SRS of Car Charger Manager

Written by Lucas Keizur

1/09/2022

Table of Contents

1. Introduction 2
   1. Purpose 2
   2. Scope 2
   3. Definitions, Acronyms, and Abbreviations 2
   4. References 2
   5. Overview 2
2. Overall Perspective 2
   1. Problem Statement 2
   2. Product perspective 3
   3. Product Function 3
   4. User characteristics 3
   5. Constraints 3
   6. Assumptions and Dependencies 3
   7. Apportioning of Requirements 3
3. Specific Requirements 4
   1. External Interface Requirements 4
   2. Functional Requirements 4
   3. Performance Requirements 4
   4. Design Constraints 4
   5. Software System attributes 4
   6. Other Requirements 5
   7. Requirements Hierarchy 5
4. Checks 6
   1. Consistency Check 6
   2. Validity Check 6
5. Design 7
   1. Graphical User Interface 7
   2. Software Architecture 8
   3. User Relation Architecture 9
6. Conclusion 10
7. Introduction
   1. Purpose

The purpose of the SRS is to clarify and elicit all necessary requirements to accomplish the task of creating a car charger finder.

* 1. Scope

The Car Charging App will allow customers to find available car chargers based on their geography, membership status, and plug type.

* 1. Definitions, Acronyms, and Abbreviations

EV: abbreviation for ‘Electric Vehicle’.

AC: abbreviation for ‘Alternating Current’.

DC: abbreviation for ‘Direct Current’.

kWH: abbreviation for the number of kilowatts transferred per hour.

Fast Charging: charging that has an output of 50kWH – 120kWH.

API: abbreviation for ‘Application Programming Interface’.

IOS: an abbreviation for ‘iPhone Operating System’.

GUI: abbreviation for ‘Graphical User Interface’.

* 1. References

https://www.myev.com/research/ev-101/ev-terminology

* 1. Overview

The SRS will describe, in-depth, the problem being solved and a general overview of the solution points. It will also contain diagrams of low- and high-level product design alongside plausible frameworks and APIs.

1. Overall Perspective
   1. Problem Statement

The problem being solved is that across the industry of EVs, several different plug types exist with variable compatibility amongst brands. On top of this, charging stations themselves often require some form of membership and the downloading of an application. These memberships may cost additional money. There is no simple solution to finding the proper charger that not only fits along side the specifications of the car, but also considers the membership status and fees associated with each charger.

* 1. Product perspective

The product will take data from various APIs including information about charger locations, wattage, and plug type, and pare it with user data to return a selection of optimal chargers based on the preference of the user. The user’s preferences may include plug type, distance, affordability, or charge speed. The data will be measured in a hierarchy based on said preferences. The product will also consider membership status and fees of charging station memberships. The product will exist as a mobile app. The platform will be IOS. The method of development will be through a third-party mobile development framework such as Flutter.

* 1. Product Function

The function of the product will be to create a convenient solution to users trying to find the right charger based on their preferences.

* 1. User characteristics

The typical user will be an individual who owns an electric car and is unfamiliar with the selection of chargers within an area. This person will likely have a non-Tesla EV and an iPhone to operate the app.

* 1. Constraints

The cost or policy of a particular API may limit if not break its applicability. This issue could exist in the form of high use cost, limits on the number of calls, or limits on the free use in a business sense. APIs could also be limiting as the Car Charger App may be in competition with an associated product.

* 1. Assumptions and Dependencies

The assumptions of the product are that development will be conducted on a development framework aimed at 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.

* 1. Apportioning of Requirements

It is plausible that the development framework will allow the release of an app version on the Android Operating System. This is a requirement that is dependent on time. Additional features dependent on time includes a ‘trip feature’ which would allow users to create a multi-destination course with its subsequent chargers enroute. If time allows, features may be implemented that create monetization or advertisement space

1. Specific Requirements
   1. External Interface Requirements

* The application will exist as a mobile app running on IOS.
* It will obtain charger data from third-part APIs and compare it with data taken from the user.
* When the app is first started, the user will be prompted to make an account, followed by a request of some user data.
  1. Functional Requirements
* The product will allow user to input their cars make and model, credit card information, and charger membership status after making an account.
* The user will tell the app what condition it prefers to search for in a charger.
* The app will use said condition alongside data from the charger API, and other user data to identify the ideal chargers and return the location and other details to the user.
* The user will be able to use their voice to conduct a charger search.
  1. Performance Requirements
* The app will be able to handle a scalable number of users by using pay-what-you-use cloud products such as, but not limited to, Google Cloud, AWS, or Azure.
* The charger search function will perform its search within 15 seconds.
  1. Design Constraints
* The app will not obtain first-party information on charging station due to time, budget constraints, and scale of data.
* The user will input their own cars data, alongside their memberships.
* As credit card information is confidential, certain limitations may be placed which could prevent its full implementation.
  1. Software System attributes
* The app will contain unique user accounts for each member, stored in a database.
* The product should be completed and operational by March 8th, 2022.
* The maintenance of the product thereafter may not be conducted by the same development team.
* The station data will be obtained through the PlugShare API.
* The map data will be obtained through the Google Maps API.
* The application will be developed using the Flutter framework.
  1. Other Requirements
* The app will allow the automatic membership of car charger organizations with the user’s permission.
* The app will require the location of the user in order to function properly.
  1. Feature Hierarchy

The priority of the feature implementation is as follows.

* 1. A usable app, designed for IOS, that gathers location data to find nearby EV chargers through an API of car chargers.

Reason:

* 1. The App will feature individual user accounts.

Reason: The user will be able to save their car’s information and allow other user specific features down the line.

* 1. User accounts will collect certain data from the user including car make and model and third-party membership status.

Reason: The base product is the foundation of all features.

* 1. User Data will be Used alongside API data to determine an optimal list of chargers based on the user’s data.

Reason: This is the basic functionality of the app. It lays the framework of additional features.

* 1. User will be able to use voice to create a search.

Reason: This feature creates reasonable usability while driving a vehicle which is very important for a vehicle-based app.

* 1. Charger search will include membership data.

Reason: This adds additional user data for the app to take into account

* 1. The app will now take credit card information as part of the user data to allow easy signup of third-party charging memberships.

Reason: This adds additional usability and convenience, while also creating the first foundation for monetization.

* 1. The app will allow users to create aforementioned ‘trips’ that can include multiple stops, not necessarily involving the user’s current location.

Reason: This feature adds more functionality outside of the users location, and is dependent on many of the previous features.

* 1. The application will be available on the Android Operating System.

Reason: This will add user numbers. This feature is farther down the line as a fully developed product on IOS is more important than a partially developed app on two different operating systems.

* 1. The application will be monetizable through plausible advertisement or promotional deals.

Reason: The app is intended to be a prototype. Monetization is not required, however, if time allows, would be greatly valued.

1. Checks
   1. Consistency Check

All features are unique and don’t contradict each other. The voice controls are an addition to the manual search feature. The user will have the option for both. Voice controls exist for when the user is driving a vehicle, while manual input will be for more deliberate searches while not driving. The hierarchy of features is dependent on the prior specified feature. All features are in a logical and feasible order.

* 1. Validity Check

The expectation of the client is an application which collects user data and combines it with data about car chargers to build an EV charger search application that can find the most convenient charger based on user specification. It will have the ability of the user to control the app with their voice while driving. The previously listed features accomplish and follow along with the given requirements. The validity of the requirements is sound as it works to create an app that is effective in its goal. IOS will reach the most amount of user as the demographic of iPhone and EV owners overlap.

1. Design

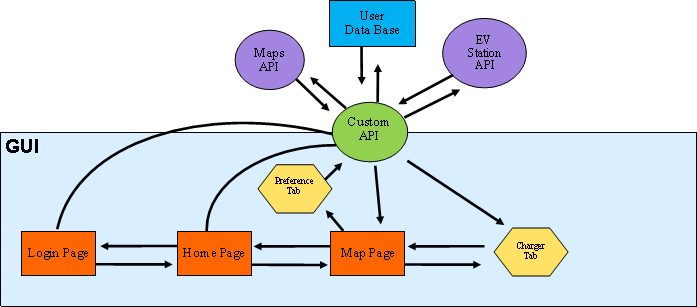


* 1. Graphical User Interface

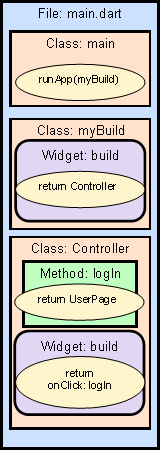
*Figure 1* is a general design for the map GUI. When the user begins a new search page, they will be presented with a default search based on membership status, followed by distance. To change this preference, the user may hit the preferences button and choose how to weight the search. They may also do this through voice commands by pressing the microphone icon. When done through voice commands, the phone will play the options through an automatic voice. The list of charger displays all necessary information to the driver in order to choose a viable option. If the user wants to know more information such as new membership costs, they may use voice commands or tap on the charging station’s info tab.

* 1. High Level Design

Figure 2 is a diagram of the high-level design. The software will begin by placing the user at the login page. Here, they will either create a new account or log into an existing one. This action will take advantage of a custom API that will access the User Data Base and return the necessary data to open the Home Page or deny an incorrect user entry. The Home Page will connect with the Custom API to display details about the user including memberships and car type. If the user wishes to start a map search, they will press a button to take them to the map page. The map page will use a default preference and all the Custom API to collect the proper data from the Google Maps API, and an EV Station API. This data will then display back to the Maps Page for the user to use. If the user wants more information about a particular charger, they can click on that chargers tab. The tab will take data from the EV Station API for the user to view.



* 1. Low Level Design:

The blue boxes are the source code files, the orange boxes are the classes, the green boxes are functions, the purple boxes are the classes’ widgets, and the yellow ovals are the code snippets the point to the next process. In Dart, the language used within the flutter framework, GUIs are made up of blocks called widgets. Each widget exists within an accompanying class that can contain functions for which the widget interacts with such as a button or input box. Each widget returns a structure. That structure can be from Flutter’s default set of objects, or a custom one. The returned structure is what adds elements to the GUI. This low-level design splits the interface into 3 distinct parts, contained in their own unique files, the Login, User, and Map pages. The controller class represents a basic login page, which brings users to the UserPage. Here, they 

have access to their personal data that they can

manipulate themselves. When the user wants to

search, they can bring up the MapPage. This page contains 3 primary widgets. The MapBlock, which contains the Google Maps image, the MapMarker, which displays pointers on the map for the location of the chargers, and the ChargerTab, a list of the most qualifying chargers which are also the same chargers being marked on the map.

1. Conclusion

Therefore, in line with the requests of the client, this application is to be built with the Flutter framework, while implementing APIs from Google Maps and PlugShare. It will solve the issue of EV owners being unable to adequately locate and use car chargers. The solution is a user-oriented application that can find the most relevant chargers based on the users’ preferences, which may include plug type, membership requirements, wattage cost, charging speed, and or distance. This application will also apply voice controls for use while operating a vehicle and the option to join third party memberships through the app.