Technical Document

for

ENRoute

Version 2.0 approved

Prepared by
Craig Turnbull
Joe Corona
Lucas Keizur
Richard DeYoung
Kirsten Boyles

Envorso - CWU

3/12/2022

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Team	1/31/22	Document Creation	1.0
Team	3/12/22	Document Finalization	2.0

1. Introduction

Electric Vehicles (EV) are cars that have the ability to operate by electricity stored within a battery. As gasoline is an unsustainable resource, the push to transition to electric cars has steadily increased. As a result, the infrastructure to support these types of vehicles has also improved, and several businesses have come into the field to fulfill this need. Tesla is one of the leaders in both supplying the cars and the infrastructure needed to charge them. As Tesla was one of the first in the field, they have their own plugs and their own charging stations that are only able to be used by Tesla customers. Tesla is not the only company in either the EV producing and charger producing field. Ford, Hyundai, Nissan, and most other car companies have started their own line of EV to sell and they all have the same standard charging plug (different from Tesla's) and also have different brands, prices, and no centralized way to find these charging stations, check prices, and pay for them. The application that will be made as a result of this document will be able to satisfy these needs.

1.1 Purpose

The purpose of this document is to provide a detailed description of the electric car charger locator application. The development of this software will be done by Seniors from CWU for the company Envorso. The document will explain the purpose and features of the software, the software's interfaces, what the software will do, and the constraints of the software. The document is intended for the developers of the software and the project managers.

1.2 Intended Audience and Reading Suggestions

The intended audience for this document are the project managers at Envorso (Bob Rapp and Parker Jones), the charging application team, and Dr. Szilárd Vajda of CWU.

Demo of app on an emulator:
https://www.youtube.com/watch?v=zSXFXhtqk1c&ab_channel=KirstenRBoyles

1.3 Definitions / Acronyms / Abbreviations

Term	Description
EV	Electric Vehicle
CC	Credit Card
CWU	Central Washington University

AC	Alternating Current
DC	Direct Current
kWH	Number of kilowatts transferred per hour
API	Application Programming Interface
iOS	iPhone Operating System
GUI	Graphical User Interface
REQ-x	Requirement number ‘x’
PID	Personal Identifiers
Fast charging	Charging that has an output of 50kWH - 120kWH.

1.4 Product Scope

This software is an electronic car charger locator. Users will be able to enter in all the services they are a part of, and then the software will search for nearby chargers. This will solve the issue of non-Tesla owners having a poor experience attempting to locate charging stations. The software will not charge the user, but will payment information. The main benefit to this software is allowing users to be able to use a single application to handle finding the best charger.

Due to this being a Capstone project and not having as much time or resources available, only Ellensburg, Moses Lake, Cle Elum, and Yakima will be considered in development. This will allow the team to properly test the functionality of the software. However, by the end of development it will be easy to add any additional services that are located outside of these locations.

1.5 References

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
- https://www.myev.com/research/ev-101/ev-terminology
- “Smartcar · API platform for connected car data” (2022). https://smartcar.com/. Retrieved 1/12/2022.
- “NREL All Stations API” (2022). https://developer.nrel.gov/docs/transportation/alt-fuel-stations-v1/all/. Retrieved 1/26/2022.
- Google Maps API: https://developers.google.com/maps
- https://pub.dev/
- https://docs.flutter.dev/

2. Overall Description

2.1 Problem Statement

Across the electric vehicle industry, non-Tesla EV owners struggle to find charging stations that work for them. This is because charging stations are not like traditional gas stations in which advertisements direct the driver to the station's location, but rather they hide behind businesses and struggle to be seen from the road. On top of that, charging stations have various types of charging plugs and speeds. Most plugs are not cross-compatible without an adaptor, which forces drivers to be more deliberate with the station they choose. The charging speed also plays a huge role in the decision as some cars aren't compatible with DC fast charging and additionally, the slower, level 1 chargers, can take up to 12 hours to charge the car depending on the size of battery, which is simply not tolerable for the majority of people. All in all, finding an electric vehicle charging station that fits the user's needs of speed, plug type, and location is a very frustrating process.

2.2 Product Perspective

This product is an Android application that can be easily ported to iOS. It's main goal is to direct the users accurately to charging stations. It will be created for Envorso Consulting by a team of students from Central Washington University for their Senior Capstone project in Winter Quarter 2022. The target users for this product will be those who have electrical vehicles that are non-Tesla as the current systems for these customers are unreliable.The goal is to help new EV owners have peace of mind by having a reliable "one-stop-shop" EV car charging app for daily use and while traveling.

2.3 Product Functions

- Users will be able to locate a car charger.
- Users will be able to determine the speed of chargers located.
- Users will be able to store the charging businesses that they are a part of.
- Users will be able to pay for most charging services utilizing this one application.
- Users will be able to create an account within the application.
- Users will be able to plan a trip with stops for charging along the way.
- Users will be able to add stops for charging in the middle of a trip.

2.4 User Classes and Characteristics

The intended users for this system are owners of electric cars, especially non-Tesla owners. Non-Tesla owners do not have an all-in-one application like Tesla does. The application should be simple enough for drivers of all technical experience to use. This will not only involve making the interface simple but providing quality documentation to the users in the form of first-time login instructions and help menus.

2.5 Operating Environment

The application is a mobile application that functions natively on Android and is made using the Flutter framework for easy porting to an iOS device.

2.6 Design and Implementation Constraints

The car charging information that is stored does not have all of the necessary PID information needed in order to properly implement some wanted features. iOS development was hampered by a lack of resources available to developers. This can be easily overcome with access to these resources and time.

Creating an account with other services presents a whole host of issues, such as needing to create a unique web-scraper for each of those sites who have browser sign ups available. Several services that users will want do not have in-browser sign ups available, so there is a hurdle of creating accounts automatically when there is only an app version to sign up with. InApp purchases of other services charging could also be a constraint.

2.7 User Documentation

Users will be provided with a self-explanatory first-time login, with small explanations given to help guide them. A help menu will also be accessible within the app (via the settings screen) for users to view if they ever need help navigating the software.

2.8 Assumptions and Dependencies

The assumptions of the product are that development will be conducted on a development framework aimed at Android and iOS applications. The product is also dependent on the limited use of APIs. The budget, time, and team size limit the viable solutions to large data retrieval processes. If time allows, Additional features dependent on time includes a ‘trip feature’ which would allow users to create a multi-destination course

with its subsequent chargers enroute. Further features can also be added with further development time.

3. External Interface Requirements

3.1 User Interfaces

3.1.1 First Sign in

On launch and without an account already logged in - If the user has an account they will go from this page to 3.1.2's "home page" using the "Sign In" button. *See figure 1 and figure 2*. If the user needs to create a new account, they will click "Sign Up" and this will prompt the next screen. This includes the initial sign up to make the account, Terms of Services and privacy policy includes warning about data usage and consent. If the user has filled out a valid email and a valid password, they can continue to the rest of the account set up. *See figure 3.*

Continue Account Setup with prompting the user to fill out necessary PID information needed to fill out other service accounts, including First name, last name, username, charger types, car brands, credit card information, and zip code. More information may be needed as more PID information is collated. Car brands will have a list populated with EV car brands where the user can select one brand. After the user clicks "continue", the Auto-Sign up page will be the final page for the user's account creation.

A list generated based on nearby charging services to the user's home ZIP code or address and other charging services. The local services are marked by a start icon and are automatically checked. The user can uncheck them and check nonlocal ones. *See figure 4.*

Charger Type brings up a list of chargers that the user can select their charging plugs from. *See figure 5.* Explanation for needing credit card info for use will display a card with the information. *See figure 6.*

3.2 Hardware Interfaces

This software will function on iOS and Android, so both operating systems need to be considered during development. This includes design of the user interface. The software also needs to properly connect to the internet to be able to interact with the chargers. This means that the application should be tested to make sure it runs on both types of devices.

The software will also need to interact with the phone's microphone for the implementation of voice commands. The software must be able to receive voice commands correctly and respond appropriately.

3.3 Software Interfaces

Envorso branding may be used in the user interface, but there are currently no hard requirements on this. The team may use creative freedom when designing the UI. These designs should be pitched to the client early to prevent lost time. Because the software is a mobile app, it should be designed with a touchscreen in mind. The app should also be voice compatible, as the user may be using it while driving.

3.3 Communications Interfaces

The application will have to communicate with other charging applications or programs in order for the user to properly pay for them. Encryption is needed due to credit card information for the user being stored and sent to these other applications. Cell Phone signal or internet (Wi-Fi) will be required in order for this application to work properly. If disconnection occurs, it will continue the task according to the last known information.

In the proof of concept phase, only chargers in Ellensburg, Moses Lake, Cle Elum, and Yakima will be considered, as they are the only ones that the team is able to easily test

4. System Features

Key:

- **H** – High priority
- **L** – Low priority
- **O** – Optional (future update)

4.1 First-Time Login

4.1.1 Description and Priority

Users will be prompted to create an account when they first open the application. Users will be able to sign in with a service such as Google or Apple to make sign-in simpler. They will be required to enter in PID such as phone number, address, charger type, and credit card info. All of this will be stored on the database.

If users are already signed up with our company, there will be an option to sign in as well. If the user has signed in already on a device, persistent sign-in will activate and login will not be shown.

4.1.2 Functional Requirements

REQ-1: The software will prompt users for PID info and the charge port their car has. **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.
- ***Implemented****:* When creating an account with the app for the first time, this information will be gathered. Can be updated in Settings as well.

REQ-2: The software will allow users to sign up with a new account. **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.
- ***Implemented****:* Will ask users to sign up if no account is associated with their email already.

REQ-3: The software will allow users to sign up using Google or Apple. **L**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* Connecting Google with the application will be easy since the team will be using Firebase as a back-end, which is a Google product. Connecting with Apple may be more difficult.

- ***Not Implemented****:* This was not implemented due to time constraints.

REQ-4: The software will generate a list of chargers surrounding the user's home address that they can sign up for. **L**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible, but will be harder. This requires prior knowledge of the surrounding chargers in a ZIP code. This would be able to work for Ellensburg, Moses Lake, Yakima, and Cle Elum, but the team may struggle with getting the application ready to handle more than those two zip codes.
- ***Implemented****:* This was implemented using the user's provided ZIP code. It takes the ZIP code and returns the list of chargers in that area and asks if the user would like to sign up for each service.

4.2 Locating a Nearby Charger (Fastest or Cheapest)

4.2.1 Description and Priority

Finding the best car charger involves searching for chargers with the city entered, and giving the user a filter to weigh each charger, according to the speed of the charger, and how expensive the charger is. This is a top priority feature.

4.2.2 Functional Requirements

REQ-5: The software can match the vehicle's charge port with chargers nearby. The software will not present users with chargers their car is incompatible with. **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.

- ***Implemented****:* For example, the users won't get a charger that does not have their charging plug.

REQ-6: The software will be able to determine the speed of chargers located. **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.
- ***Implemented****:* A charger's speed is stored in the database (Level 1, Level 2, DC fast).

REQ-7: The software will be able to determine the price of chargers located. **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement has feasibility issues. Price of chargers isn't always given up front, so the team will have to dig a little deeper to find this information.
- ***Implemented****:* Only implemented for Free not free, retrieving real time pricing was not feasible at this point of the application.

REQ-8: The software will be able to determine how long it takes for a vehicle to charge fully. **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.
- ***Not Implemented****:* Retrieving this real time data was not feasible at this stage, SmartCar API may be utilized to satisfy this feature in the future.

REQ-9: The software will be able to determine which charger is the best based on price or speed of the charger (up to the user to decide). **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.

- *Feasibility:* This requirement is feasible.
- ***Implemented****:* Allowing users to filter their search results was a feasible task once the team had located all necessary data.

REQ-10: The software will get the current charge of the vehicle from an API call or from user entry. **L**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.
- ***Not Implemented****:* Not implemented due to cumbersome requirements for a user to constantly enter their own level of charge. Solution would be SmartCar API or something similar at a later stage.

REQ-11: The software will be able to determine if a charger is currently in use. **O**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement has feasibility issues. This information is reliant upon users checking into a charging station. Also, not every owner of an EV will use this app, so the data will be incorrect most of the time.
- ***Not Implemented****:* This was not implemented because the team did not have a way to get information from the charging station itself. A solution to this is to use another API from the charging station, but not every charging station has an API that we would be able to use.

REQ-12: The software will have a report feature to allow users to report on the status and quality of the charger. **O**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.
- ***Not Implemented****:* While this may be a good feature to have in the future, it was not feasible to create it in this amount of time.

4.3 Voice Command Navigation

4.3.1 Description and Priority

Drivers will have their hands occupied on the steering wheel so having a system that is able to have voice command navigation is of high priority. The benefits are boundless because this will prevent the people in the automobile and on the road from having an accident. This feature will work in conjunction with feature 4.2.

4.3.2 Functional Requirements

REQ-13: The software will respond to voice commands. **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Not Implemented****:* Flutter framework for Voice activation is set up within the app, but the feature is not currently fully functional.

REQ-14: The software will be able to navigate users to a charger based on what a user asks for in the voice prompt. **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Not Implemented****:* Trip planning was not a priority during the development of the project, so this was not prioritized either.

4.4 Planning a Trip with Stops for Charging

4.4.1 Description and Priority

Users will also be able to plan out a route. This involves users entering in a starting point and destination, along with when they want to stop for a charge. The software will then modify the travel time based on how many stops the user will make, and how long a charge will take based on the method developed in section 4.2. The team believes that this

feature will be able to be implemented if the team has time. It also won't be able to function until section 4.2 is implemented properly.

4.4.2 Functional Requirements

REQ-15: The software will be able to function like a mapping software (think Google Maps or Waze). **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* Assuming that the development team has access to the Google Maps API or something similar, this requirement is feasible. Otherwise, the development team does not have the time or resources to create a map-like API in the given timeframe.
- ➢ ***Implemented****:* The app does look and feel like a mapping software. The Google Maps API does not support in-app navigation, so a deep-link was implemented. The deep-link takes users to Google Maps at the correct location.

REQ-16: Users will be able to add stops in a trip based on when they want to charge. **O**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement has feasibility issues. This may not be able to be implemented in the given time-frame due to the complexity of what is required for the feature.
- ➢ ***Not Implemented****:* Trip planner functionality not fully implemented due to limited range of current database and time constraints.

REQ-17: The software will present users with the best charger near the requested stop (See section 4.2). **O**

- ➢ *Consistency:* This requirement is consistent with other requirements.

- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Implemented****:* The app does present users with a list of chargers in and around the searched city. Since trip planning functionality was not implemented, there is not a way to search near a requested stop.

REQ-18: The software will modify the travel time based on how long each stop will take. **O**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Not Implemented****:* PID information for this requirement could not be required at this stage. Possibly can be implemented in the future.

REQ-19: The software will allow users to add stops while on the trip. **O**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Not Implemented****:* Trip planner functionality not fully implemented due to limited range of current database and time constraints.

4.5 Connecting to existing subscription services

4.5.1 Description and Priority

There are a lot of different subscription services that electric car owners can be a part of, the software should be able to store these. This makes it easier for the software to know which chargers to offer up to the user.

4.5.2 Functional Requirements

REQ-20: The software will have a place for users to enter in pre-existing subscription services. **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Implemented****:* The app does let users select which services they have an account with. However, the app does not connect to these services due to time constraints.

REQ-21: The software will need to store each user's subscription service on a database. **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Implemented****:* Any services that users select are stored in the database.

5. Other Nonfunctional Requirements

Key:

- **H** – High priority
- **L** – Low priority
- **O** – Optional (future update)

5.1 Performance Requirements

REQ-22: The charger locator must return results to the users in a quick and efficient manner. **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.
- ➢ ***Implemented****:* Results are presented to the user in a real-time fashion.

5.2 Safety Requirements

REQ-23: User Credit Card information will be encrypted. **O**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is not feasible in the given time frame.
- ➢ ***Not Implemented****:* Since this app was a proof-of-concept, security was never a priority. Should development continue, security will become a primary concern. Since Firebase was used, adding security would be an easy task.

5.3 Security Requirements

REQ-24: The database that stores user data must be protected to prevent user data from leaking to external sources. **O**

- ➢ *Consistency*: This requirement is consistent with other requirements.
- ➢ *Validity*: This requirement is valid.
- ➢ *Feasibility*: This requirement is feasible.
- ➢ ***Not Implemented****:* Since this app was a proof-of-concept, security was never a priority. Should development continue, security will become a primary concern. Since Firebase was used, adding security would be an easy task.

5.4 Software Quality Attributes

REQ-25: The software will be developed for iOS, but will be developed to make porting to Android easy. **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible. The team is using a framework that supports cross-platform development.
- ➢ ***Implemented****:* Developed and tested for Android due to hardware constraints, but made for using easy porting to iOS at a future stage.

REQ-26: The software must be usable by people of all technical skill levels. **H**

- ➢ *Consistency:* This requirement is consistent with other requirements.
- ➢ *Validity:* This requirement is valid.
- ➢ *Feasibility:* This requirement is feasible.

- ***Implemented****:* The current UX/UI design helps facilitate a user experience on the application. Testing with those of different skill levels needs to take place.

REQ-27: Help menus and tutorials must be present in the software. **H**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement is feasible.
- ***Implemented****:* Help menu is available in settings and support is also available there.

REQ-28: The software must automatically pay for the charger once the user confirms they wish to use the charger. **L**

- *Consistency:* This requirement is consistent with other requirements.
- *Validity:* This requirement is valid.
- *Feasibility:* This requirement has a feasibility issue. This may be difficult depending on how secure other charging service's websites and payment methods are. It is also dependent on how the app creates accounts for users in these various charging services.
- ***Not Implemented****:* Feasibility issues with interactions with other services made this requirement difficult to implement. With further testing, research, and possible collaborations with services could make this requirement much more feasible.

6. Software Tools

This app was developed using the Flutter framework. Flutter is developed and maintained by Google. It is used to develop cross-platform, mobile and web applications from a single codebase. Flutter is written in the Dart language, which is an object-oriented, class-based, garbage-collected language.

6.1 Dependencies

6.1.1 Packages

All Dart and Flutter packages are uploaded and maintained on pub.dev. In the project, all packages are stored in the pubspec.yaml file. Below is a list of all packages that were installed.

- dio
- flutter_polyline_points
- google_maps_flutter
- location
- map_launcher
- flutter_svg *(Not in Use)*
- flutter_map *(Not in Use)*
- latlong2 *(Not in Use)*
- google_place (*Not In Use*)
- cupertino_icons
- firebase_core
- cloud_firestore
- firebase_auth
- avatar_glow
- highlight_text
- speech_to_text
- flutter_signin_button
- country_state_city_picker (*Not In Use*)
- flutter_credit_card (*Not In Use*)
- speech_recognition
- english_words
- permission_handler

6.1.2 Libraries

For building the web scraper, the BeautifulSoup and Mechanize libraries were used for Python.

6.1.3 Languages

The languages that were used:

- Dart

- Python

6.1.4 Operating System

Windows 10 64-bit, since it was a minimum requirement on the recommended specifications of Flutter. For the Mac system, the latest Apple M1 Processor is required for flutter.

7. Hardware Requirements

Hardware Requirements will be related to what the Flutter minimum requirements entail. To run the application, a smartphone (Android or Apple), or smartphone emulator. A stable internet connection is also required.

7.1 Hardware Minimum Requirements

7.1.1 CPU

Physical Phone - Snapdragon

Emulation - The CPU minimum requirement to be able to support the Flutter system requirements is Intel Core i5-8400.

7.1.2 Memory

Physical Phone - Minimum tested 4 GB of RAM, 200 KB of disk space needed.

Emulation - Disk Space required is 1.64 GB and 2GB of RAM according to Flutter specifications.

8. Software Documentation

In the subheadings below are the written text that will help illustrate what is helping with the computer software and show what is embedded in the source code. Code organization consists of the different modules, functions, parameters, and the algorithms that were used in the car charger app.

8.1 Code Organization

Code is properly organized, imports are at the top all together, main method is concise, classes have proper spacing, widgets receive their BuildContext parameters, etc. Separation of the different files helped divide the work which resulted in making the code look less crammed, that was the goal.

Dart supports asynchronous programming. Asynchronous programming allows programs to continue execution, while waiting for another operation to finish, such as accessing a database. In Dart, there is a class called **Future<T>**, where **T** is the type (int, double, String, etc.) This works with asynchronous programming as a function labeled as **async** won't return immediately; it will return a future result.

There are a few other important keywords or symbols to understand in Dart. One is the '?' symbol. The '?' signifies that a variable may be null. Saying "int? a;" means that 'a' can be null. This is required since Dart supports null safety. The "dynamic" type indicates that the variable type can be changed.

8.1.1 Different Modules

Modules that were created:

- aboutUs.dart
- addAcounts.dart
- chargeStation.dart
- dateValidation.dart
- enRouteAccountSettings.dart
- firebase_options.dart
- firebaseFunctions.dart
- firstlaunch.dart

- help.dart
- main.dart
- mapScreen.dart
- newUser.dart
- newUserEmail.dart
- savedLocations.dart
- searchPage.dart
- servicesList.dart
- settings.dart
- speech_recognition.dart
- startUp.dart
- userAuth.dart

8.1.2 Functions

Functions that were used:

- build()
- dispose()
- goToAddAccounts()
- goToHelp()
- goToInformation()
- goToMap()
- goToSearchPage()
- goToSettings()
- initState()
- itemChange()
- run()
- setState()
- signInWithEmail()
- updateUserState()

A number of these functions were used more than once.

Chargers() Class

Future<List<Map<String, dynamic>>>	**pullCharger(double, double)** Returns a list of charging stations within a set range away from the given double values, latitude and longitude.
List<Map<String, dynamic>>	**filterChargers(List<Map<String, dynamic>>, List<bool>, List<bool>)** Returns a list of charging stations based on the boolean Lists set by the filters.
Future<List<Map<String, dynamic>>>	**findCity(String)** Returns all the chargers within the given city.
Future<List<String>>	**pullServices(double, double)** Return a list of services within a range of the given latitude and longitude coordinates.
Future<int>	**activateAccount(String)** Activates the plugs and networks class by populating them with the user's data.
void	**setCarPlug(List<String>)** Set the list of Car plugs to be filtered by.
void	**setMemberships(List<String>)** Set the list of memberships to be filtered by.
void	**setRange(int)** Set the range for which chargers will be searched for.
void	**setMinSize(int)** Set the minimum size of the list of chargers to be returned. The search will continue to go until this number is met or it has reached the max range.
List<Map<String, dynamic>>	**maskPlugs(List<Map<String, dynamic>>)** Returns a list of charging stations that don't include the plugs not owned by the user.
List<Map<String, dynamic>>	**orderDistance(double, double)** Returns a list of charging stations in the order from nearest to farthest based on the given longitude and

	latitude coordinates.
int	**geoHash(double, double**) Returns an integer value which represents a .005 by .005 longitude by latitude block based on the given coordinates.
List<List<int>>	**getGeoSet(int geohash, int range)** Returns a List of Lists of geohash values with the given geohash as the center and the range as the radius. The Lists within the List are at most 10 values long since firebase only allows querying for only 10 value arrays.

firebaseFunctions() Class

String	**getUId()** Returns the user's unique ID.
Future<String?>	**createAccount(String email, String password)** Creates a new account in Firebase with the provided email and password. Returns the user's ID
void	**createUser(String uId, String email, String password, String username, String phoneNumber, String street, String city, String zip, String state, String name, String creditCard, String expiry, String cvv, List<String> chargers)** An asynchronous function that will create a new user in the database with their personal information.
void	**updateAccount(String uId, String email, String password, String username, String phoneNumber, String street, String city, String zip, String state, String name, String creditCard, String expiry, String cvv, List<String> chargers)** An asynchronous function that will update a user's account in the database. Will not overwrite any existing data.
void	**addServices(String uId, List<String> services)**

	An asynchronous function that will add charger services to a user's account
Future<List<String>>	**getChargers(String uId)** An asynchronous function that get the list of charger ports in a user's account
Future<Map<String, dynamic>>	**getPID(String uId)** An asynchronous function that gets all of the user's PID info from the database
Future<List<Map<String, String>>>	**getServices(String uId)** An asynchronous function that gets all of the user's PID charging services from the database. It returns a list containing the name of the service, whether it is networked, and the price.

firebaseFunctions() Class

Future<String?>	**registerWithEmail(String email, String password)** An asynchronous function that creates a new user account with the given email and password.
Future<int>	**signInWithEmail(String email, String password)** An asynchronous function that signs in a user with the given email and password

8.1.4 Geohashing

Geohashing is the process of splitting the earth into geographic hashes defined by a unique key that represents that location. The way this was implemented in ENRoute was that each 0.005 latitude by 0.005 longitude acted as a single hash, represented by an integer value. Since there are 180 points of latitude and 360 points of longitude, based on the system of geohashing, there are a total of 907,200,000 possible geohashes on earth.

8.1.4 Widget Interface

Refer to Appendix D for documentation on ENRoute's Widget Interface

9. Installation

9.1 Steps

9.1.1 Android

1. Download the Flutter SDK. This can be done by going to https://docs.flutter.dev/get-started/install
2. Now download Visual Studio Code at https://code.visualstudio.com/download
3. Install the Android studio, along with the Android SDK. This can be done at https://developer.android.com/studio
4. As of 3/12/2022, you must pull the git repository from https://github.com/Kboyles138/envorsocarcharger (The repository location will change by 3/14/2022)
5. Once steps 1 through 4 are complete, open the shell and type "flutter doctor". The shell will prompt the user with any additional packages required to run
6. Open Visual Studio Code
7. Go to the extensions tab and search Flutter, then install the first result called 'Flutter'
8. Open the repository in VSCode
9. Open Android Studios and go to 'more actions' and then click 'AVD manager'
10. click 'Create Virtual Device' and choose a pixel 2 along with the most up to date API.
11. Then hit the green action button to start the emulator.
12. On the bottom right corner of VSCode, the name of the emulator should appear after some time.
13. When the emulator appears, run the app from the "Run and Debug" tab.
14. The app is now running…

9.1.2 Apple

Tutorial from Flutter for installation: https://docs.flutter.dev/get-started/install/macos

1. Download the Flutter SDK. This can be done by going to https://storage.googleapis.com/flutter_infra_release/releases/stable/macos/flutter_macos_2.10.3-stable.zip
    a. extract file
2. Download Visual Studio Code at https://code.visualstudio.com/download
3. Install xCode: https://apps.apple.com/us/app/xcode/id497799835?mt=12
4. As of document version 2.0, you must pull the git repository from https://github.com/Kboyles138/envorsocarcharger (The repository location will change at a future date)
    a. add flutter tool to the path in Visual Studio terminal
5. Run ‘sudo gem install cocoapods’ in VS studio Terminal
6. Follow the xCode signing flow to set up provisioning for the project
7. Run ‘flutter doctor’ in terminal to make sure that things have been installed and set up correctly
8. Run ‘open -a Simulator’ in terminal to start emulation

9.2 Troubleshooting

If the app won’t start after hitting ‘run’ in VSCode, open the shell and navigate to the location of the head Flutter file. This means the folder ‘envorso_charging_app’, one level higher than the ‘lib’ folder. Once there, enter “flutter pug get” to download any needed packages.

10. User Documentation

10.1 Operating the Software

10.1.1 First time launched - Sign up

1. When the app has been loaded, the user should first hit ‘signup’
2. The page changes, and the user will be prompted to give their email and password, along with accepting the terms of service and privacy policy.
3. After hitting Continue, the user will be prompted to give their username, address information, phone number, credit card info (optional) and the charging plugs they have.

4. The user will then be presented with a list of local (according to ZIP) charging networks. Their input currently has no functionality. The user should then press continue.
5. The user is now on the main map page of the app.

10.1.2 Log-in

1. When the app has been loaded, the user can be prompted to submit their email and password and login.
2. If the password and email information are correct, the user will go to the main page of the app

10.1.3 Map screen

1. The user is now on the map page where it centers on the users current location
2. The map displays yellow and red pins for chargers nearby. Red meaning super charger available, yellow meaning supercharger not available.
3. There is a filter button that allows the user to choose what chargers are displayed, by speed and by free or costs money.
4. Button on the right hand side opposite of the filter button allows the map to recenter on the user.
5. Search bar links to the search bar in 10.1.4
6. Saved Locations links to the the page linked in 10.1.6
7. Settings links to the page linked in 10.1.5
8. When the charger pin is selected by the user, a card with the charger name, location, charge speeds, and plugtypes appears. The card also has the ability to be closed (top right), deep link to google maps for turn by turn navigation to the charger, and be saved/unsave in collaboration with the “saved locations” page.

10.1.4 Search Bar

1. Clicking on the search bar on the map screen will bring users to the search page.
2. Here, they can enter in a city name (currently only Yakima, Ellensburg, Cle Elum, and Moses Lake). The search bar only supports these cities at this moment. Entering in anything other than a city will return no results
3. Click on the “Search” button will display the results below. These results are pre-filtered based on the charger types given by the user. A user can filter

further based on if the charger is free or has a fare, as well as charger speed (DC fast, Level 2, Level 1).
4. A user can save a charger on this page, and it will appear in the “Saved Locations” page. Clicking on one of the results will navigate the user to the charger on the map.

10.1.5 Settings

1. In “Settings” there are four options: “Add Service Accounts”, “EnRoute Account”, “Information”, and “Help”.
2. “Add Service Accounts” displays all charging services in the database. If the service is non-networked, it will display the name of the charger, as well as the city name
3. “EnRoute Account” will allow users to update their account info. This page also contains the sign-out button
4. “Information” contains information about the app, the development team, and Envorso.
5. “Help” contains useful information about the app, such as what certain symbols mean and how to navigate through the app.

10.1.6 Saved Locations

1. The user will be able to see all their saved chargers. The information will include the levels of charge, the plug types, along with the network name and address.
2. From this page, they can unsave any chargers they want or click on the charger to navigate to its location on the map screen.
3. The user can also filter the results based on the charging speed and if there is a cost to use it.
4. To leave this page, the user may hit the back button on the top left of the page.

11. Conclusion

This document covered all the technical and non-technical aspects of the document. It began by giving an overview of the problem trying to be solved by ENRoute and the circumstances around the development of the app. The document then covers the interfaces of the app, including hardware, software, and user. The document then discusses all the application’s features. This includes both functional and nonfunctional

requirements along with the feasibility and priority of each feature. Following that, are the dependencies used by ENRoute along with the hardware requirements to run. The document then covers the instructions on how to install and operate the app.

During this project, the students learned how to use the Flutter framework along with implementing various APIs and packages. This means the general structure of Flutter user interfaces, along with how to implement Google Firebase as a database management system, Google Maps as an application function, and the overall object oriented programing of dart. Students also learned how to communicate appropriately with a client in a professional manner and interpret their software needs. Additionally, the team learned how to deal with the ongoing issues of software development such as coming upon feasibility issues with finding and implementing certain charging station data along with having to deploy the app on iOS. An idea that was brought to the team by the client was that it is best to fail early in the project rather than later. The client was referring to the implementation of the app in IOS and just as a general lesson of software development. Overall, the team also refined their skills in communicating ideas and implementing software together as a group.

The primary thing that the team would do differently is start earlier. Part of this may include changing the due date of the initial individual SRS to an earlier date in order to move up the design process and allow the development to start earlier. The development for iOS should have also started at a much earlier stage, as the pitfalls to overcome for this aspect of the project were greater than the students initially realized. The team also realized that much of the designing and research was dependent on the actual development process as packages and API’s changed as it was made clear that not everything worked as expected. This would also allow the possibility of more testing and increased features.

Appendix A: User Interface Design

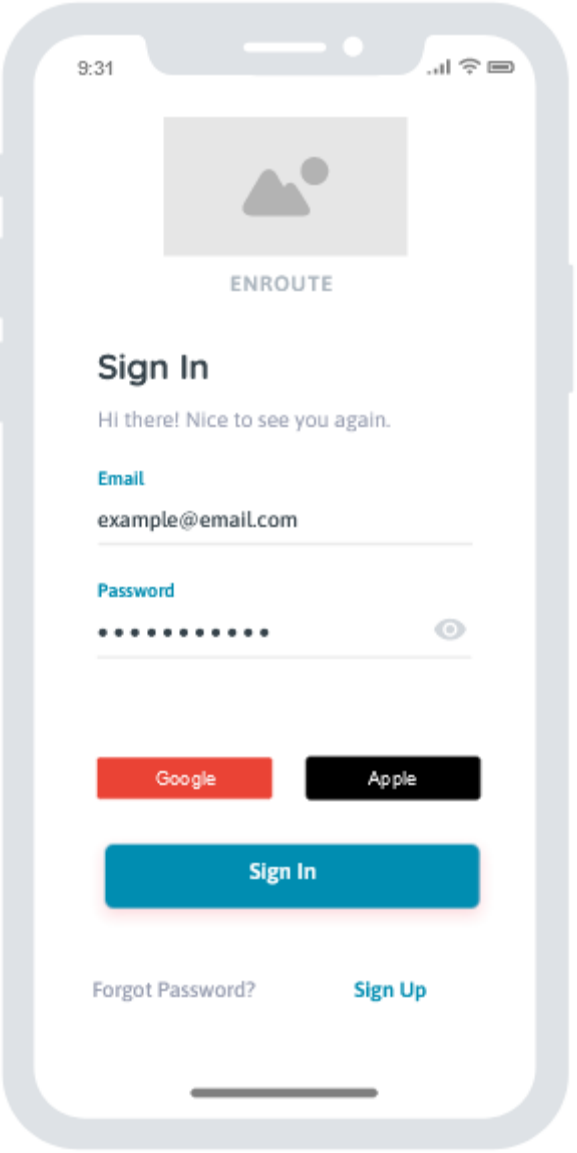

Figure 1: Sign-in page Mockup

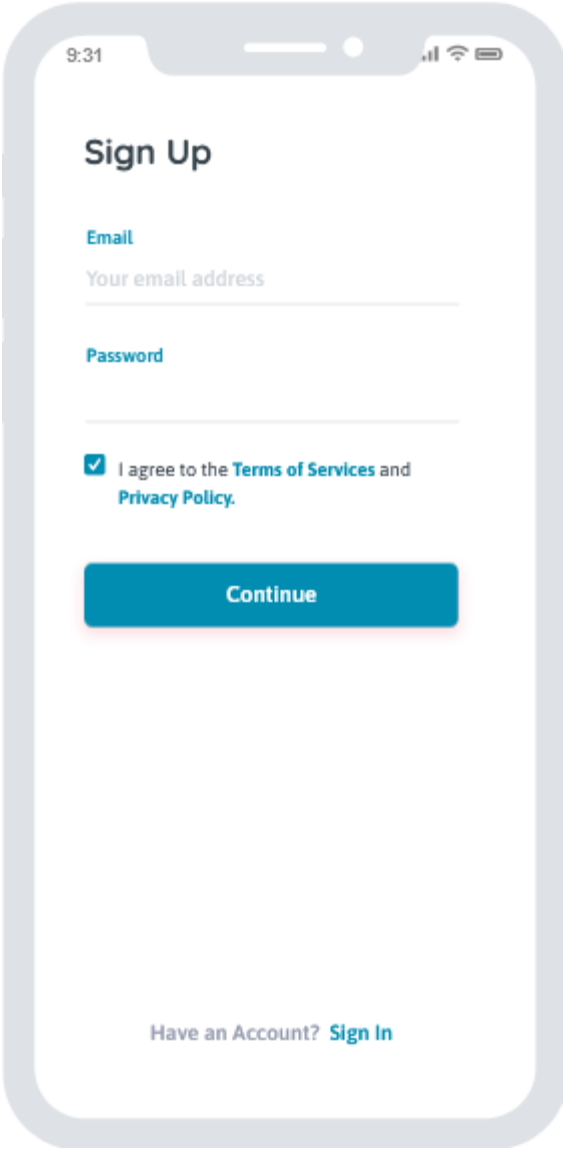

Figure 2: Sign-up page Mockup

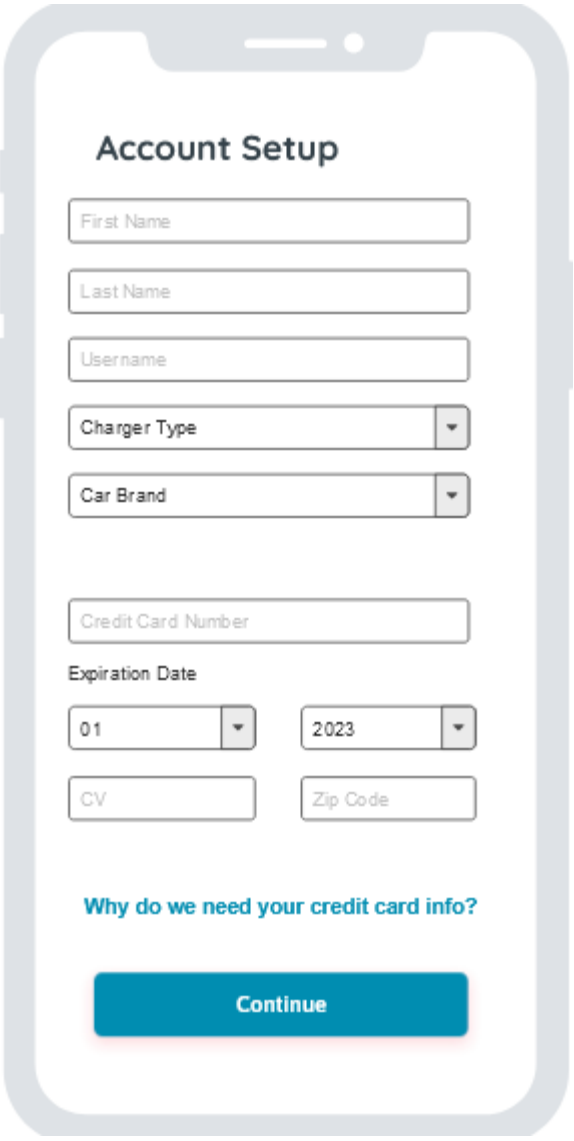

Figure 3: Account Setup Mockup

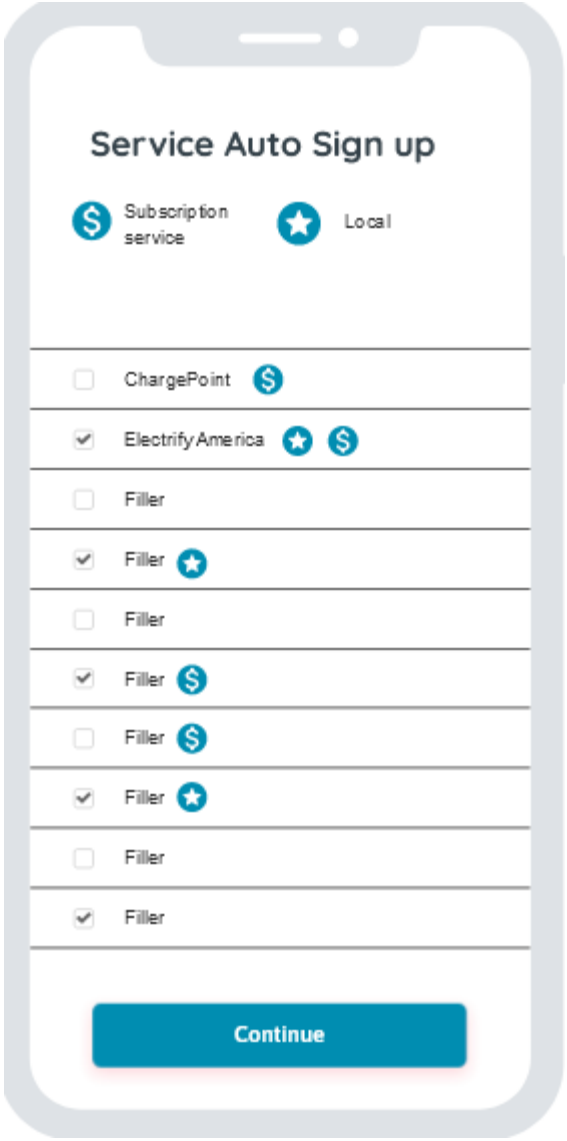

Figure 4: Services Sign-up Mockup

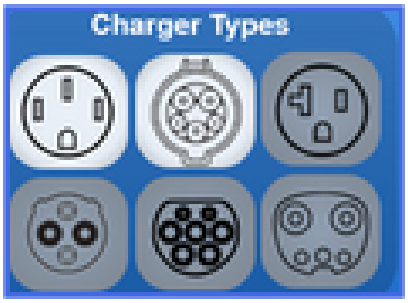

Figure 5: Charger Selection Mockup

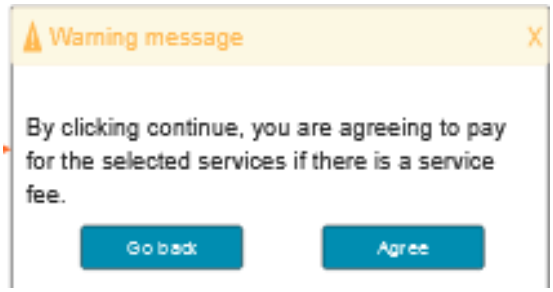

Figure 6: Service Fee Warning Mockup

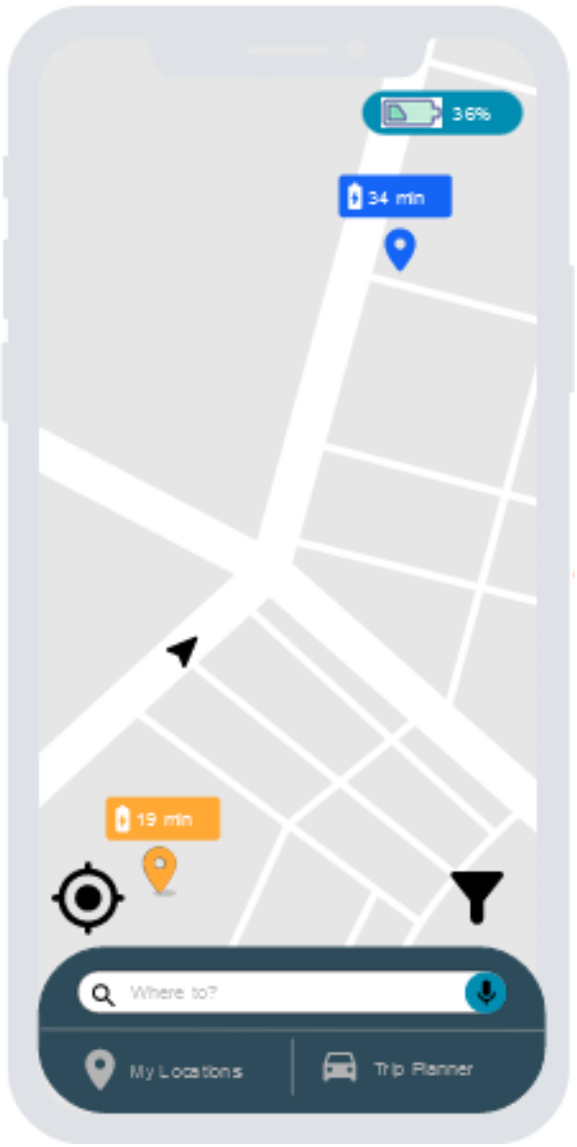

Figure 7: Navigation Mockup

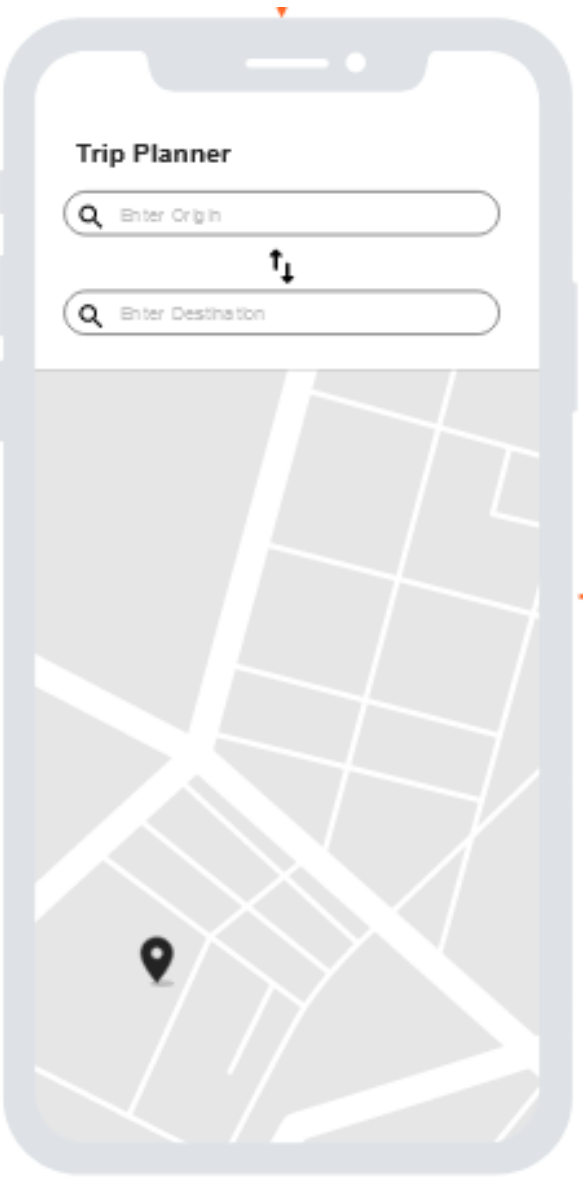

Figure 8: Trip Planner Mockup

Figure 9: Recent Chargers Mockup

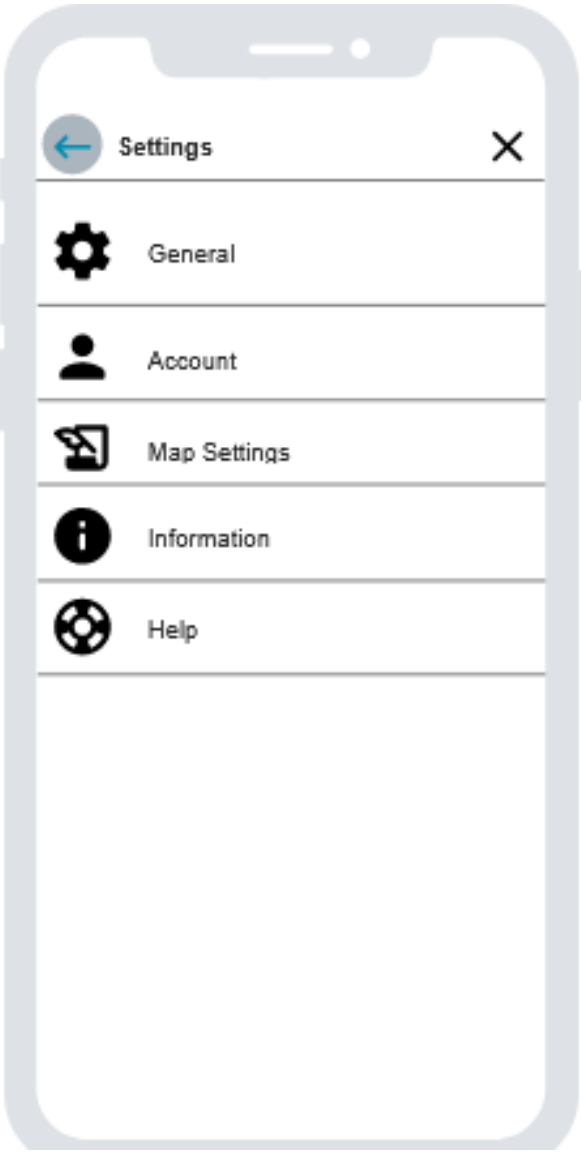

Figure 10: Settings Mockup

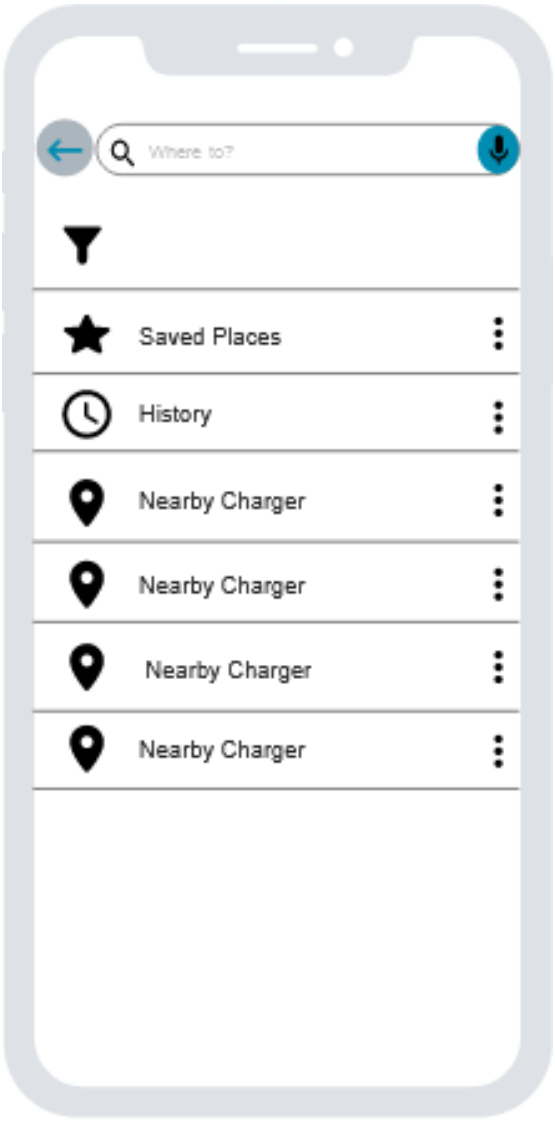

Figure 11: Saved Chargers Mockup

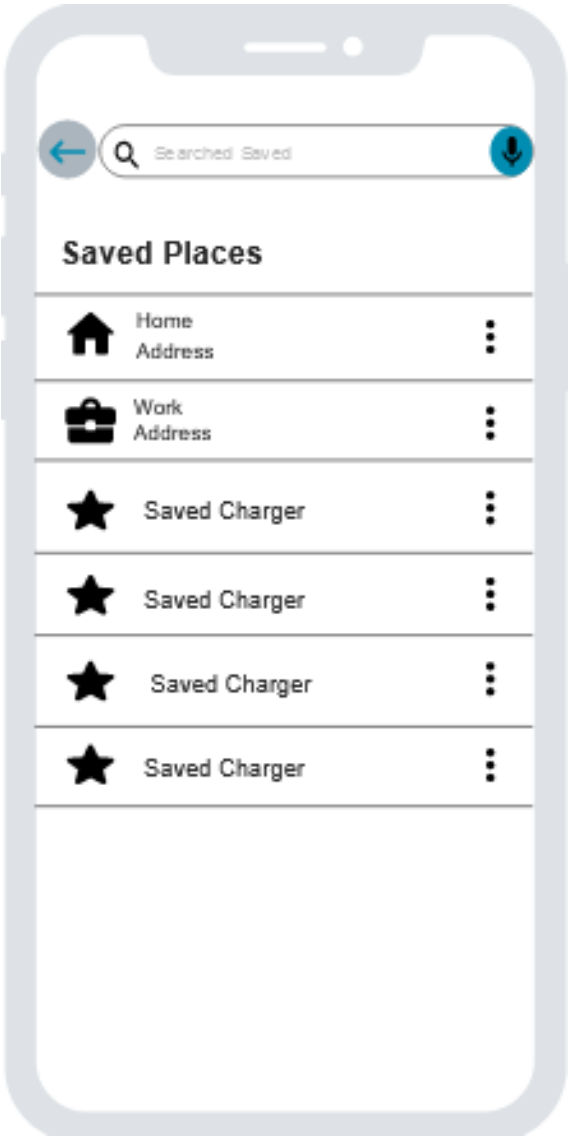

Figure 12: Saved Places Mockup

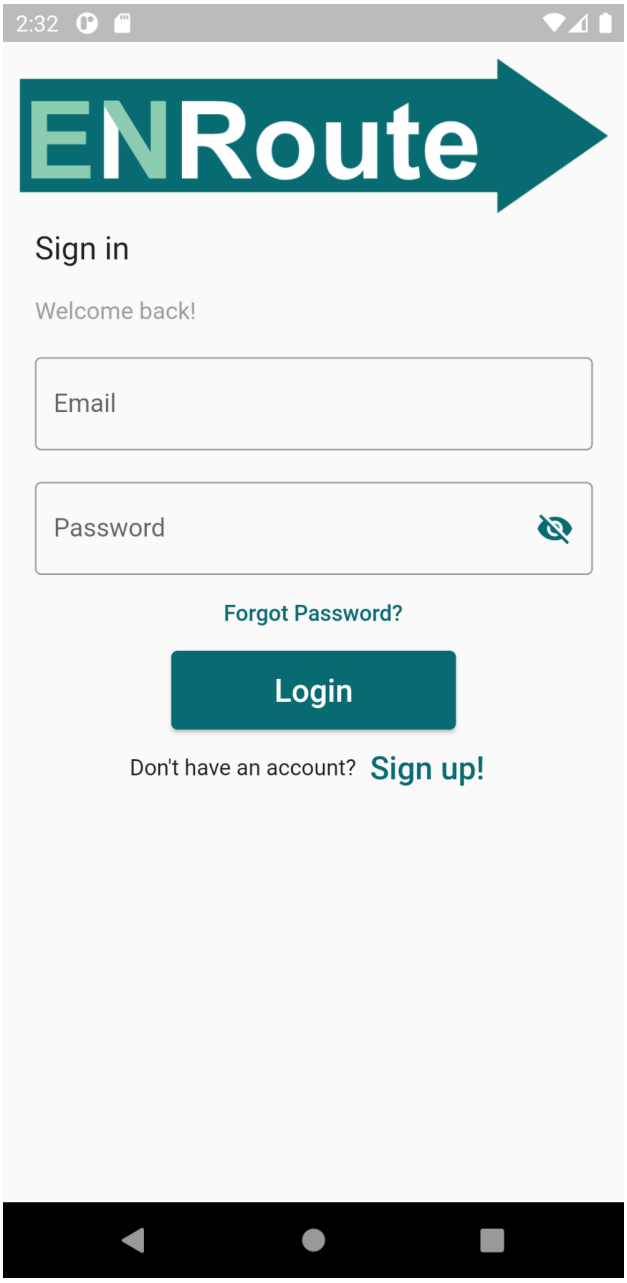

Figure 13: Launch Screen

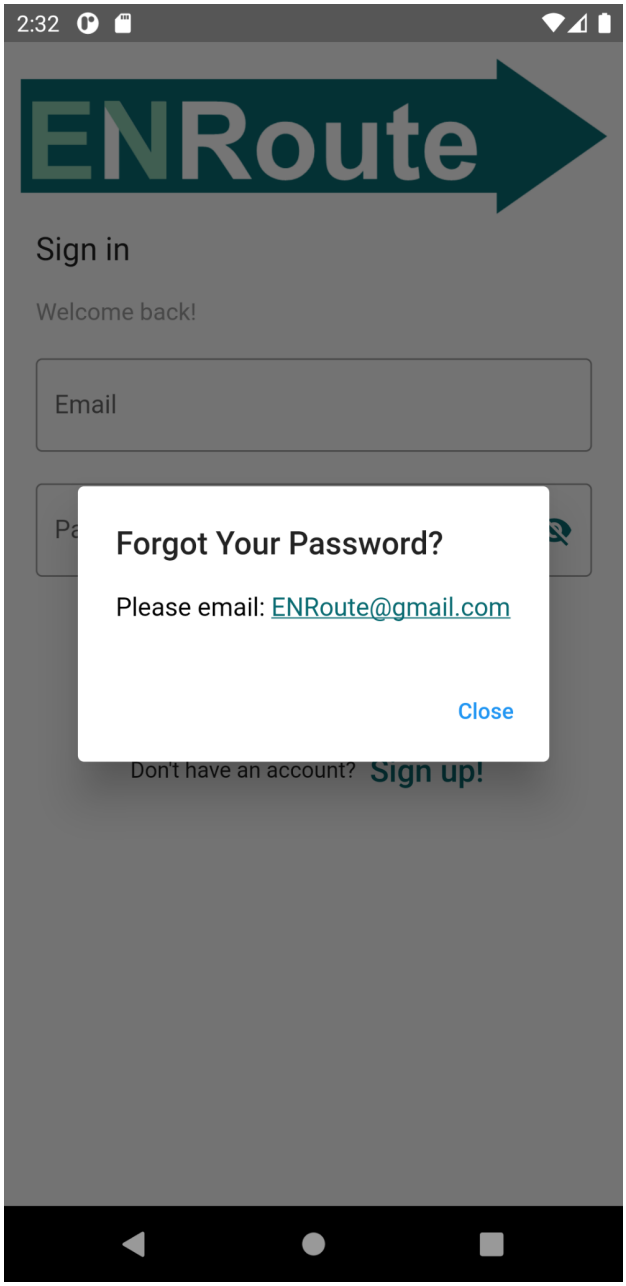

Figure 14: Forgot Password

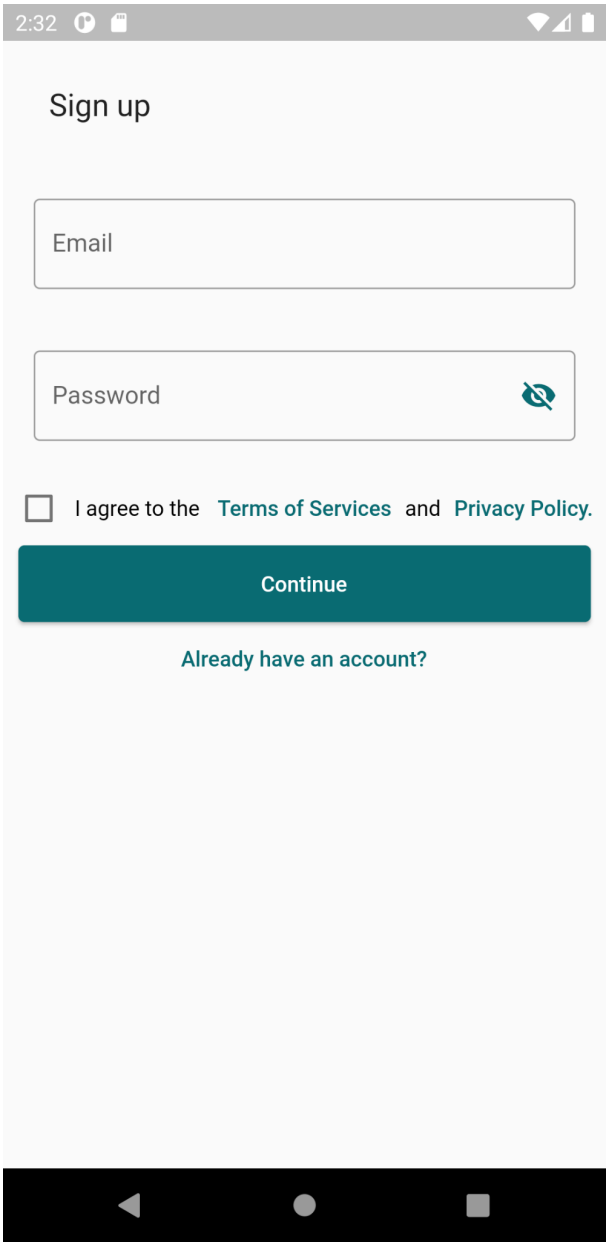

Figure 15: Sign-up Screen

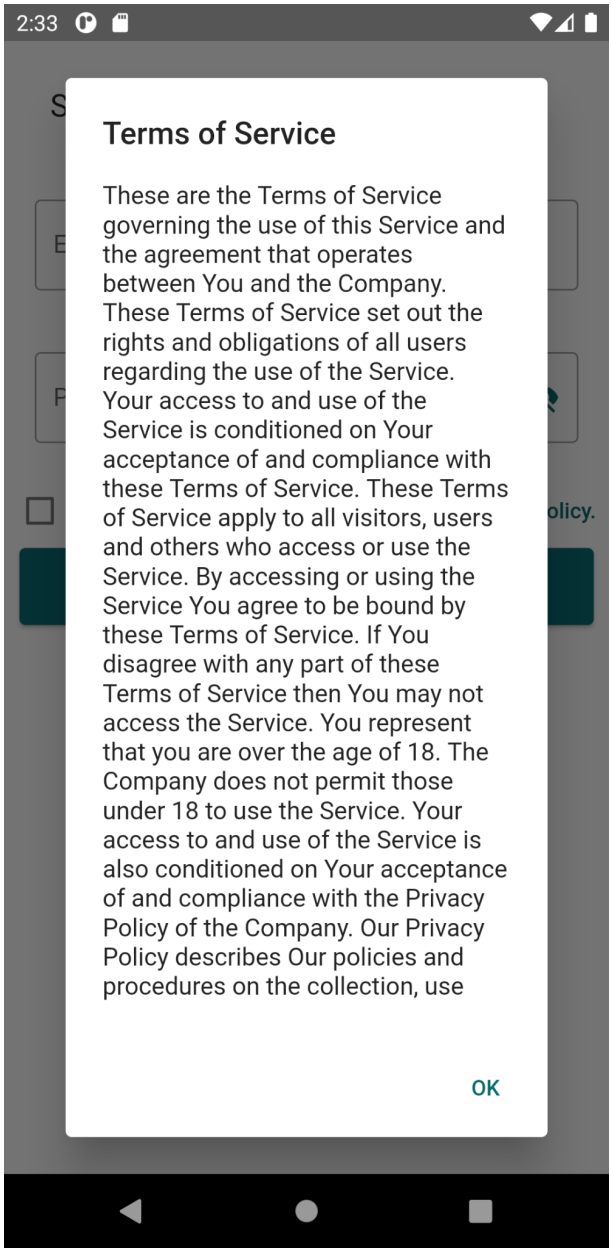

Figure 16: Terms of Conditions

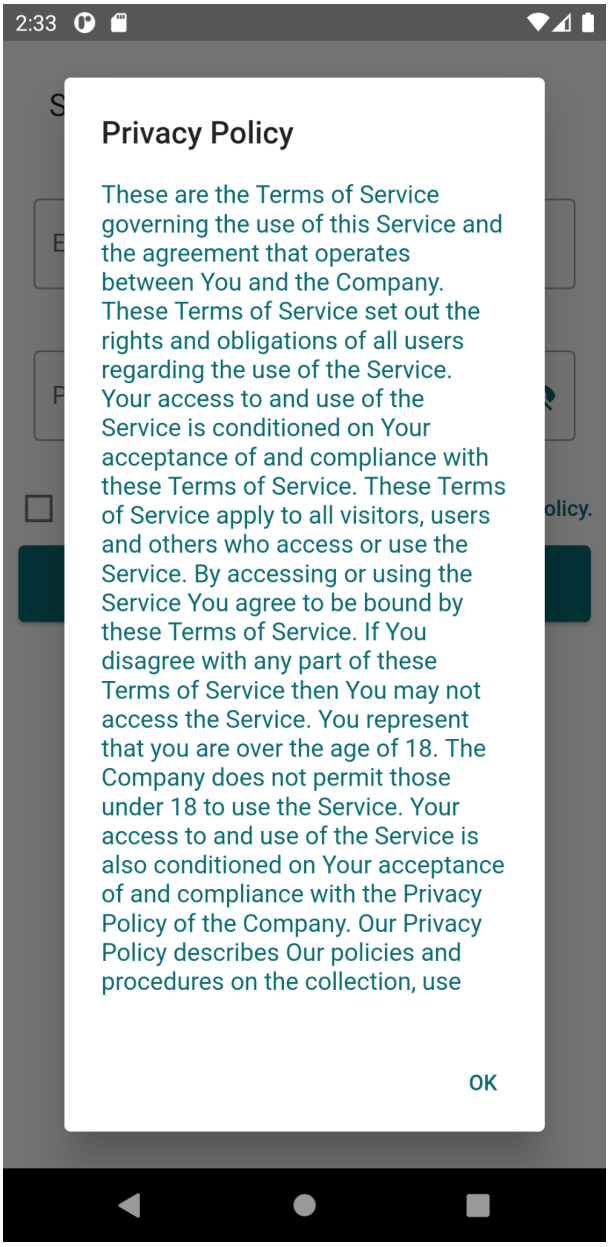

Figure 17: Privacy Policy

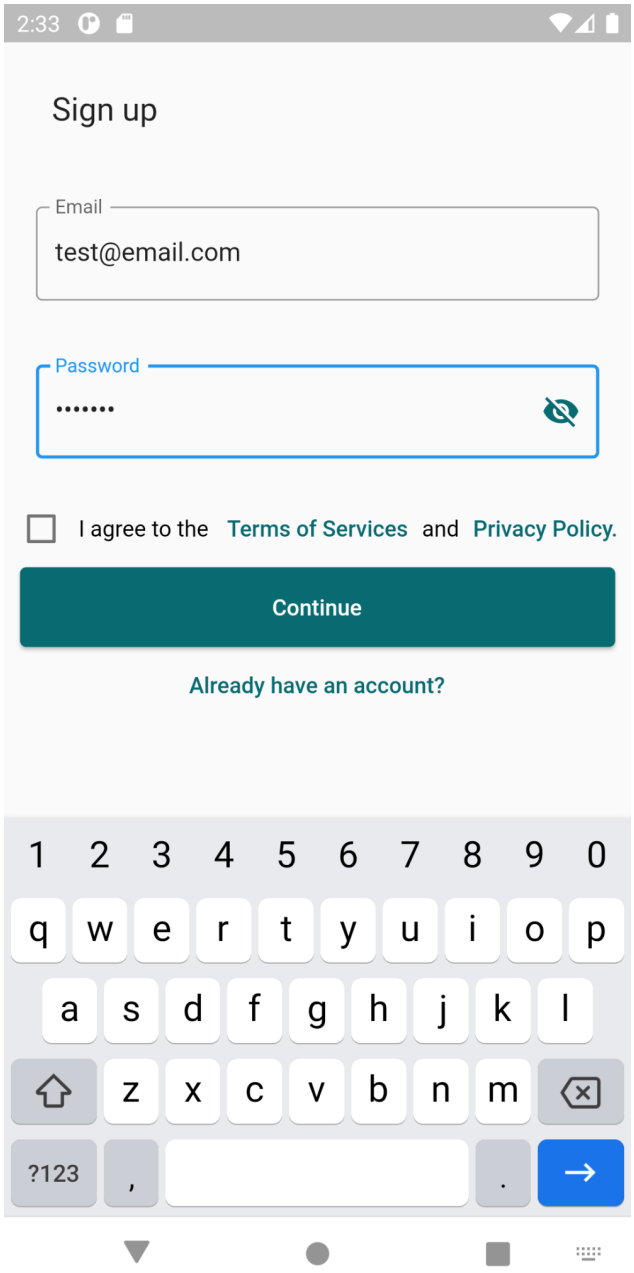

Figure 18: Fill out email and Password

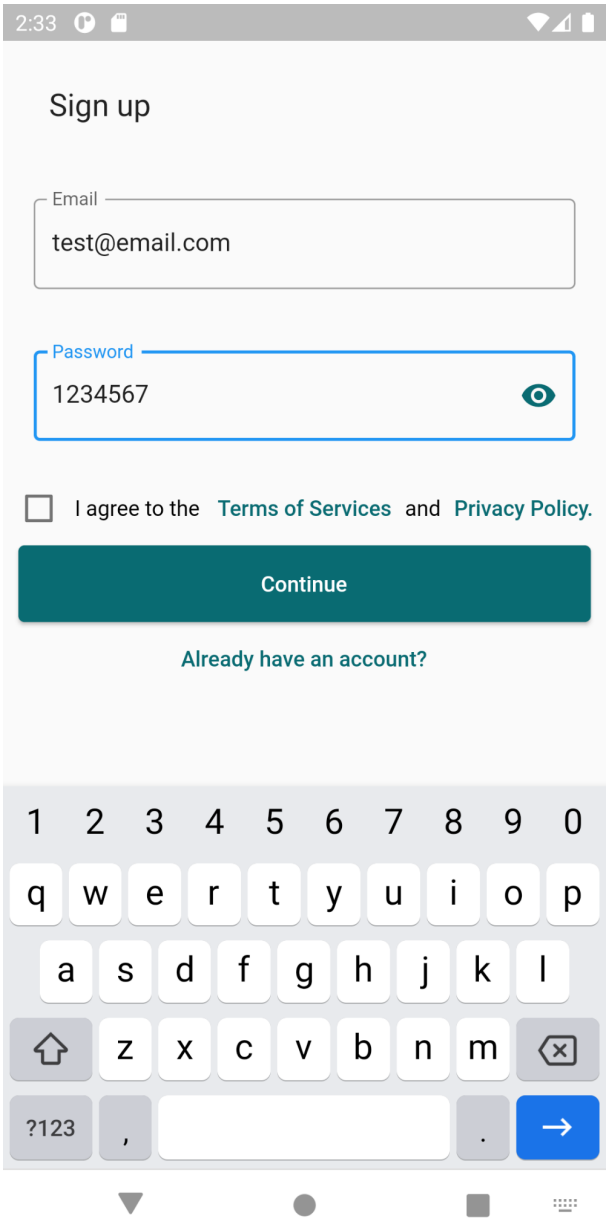

Figure 19: View Password

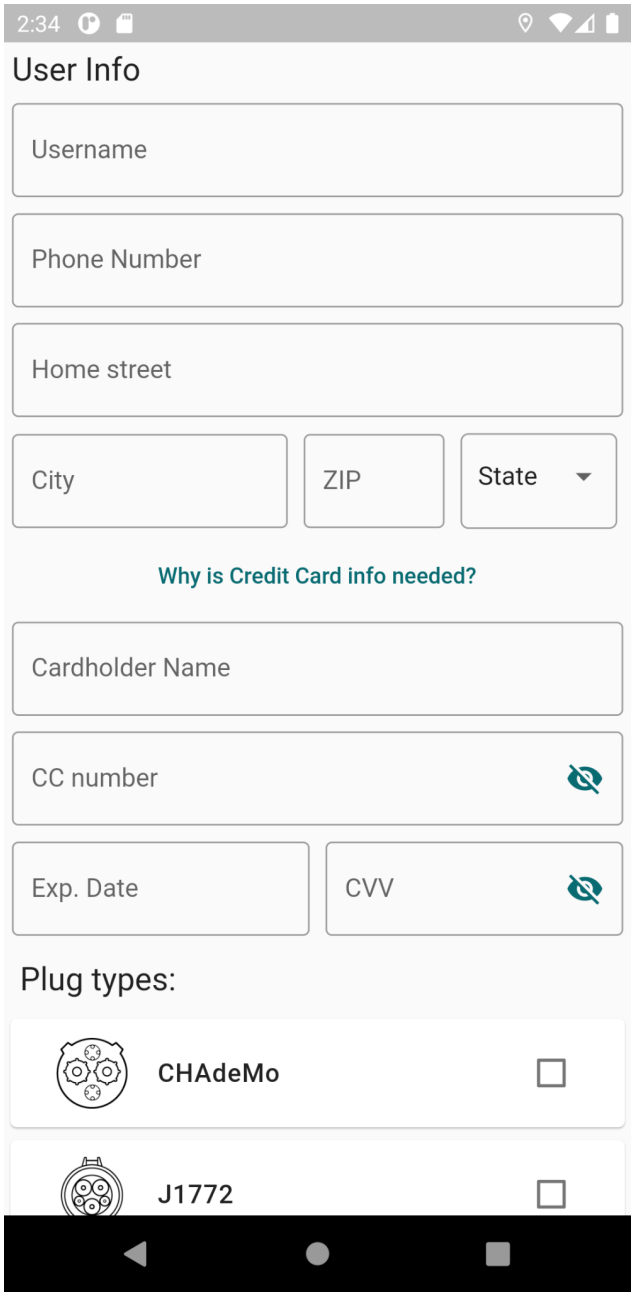

Figure 20: Top of Add PID

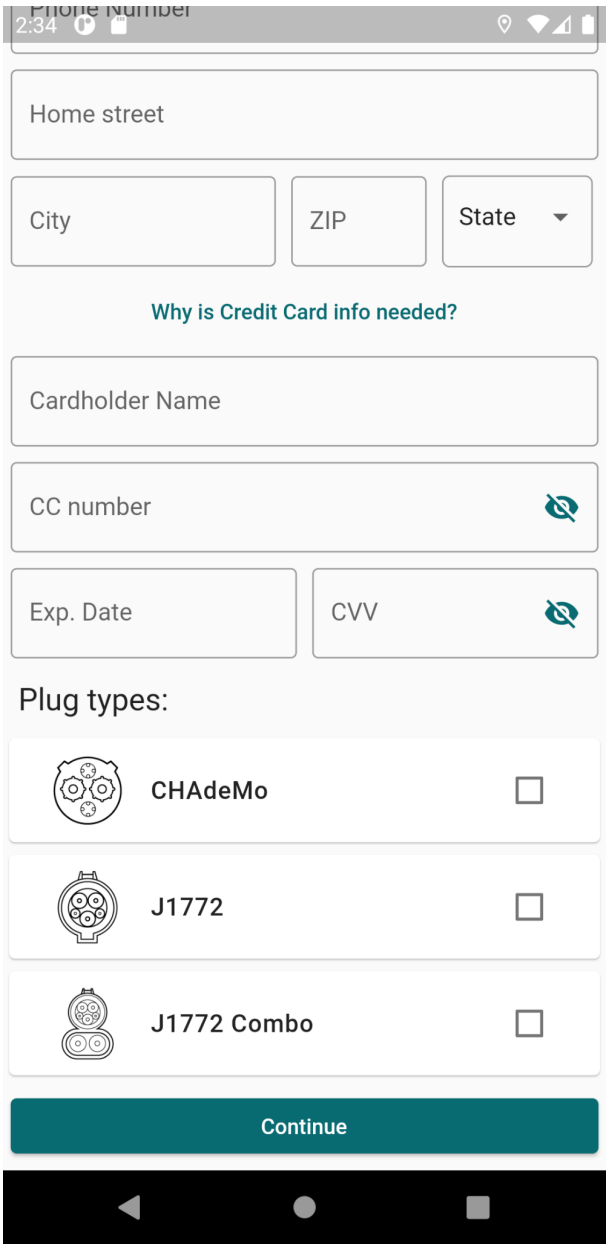

Figure 21: Bottom of Add PID

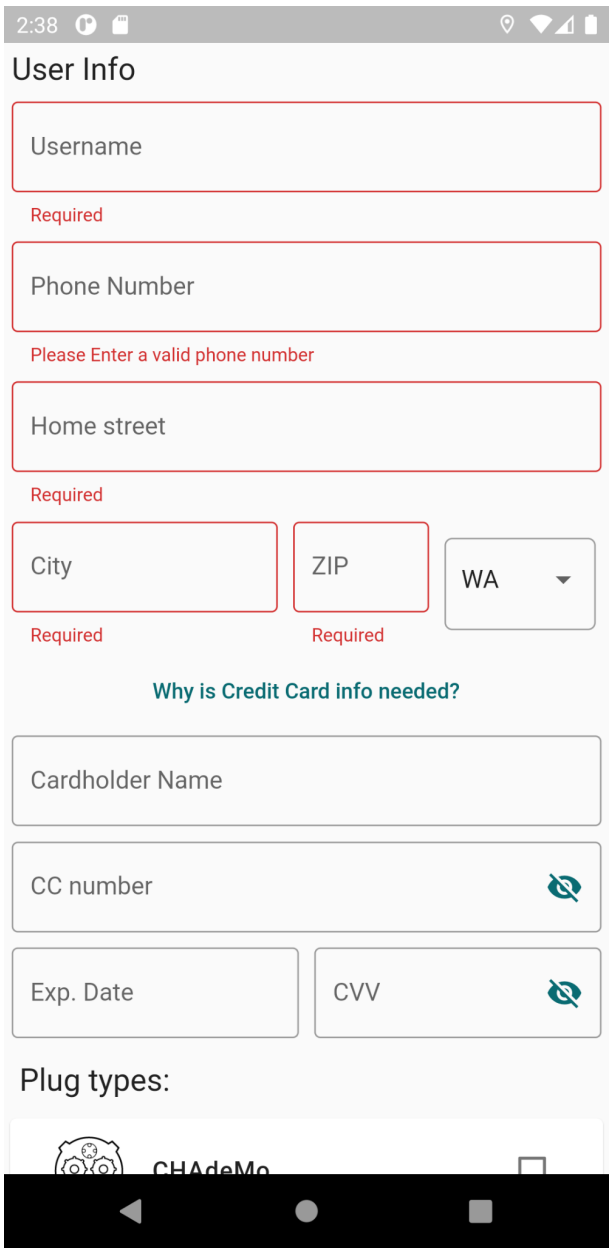

Figure 22: Required Fields

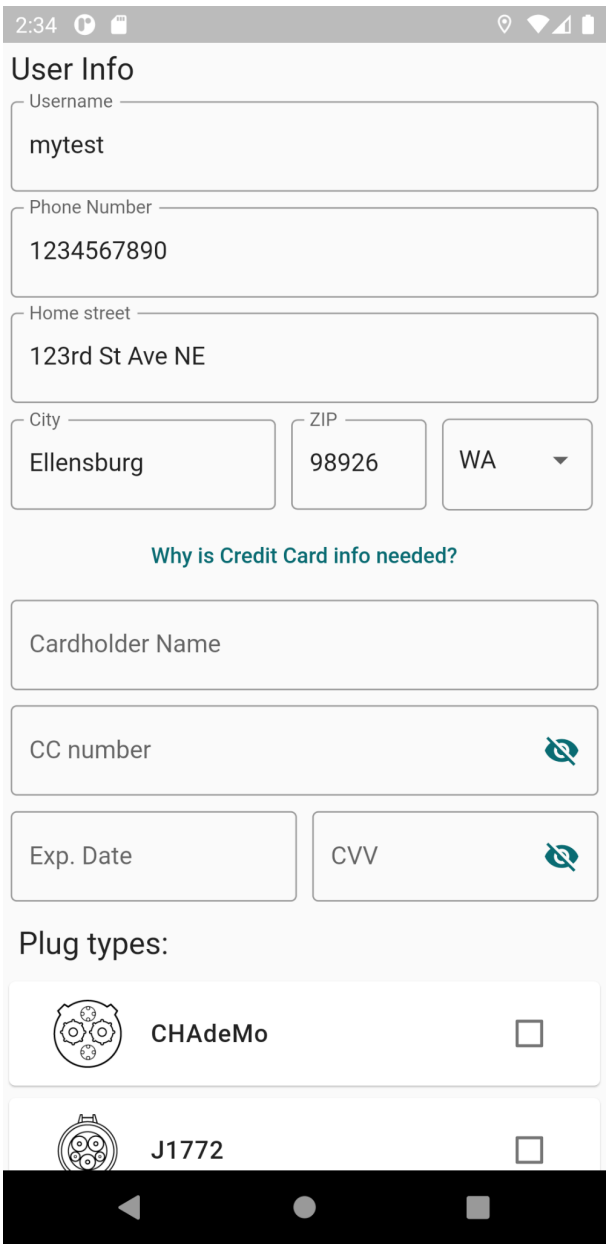

Figure 23: Adding in Necessary PID

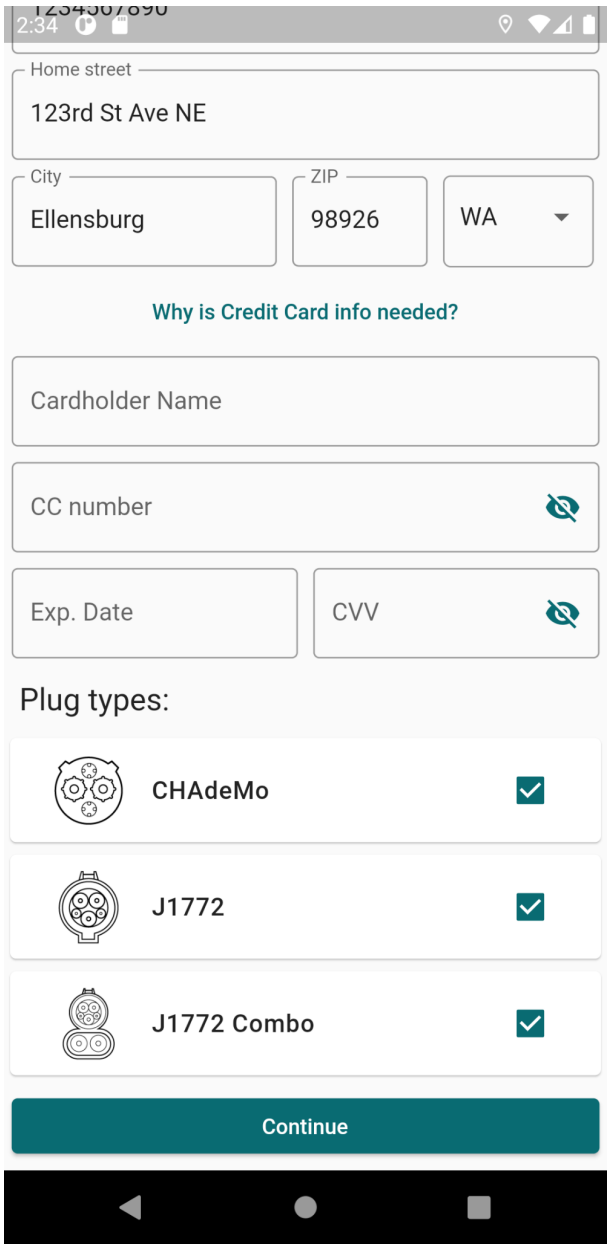

Figure 24: Adding Plug Types

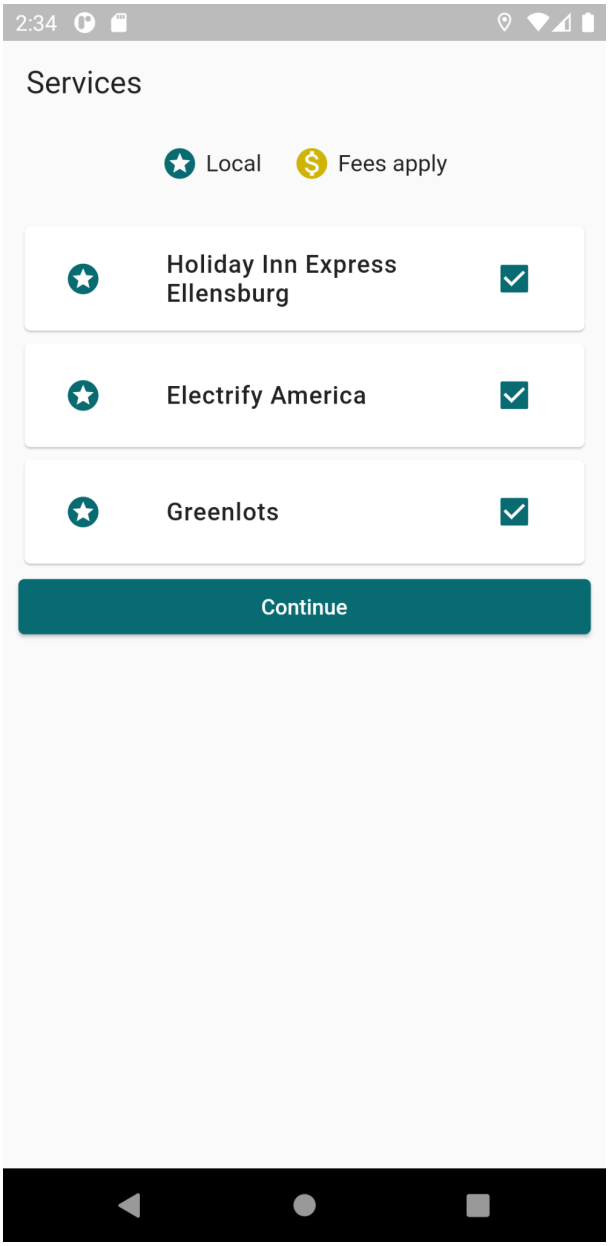

Figure 25: Local Services based on Zip Code

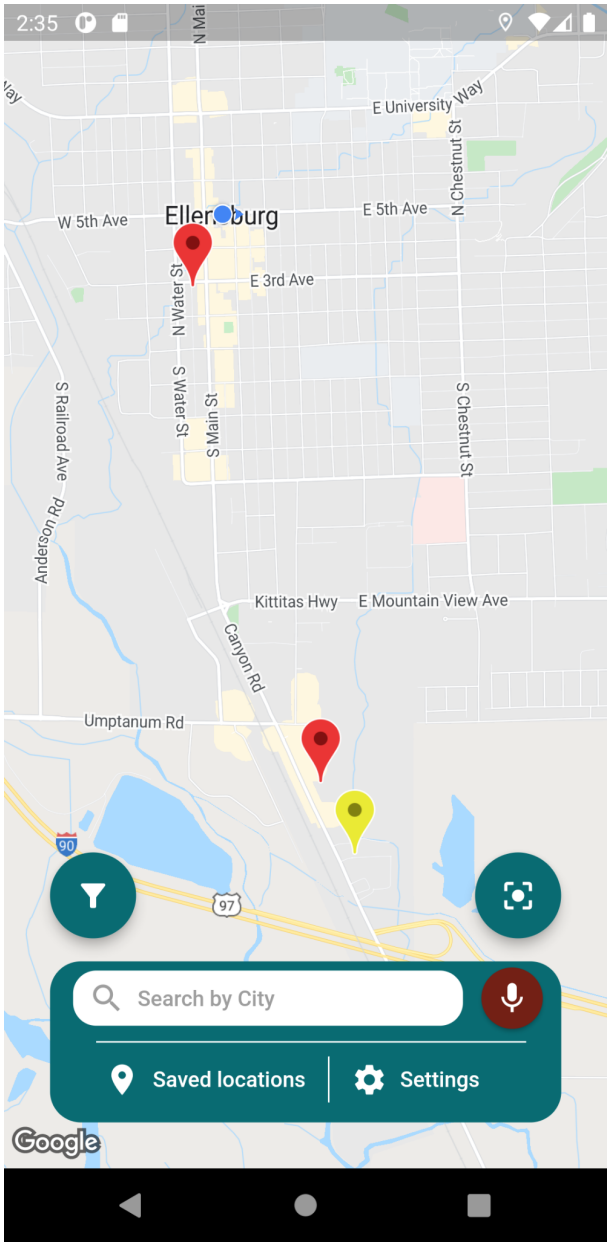

Figure 26: Map Screen With Pins for Chargers

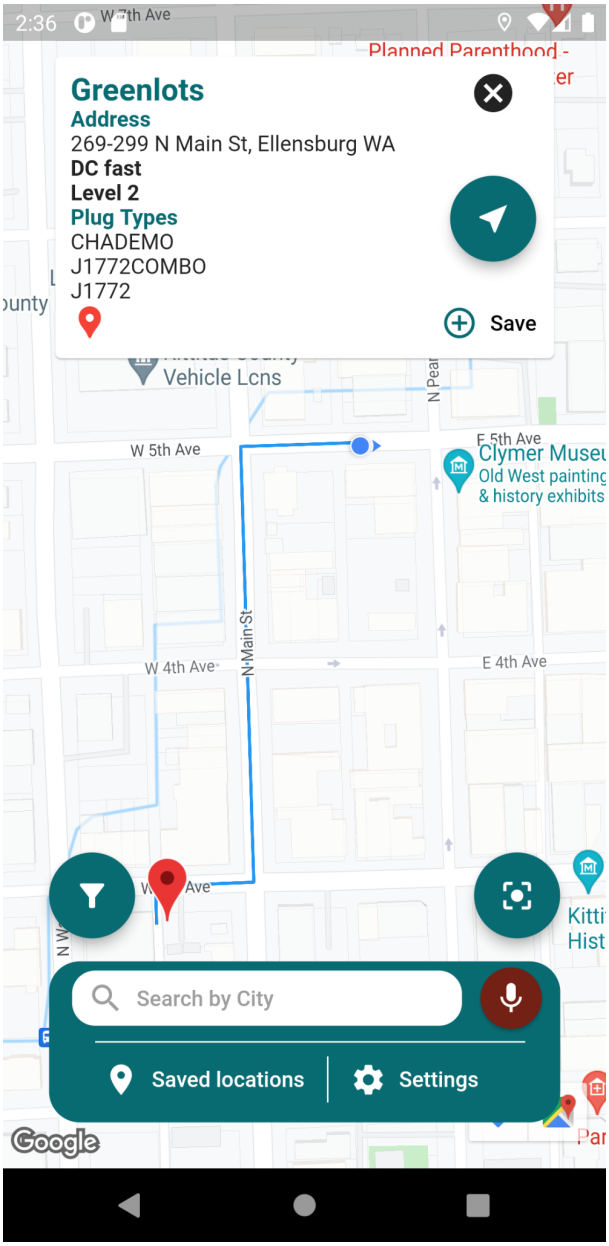

Figure 27: Path to a Charger and the Charger's Info Card

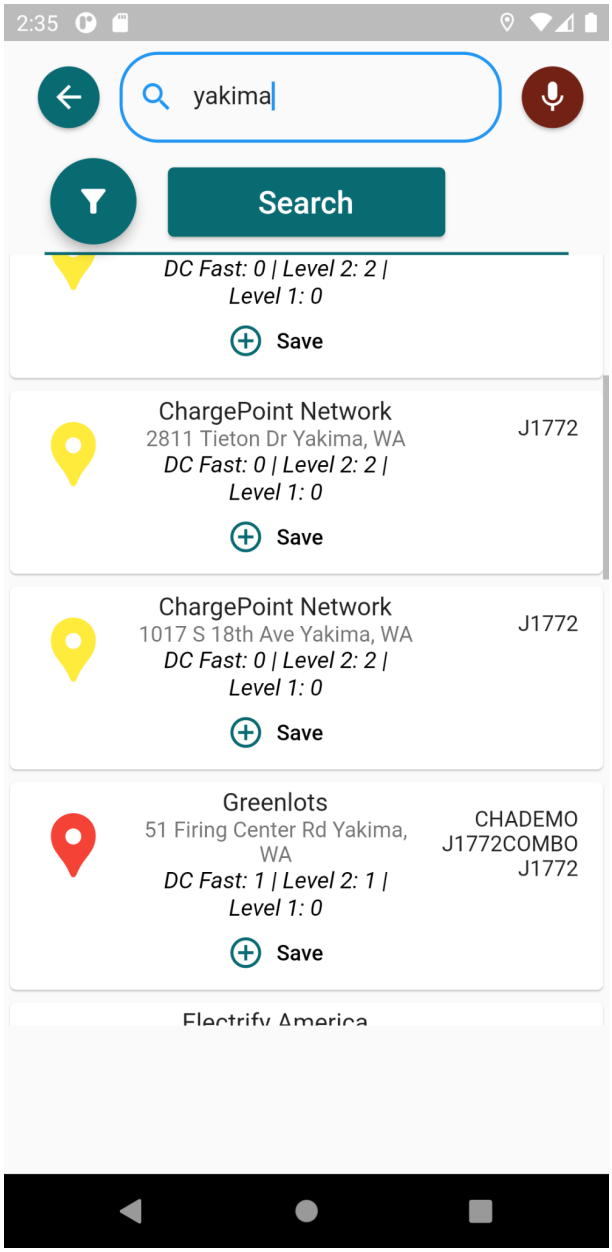

Figure 28: Searching for Chargers in Yakima

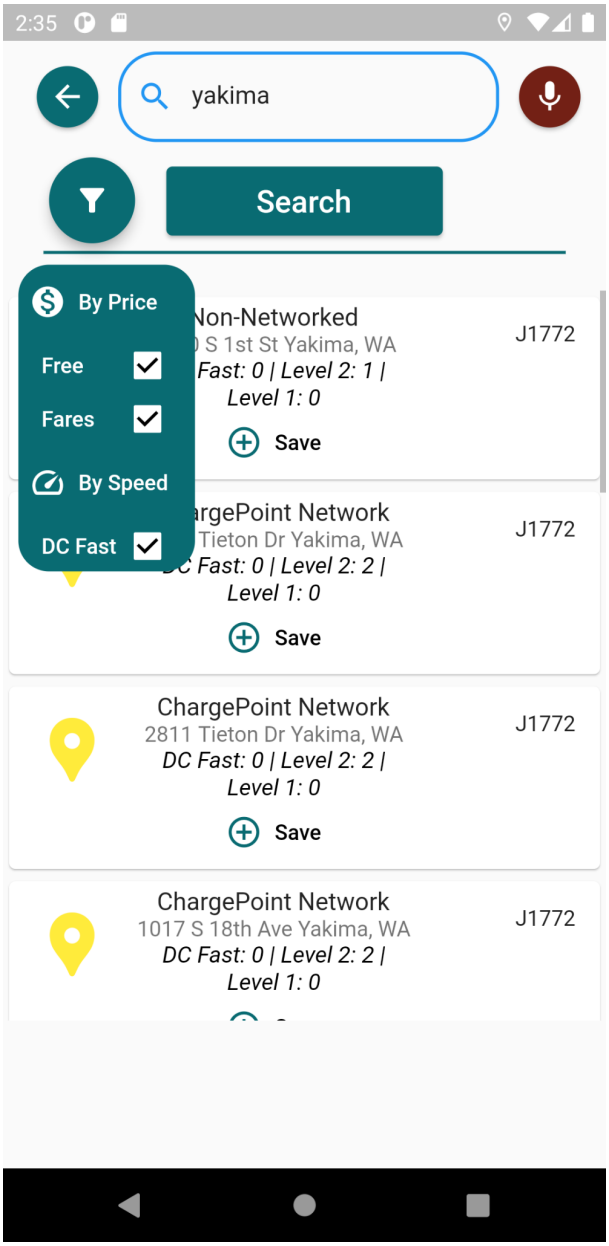

Figure 29: Filter Menu

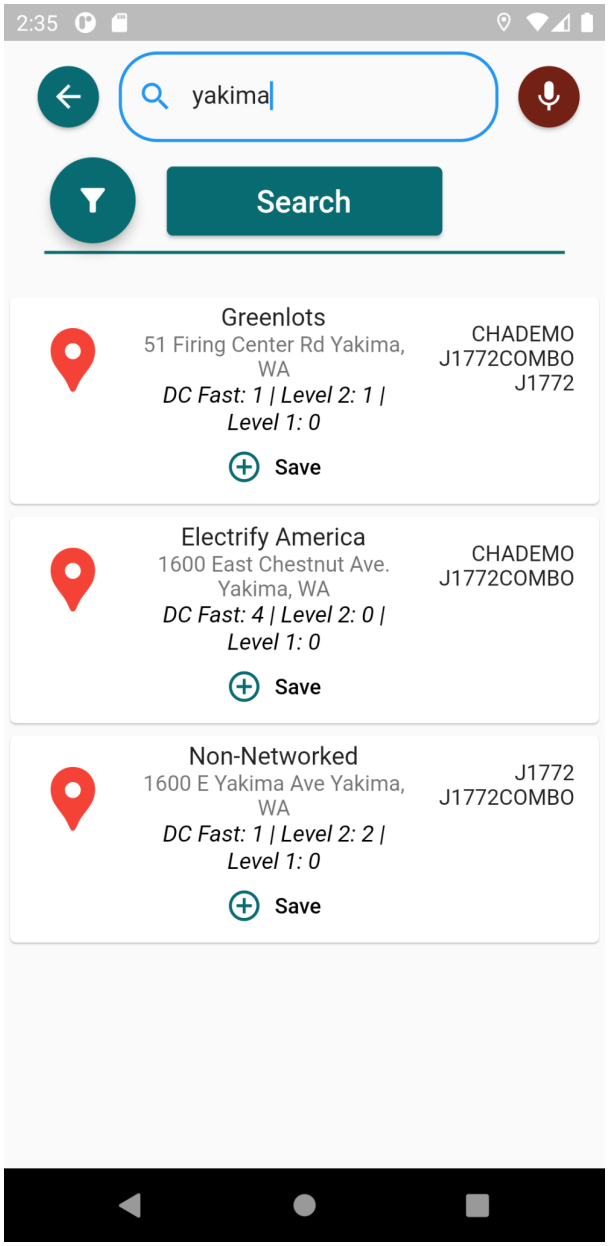

Figure 30: Filtering for SuperChargers in Yakima

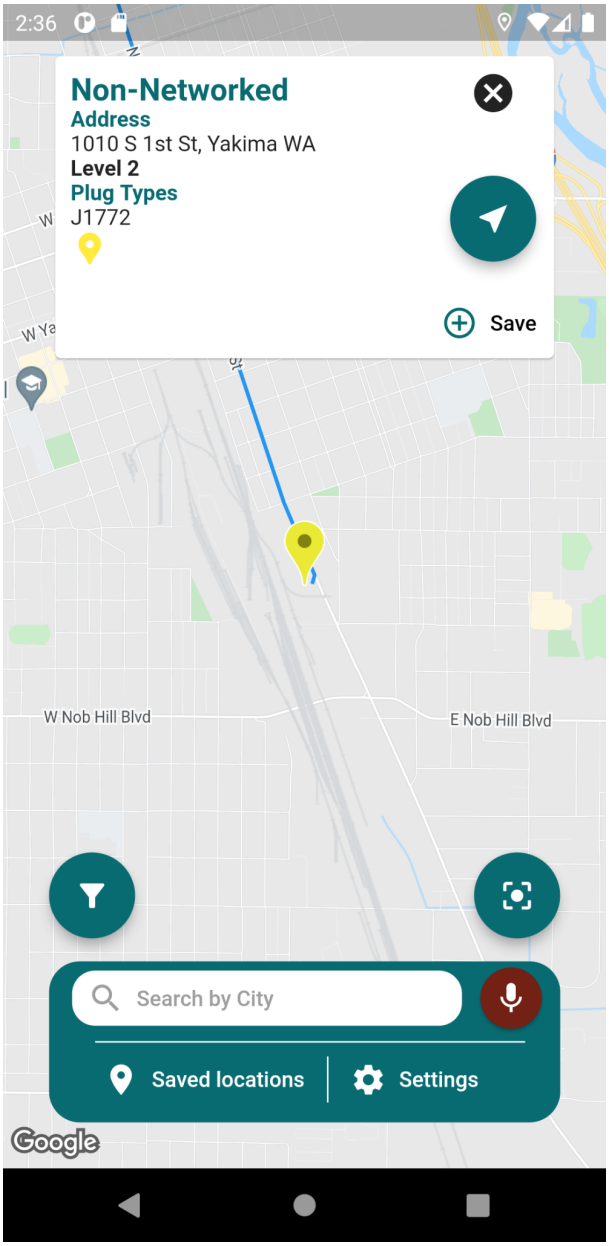

Figure 31: Charger in Yakima

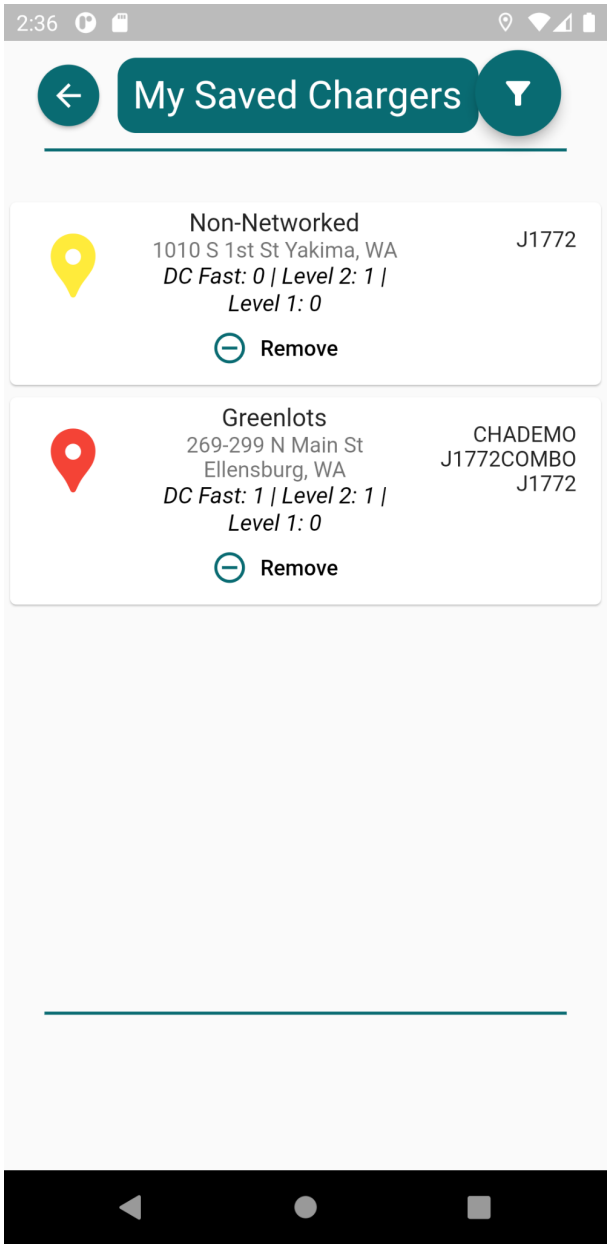

Figure 32: Saved Chargers

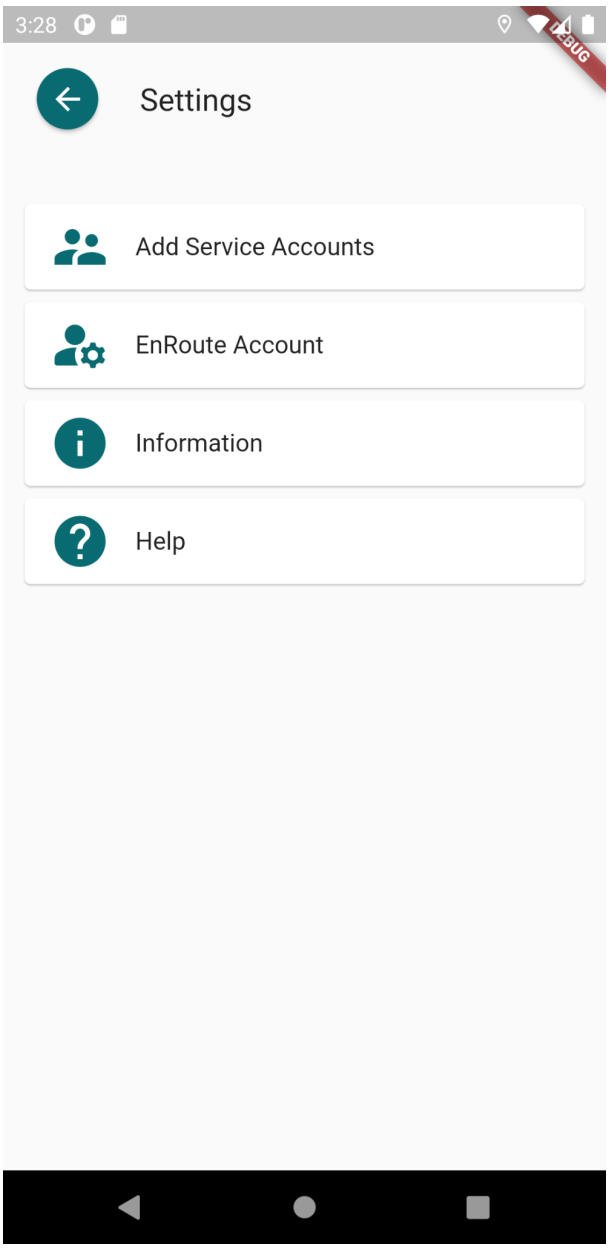

Figure 33: Settings Page

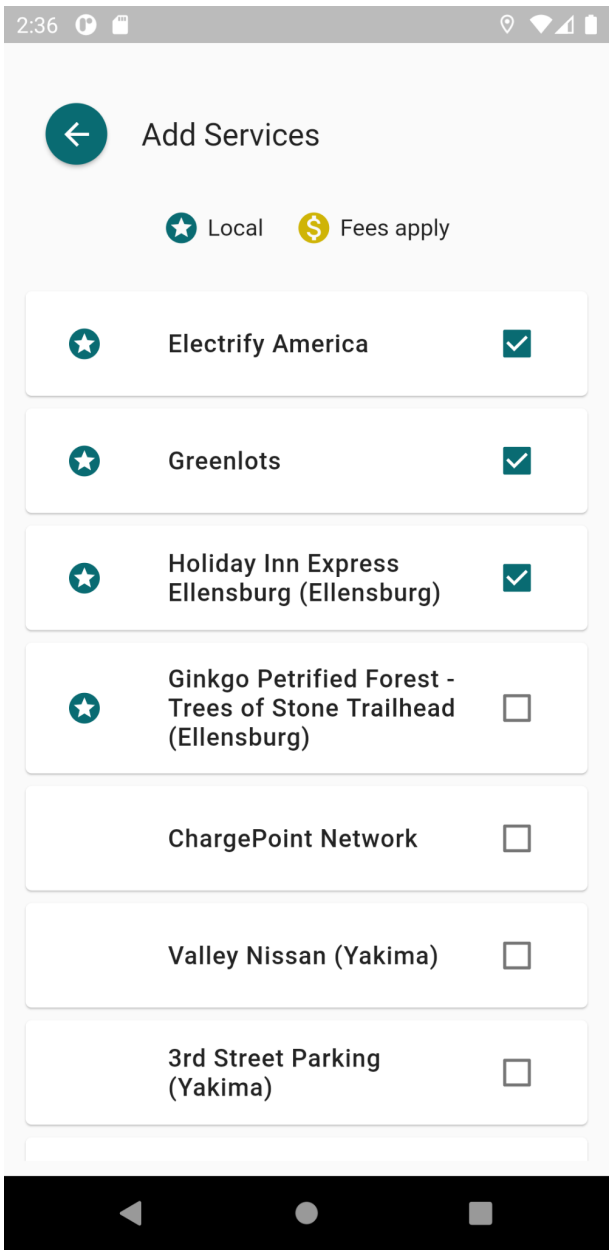

Figure 34: Add Services to Account

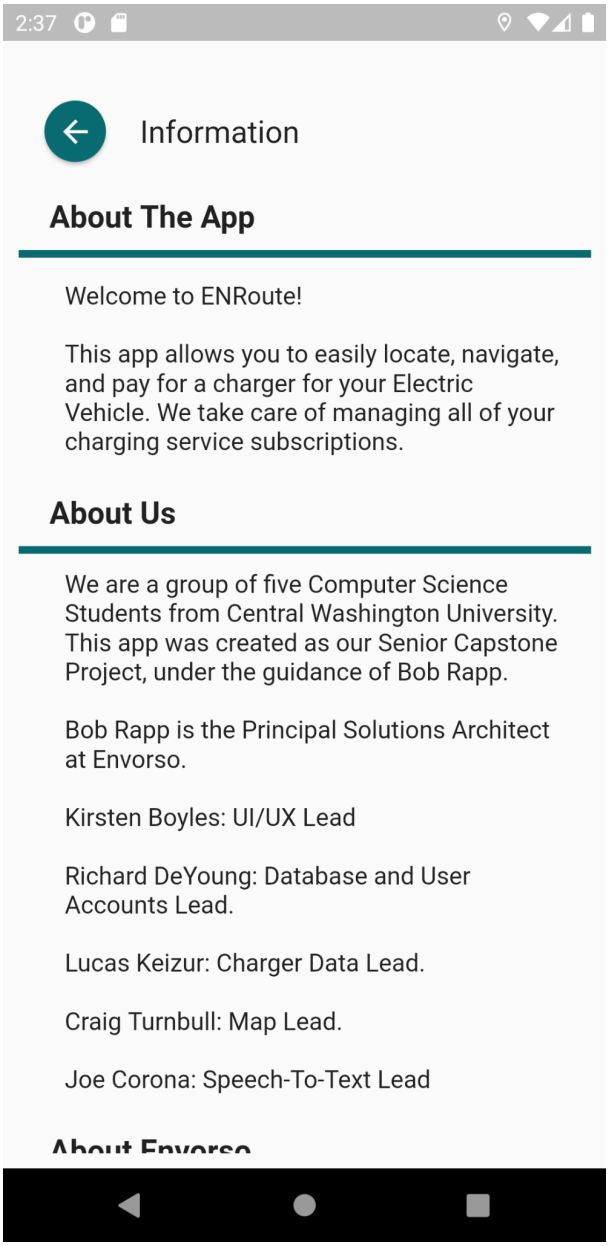

Figure 35: Information Page

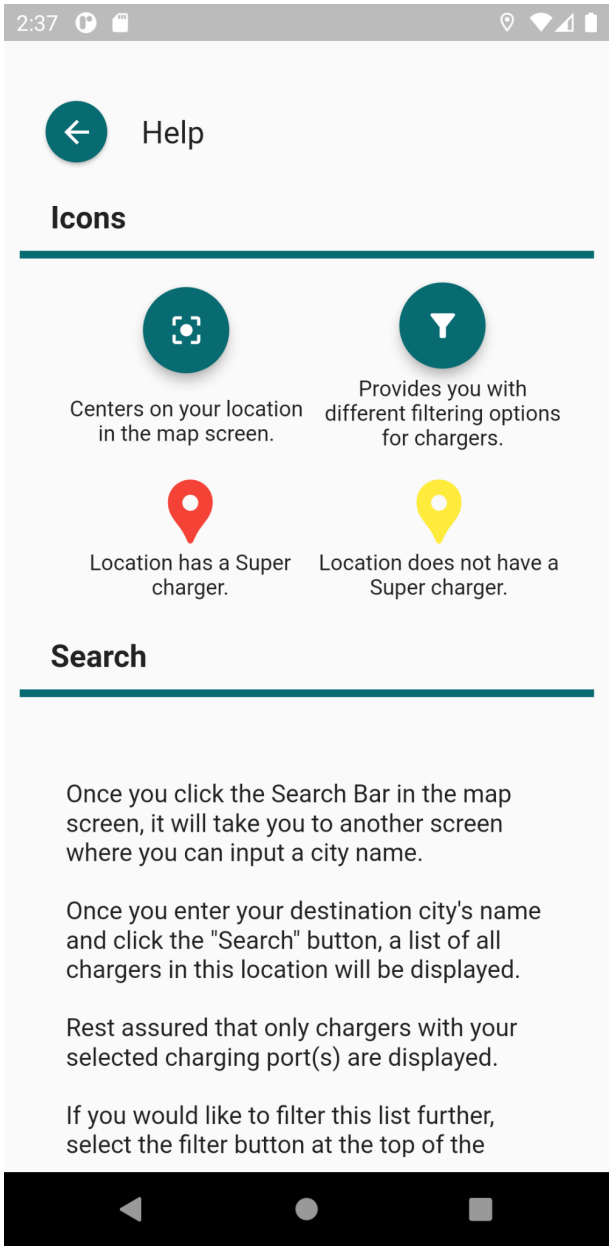

Figure 36: Help Page

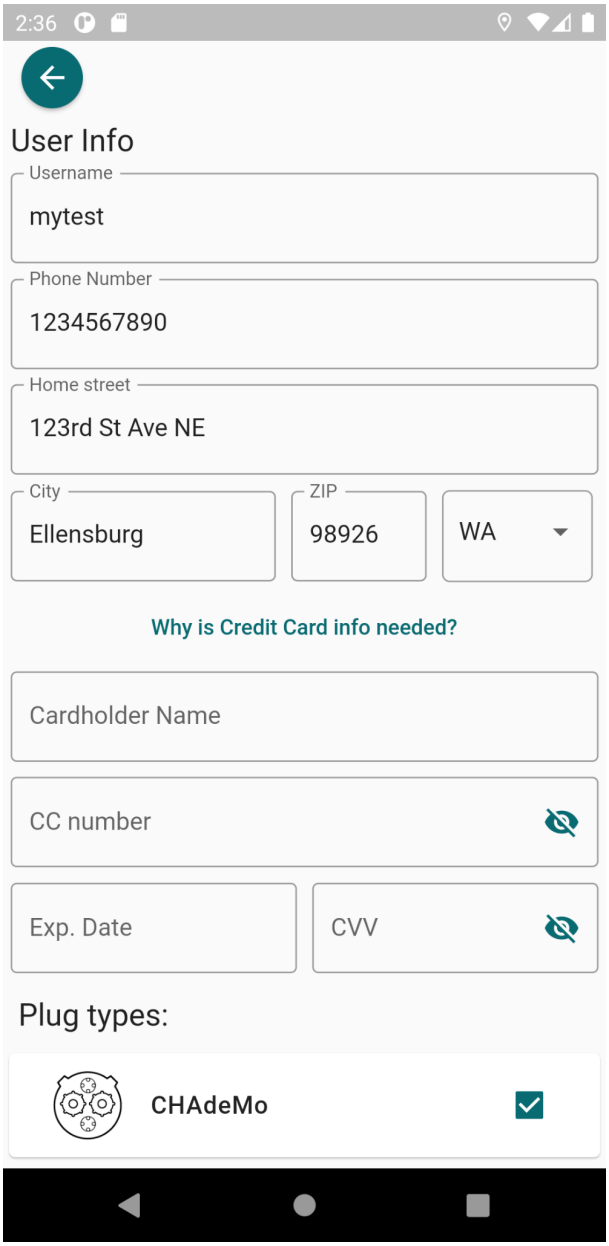

Figure 37: Update PID

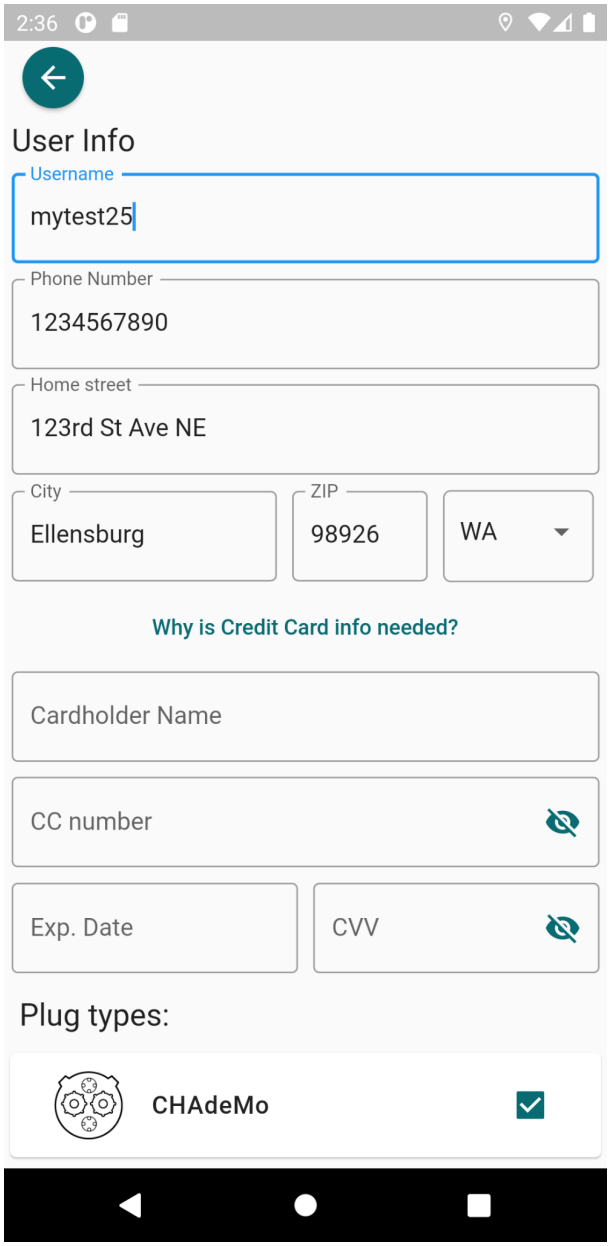

Figure 38: Changing User Name

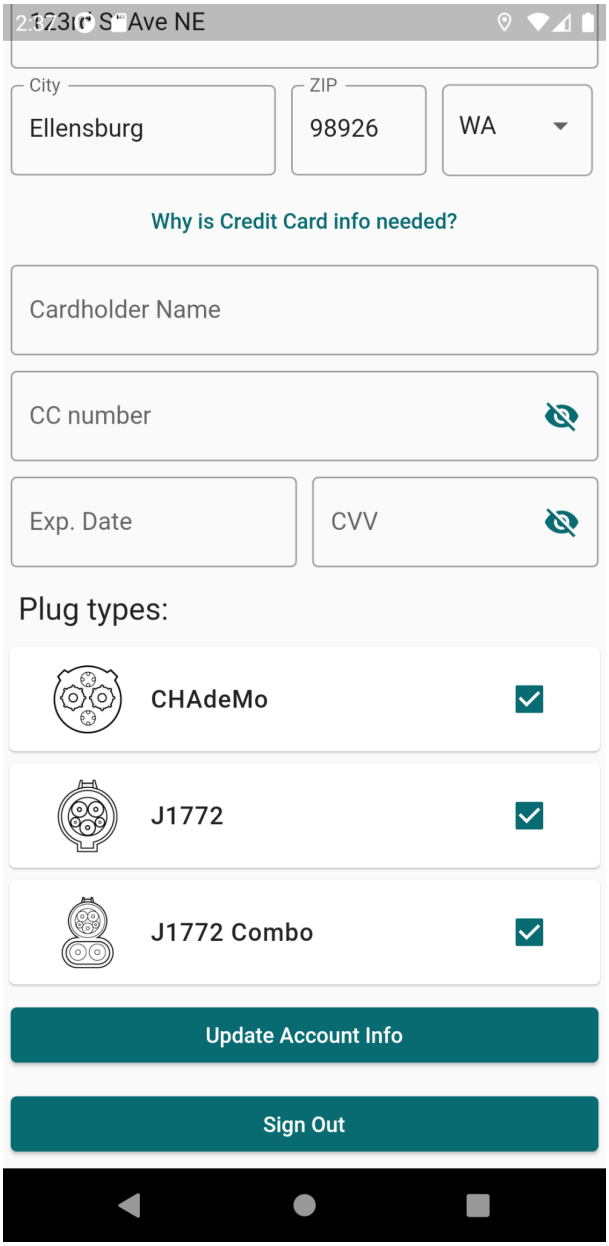

Figure 39: Bottom of Update PID Page

Appendix B: High Level Design

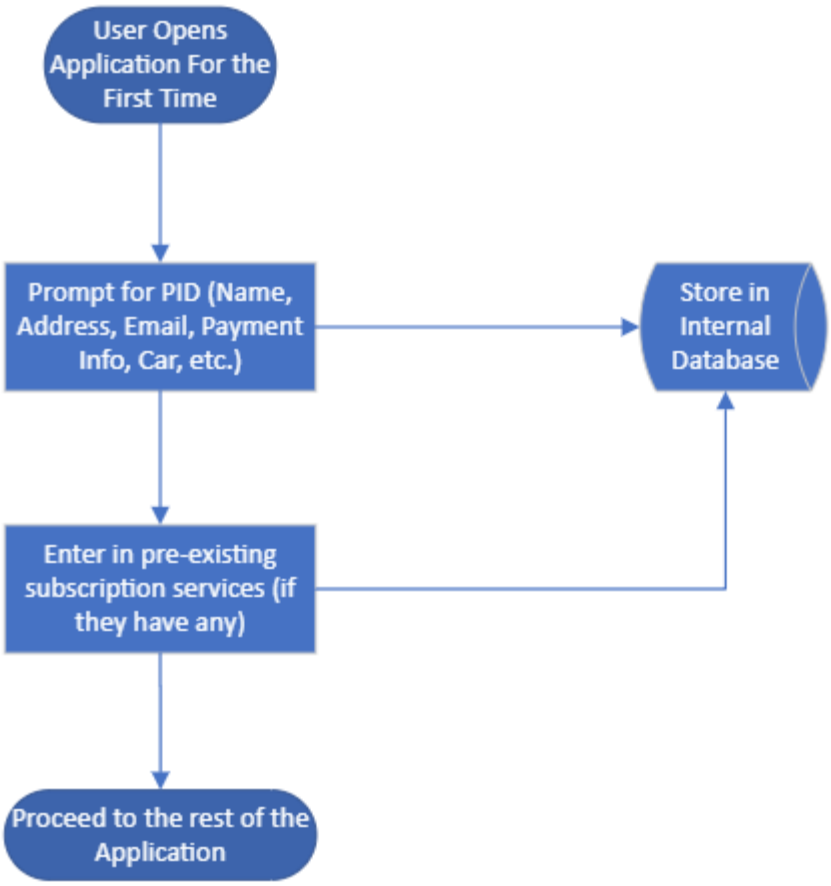

Figure 40: Flow Chart of Sign-up Process

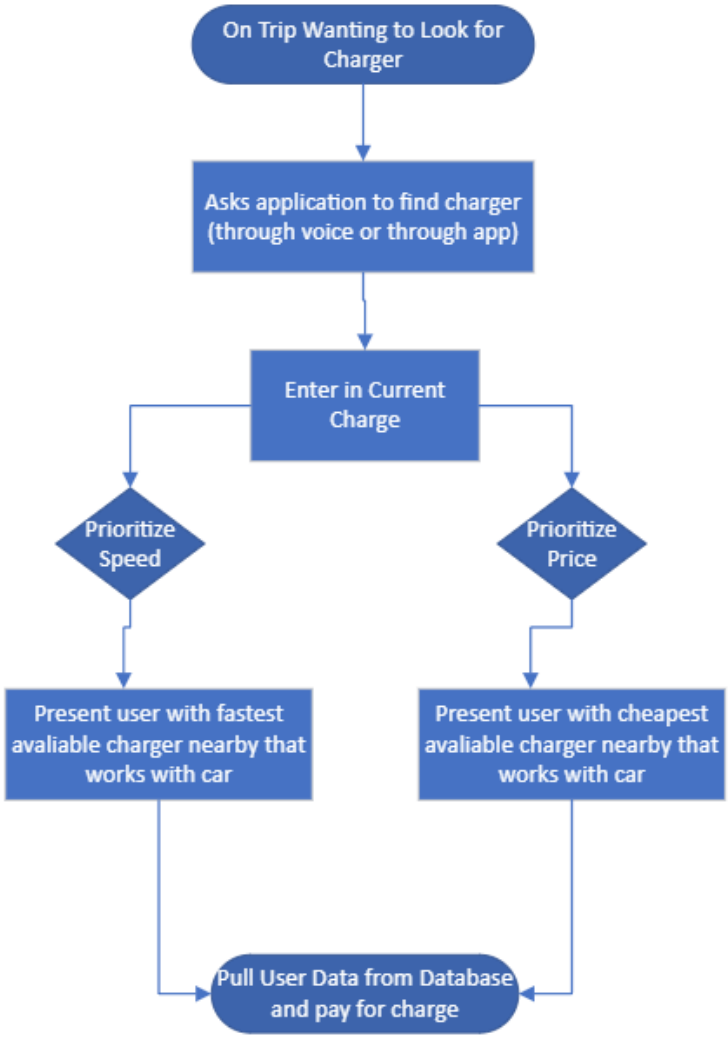

Figure 41: Flow-Chart of a User Locating a Charger while driving

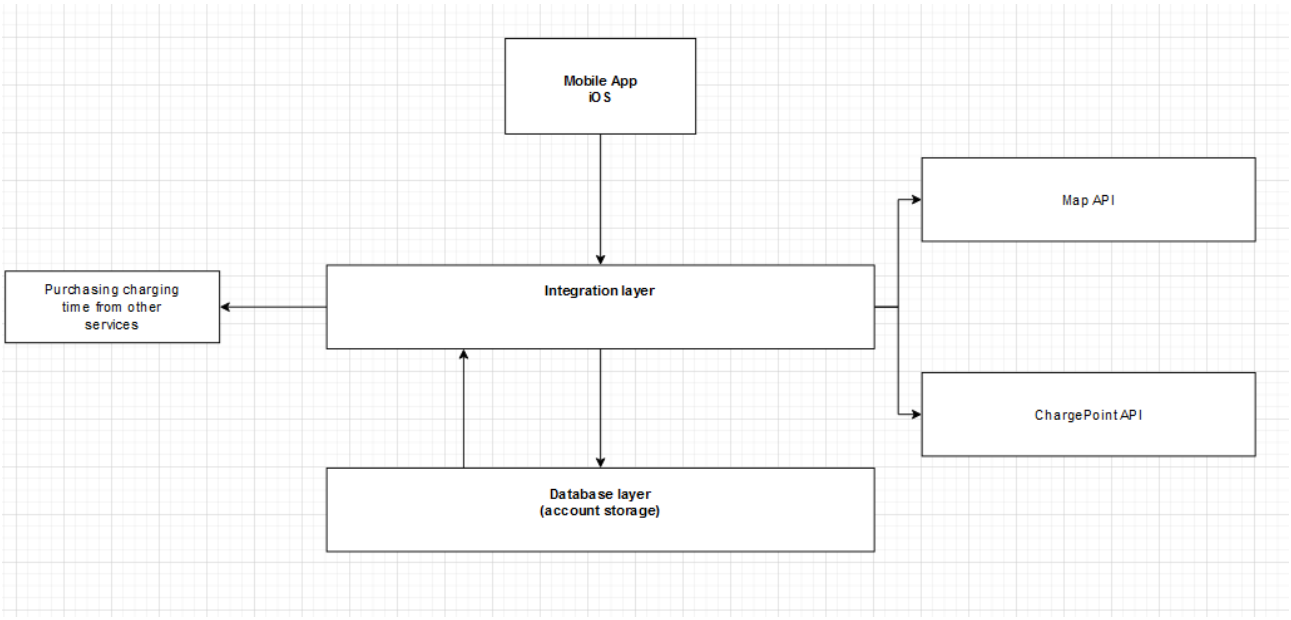

Figure 42: Software Interface

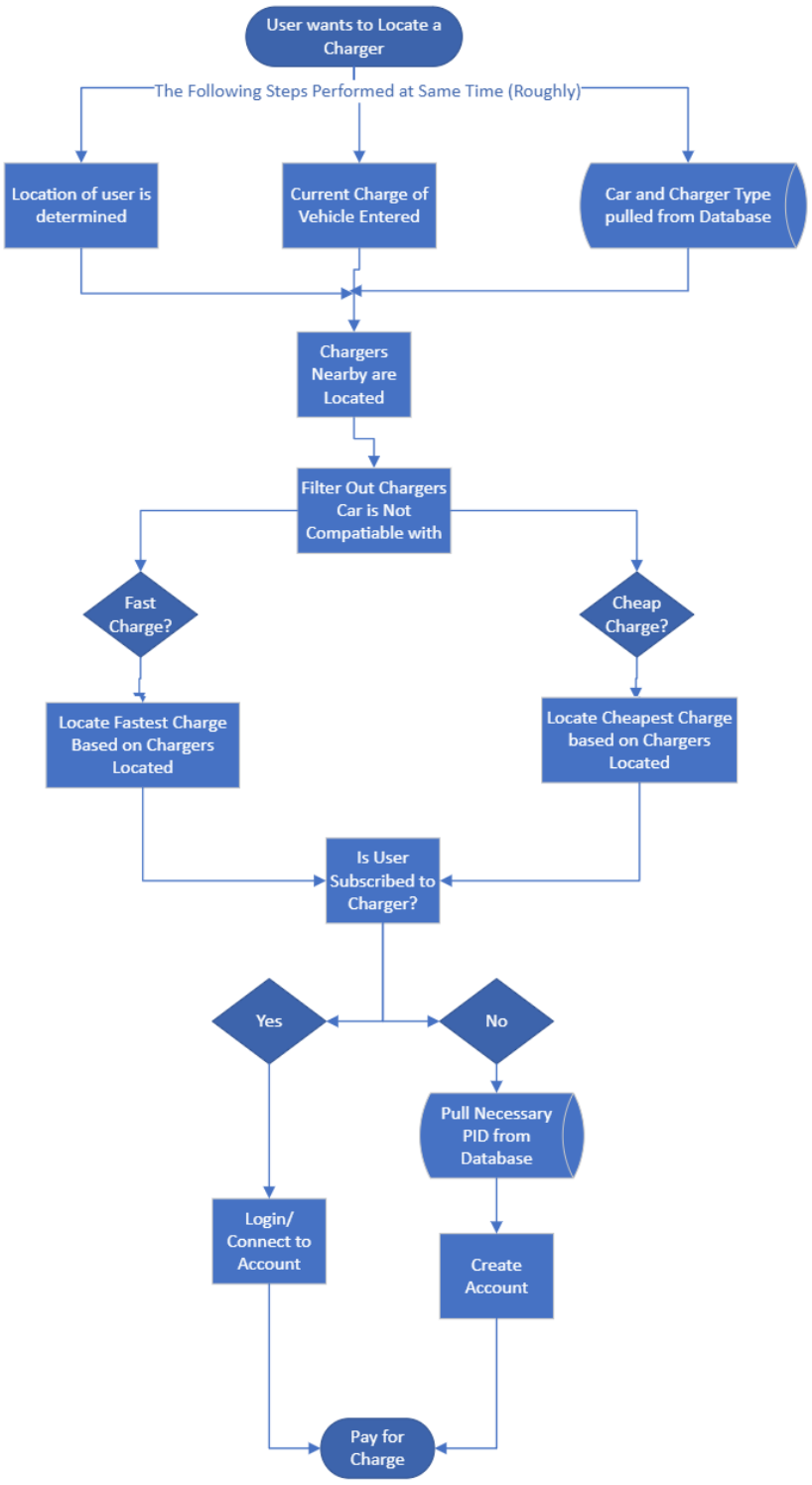

Figure 45: Charger Location Flow Chart

Appendix C: Low Level Design

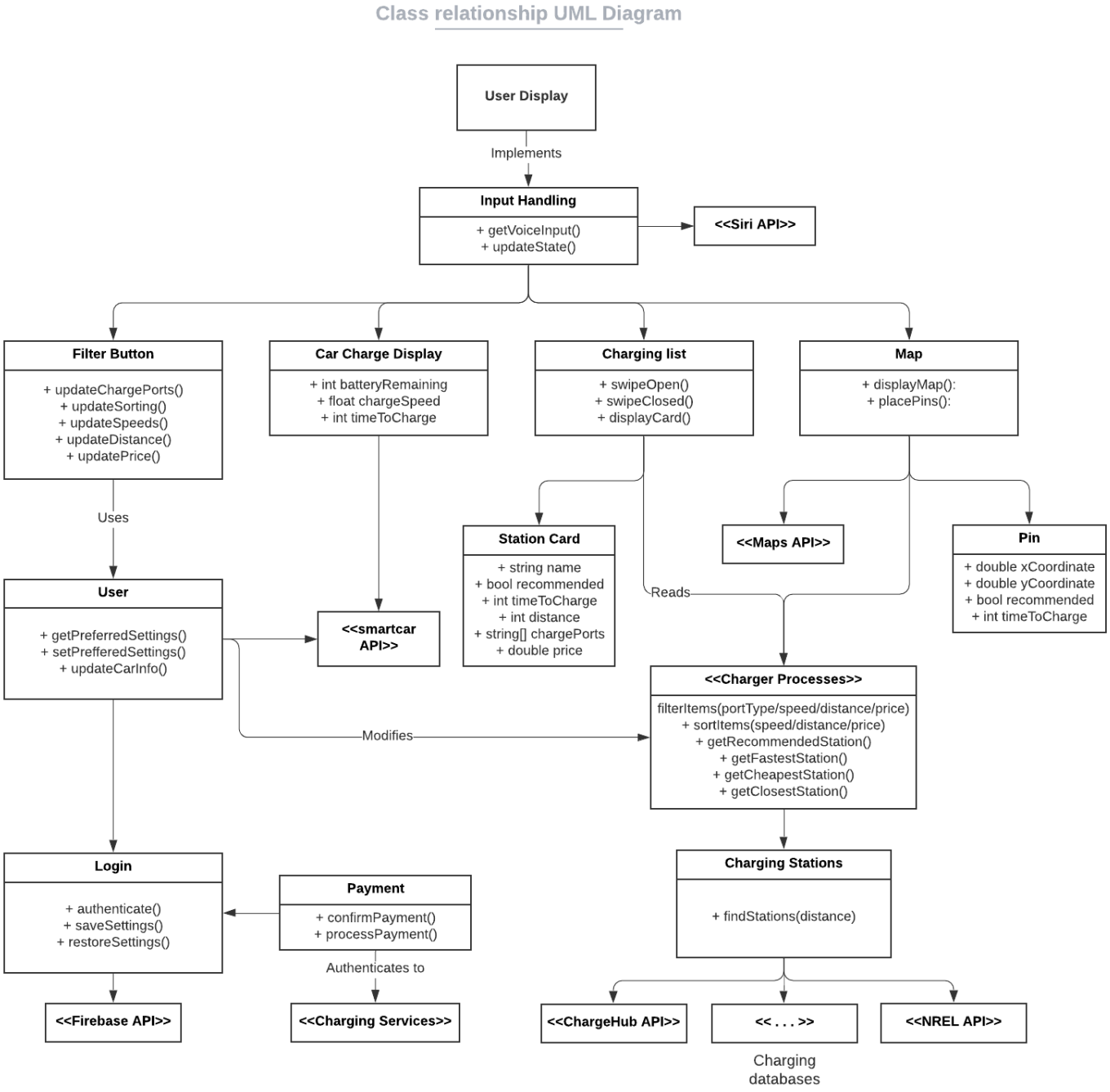

Figure 46: Class Design Mockup

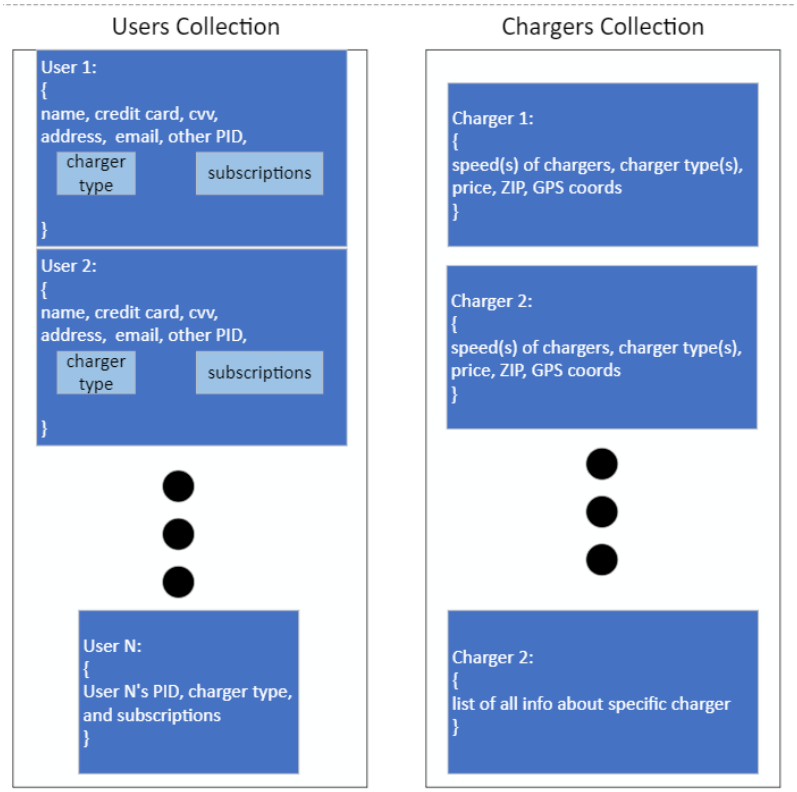

Figure 47: Database Design

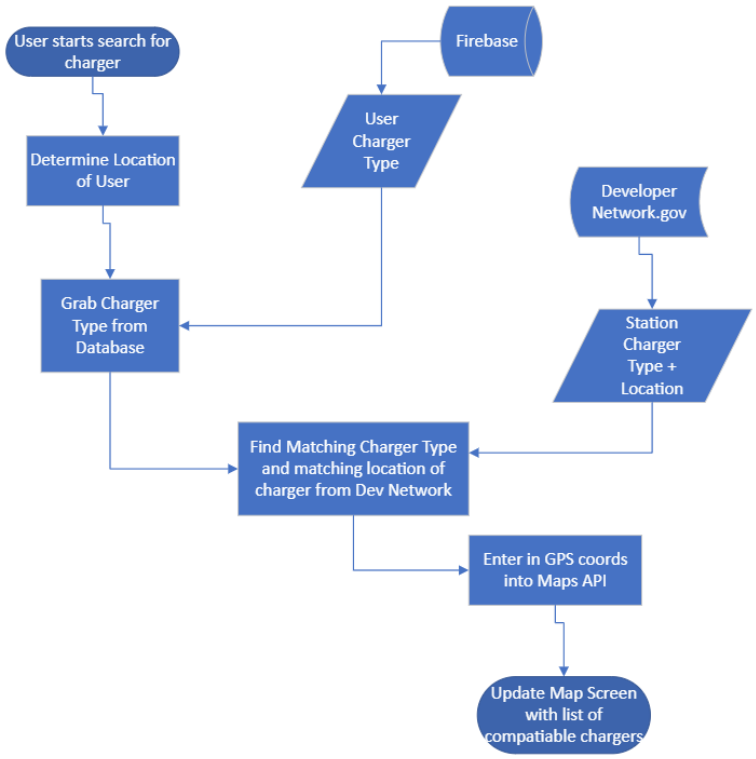

Figure 48: Maps API to Database Connection

Appendix D: Application Interface

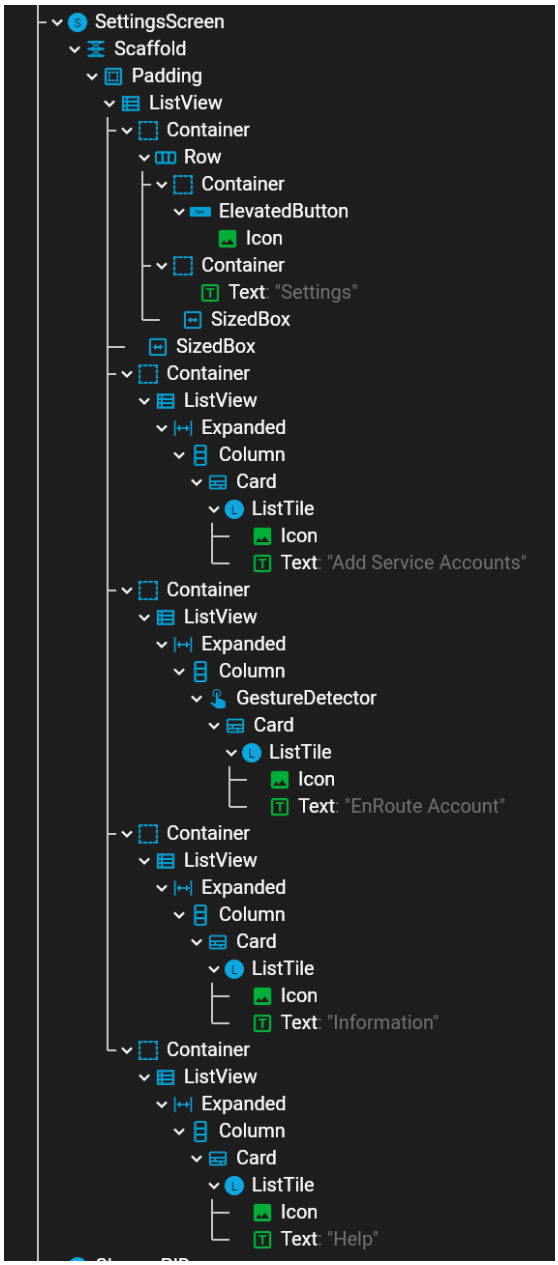

Figure 49: Settings Screen

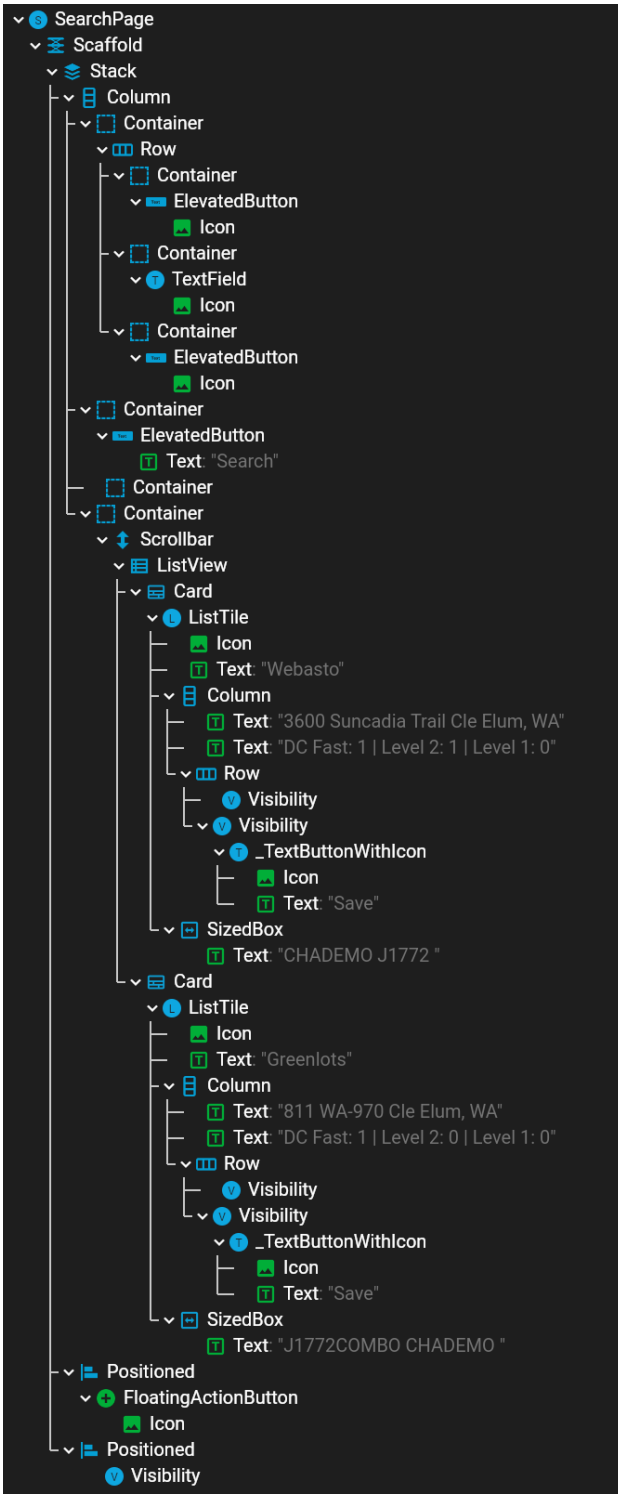

Figure 50: Search Page

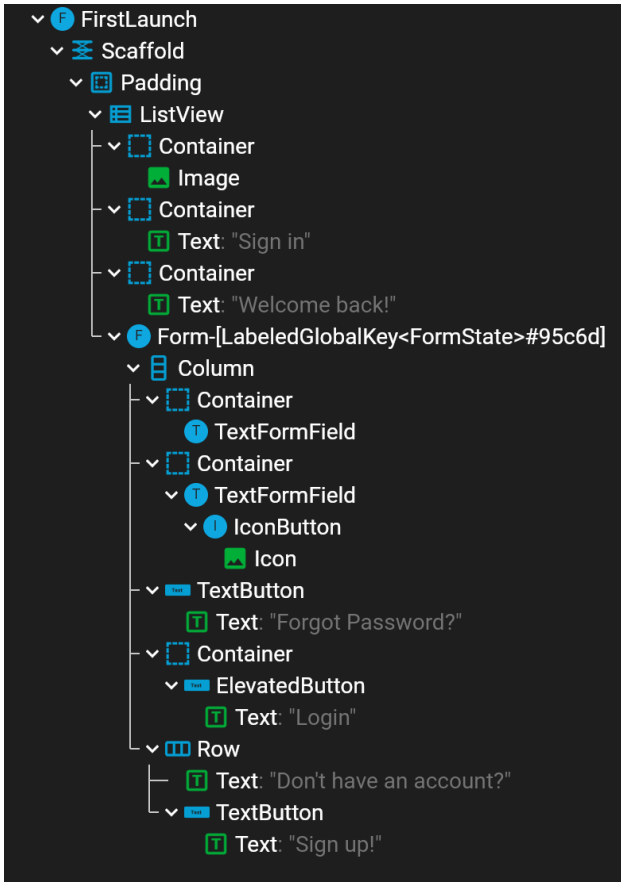

Figure 51: First Launch

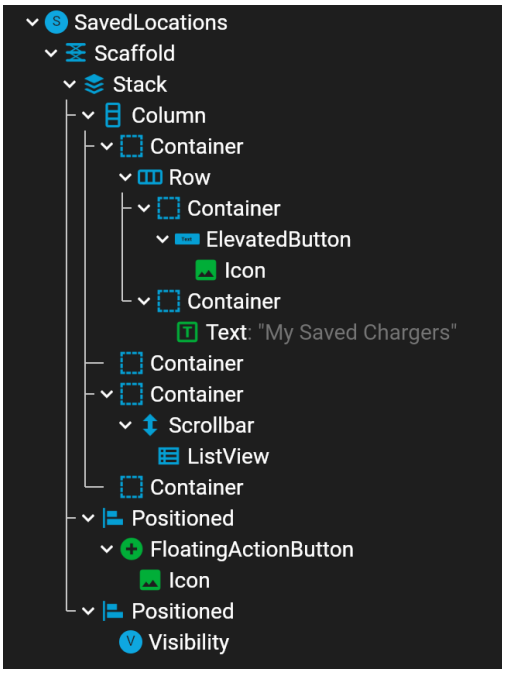

Figure 52: Saved Locations

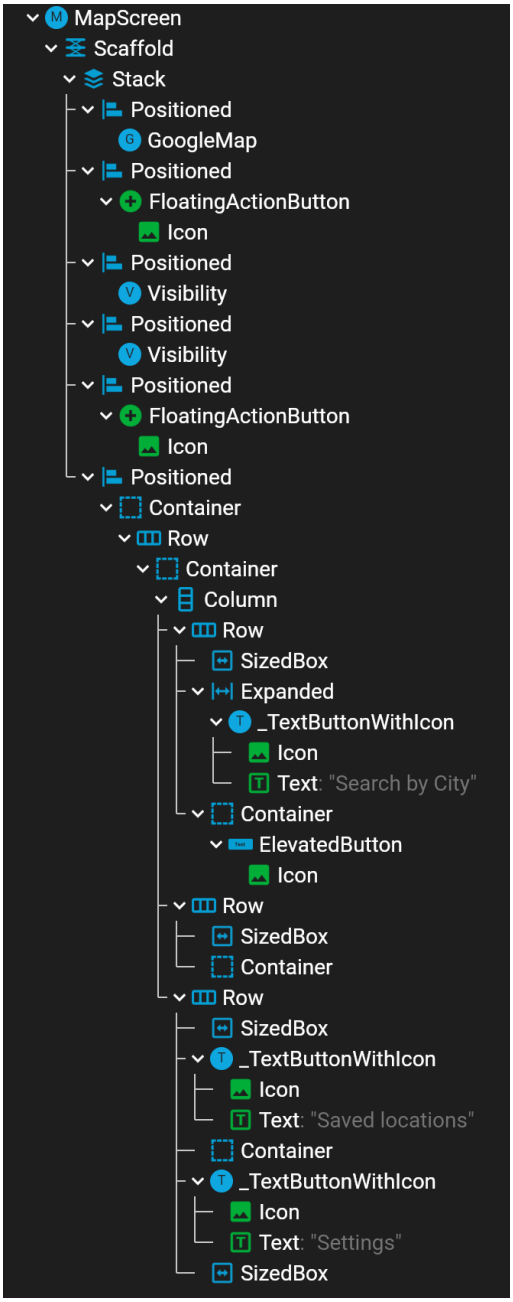

Figure 53: Map Screen