CA326

Group cd1

Conor Nugent
Jason Madden

Software Requirements Specification

Document

Date: (30/11/2017)

Table of Contents

Section 1

1.1 Purpose

The purpose of this document is to clearly and efficiently explain all aspects of the project from the overall functionality to the estimated schedule for the project. It should also provide a very detailed look into the software and hardware requirements along with the technical ideas of how the project will be developed. The intended audience of this document is EV drivers and technical minded individuals who are interested in the details of the project.

1.2 Scope

ChargeGuide is an Android application to help EV drivers get to their destination using optimal charger(s) and estimate how long they will be charging for. This includes having a database which contains efficiency, battery size and connector of all the EV's available in ireland. There will also be a table containing all the rapid chargers in Ireland.

The optimal path will be one or more chargers that will then be sent to google maps as gps waypoints. Recommended chargers will be based on how close the charger is to the estimated point of the car battery dying and how far the charger is to the starting point. The charger can't be too close to the start point as there is little point in stopping and charging while the charger can't be out of range as the car won't make it to the charger.

Users will have the opportunity to select which route they want to take out of the routes recommended by the app.

1.3 Definitions, Acronyms, and Abbreviations.

Ev: Electric Vehicle.
SOC: State of Charge.
CCS, Chademo, FastAc : The main 3 standards for rapid charging in europe.
Wh: Unit of energy.
Wh/km: Amount of energy required to drive 1km
ICE'd: When a charging bay has been blocked by a car that can't be charged.
Regen: Motor slowing the car down to gain charge.

1.4 References

Organisation: ESB
Last Accessed: 30/11/17
http://esb.ie/electric-cars/kml/charging-locations.kml
(now referred to as "the kml file".)

1.5 overview

ChargeGuide aims to provide EV drivers with a tool that allows them to quickly and efficiently plan a journey. This means that the user will not have to make any unnecessary stops or have to stay at a charger for longer than needed. Our application will also alleviate the hassle of showing up to a charger and realising it is either broken or occupied.

The main factor that affects our application is our dependency to ESB for the charger information. Since there is no API for the information it is up to our team to parse the information and display what data we need. However obviously if the layout of the information changes then we would have to change how our application works.

Requirements were gathered by using apps which are of a similar nature and pointing out features and aspects which are lacking. We also have a real EV user which could aid in gathering more requirements which we may have missed.

(background for each of the requirements will need to be done when section 3 is done)

Section 2 is the section which describes how the user will interact with the system. It describes what the target audience is and what their abilities are. This section also puts ChargeGuide in perspective by comparing it to other applications similar to it on the market.

Section 3 is the section aimed at the developer. It gives a description of what classes and objects are required and what their function is in the overall system. It also includes other non functional requirements for the project such as server and hardware requirements.

Section 4 is a high level system overview using models such as class models and data flow diagrams.

Section 5 is the preliminary schedule which describes the estimated timeline of the project. Each stage of the project is graphed on the gannt chart provided and shows the timescale for each of the section.

Section 2

2.1 Product Perspective

ChargeGuide is self contained.
ChargeGuide will be somewhat related to few other apps notably plugshare and esb ecars, it will share most of its features with Tesla's inbuilt navigation.

Plugshare has the ability to direct users from point A to point B but has a few missing features such as no charger status. Plugshare cannot tell the user if a charger is broken available or occupied.

Plugshare knows the car that the user is driving however when the user inputs a route Plugshare won't suggest where to stop. So the user picks one that's on route rather than having them selected automatically.

ESB's ecar connect is a fairly limited application. There is no option to add a route so the user is ambushed with pins all around Ireland and has to pick one pin as a destination in google maps similar to plugshare. The button which allows the user to filter the types of chargers is by no means intuitive. The app is similar to ours in the sense that it only shows pins in Ireland while plugShare shows pins around the world.

Tesla inbuilt navigation automatically routes the car to a charger and says how long you will be charging for and the status of a charger and is the most similar feature wise.

2.1.1 System Interfaces

The car data, user information, car charger status and charger location will be pulled from a server.

The google Maps API will be used to display the map in the application and displaying the pins of the charger location.

The app will be using google cloud sql for the database. This database will be used to store the car data, user data etc.

2.1.2 Interfaces

The users will have to interact with the system via a GUI. There will be no special requirements as the user will be expected to be able to see as they they need to be able to drive. The interface the user will be using is touchscreen.

2.1.3 Hardware Interfaces

The system has no hardware interface requirements.

2.1.4 Software Interfaces

Google Maps

- The purpose of google the google maps API is to provide the map and the map functions that are needed for the application to work. Examples of this include the ability to add markers and information associated with the marker to the map and displaying the map itself. The application also allows the user to visualise the proposed routes the application recommends to the user.

Google Cloud

- The purpose of this is to provide a cloud based database which the application will use to access data that is needed for the application to function. Information that is stored on the cloud includes data about the electric cars that are available in Ireland, information about the chargers such as type, location and status and where the user's "home" is. The application will be able to access this data when needed.

2.1.5 Memory Constraints

512 MB of RAM is the minimum requirement

2.1.6 Operations

Data will be processed by a python script which will parse the data out so we just get what's needed, sort it into SQL tables so the phone can access the data. Data will include the charger status, location, type and availability.

2.2 Product Functions

1. Setup of user profile
    a. State what car the user is driving.
    b. State where the user lives
2. Show the map of Ireland with all fast chargers.
3. Clicking a pin would display charger information a button to navigate current location
4. Search bar enables user to search for destination in Ireland.
    a. Show the routes the user can choose from
5. The app will suggest routes the user can take with pins to show where the charger is situated.
6. The user picks a route.
7. The app will then display a route summary which includes how long they are stopping for and where and their arrival soc and departing soc. There will be a button to open this route in google maps.

2.3 User Characteristics

The user will be expected to be a new to ev's. They are not expected to know what type of connector their car uses or what the range is of their car at different speeds. This means that there will have to be a table of cars rather than the user being asked about what the range and conector their car has.

The user is likely not familiar with all the terms that other ev drivers use so a help page about electric cars would will be provided explaining terms like regen icing and how to maximize range.

2.4 Constraints

The only safety constraint is that the user should not use this while driving. This message will be displayed on the splash screen of the application.

Google maps will be doing the navigation we will be relying on it to navigation and to calculate journey time and distance.

The system is limited to 150 000 map loads per day on the free tier for the google maps api.

2.5 Assumptions and Dependencies

The format of the kml file with the chargers information doesn't change. We depend on this to as the parser that obtains the charger information will no longer work.

Features of the google maps api aren't deprecated such as the way the pins on the map are handled. This would mean some features of our application would have to be dropped or modified.

2.6 Apportioning of Requirements.

With all the data from the charger status it would be possible to rank chargers from most to least available. This would be done by monitoring them over a few days to gage the average utilization to give them an available ranking. This would be used to prioritise paths with higher availability chargers.

Drawing the different routes may be challenging so initially coloured pins will denote the different paths as this is quite straightforward. If there is time an approach like google maps with a blue highlighted paths that can be selected will be used.

Being able to report a charger as not working or inaccessible would be feature that would be a nice to have as the chargers at the moment are notoriously unreliable and more often than not don't report themselves as broken

Section 3

3.1 External Interfaces

3.11 user interfaces

When the user opens the app for the first time the system will will prompt with a create user account. This will request the user to input what car they are driving and where they live. There will be an button to automatically set this to current location. A submit button will also be shown to save the user's details. This will only come up the first time the user opens the app.

Once the user inputs their details. The activity displaying the map will appear. This will be the default activity from now on. Markers will show charger locations in Ireland that are relevant to the user depending on the car they drive. These markers are color coded depending on their status and type. A search bar will be on the top of the activity. This allows the user to search for locations in Ireland. While the user is inputting a location, the app will clickable display suggestions. When the user clicks search (or enter) the app will calculate suggested routes and display them as colored pins.

When a route of a particular color is clicked the user will be redirected to a new activity displaying all relevant information about that route. This will include suggested time to charge (an int in minutes) , car charge (from int 0 to 100) when arriving at a charger, car charge (from int 0 to 100) when leaving the charger, the name of the charger (a string) and the arrival state of charge (int from 0 to 100) at the destination. There will also then be a button to open the route in google maps.

The system will have a navigation drawer which will give the user the option to change their details such as what car (string) they are driving and where they live (latLng object) . There will also be a help button which will direct the user to a help activity. Here the user will be able to find information about different terms such as icing and Wh/km.

On the main map page the user will be able to click on a charger marker to view its status and navigate to it directly. The user may be able to report this charger as broken.

3.1.2 Hardware Interfaces

The only hardware interface will be the touchscreen on the user's phone.

3.1.3 Communications Interfaces

On the login page, when the user clicks submit, the data will be sent to the database where the application will be able to access it when needed.

The server will also hold information about EV's. A car object will be made of the user's inputted car. The object will get the data of the user's car for its attributes from the database of cars. This will later on be used for range calculations and time to charge calculations.

optimalPath will take in the start and end goal and the user's car. From this the application will be able to work out what routes to suggest to the user and what chargers the user should stop at.

journeyInfo will take in the best paths form optimal path and will be used to display the how long the user will be stopping for and what soc they will be arriving with.

3.2 Functions

3.2.1 Class/Object

Map:

Methods:
displayMarkerDetails() shows the user the information associated with the particular marker that they clicked on.

goHome() uses the optimalPath class but fills in the start goal as the user's current location and the goal as the user's inputted "home" data supplied when the user opened the application first.

getCurrentLocation() gets the user's current location.

Attributes:
This class doesn't have any attributes associated with it.

Functional Requirements:

The functional requirements for this class is to guide the user home from their current location, supply other classes with the user's current location, display the markers of the location of the chargers on the map and show the details of the charger at the particular marker when the user clicks on it. The Map class can also search for the optimal routes to suggest to the user. Each route will have a color and each pin that is relevant to each route will be colored accordingly. The user will be able to tap on a pin of a particular route to get more details about the route. The option to show the route in google maps will also be there once the user selects a route.

Criticality:

The map class is reasonably important but not the most important class as it doesn't do much and some of its features could easily fit into other classes.

User:

Methods:
This class has no methods associated with it.

Attributes:
setup is a boolean variable that is true if the user has gone through the setup process or false if it is the first time the user has opened the application.

home is a LatLng object which is the location the user supplies in the setup or settings of the application.

car is a string variable which stores what car the user is driving. This is used for calculating range for the journey the user is taking.

Functional Requirements:

The user class is used to store the user's data for easy access across the application. Since there are no methods with this class there is limited number of functional requirements for this class.

Criticality:
This class isn't particularly important as it is quite small and has no methods and the rest of the system would mostly work without it with minor changes.

Functional Requirements:

The functional requirements for this class is to store parameters about the user. This data will be accessed across the application when the user's home location or specific car is needed for route planning or calculations.

Main:

Methods:
getChargers() builds the listOfChargers from the server information. It will limit the list of chargers to only the chargers that are applicable to the car the user said they were driving.

toObjChargers() takes the charger data from the server and makes a charger object for each of the chargers and stores them in listOfChargers

displayMap() uses the google maps api to display the map.

displayMarkers() displays all markers that are in the listOfChargers using the coordinates of the charger object.

openHelp() starts the activity to display help information to the user.

Attributes:

listOfChargers is the list of all charger objects in Ireland which are applicable to the user based on the car that they are driving.

Functional requirements:

The main class is build up array of chargers that are compatible with the users car and to display the map. On the likely chance the server is down or the app cannot connect to the server the app will display a message "cannot connect to server".

Criticality:

The main class is very important as it is responsible for making an array of charger objects this is used by several methods. It is also responsible for displaying markers which are used to denote the path and to route directly to the destination. Very little would work without this class.

Charger:

Methods:
There are no methods associated with this class

Attributes:
Location is a string representation of where the charger is located. This is not the coordinates but more of a "human readable" version of where the charger is situated.

Coordinates is a LarLang representation of the geographical location of the charger. This is the attribute that will be used to construct LatLng objects to draw the markers on the map.

Connector is a list of the type of chargers that are available at the particular charging stations. Each charger can have more than one type of connector available to use.

Status is a string representing the state the charger is in. The status a charger can be in is either available, in use or out of service.

Functional Requirements

The charger class is responsible for supplying the all details about the charger. This information will be used across the application for functions such as estimating which charger should be recommended to the user and displaying the marker to denote the particular charger.

Criticality:

The charger class has no methods but is important as without it a the main class would not be able to make a charger object to be used by several other classes.

Server:

Method:
parseData() Takes the kml file and parses the data to a SQL table of chargers.

getNewKml() requests new kml file from the link in the reference to get the most up to date data about the chargers.

Attributes:
chargerLocation is the string representation of the charger geographical location. This will be used for an attribute in the charger object.

chargerName is the string representation of the human readable location of the charger.

chargerStatus is the string representing the status of the charger. The information of the charger status is refreshed every 10 minutes.

chargerType is the string which contains the type of charger that is available for the user. The server will contain all chargers in Ireland but the application will limit this to only those applicable to the user.

userData: is a sql database which contains the user's home location and car.

Functional Requirements:
The server hosts sql database with the user data and parses out the kml file into a sql table

Criticality:
This is an important class as without it there would be no chargers to on the map and the app would become useless as the server is responsible for pulling and parsing the kml file.

optimalPath:

Methods:
coordinatesToKm(): This take the two coordinates and calculates the distance in km and returns an int.

googleMapsCheck(): takes in the possible paths and gets the distance over the roads to give a more accurate estimate of the distance to the destination and to sort the possible paths in terms of distance.

aStarSearch(): takes in the array of all chargers , the user's car and destination. Using point to point distances makes several estimates of possible paths and appends them to the possible paths array

calculateConsumption() : estimates the soc between the chargers in the selected paths array using googleMapsCheck to get distance and average speed.

openInMaps(): Sends the coordinates in the selectedPath to google maps as waypoints.

Attributes:

possiblePaths : is an array made up of arrays of chargers which make up a path to the destination

selectedPaths: is the list of routes which the user is presented with after only selecting the best.

Functional requirement

optimal path has to create an array of different paths to the destination. The array is made up of chargers. aStarSearch needs to handle the event where no path is found so a message will need to be displayed to the user.

Criticality:

This is the most critical class as it will be the most difficult to get right as it has to find optimal paths. This will be a difficult to as it will be a fairly long processes going through several steps to find optimal paths.

journeyInfo:

Methods
chargeTime() estimates the time in minutes the user spends charging at a charging station.

Attributes
chargerName: This is the name of the charger
arrivalSoc: The soc when the car reaches the charger
departSoc: The soc when the car departs the charger
ChargeTime: The time in minutes that the user will be charging for at the charger

Functional requirements:

journeyInfo takes input from the optimal path to create an estimation of how long the car will be charging for, what their arrival and departing soc will be.

Criticality:

This is not a reasonably important class but not critical as it the final stage of the process and it won't be very difficult to implement.

Car:

Methods:
There are no methods associated with this class.

Attributes:
Name: String representation of the car.
Battery: Size of the battery in W/h
Consumption: a rating of how efficient the car is.
Connector: A string representation of the connectors the charging station has.

Functional Requirements:

Model the attributes of a car. The information for each of the cars will be stored on the server.

Criticality:

This isn't a very important class as it has no methods and will be quite small and easy to implement.

3.3 Performance Requirements

The system will be able to function at a minimum support the current ev driving user base in ireland which is 3500.

The number of simultaneous users all pulling data from the server will be at least 100.

The information handled from the server to the user will be in plain text under 1 mb.

The amount of cars the server will have to store will be approximately 30. The amount of user data the server will have to handle is two times the amount of users (each user will have a home and a car). Most of the data will be stored as a string. The energy consumption and the battery capacity of the car is an integer.

The application does not have specific numerical requirements for performance.

3.4 Logical Database Requirements

The user class will pull user's inputted data from the server which is the car that they are driving and the location of "home". The user class will be accessed when the home data or the car data is needed. This information will be needed when the user selects the button to "go home" and when the application is calculating range.

The Main class will retrieve the list of chargers from the server. This will then be used to trim the array down to the applicable charger's depending on the car the user is driving. This will be called everytime the app is opened.

Only these classes will be able to access the database as these are the only classes that need to. Once these retrieve the data they need any subsequent access of the data will be through these classes.

The data will be kept perpetually.

3.5 Software System Attributes

The system must be accurate at estimating how far a car can travel on a given path and estimating how long a car will take to charge to a point where the user will reach the destination.

The system must be able to find the an efficient path to the destination and if possible multiple paths so the user can chose a path to their preferences.

3.5.1 Reliability

The system does not have not have any reliability requirements.

3.5.2 Availability

Our use of the cloud service means that the uptime of the server depends on reliability of the google cloud service and is therefore out of our hands.

Since there is very little opportunity for the user to input anything, there is very little chance that the user will lose any data while a particular service is unavailable however the user will lose the ability to actually use the application as the system is fairly heavily dependent on network connection.

3.5.3 Security

User data will be stored in the server. The only personal details about the user is the location of where they live. This is sensitive information which will have to be stored in the database as ciphertext rather than plaintext.

3.5.4 Maintainability

The car class makes it easy to add a new car to the system as new cars are launched in the coming years. The developer will input the car's name and usable battery capacity in Wh. The developer will then enter the MPGe for the car. This will then be used to work out the range of any car at a given speed as range is inversely proportional to speed. This is more scalable approach compared to entering speed vs range graphs for each new car.

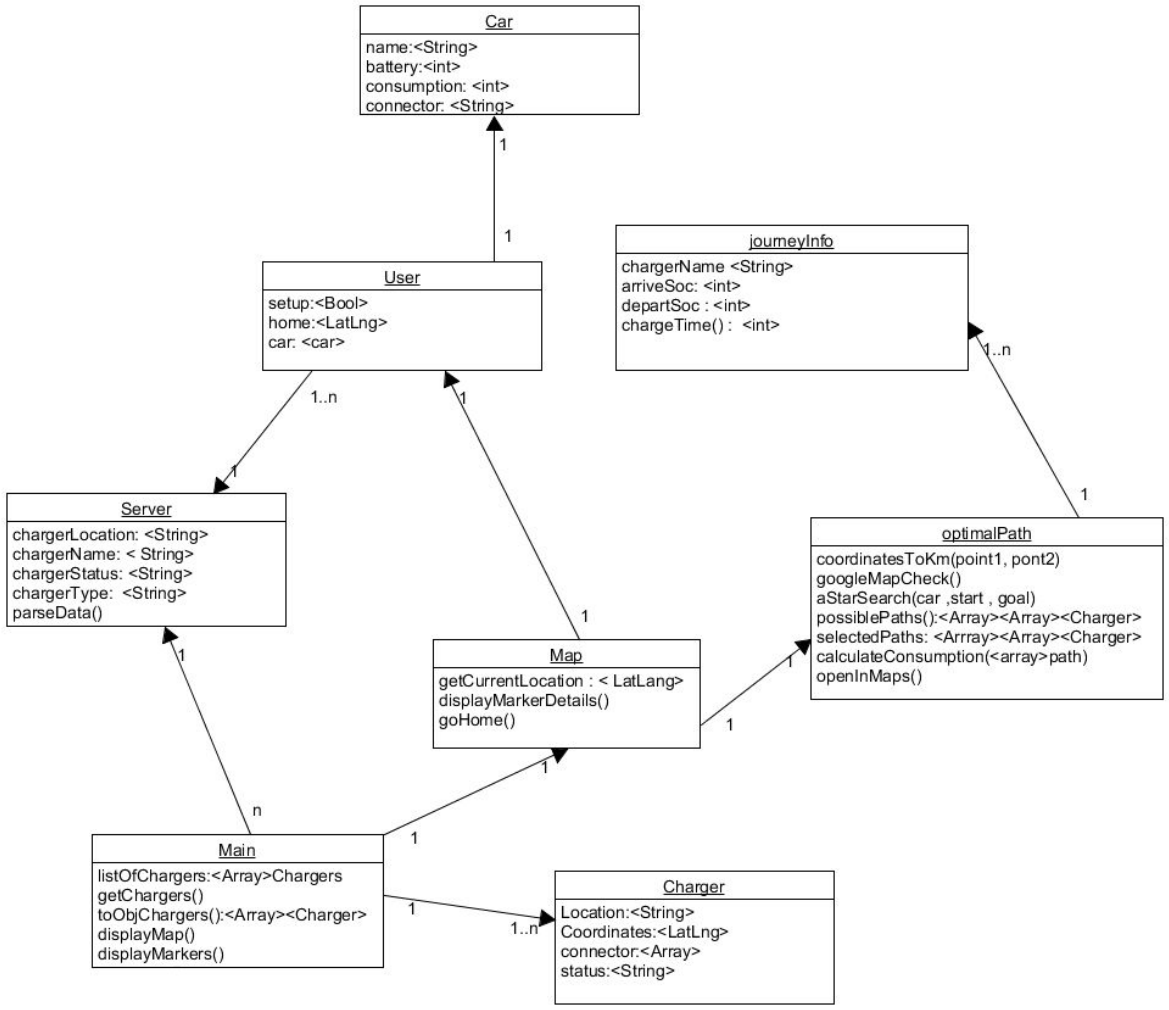
3.5.5 Change Management Process

The srs document can be changed if there is a consensus with the team. If it's a trivial change the srs won't be changed eg a small function is added that was part of a larger function. If the change is larger eg adding a new important function then the srs will be changed. The team member who requested the change must change the srs. The commit message will log what changed.

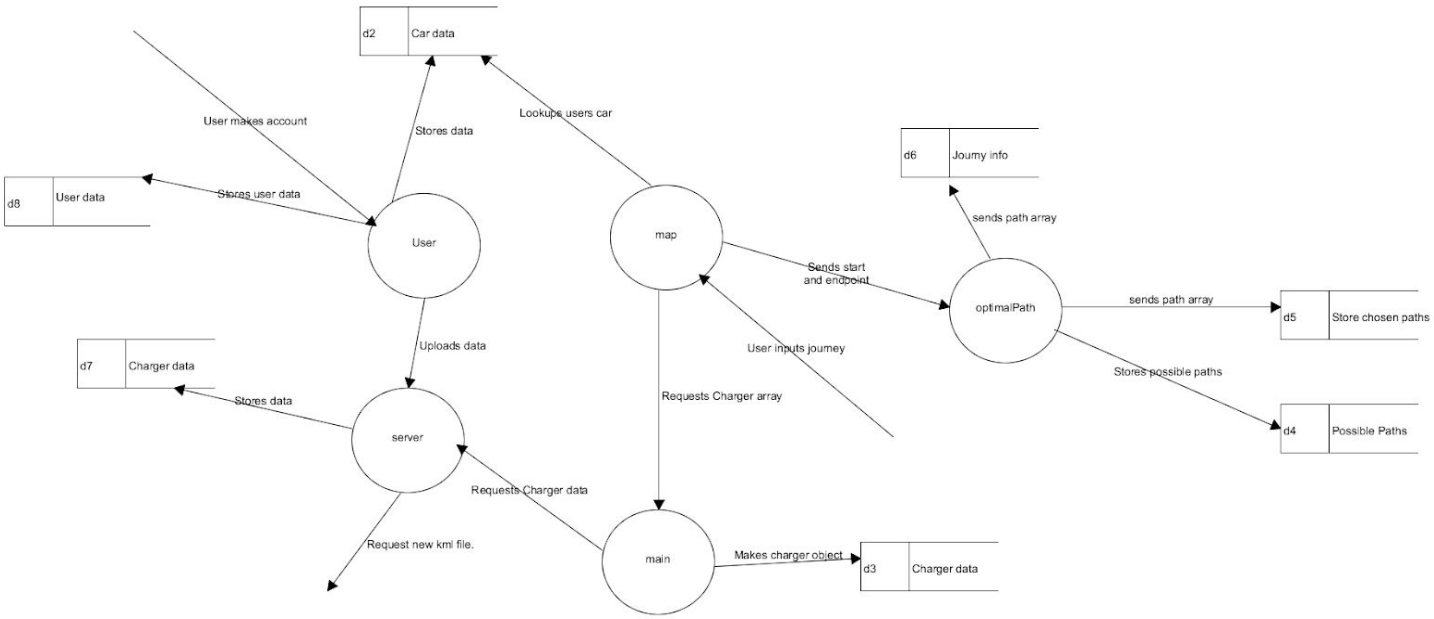
Section 4

High Level Design

Class Diagram



data flow diagram



Section 5

5.1 Preliminary Schedule