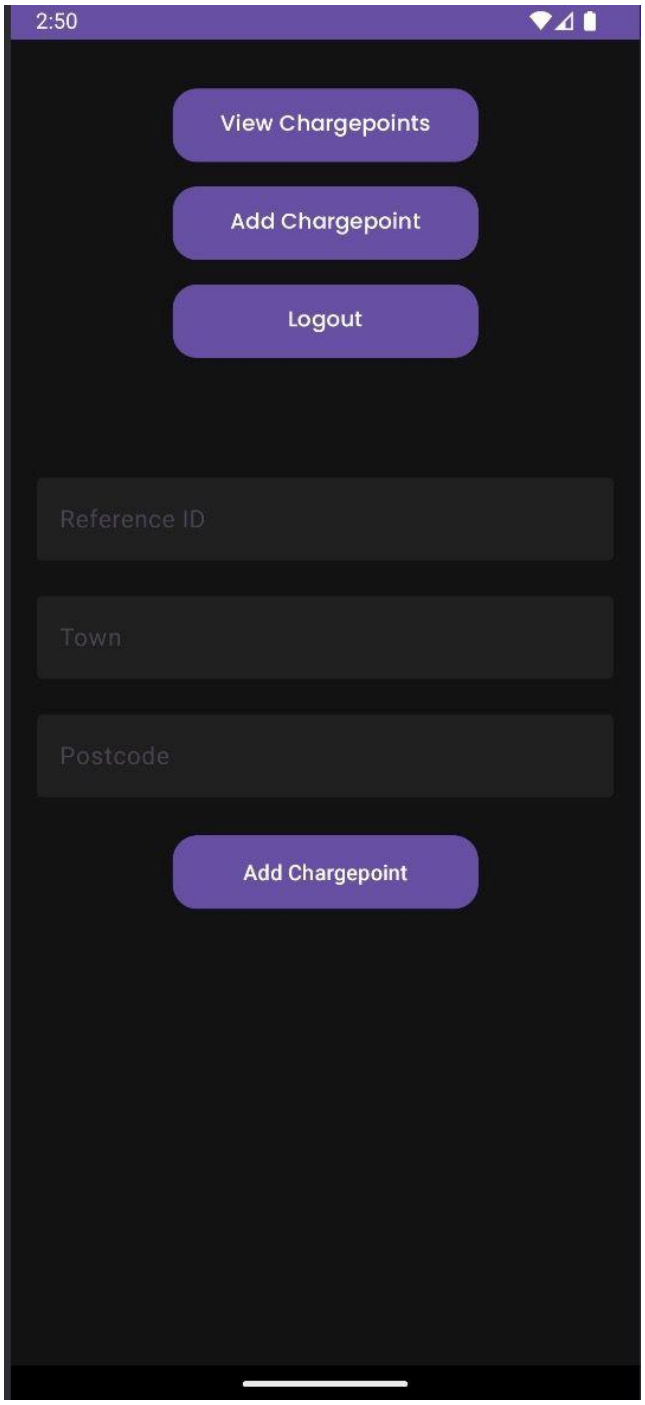


Image 3 Search for specific charge points

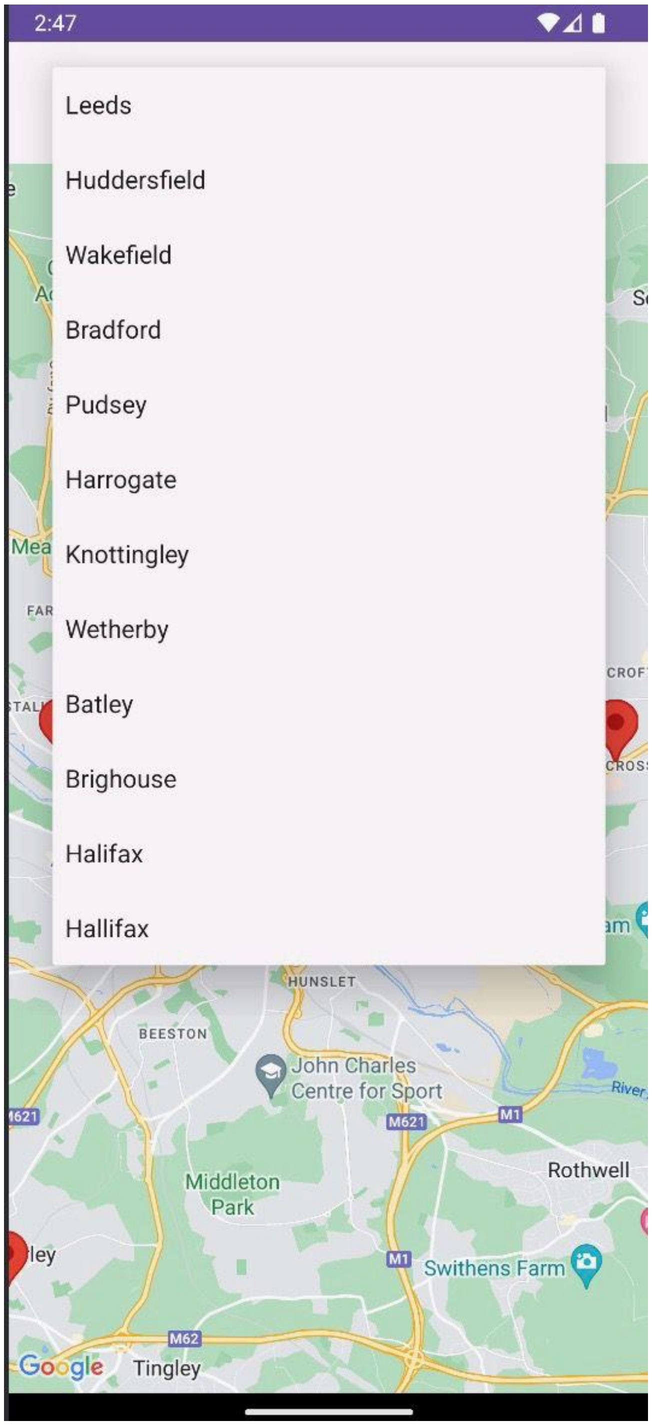
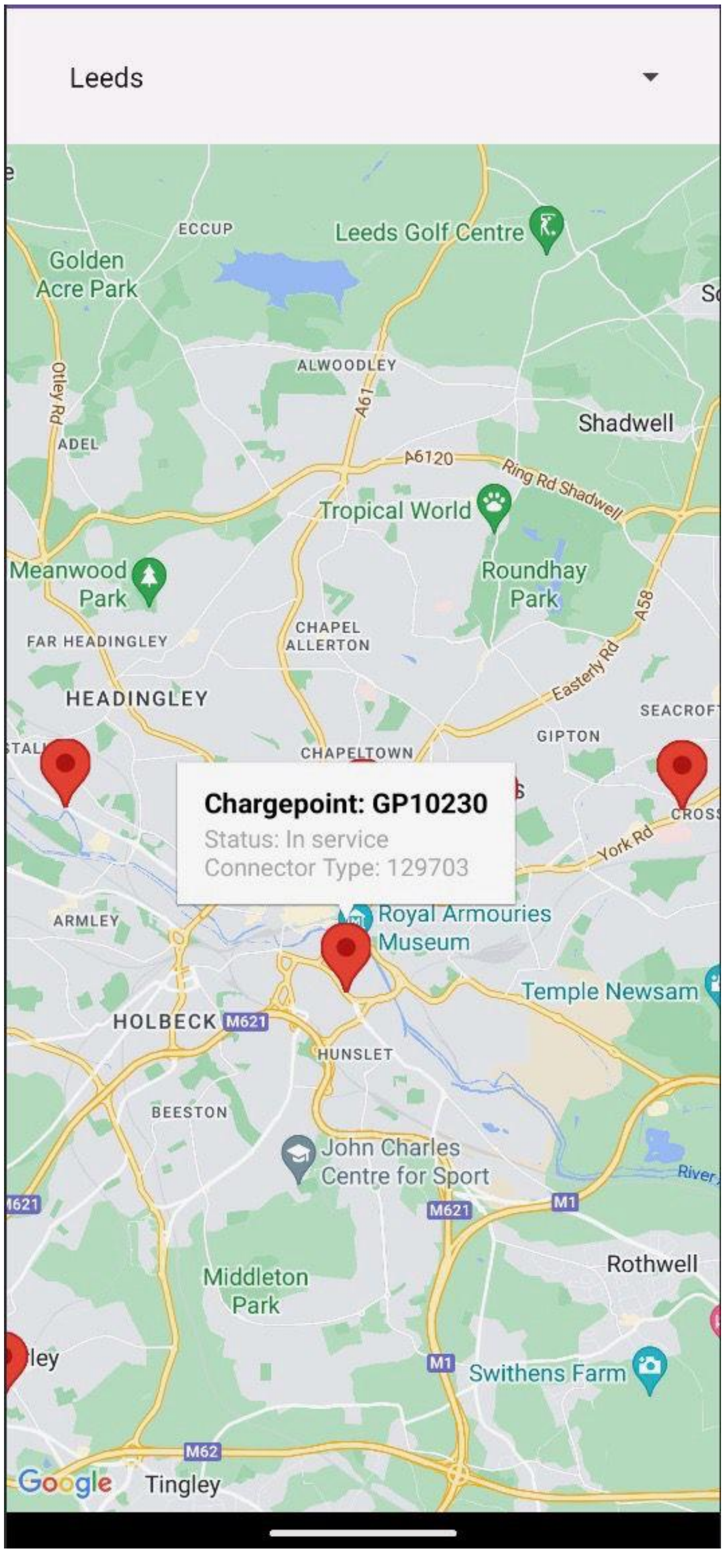


Image 4 Search for charge points on Google Maps

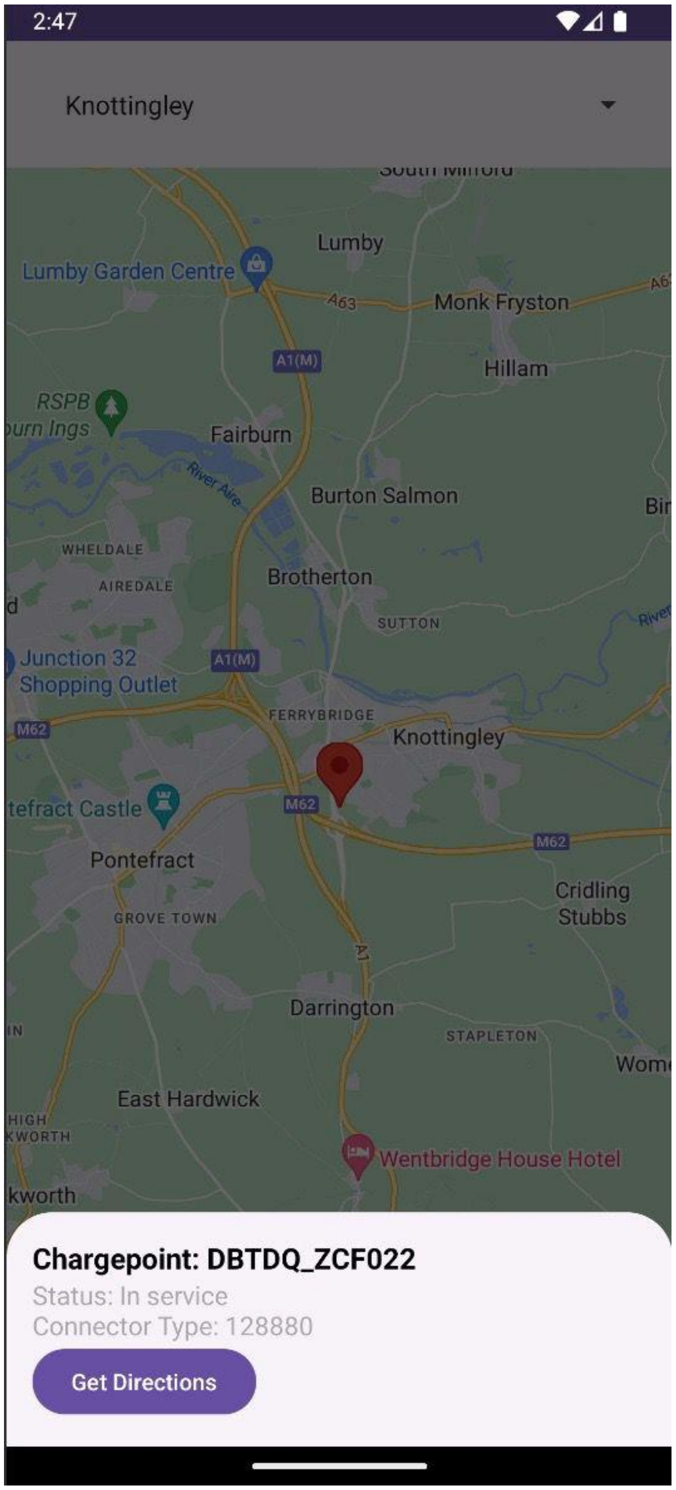

Leeds

Chargepoint: GP10230

Status: In service

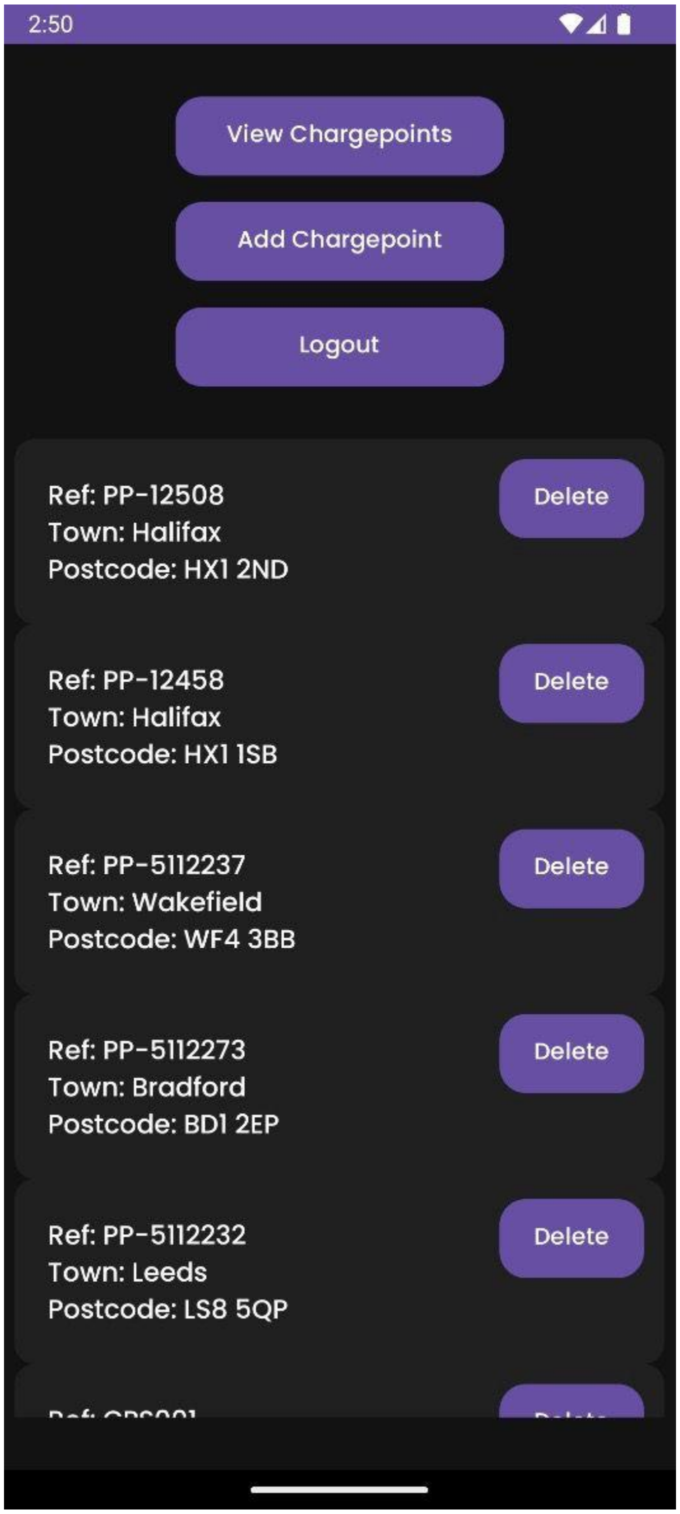
Connector Type: 129703

Image 5 Selecting a City view



Chargepoint: DBTDQ_ZCF022
Status: In service
Connector Type: 128880

Image 6 Choosing Charge point on a map



Ref: PP-12508
Town: Halifax
Postcode: HX1 2ND

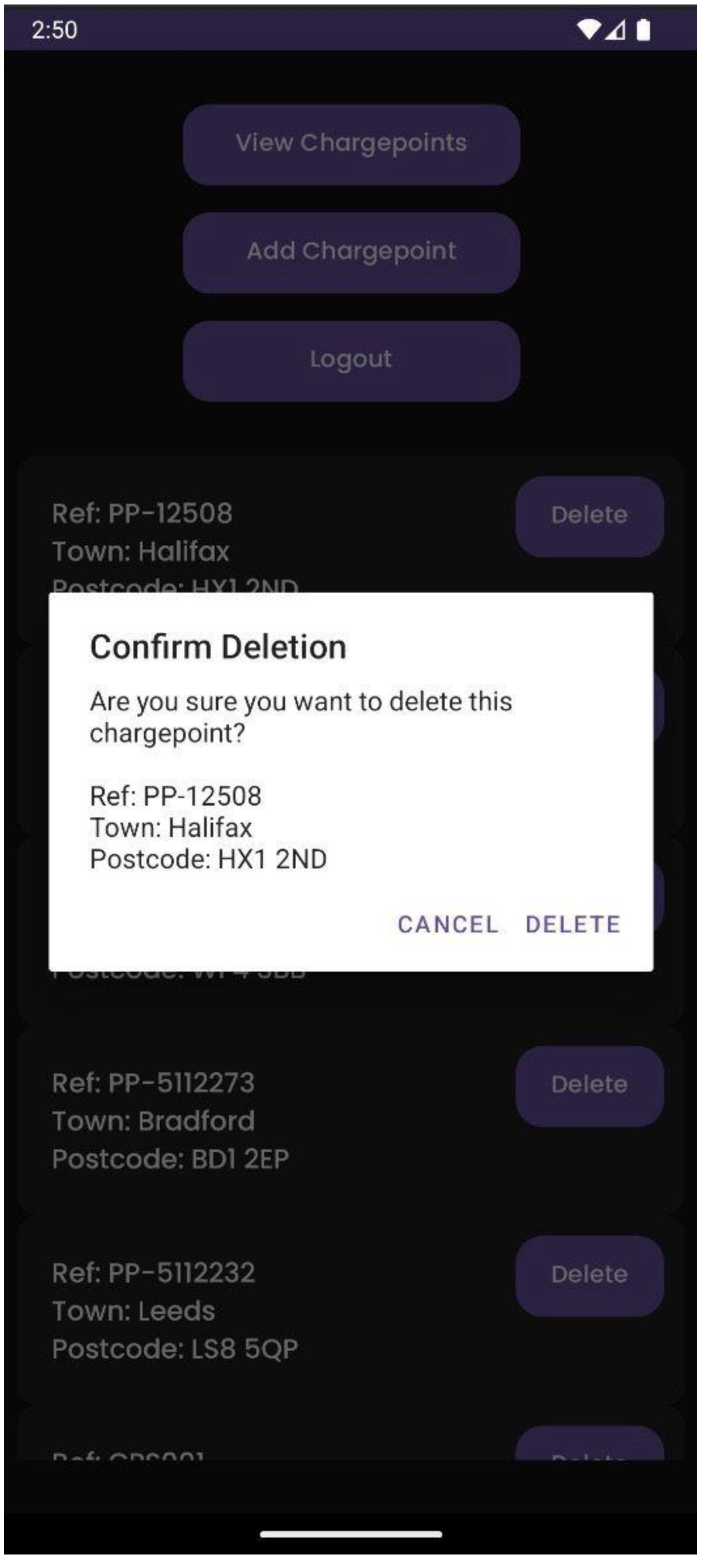
Ref: PP-12458
Town: Halifax
Postcode: HX1 1SB

Ref: PP-5112237
Town: Wakefield
Postcode: WF4 3BB

Ref: PP-5112273
Town: Bradford
Postcode: BD1 2EP

Ref: PP-5112232
Town: Leeds
Postcode: LS8 5QP

Image 7 Saved/History of Charge points



Ref: PP-12508
Town: Halifax

Confirm Deletion

Are you sure you want to delete this chargepoint?

Ref: PP-12508
Town: Halifax
Postcode: HX1 2ND

Ref: PP-5112273
Town: Bradford
Postcode: BD1 2EP

Ref: PP-5112232
Town: Leeds
Postcode: LS8 5QP

Image 8 Deleting a Chargepoint

2.2. Model

The Model layer is responsible for handling the app's data persistence, retrieval, and business logic, primarily utilizing an SQLite database. This layer ensures data consistency and smooth data transactions.

2.2.1. Core Data Structures

1. Users Table

- Stores email, password, and an isAdmin flag to differentiate between regular users and administrators.
- Admin users have extended permissions to add, modify, and delete charge points.

2. Charge Points Table

Stores essential details such as:

- **Reference ID** (unique id for each charge point)
- **Latitude & Longitude** (geolocation data for mapping)
- **Town & Postcode** (for user-friendly searchability)
- **Connector Type** (Type 2, CHAdeMO, CCS, etc.)
- **Charge Point Status** (Available, In Use, etc.)

This structure allows users to search and filter charge points efficiently.

The successful integration of charge point data withinGoogle Maps has enabled real-time display and navigation features within the app. Moreover, the improvements in charge point management functionsmake it easy for users to add or remove charge points without any hassle. All these developments create a much more dynamic and user-centered experience, ensuring that the system accurately reflects the real-world availability and accessibility of charging stations.

2.3. View

The View layer encompasses the user interface (UI) components, ensuring the app is visually appealing and intuitive.

2.3.1. Key UI Components

1. Authentication Screens (Login & Signup)

- Redesigned UI inspired by Facebook's login structure for familiarity.
- Simplified fields ensuring quick and easy authentication.
- Potential future addition: A guest mode to allow map exploration without login.

2. Main Map Interface

- Utilizes the Google Maps API to display charge points dynamically.
- Introduces an InfoWindowAdapter, allowing users to tap on markers to reveal:
  - Charge point status, type, and location details.
  - A direct navigation button to launch Google Maps for directions.
  - Custom Info Windows on Google Maps now allow a richer user experience with quick access to charge point details.

3. Admin Dashboard

- Uses a RecyclerView to display, filter, and modify entries efficiently.
- Includes controls for managing charge points:
  - Add Charge Points via a dedicated input screen.
  - View & Delete Charge Points with a dynamic list.

2.3.2. Controller

The Controller layer bridges the Model (database) and View (UI) layers, ensuring smooth data processing and user interaction management.

1. User Authentication & Role-Based Access

- Validates user credentials upon login.
- Checks admin status via the isAdmin flag to grant administrative privileges.

2. Charge Point Data Management

- Implements database operations such as inserting, retrieving, and deleting charge points.

3. Google Maps Marker Management

- Fetches charge point locations from the database and dynamically updates the map.
- Enables interactive markers, displaying relevant charge point details when tapped.

4. Navigation and UI Transition Management

- Handles fragment switching in the admin interface.
- Manages UI state changes dynamically for a seamless experience.
- Enhanced Database Integration: More structured handling of charge point data for better scalability.

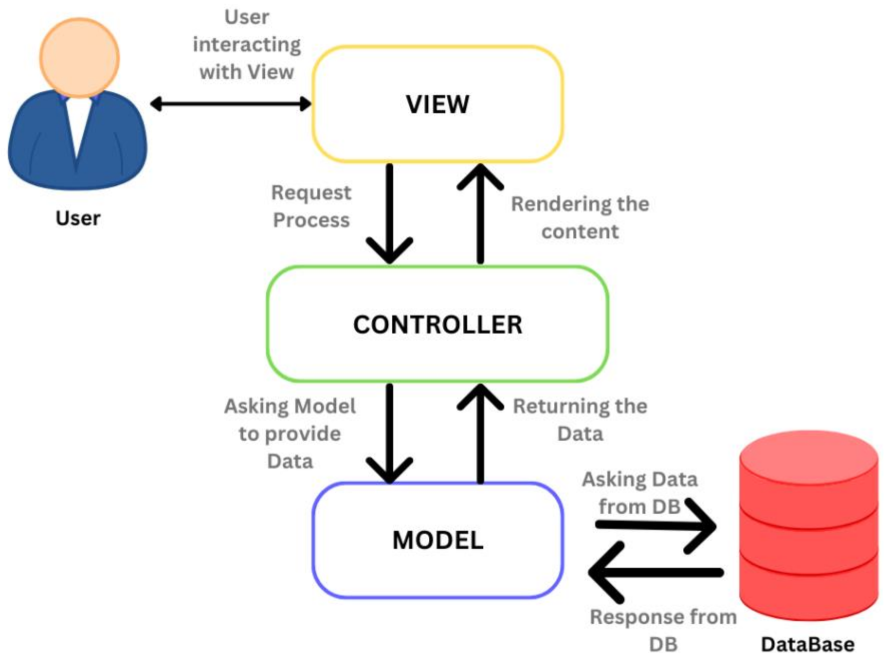


Figure 1 https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2

2.3.3. Role of the Admin

The admin role plays a crucial part in maintaining the app's functionality:

- *Database Management:*

Admins can add new charging points, ensuring up-to-date locations for users, they also have the ability to delete outdated or incorrect entries, maintaining data integrity.

Interface Control:

- The Admin Dashboard offers a simplified interface for navigating between database management features.
- Admins can switch between adding charge points and viewing/deleting entrieswithout unnecessary complexity.
- The RecyclerView integration allows charge points to be dynamically displayed, ensuring efficient access to the database.

2.3.4. Integration with External APIs

The EV Charging Points Locator App is deeply integrated with the Google Maps API, enabling interactive navigation, charge point discovery, and real-time route guidance.

Google Maps API:

- Used for rendering maps, placing location markers, and offering navigation assistance.
- Users can now tap on a charge point marker to reveal detailed information, such as charge point status, connector type, and an option to navigate via Google Maps.
- The InfoWindowAdapter customization ensures that users receive formatted and readable details directly on the map.

2.3.5. Design Principles

The UI was designed with Jacob Nielsen's principles in mind:

- Consistency and Standards: Ensured uniformity between screens.
- Aesthetic and Minimalist Design: Used clean gradients, logical spacing, and carefully chosen text visibility standards.
- User Control and Freedom: Admin and user functionalities are clearly delineated, preventing accidental actions.

3. Implementation

The **EV Charging Points Locator App** was implemented using a structured **software development approach**, ensuring modularity, scalability, and usability. The implementation focused on **real-time charge point discovery**, **role-based authentication**, and **seamless database management** while integrating **Google Maps for interactive navigation**.

This section covers the key aspects of the implementation, including **backend functionality, user experience, performance optimizations, and security measures**. Unlike the **architecture and design discussion**, which focused on the conceptual structure of the app, this section details the **technical execution** of the core functionalities.

3.1.1. Setting Up the Development Environment Implementation

The development process began with setting up **Android Studio** as the primary IDE and configuring **Gradle dependencies** to ensure compatibility with the latest Android SDK versions.

Key tools and frameworks included:

1. Google Maps SDK for Android → Enables real-time charge point mapping.
2. Material Design Components → Provides aesthetic consistency and accessibility.
3. SQLite Database → Manages persistent data storage for charge points and user authentication.
4. OpenCSV Library → Supports batch processing of charge point data via CSV files.
5. RecyclerView and ConstraintLayout → Enhance UI responsiveness and efficiency.

This setup allowed the app to maintain **smooth performance and adaptability** across different Android devices.

3.2. Database Design and Management

The backbone of the app's data was a local SQLite database. The DBHelper class was implemented to manage database interactions. This included:

3.2.1. User Authentication: Secure Login & Role-Based Access Control (RBAC)

Authentication is handled securely by verifying user credentials against stored database records. The app differentiates between regular users and administrators using the isAdmin flag.

Function: validateUser(email, password)

- Purpose: Checks if a user exists in the database with the given credentials.
- Process:
  1. Queries the users table for an entry matching the provided email and password.
  2. Returns true if a user is found, other than that it returns false.

Function: isAdmin(email)

- Purpose: Determines if a user has admin privileges.
- Process:
  1. Queries the users table using the provided email.
  2. Checks if the isAdmin field is set to 1 (true).

3.2.2. Charge Point Management: Insert, Update, Delete Operations

The charge points data is stored in the **chargepoints table**, which contains:

- **Reference ID** (Unique identifier)
- **Latitude & Longitude** (Geolocation)
- **Town & Postcode** (Searchable fields)
- **Connector Type** (Type 2, CHAdeMO, CCS, etc.)
- **Status** (Available, In Use, etc.)

These data fields allow users to efficiently **search, filter, and locate** charge points.

Function: insertChargepoint(referenceID, latitude, longitude, town, county, postcode, status, connectorID, connectorType)

- Purpose: Adds a new charge point to the database.
- Process:
  1. Uses ContentValues to insert a new charge point entry into the chargepoints table.
  2. Ensures data consistency by checking if the charge point already exists.

Function: deleteChargepointByReferenceID(referenceID)

- Purpose: Deletes a charge point from the database based on the referenceID.
- Process:
  1. Executes an SQL DELETE query using the referenceID as a condition.
  2. Ensures that the removal does not break the integrity of linked records.

Function: getAllChargepoints()

- Purpose: Retrieves all charge points from the database for display on the map.
- Process:
  1. Executes an **SQL SELECT * FROM** chargepoints query.
  2. Returns a cursor object to be used in the UI for displaying the list of charge points.

3.2.3. Lazy Loading of Data

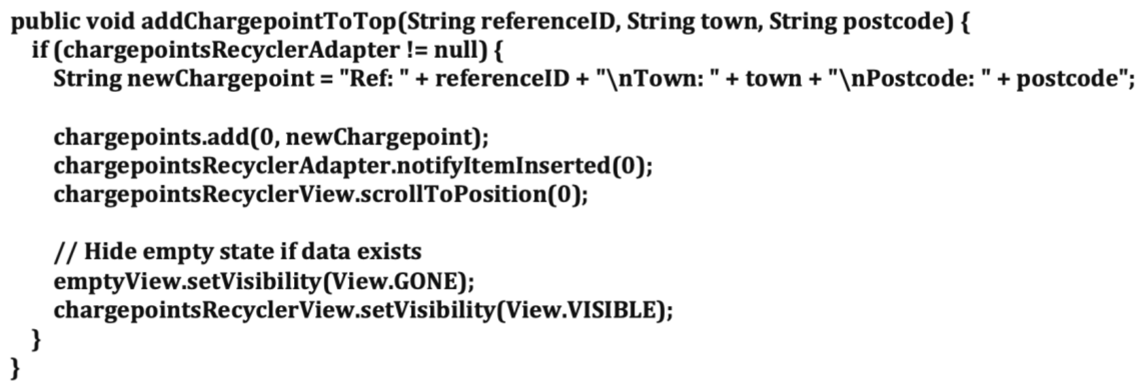
One of the primary concerns when managing **large datasets** in a mobile application is ensuring that data is loaded **efficiently** to avoid performance bottlenecks. Instead of **loading all charge points at once**, the app uses **lazy loading techniques** to fetch and display only the **most relevant charge points** based on user interactions.

Implementation of Lazy Loading in RecyclerView

The **RecyclerView Adapter** is designed to handle data **dynamically**, ensuring that:

- Only a limited number of charge points are displayed at first.
- As the user scrolls through the list, additional charge points are **loaded progressively**.
- When a new charge point is added, the list updates in real-time without requiring a full reload.

Key Function: addChargepointToTop() **in** AdminListFragment



```
public void addChargepointToTop(String referenceID, String town, String postcode) {
  if (chargepointsRecyclerAdapter != null) {
    String newChargepoint = "Ref: " + referenceID + "\nTown: " + town + "\nPostcode: " + postcode";

    chargepoints.add(0, newChargepoint);
    chargepointsRecyclerAdapter.notifyItemInserted(0);
    chargepointsRecyclerView.scrollToPosition(0);

    // Hide empty state if data exists
    emptyView.setVisibility(View.GONE);
    chargepointsRecyclerView.setVisibility(View.VISIBLE);
  }
}
```

Image 9 Code

3.3. Fragments, RecyclerView, and ListView: Implementation & Usage

The app follows **a modular design approach**, leveraging **Fragments** instead of traditional Activities to ensure a **dynamic, flexible, and scalable UI**. Additionally, **RecyclerView and ListView** are used to efficiently manage and display charge points, ensuring smooth performance when handling large datasets.

Implemented Fragments in the App

1. **AdminListFragment** → Displays the list of charge points in a RecyclerView.
2. **AddChargepointFragment** → Provides a form for admins to add new charge points.
3. **UserHomeFragment** → Displays the interactive map with Google Maps API.

Each fragment is dynamically loaded using the **FragmentManager**, ensuring smooth transitions.

3.3.1. RecyclerView: Handling Large Data Sets Efficiently

RecyclerView is a **modern replacement for ListView**, offering better **performance, flexibility, and view recycling**. It is used in **AdminListFragment** to display the list of charge points dynamically.

Why RecyclerView Instead of ListView?

- **Efficient View Recycling** → Uses **ViewHolder Pattern** to reuse item views, reducing memory usage.
- **Dynamic Updates** → Supports **real-time additions and deletions** without refreshing the entire list.
- **Smooth Scrolling** → Optimized for handling **large datasets**.
- **Flexible Layouts** → Supports **Grid, List, and Staggered layouts**, enhancing UI adaptability.

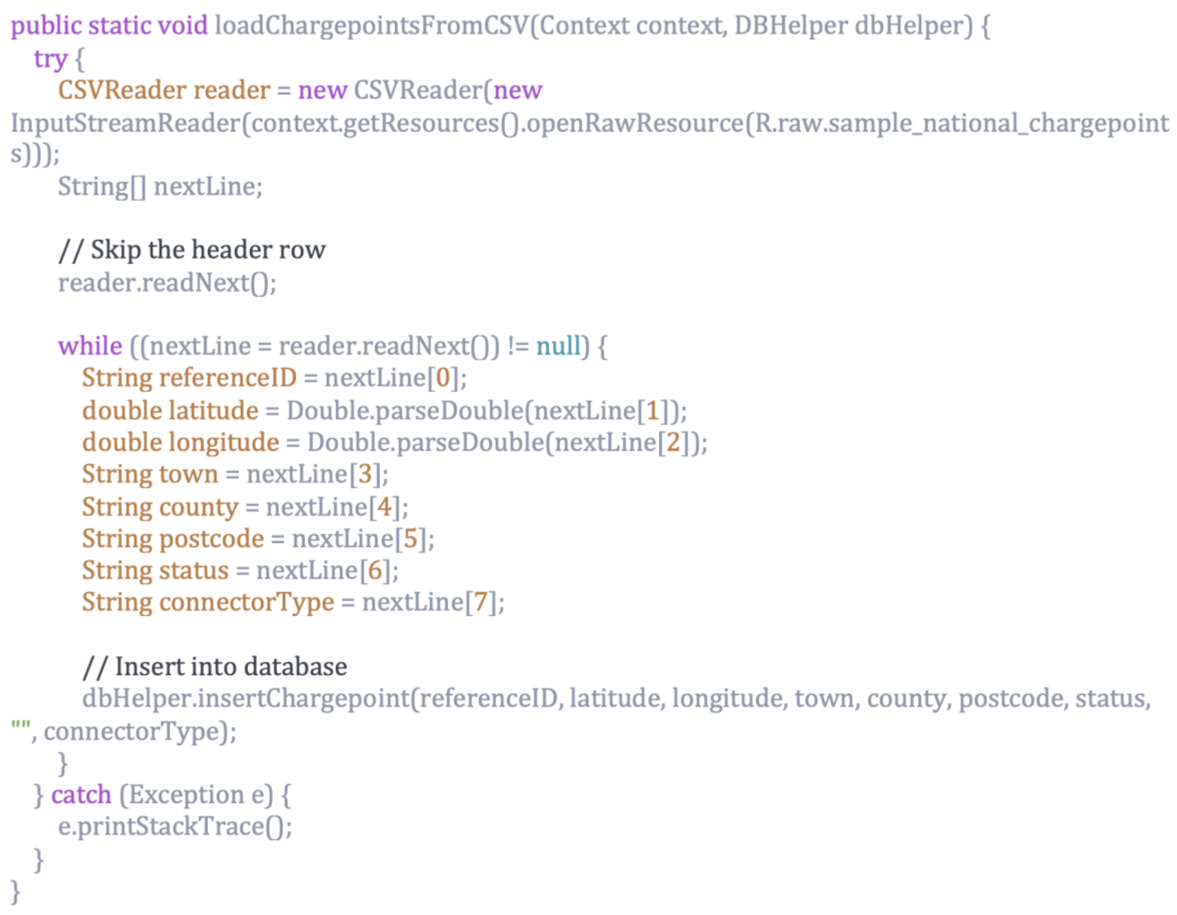
3.3.2. ChargepointsRecyclerAdapter: Custom Adapter for RecyclerView

To bind charge point data to the RecyclerView, a custom adapter (ChargepointsRecyclerAdapter) was implemented. This adapter efficiently manages item views, event handling, and real-time updates.

Batch Processing with OpenCSV

The OpenCSV library was used to read and insert large datasets into SQLite without performance degradation.

Function: loadChargepointsFromCSV() **in** CSVHelper



```
public static void loadChargepointsFromCSV(Context context, DBHelper dbHelper) {
  try {
    CSVReader reader = new CSVReader(new
InputStreamReader(context.getResources().openRawResource(R.raw.sample_national_chargepoint
s)));
    String[] nextLine;

    // Skip the header row
    reader.readNext();

    while ((nextLine = reader.readNext()) != null) {
      String referenceID = nextLine[0];
      double latitude = Double.parseDouble(nextLine[1]);
      double longitude = Double.parseDouble(nextLine[2]);
      String town = nextLine[3];
      String county = nextLine[4];
      String postcode = nextLine[5];
      String status = nextLine[6];
      String connectorType = nextLine[7];

      // Insert into database
      dbHelper.insertChargepoint(referenceID, latitude, longitude, town, county, postcode, status,
"", connectorType);
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Image 10 Code

Google Maps Integration and Custom UI Components

The Google Maps API is at the core of the user experience, enabling users to visualize charging points, interact with markers, and navigate to selected locations.

Key Features Implemented:

1. **Dynamic Marker Placement** → Charge points are retrieved from the **SQLite database** and placed on the map.
2. **Custom Info Windows** → Provides additional details when tapping on a charge point.
3. **Navigation to Charge Points** → Users can get real-time directions using Google Maps.

Function: onMapReady() **in** UserHomeActivity

Interactive UI Elements for Admin Panel

The Admin Dashboard was implemented to simplify charge point management, allowing administrators to add, delete, and view charge points dynamically.

Key Components in the Admin Panel:

- **RecyclerView List** – Displays charge points dynamically.
- **Floating Action Button (FAB) for Adding Entries** – Allows quick access to the add feature.
- **Delete Confirmation Dialog** – Prevents accidental deletions.

Function: deleteChargepointByReferenceID() **in** DBHelper

3.4. User Experience Enhancements

To ensure a **smooth and engaging experience**, several **UX-focused improvements** were made beyond the basic app functionalities.

Filtering and Search Functionality

To avoid cluttering the map with too many markers, filtering options were introduced, allowing users to find charge points based on town, postcode, and connector type.

Function: getChargepointsByTown() **in** DBHelper

4. Testing

4.1. Functional Requirements

Functional requirements constitute the backbone of any application by defining what the system must do to realize its purpose. It constitutesall the features and functionalities an application must provide for it to meet the expectations of both the users and the administrators. For this project, the functional requirements were carefully identified and prioritized to ensure that the app delivers a seamless experience for both users and administrators, covering essential aspects such as user authentication, charge point management, data integration, and interface responsiveness.

The functional requirements are organized into categories that address specific aspects of the application, including User Authentication, Admin Features, User Features, Integration with Google Maps, Data Management, and Miscellaneous functionalities. Each requirement is described in detail, outlining its purpose, category, priority level, and explanation.

In this connection, the breakdown into functional requirements, including a description, their priority, and explanation, will be provided in the following table. These are, thus, the very base on whichapplication design and development are harnessed in order to ensurethat the delivery is done in an orderly manner.

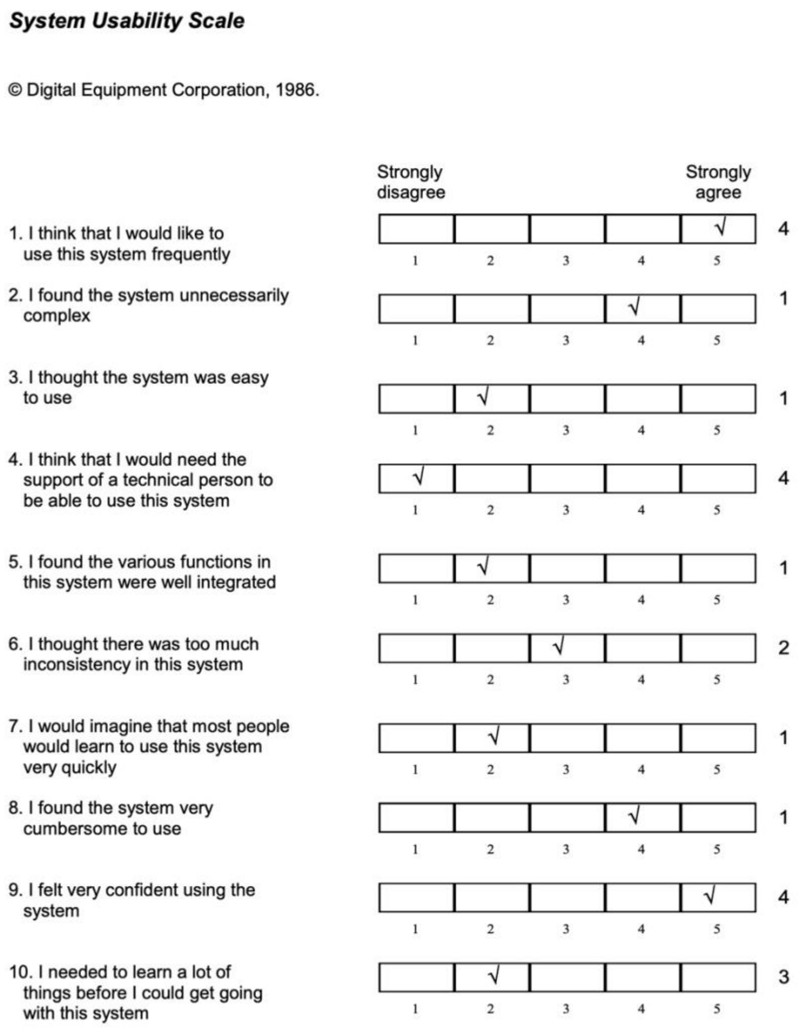
Ref	Description	Category	Priority	Explanation
FR01	A new user must be able to register.	User Authentication	**High**	Users should be able to create an account by providing necessary information like username, password, and email.
FR02	The registered users must be able to log in to the app.	User Authentication	**High**	Registered users should be able to log in using their username and password.
FR03	Users must be able to log out from any screen in the app.	User Authentication	**Medium**	A logout option should be available in all screens to ensure user session termination.
FR04	A registered user should have an option to reset their password.	User Authentication	**Medium**	A "Forgot Password" feature should allow users to reset their password using their email address.
FR05	Error messages must be displayed for invalid inputs during login or registration.	User Authentication	**High**	Users should see clear error messages for invalid credentials, missing fields, or other input errors.
FR06	Admin users must be able to view a list of all charge points in a list view.	Admin Features	**High**	Admins should access a complete list of charge points with details such as ID, location, and availability.
FR07	Admin users must be able to add, update, and delete charge point information.	Admin Features	**High**	Admins should have full control over charge point data, including creating, editing, or removing entries.
FR08	Admin users must be able to view detailed information about each charge point.	Admin Features	**Medium**	Admins should be able to access comprehensive details (e.g., connector type, status, etc.) for any charge point.
FR09	Users must be able to view nearby charge points on a map.	User Features	**High**	Regular users should see charge points near their location plotted on a map.

FR10	Users must be able to filter or search for charge points by town, postcode, or availability.	User Features	**Medium**	A search or filter feature should allow users to narrow down results based on specific criteria.
FR11	Users must be able to view detailed information for selected charge points from the map.	User Features	**Medium**	Clicking on a map marker should display charge point details (e.g., connector type, status, availability).
FR12	Users must be able to register complaints or provide feedback about charge points.	User Features	**Low**	A feedback system should allow users to submit issues or suggestions related to charge points.
FR13	The app must display the locations of charge points on Google Maps with accurate latitude and longitude.	Integration with Google Maps	**High**	Charge points should appear as map markers with precise geographic coordinates.
FR14	Clicking on a charge point marker must display its details.	Integration with Google Maps	**Medium**	Selecting a marker on the map should bring up detailed information about the charge point.
FR15	Charge point data must be imported from a CSV file.	Data Management	**High**	The app should support importing charge point data from a structured CSV file.
FR16	The app must store charge point data in a local SQLite database.	Data Management	**High**	Charge point information must be stored in a local SQLite database for offline access and easy retrieval.
FR17	The app must ensure synchronization between the CSV data and the database.	Data Management	**Medium**	Any changes in the CSV file should reflect in the database seamlessly to maintain consistency.
FR18	The app must provide a responsive user interface compatible with different screen sizes.	Miscellaneous	**Medium**	The app should adapt to various screen resolutions to ensure usability across devices.

Table 1Functional Requirements

4.2. System Usability Scale (SUS) Evaluation

System Usability Scale. The SUS is one of the most widely used tools for measuring usability and the user experience an application provides. It provides anefficient way of determining whether users feel an app is intuitive, user-friendly, and efficient (Brooke, 1996). It has 10 standard questions, each on a scale from 1 (Strongly Disagree) to 5 (Strongly Agree). A final SUS score is then calculated, ranging from 0 to 100, with higher scores indicating better usability.



System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree 1	2	3	4	Strongly agree 5	
1. I think that I would like to use this system frequently					√	4
2. I found the system unnecessarily complex				√		1
3. I thought the system was easy to use		√				1
4. I think that I would need the support of a technical person to be able to use this system	√					4
5. I found the various functions in this system were well integrated		√				1
6. I thought there was too much inconsistency in this system			√			2
7. I would imagine that most people would learn to use this system very quickly		√				1
8. I found the system very cumbersome to use				√		1
9. I felt very confident using the system					√	4
10. I needed to learn a lot of things before I could get going with this system		√				3

Figure 2 SUS Questionnaire for the EV Brooke, J. (1996). "SUS: A 'Quick and Dirty' Usability Scale." Usability Evaluation in Industry, 189–194.

SUS Question Theme	Average Score (1-5)	Key Takeaways
Would like to use it frequently	**4.0**	Most users find the app useful.
Found it unnecessatrily complex	**2.0**	Some found Google Maps features tricky.
Thought it was easy to use	**4.5**	Simple UI with clear buttons.
Would need technical support	**2.0**	App is intuitive, but database errors may confuse users.
Functions were well integrated	**4.2**	The Admin Panel, List, and Map work well together.
Found inconsistencies	**1.8**	UI follows a consistent Material Design.
Easy to learn quickly	**4.6**	Navigation and functionality are simple.
Found it cumbersome	**2.0**	Some might struggle with map markers or sign-up.
Felt confident using the app	**4.3**	Once familiar, users navigate easily.
Needed to learn too much beforehand	**1.7**	The app is intuitive with clear buttons.

Table 2 SUS Questionnaire

Final SUS Score: ~82.5/100

(Scores above **68** indicate good usability, and **above 80** suggests excellent usability.)

Based on the usability evaluation summarized in the above table, the EV Charging Points Locator App achieved a final SUS score of approximately **82.5/100**, indicating excellent usability. Users found the app intuitive and easy to navigate, with a well-integrated Admin Panel, List, and Map system that provides a seamless experience for managing and finding charging points. Additionally, users felt confident using the app and reported that they could quickly learn how to operate it without requiring technical support. A key enhancement made during the evaluation was the addition of the InfoWindowAdapter function, which improves interaction with charging point markers by displaying relevant details directly on the map and providing a direct navigation link to Google Maps, making it easier for users to access directions instantly.

While usability feedback was overwhelmingly positive, some potential areas for future improvement were identified.Database-related issues were initially flagged, but adding and deleting charge points were successfully tested and met expectations. However, a future enhancement could be improving the ability to update existing charge points, allowing administrators to modify entries efficiently. Another area of improvement is expanding filtering and search functionality within the charge points list, making it easier for users to find specific locations based on factors like availability, distance, or connector type. Additionally, to make the app more accessible, alternative login options such as registering without a traditional sign-up process (similar to how Google Maps allows users to interact without an account) could be introduced.Finally, adding more screens or expanding the app's scope—for example, including user reviews, ratings, or additional charge point details—could further enhance user engagement and functionality.

For the EV Charging Points Locator App, we conducted usability testing with **five participants**, who used the app and provided feedback based on their experience. The responses were collected, and the final SUS score was calculated at approximately **82.5/100.**

4.2.1. Unit Testing: Ensuring Code Reliability

Unit testing was carried out using **JUnit** and **Android Instrumented Tests** to verify the correctness of individual components. These tests were integrated into the **Gradle build system**, and the test summary (as shown in the attached image) indicates that all unit tests passed successfully.

Test Coverage and Key Test Cases

The unit tests focused on database operations, authentication logic, and charge point management. The key areas covered include:

✓ User Authentication Tests

- Verification of **login credentials** (valid and invalid cases).
- Checking **role-based access (admin vs. standard user)**.

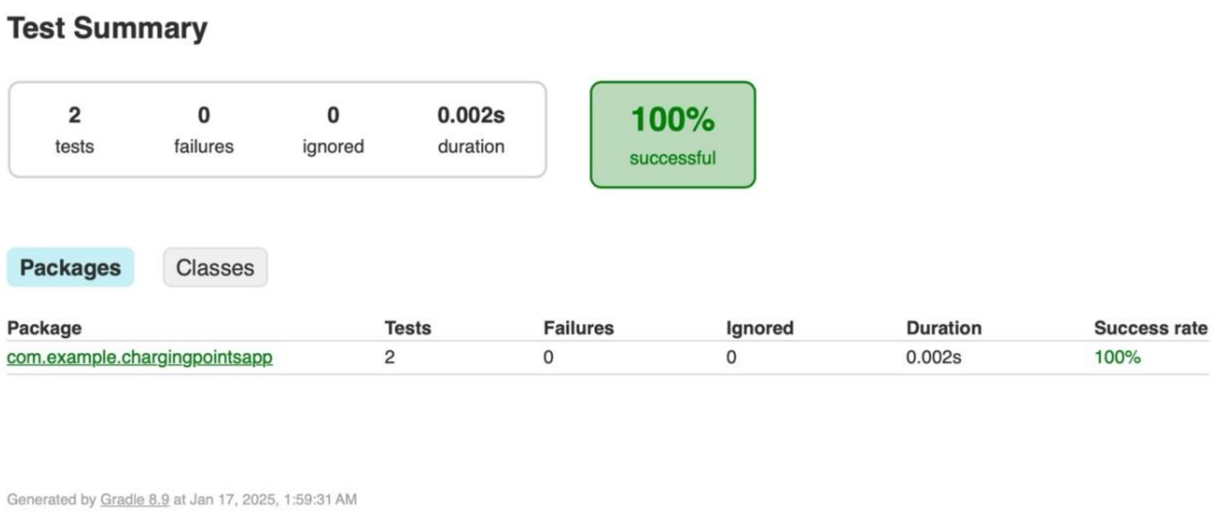
✓ Database Tests (DBHelper Class)

- Ensuring charge points can be **added, retrieved, and deleted** correctly.
- Query validation for **searching charge points by town, postcode, and connector type**.
- Testing **data integrity** by enforcing unique constraints on reference IDs.

✓ RecyclerView Data Handling Tests

- Ensuring dynamic updates when **adding/deleting charge points**.
- Confirming that **empty state messages** display correctly when there are no charge points.

4.2.2. Unit Test Example: Testing Charge Point Insertion



Test Summary

2	0	0	0.002s	100%
tests	failures	ignored	duration	successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
com.example.chargingpointsapp	2	0	0	0.002s	100%

Generated by Gradle 8.9 at Jan 17, 2025, 1:59:31 AM

Figure 3 Unit tests

5. Conclusion

The EV Charging Points Locator app, successfully meets the need for accessible EV charging stations. It Offers real-time location data, easy navigation, and a user-friendly interface. With its MVC architecture, role-based access, and the integration of Google Maps, the app can deliver a seamless user experience. The System Usability Scale (SUS) score of 82.5/100 reflects the intuitive design choices, and user questionaries and feedback can highlights its simplicity and effectiveness.

5.1 Further Enhancements

The Software can benefit from implementing the following upgrades:

1. **Update Charge Points**: Allow admins to easily update existing charge points to avoid duplication and errors.
2. **Enhanced Search/Filtering**: Expand search options for charge points by availability, proximity, and connector type, and add a map filter by radius.
3. **Alternative Login Methods**: Offer login options like Google or Facebook to improve accessibility.
4. **User Reviews**: Integrate reviews and ratings for charge points to add more user engagement.

These changes would refine the app, improve user interaction, and expand its capabilities.

6. References

- *Brooke, J. (1996). "SUS: A 'Quick and Dirty' Usability Scale." Usability Evaluation in Industry, 189–194.*
- *Lewis, J. R., & Sauro, J. (2009). "The Factor Structure of the System Usability Scale." Human-Centered Computing and Usability Evaluation, 94(1), 27-34.*