Imperial College London

ELEC96012 - Year 3 MEng Electrical Engineering Group Project

Imperial College London

Department of Electrical and Electronic Engineering

DG Cities Electric Vehicle Optimisation

Author:	*CID:*
Sheng Yu	01413225
Weiyue Leng	01351038
Igor Fomenkov	01345356
Joshua Afengbai	01353681
David Adeyemi	01364508

Supervisor:
Dr. Philip Clemow

June 25, 2020

Abstract

This document provides full documentation of the penultimate year EE group project in association with DG cities and Royal Borough of Greenwich. The goal of this project was to create a two-side application which enables a smooth and efficient control of an electric vehicle fleet, both from management and employee perspectives. The project mainly involved using programming for the development of mobile and web application, mainly covering such languages as TypeScript and HTML as well as Angular app development framework. A working demo version of the application was created by the end of the project, and was successfully tested with various pieces of data.

Contents

1 Introduction

1.1 Background

The client for this project was DG Cities - an urban innovation company, which specialises in 'smart city' technologies, and cooperates with Royal Borough of Greenwich. As part of the '*Greener Greenwich Strategy*' which is the council's response to climate change, the Greenwich council plans to increase its fleet of Electric Vehicles from 11 to around 550 to achieve an 100% electric fleet by 2030. However, developing electric vehicles may face several constraints, including spatial limitation of building addition charging points and parking spaces, risk of overloading on-site depot grid capacity and lack of reliable electric equivalents to replace special vehicles (e.g refuse collection vehicles). The fleet electrification is divided into four phases for researchers to fully understand its potential. One of the phases of this transaction to electric vehicles is the development of an application, that will be used to optimize their fleet management. This became the main goal of the project.

1.2 Project specification

According to the initial specification given to the team, the app should entail the following functions:

1. Have an employee side and a management side
   (a) The employee side should consist of tasks, routes and charging sites
   (b) The management side should consist of an overview of the vehicles, onsite charging points, employee journeys and vehicle tracking on maps
2. Dynamically allocate vehicles to employees
   (a) This should consider the route, vehicle charge level and service group
3. Optimise the route employees will follow to complete their tasks
4. Notify the employee with any available charging sites around the vehicle
5. Be adaptable
   (a) Should be able to tell when a vehicle is being used more than expected on the original schedule
   (b) Should be able to re-evaluate the need to charge

The criteria above were procured from the DG Cities project brief (shown in Appendix A.1) and our initial client meeting.

2 Project Management

To achieve the project specification set out by DG Cities, project management was a key tool used. It helped with keeping a clear focus on the objectives, maximising resources available and having a realistic project plan. This was especially important given the current COVID-19 situation which meant that remote working was the only possible way of carrying out this project.

Over the course of the project, there were at least 2 regular meetings to set the plan for the week and discuss any difficulties faced in the previous week. The group academic supervisor was always included in one of the meetings which was held weekly, he served as a guide throughout the project. In addition to the supervised weekly meetings, a WhatsApp group chat was created to enable constant communication between group members. Group members met more frequently not only to review suggestions given by the supervisor but also to hold seminars to discuss research results and make significant decisions. Other minor decisions and addition/removal of features were also discussed in the format of group chat. A OneDrive folder was used to store useful resources and files in an organized way, the documents were sorted according to the week they were due and the category they fell into e.g. client files, academic files. Two Github repositories were also created for each side of the application, so that team members could work on the code respectively and it could be merged and shared at any point.

Table 1 shows the summary of all the group meetings and Table 2 shows the decisions and assumptions taken and the rationale behind those thoughts. These two tables show how the group went from the initial tasks set out by DG Cities to the final project presented in the final demonstration.

Date	**Summary of meeting**
28/04/2020	• Introductory meeting with DG Cities • Discussed initial tasks given by DG Cities • First Team Meeting • Discussed the deliverables for the project
30/04/2020	• Shared initial ideas for project • Scheduled Meeting with Academic Supervisor and DG Cities • Discussed the use of telematics in final solution • Discussed clarifying questions for pending meeting with academic supervisor and DG Cities
03/05/2020	• Discussed on Key assumptions for algorithm • Discussed how to simplify the initial tasks • Decided outline for task list and structure of map page

04/05/2020	• Meeting with Academic Advisor • Researched Smartphone Apps for Electric Vehicle Owners • Assigned research to each team member: – Model Parameters for Nissan – David – Transmission Information – Joshua – Charging Points Location/app - Leng – Model Parameters Renault – Jason – Charging Points Type and Speed – Igor • Research structure is shown in Appendix A.3
05/05/2020	• Meeting with DG Cities • Asked clarifying questions • Questions can be found in Appendix A.4
07/05/2020	• Discussed the leaflet • Split up the research for the leaflet according to the mark scheme – Matching the proposed concepts with the client criteria - Jason – Social, economic, environmental Context – David – Competitor Analysis – Joshua & Leng – What does the app do? - Igor
10/05/2020	• Discussed High level Algorithm • Discussed individual research for leaflet • Discussed Design for leaflet
11/05/2020	• Meeting with Academic Advisor • Clarified assumptions for project • Finalized research on the two models of vehicles
13/05/2020	• Meeting with DG Cities • Received further clarification of tasks given by DG Cities • Looked at competitor Analysis

15/05/2020	• Discussed structure of leaflet • Decided on final leaflet design
18/05/2020	• Discussed the app development process • Decided on the key pages required for the app, they were: – Main Menu – Vehicle – Map View – Charging projection – Fleet and Charging Management
20/05/2020	• Discussed the app structure • Created a document detailing the programming language, device required, back-end tool, front-end tool and tasks to complete for the Management and Employee side of the app. This can be found in Appendix A.5
25/05/2020	• Each team member updated the group with their progress so far • Discussed any difficulties faced and shared useful resources • Decided to add charging points as pins on the map • Decided on making the status of each charging point static for the time being
27/05/2020	• Meeting with Academic supervisor • Discussed issues faced with pulling data from google maps • Decided on using an excel sheet to mimic the google maps data
02/06/2020	• Discussed progress so far • Decided to add more Vehicle Detail to the Employee side on the "Site" page • The "Site" page should display Vehicle ID and Model

03/06/2020	• Meeting with Academic Advisor • Showed Final draft of leaflet • Discussed progress so far • Decided to add onsite charging points status in the management side
10/06/2020	• Meeting with Academic Advisor • Discussed progress so far
15/06/2020	• Meeting with DG Cities • Received feedback on app and suggestions for presentation • Team meeting to discuss video demo and documentation deadline
17/06/2020	• Meeting with Academic Advisor
18/06/2020	• Video Demo • Meeting with DG Cities to discuss documentation suggestions

Table 1: Summary of Meetings

Date	Decision	Rationale
08/05/2020	Ignoring the discharging rate of vehicles	The difference in discharge rate between the two vehicles currently in the Royal Borrow of Greenwich fleet is marginal. This helps to simplify our final product
09/05/2020	Charge as a % of maximum should be aimed to be uniform across all vehicles	An assumption made to simplify our design process.
12/05/2020	Using Back4app for our Backend	It's cheap (free to start) but has many features. It's not limited to Android nor IOS. It can sync codes like GitHub.
13/05/2020	Checking the charge level is above a certain threshold	This automates part of the morning checklist for the client. They don't have to worry about charge timetabling with this feature.
13/05/2020	Charge Timetable should be added	This was suggested by our clients during our meeting.
13/05/2020	Ionic for the frontend	It provides a web platform in addition to Android and IOS
23/05/2020	Assuming the status of charging points is static	Unable to obtain real-time data of the charging points now
25/05/2020	Ability for employees to create tasks in the database	DG Cities informed us that it is the employees who put all the tasks in the database
31/05/2020	Adding exact car details on the 'Site' page	This will provide more details to the employees and keep them informed.
03/06/2020	Adding onsite charging point status in the management side	Provides with management side with details of how much power is consumed, which service uses the most power, etc. This was asked for by the client.
10/06/2020	Decided not to use telematics in final solution	The Covid-19 situation made access to OBD scanners and the borough's fleet impossible. Therefore, testing and calibration couldn't happen.

Table 2: Decisions and Assumptions Taken

3 Key Decisions Taken

This section outlines some key decisions and initial thoughts before the app development stage. Considering the current situation and time limitations, the team had to decide what features needed to be included or cut off while maintaining and optimising the functionality of the product.

3.1 Front-end

Front-end, also called as client-side, is the part that users interact with directly. In the early stage of the project, the team was initially planning to develop both management and employee applications that can run on Android and iOS. Therefore, it was crucial for the team to build the code that could share and run over various platforms. With this in mind, two major solutions that offer cross-platform app development were considered, **Ionic/Angular/Cordova** and **Xamarin**.

Angular is a popular platform and framework that largely simplifies the app development and testing process. Cordova allows the mobile application to fully access and invoke native functionality on a mobile device. The combination of these two frameworks makes the mobile app to be hybrid and the core of the application is written using web technologies, which are then encapsulated within a native application.[1] Ionic framework builds on top of these two and provides user interface components, such as icons and theming. Therefore, the features of this solution can be summarised as following:

- Written in HTML, CSS and TypeScript (JavaScript)
- Combination of web and native application development
- One Angular-based code can run on multiple platforms such as web, iOS and Android

Xamarin is an open-source source platform for developing modern application for iOS, Android and Windows with .NET.[2] It has the following features:

- Written in C#
- Build mobile and desktop application with one shared-code
- Built with native interface controls thus the app is closer to native, leading to a higher performance and user experience.

DG Cities had stated that a website is a preferable platform for the management-side application, so that more information can be displayed and processed. Hence, **Ionic/Angular/Cordova** became a feasible solution, since Xamarin could not develop a web application.

3.2 Back-end

Back-end, also called as server-side, is the part that users cannot see or interact and everything that operates behind the scene, such as servers and databases. Due to the time limitation, it was not realistic for the team to build and maintain a back-end from scratch. Therefore, **Backend-as-a-Service** (or BaaS) was introduced in this project. **BaaS** vendors provide pre-written software for activities that take place on serves such as database management and push notifications; in which case, the team can outsource all the behind-the-scenes aspects of the application and focus more on the front-end development.

Parse became the back-end framework in this project. It is a common used open-source framework to develop application backend. It supports many useful features, such as data modeling and real-notifications, which are key implementations regarding this project.

Back4App was then discovered. It is a platform that builds on top of Parse framework, but includes extra properties such as multi-tenant dashboard. This saves plenty of time for developers to create new app and database with few clicks, whereas self-hosted Parse requires to set up the infrastructure for each new app. This is a advantageous feature, in case that the database experiences any destruction accidentally thus less time is taken for initialising a new server.

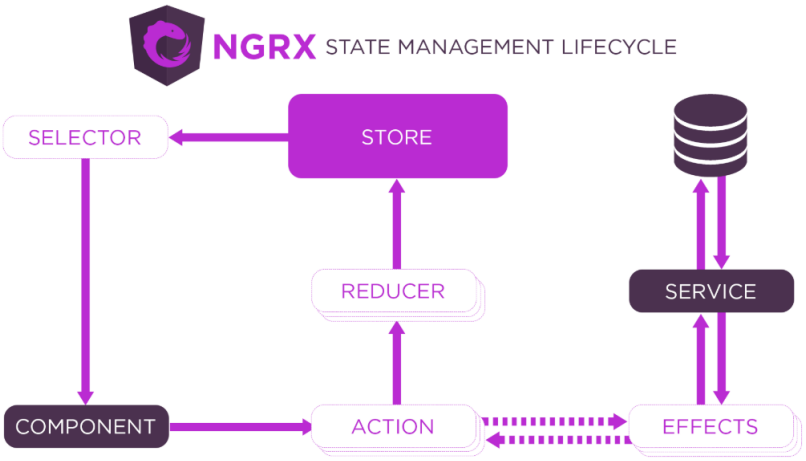


Figure 1: NgRx/Store

However, since the team used the free plan of Back4App, it has limited requests of obtaining the data from the database. In order to reduce this, instead of accessing the back-end and retrieving objects from each page, the team used **@ngrx/store** module to handle the data storage. The application fetches all the data required after login and transfer it between pages using this module. Figure 1 represents that general overflow of application state of NgRx. The following descriptions explain the key concepts of this implementation:[3]

- **Action** describes unique events that are dispatched from components and services.
- **State** changes are handled by pure functions called reducers that take the current state and the latest action to compute a new state.
- **Selectors** are pure functions used to select, derive and compose pieces of state.
- State is accessed with the **Store**, an observable of state and an observer of actions.
- **Reducers** in NgRx are responsible for handling transitions from one state to the next state in your application

More detailed explanations of the code will be demonstrated in the Section 4.1.2 later.

3.3 Coding and Testing Tools

The team chose Visual Studio Code to be the code editor. Visual Studio Code is easy-to-use and contains various package extensions, which include ionic and angular snippets. The Cordova Tool extension is also supported, so that debug console can be used to see the output messages when running the code on a emulator. The command '*ionic serve*' is executed in the terminal to run the program. This boots up a development server on local host, which then launches in the browser. Any changes made in the source files will automatically reload and reflect in the browser.

The browser might be a great tool for visualising the outcome of the management interface. However, the employee side application needs to run on a mobile device and access to the mobile device's functions. The team chose Android Emulator for the employee side to deploy the program to the device, since iOS Emulator could not run on the Windows or Ubuntu environment. To achieve this, Android Studio was installed to set up the Software Development Kits (SDK) and Android Virtual Device (AVD). By running the command '*ionic cordova emulate android --l*', the application is then available to run on the emulator.

3.4 Initial Designs

3.4.1 Initial Ideas

After inspecting the project objectives and discussing with the client, the team came up with preliminary ideas for user interface design and algorithm in a high level perspective.

Various possible factors when developing the algorithm for the route optimisation part of the app were considered. For the employees involved, the team needed to know the qualifications (e.g. can only do certain type of work), the working schedule and any working restrictions (e.g. lunch break). For services, the team needed to know if it was regular or non-regular. The information for the cars needed for the algorithm would include the maximum charge stored, the charging speed and the maintenance required. Charging of the vehicles would depend on the type of charge, the charging speed, charging location and cost.

Moreover, some initial assumptions (some of which were clarified by the clients) would need to be made, as the algorithm for the optimised route planning was contemplated. These assumptions include cars not getting switched during the day, tasks not being able to be cancelled and ignoring traffic when planning the drivers' routes; in addition, assuming employees are equivalent and no new task appearing during the day.

3.4.2 App System Flowchart

The app system flow chart (Figure 2) shows a high-level initial design on how the app would be designed to work. The real time vehicle information is obtained from an on-boarding diagnostic scanner which is then sent to the cloud server – Back4App. The task details which are set by the employee would be also transferred to the cloud. Charger information is researched before creating the app and is fixed in the cloud. The cloud then passes this data as inputs to the algorithm which in turn then allows for optimised route planning, notification of when to charge and charge level for each vehicle on the employee side and fleet management on the management side.

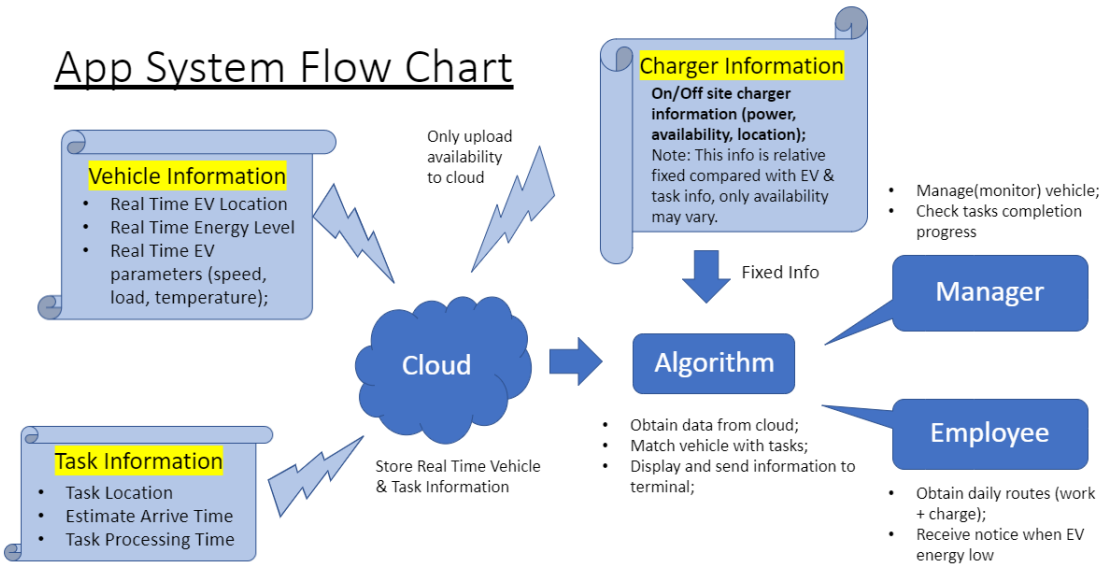


Figure 2: App System Flowchart

3.4.3 Employee Interface

In general, the team decided that it will have two main streams. The first is to allocate tasks to vehicles according to the morning charging levels, possible routes to be taken for tasks and the battery consumption (per km). Furthermore, the second algorithm will decide on whether vehicles use off-site or on-site charging points. Off-site charging will then depend on the urgency of charge and the best charging point locations. These are shown in Figures 3 and 4.

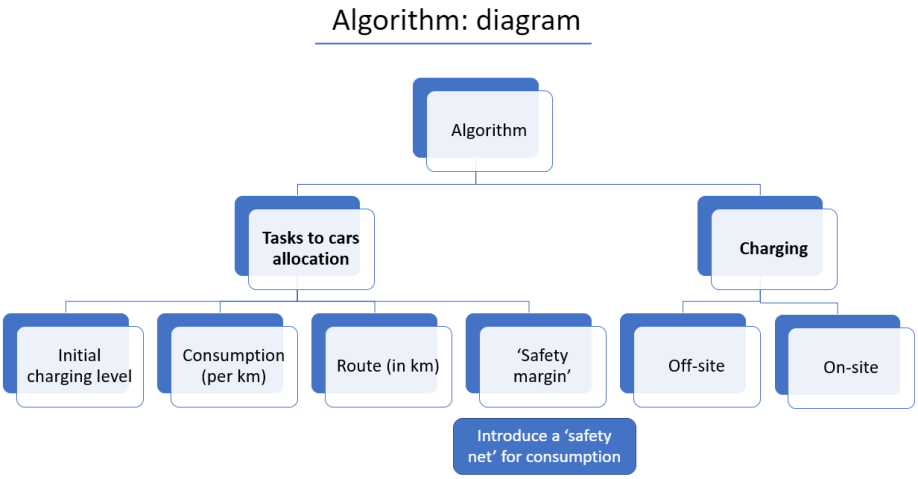


Figure 3: Employee-side Algorithm Flow Chart

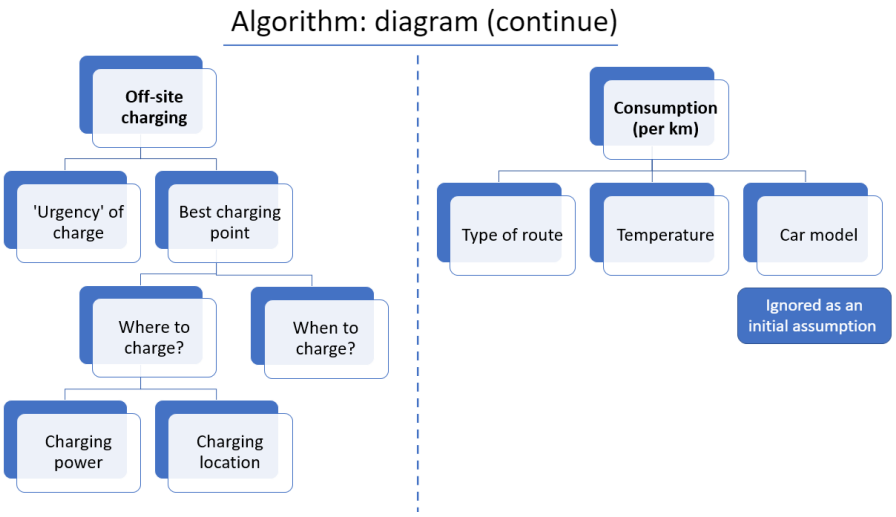


Figure 4: Employee-side Algorithm Flow Chart(Continued)

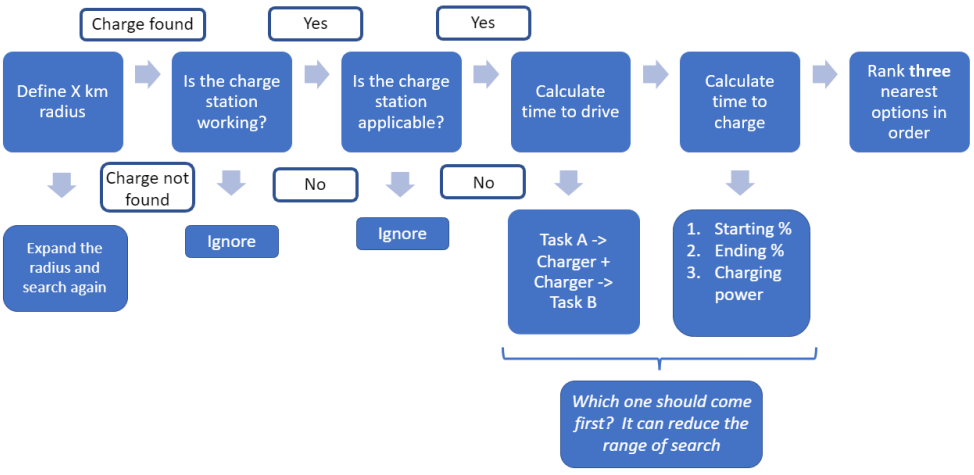


Figure 5: Route Planning Algorithm for Employee Interface

For route planning, the flow chart in Figure 5 shows the 'thought process' of our algorithm. Initially, the algorithm defines the distance and returns 3 nearest charging stations. Once a few charging points are found, it checks if the charging stations are working and calculate the best charging point using the flowchart below.

In order to make the best charging point algorithm, the team sought to find the best charging point based on two major factors – the time to get to the charging point and the time duration of charging the vehicle. The first factor will be dependant on the distance from the first task (task A) to the charging point and the distance from the charging point to the next task (task B). The second factor will then depend on the charge type, starting charge level and required end level. A mixture of these conditions will determine the best charging point. This is shown in Figure 6.

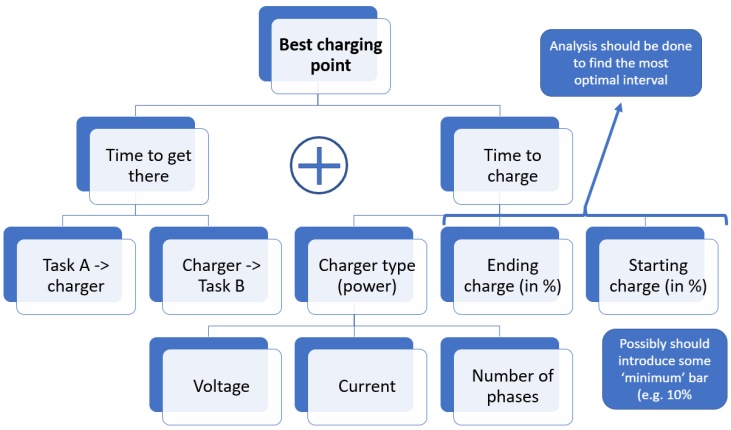


Figure 6: Best Charging Point Algorithm

3.4.4 Management Interface

For the management-side interface part, the initial design contained three pages on the main screen: *Map view* - *Vehicles*, *View Tasks* and *Charging information* (later re-named to *Allocate Vehicles*), which are shown in Figures 7, 8 and 9. In the later stage of the project, the team has made a decision to include two more pages - *Fleet management* and *Onsite Power Consumption* (described in the Section 4.1 of the report).

Figure 7 illustrates the initial design of *Map view* page - it merges both the overall view of all vehicles with individual information on every car, which was inspired by the airplane live tracking map. Every vehicle would be shown on the main map (featuring Borough of Greenwich), its colour would represent the level of charging: green for high (>65%), orange for medium (>20%) and red for low (<20%). This feature would enable the managers to quickly view the overall status of fleet charging. Moreover, individual information on every vehicle (e.g. car model, car number, task allocated, driver, etc) would become available upon clicking on it.

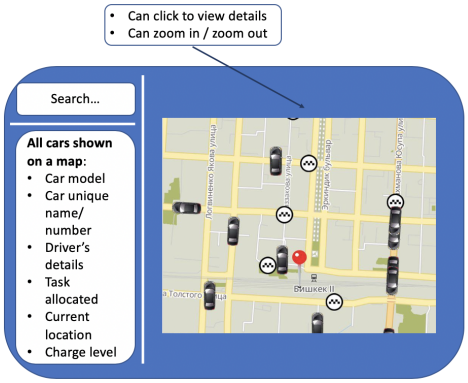


Figure 7: Page: Map view - vehicles

Figure 8 shows how tasks were initially planned to be displayed. One large timetable of all tasks completed over the past month would be shown, and then more detailed information per day would be available by choosing a date in the timetable. Individual information on every task would be displayed, which allows every manager to quickly view the required piece of data about any task. Although the outer design was moderately adjusted, the main features of the page remained closely the same.

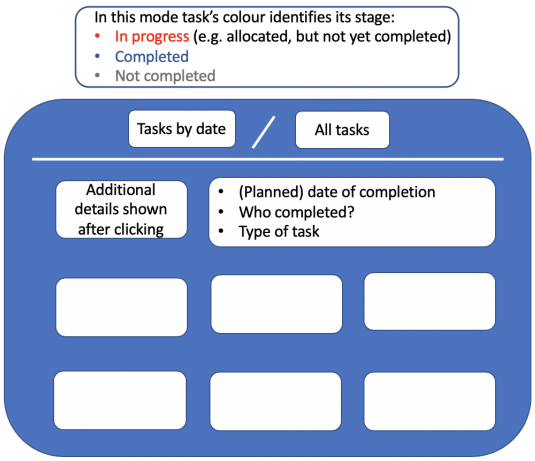


Figure 8: Page: View Tasks

Figure 9 displays the way cars' charging was shown for the initial design. In the first column, cars would be sorted by their current level of charge (from highest to lowest), whereas the second column would display the same information but for the next day; the last column would give the recommended charging points allocation to vehicles. In addition, there would be some analytics provided about charging consumption in the bottom of the page. However, it was adjusted heavily at a later stage of the project in order to address the vehicle allocation process more effectively: one page was split into two separate ones - 'allocate vehicles' and 'onsite power consumption'.

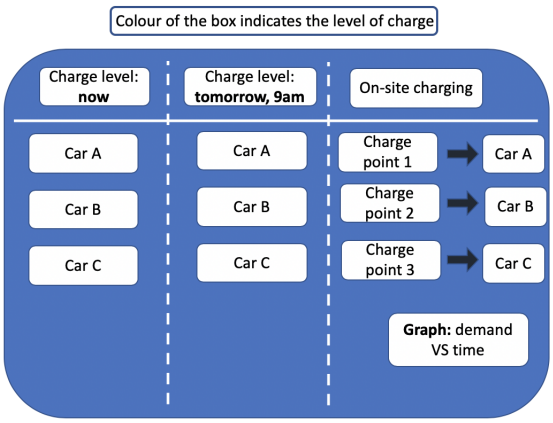


Figure 9: Page: Map view - charging

4 Final Product

In this section, product details including interfaces, codes, and algorithms will be explained. The product is composed mainly of two applications. One application is a website page based tool targeting for high efficiency electric vehicle(EV) fleet management; while the other application is an mobile application which is designed to facilitate employees' daily working.

4.1 Management Interface

Management side application is designed for monitoring and managing the fleet. As Figure 10 shows below, the application can enable the manager to check locations of electric vehicles, view task detail information and allocate vehicles to tasks; in addition, manage the electric vehicle fleet and analyse on-site power consumption.

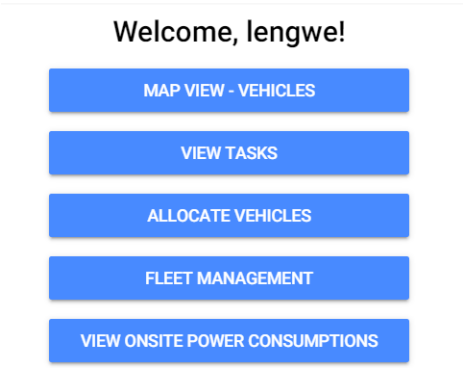


Figure 10: Management Application Home Page

4.1.1 Manager Profile

Managers are supposed to register their personal information in first use. After registration, the manager can use the username and password to login. The username will also be displayed on the main page which was shown in Figure 10. Besides, there is a profile page for manager to check and update information. As demonstrated in Figure 11, not only personal information such as full name and contact details but also security data including account password and ID are listed according to registration records. These pieces of information are then saved in the database for general management and security purposes and can be updated by the user in this page .

The manager account used for the demo purpose is given below:

Username: lengwe
Password: ic

Please either use the above login information to login from the management application or sign up new manager accounts.

4.1.2 View Electric Vehicle Locations

In this page (Figure 12), vehicle locations are displayed on the Google Maps. Each vehicle is marked by a red icon which indicates its real-time physical location according to its latitude and longitude coordinates, which are stored at the database. By clicking the icon, the manager can check further information of the vehicle including vehicle ID, vehicle model and charging level. Therefore, the latest fleet information can be accessed. Since the On-board diagnostic (OBD) system is assumed not working in this case, therefore the geopositions of all the vehicles are stored in a spread sheet.

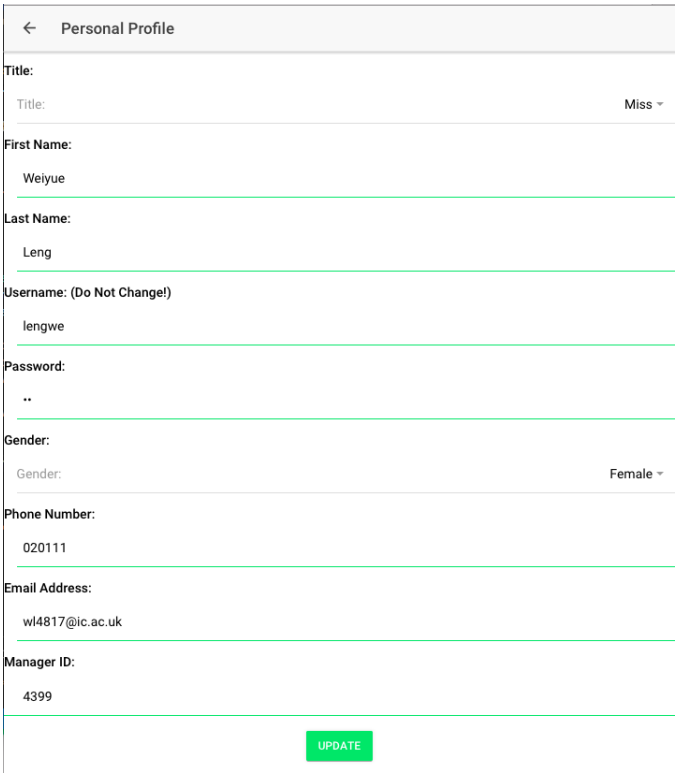


Figure 11: Manager Profile Page

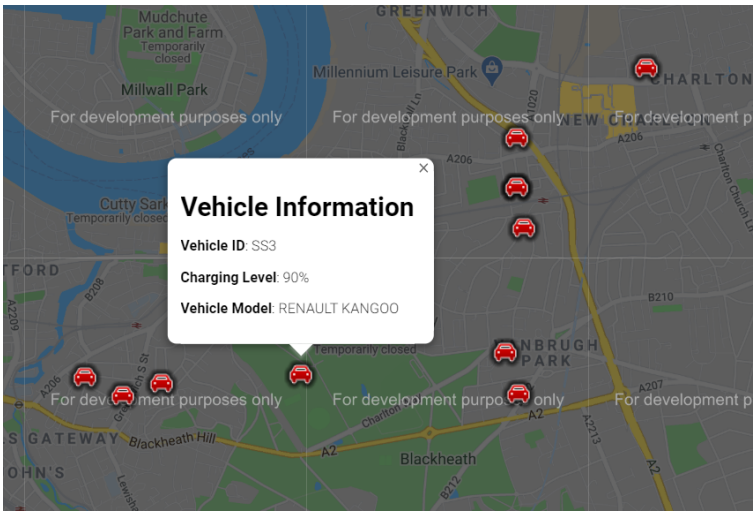


Figure 12: Vehicle Locations on map

The Code Snippet Figure 13 shows how to obtain information from the back-end database. A query is defined by the '**Parse.Query**', which is '**FleetT**' in this example. This query is used to fetch Parse.Objects from the '**FleetT**' data class. All objects that matched the query are then retrieved through '**find**' method.

As previously mentioned about the use of **ngrx/store**, line 76-79 illustrates this implementation. The '*type*' describes the action that will be dispatched in the application and the '*payload*' is for sending the data to the reducer. When the Action 'VEHICLE' is dispatched, ngrx will go through all the reducers until it finds a case for that Action. Figure 14 shows the part of the reducer function. In this instance, the case 'VEHICLE' is found and the data is then stored into the '*vehicle_info*'.

To fetch the data from the store, '**Selector**' is used to obtain the slices of the state. As Figure 15 shows, the Selector selects from the '*ManagerReducer*' and returns the properties from the '*vehicle_info*'.

```
getAllVehicle() {
  //obtain information from backend database
  let query = new Parse.Query('FleetT');//visit 'FleetT' class
  query.find().then(vehicles => {       //find result
    let fleet = vehicles.map(v => {
      return {
        lat: v.get('GeoPosition').latitude,  //get latitude of the vehicle
        lng: v.get('GeoPosition').longitude, //get longitude of the vehicle
        label: v.get('VehicleID'),           //get ID of the vehicle
        ChargingLevel: v.get('ChargingLevel'),// get charging level of the vehicle
        Model: v.get('VehicleModel')         // get model type of the vehicle
      };
    });
    this.store.dispatch({
      type:"VEHICLE",
      payload:fleet
    });
    this.navCtrl.push('MapsPage');
  }, err => {
    console.log('Error getting Evs', err)
  })
}
```

Figure 13: Code Snippet: Get vehicle information

```
export function reducer(state = initialState,action){
  switch(action.type){

    case "VEHICLE":
      return{
        ...state,
        vehicle: true,
        vehicle_info: action.payload
      }
```

Figure 14: Code Snippet: Reducer

```
this.store.select('ManagementReducer').subscribe(state => {
  this.markers = state.vehicle_info
});
```

Figure 15: Code Snippet: Selector

4.1.3 View Task Details

In this page, the manager can check every task information in detail and track the progress by selecting service department and date. It filters out tasks that do not meet the conditions and only display tasks that the manager is interested in. In each task card, the date and completeness of the task will be highlighted at the header. If the task has been completed, the header will be displayed in green, otherwise displayed in red. Other task specifications include **Employee's Username**, **Estimated Time**, **Destination** and **Service Type**. In addition to those compulsory specifications, extra information can be provided by the employee. **Stop** means any stopping points required during the task and **Reference Number** can provide a practical and efficient method to recognise and identify tasks for both employee and manager.

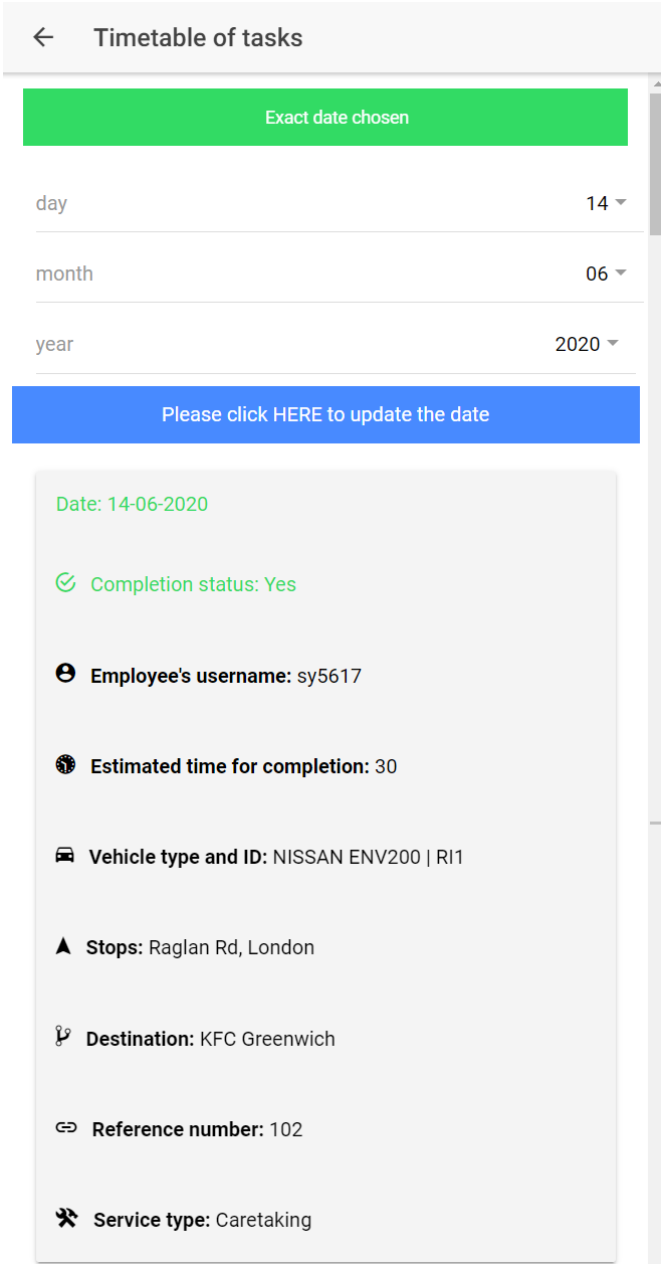


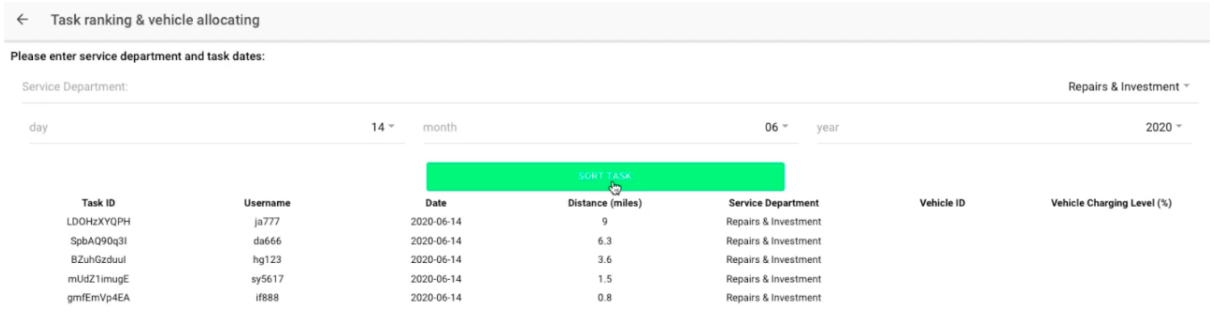
Figure 16: Task List Sorting Result

4.1.4 Allocate Vehicles

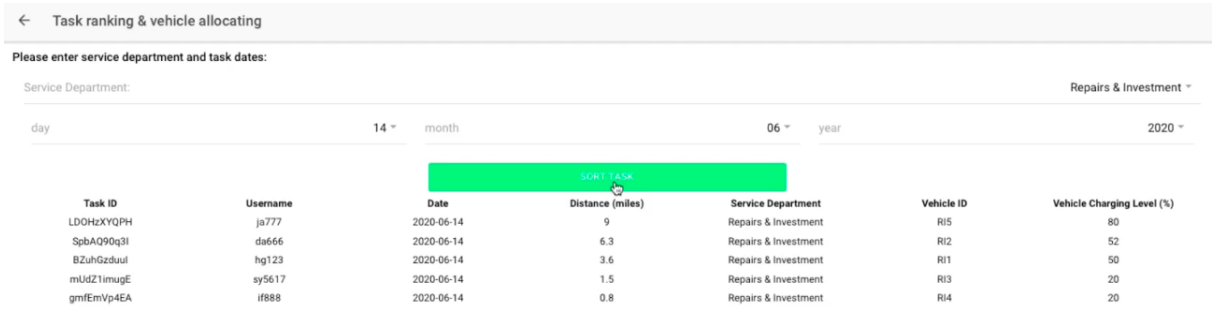
This page mainly focuses on how to maximise the general efficiency of the journey by allocating the task to a specific vehicle. Electric vehicles generally require longer time to recharge or refuel than traditional diesel vehicles. With the help of our system, the probability of recharging during a task can be considerably minimised to avoid long waiting time for charging during working hours. In addition, the economic costs can also be reduced since the shortage of the power is kept to its minimum.

Vehicles are organised to certain tasks (i.e. vehicles equipped with street service devices can only be allocated to street service tasks), thus the manager is supposed to first select a service department to filter unqualified vehicles. Figure 17a and 17b illustrate the difference of a before and after vehicle allocation.

These tasks are listed by the estimated distance in descending order, and those with the longest journey will be assigned to the vehicle with the highest morning charging level.



(a) Before Allocating



(b) After Allocating

Figure 17: Allocating vehicles to tasks

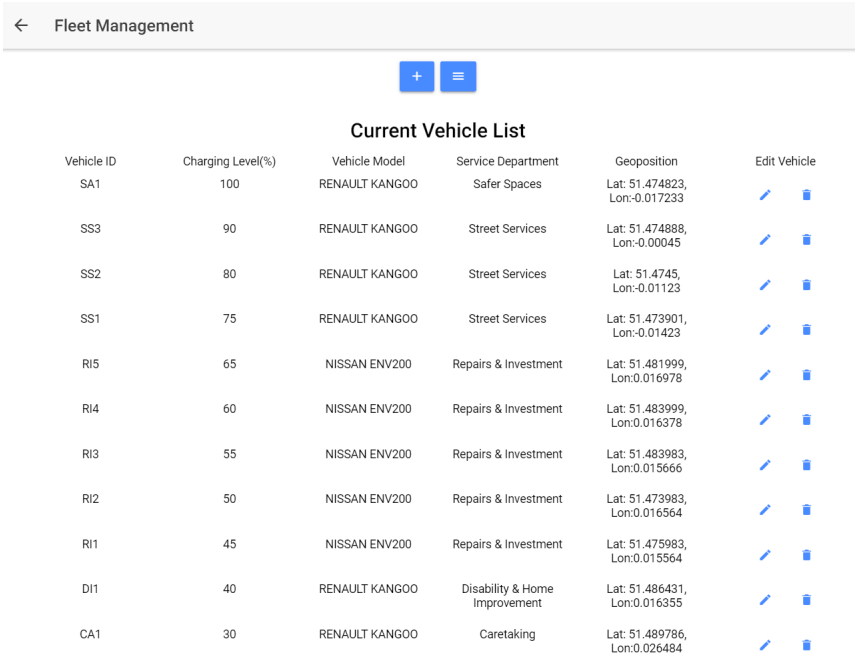
The Figure 18 describes a function that is used to rank the charging level of the vehicle fleet. Two vehicles represented by **a** and **b** are compared by their charging levels and then the comparison result is sorted by array's property function **sort** in Line 116. Similar comparison functions were also applied while ranking tasks by journey distances and dates.

```
116     this.fleetinfo.sort(this.chargcompare);
117
118 >   if(this.taskinfo.length>this.fleetinfo.length){…
131 >   }else{…
139     }
140   }
141   chargcompare(a,b){
142     const charginga=a.charging;
143     const chargingb=b.charging;
144     let comparison=0;
145     if((charginga-chargingb)>0){
146       comparison=-1;
147     }else if((charginga-chargingb)<0){
148       comparison=1;
149     }
150     return comparison;
151   }
```

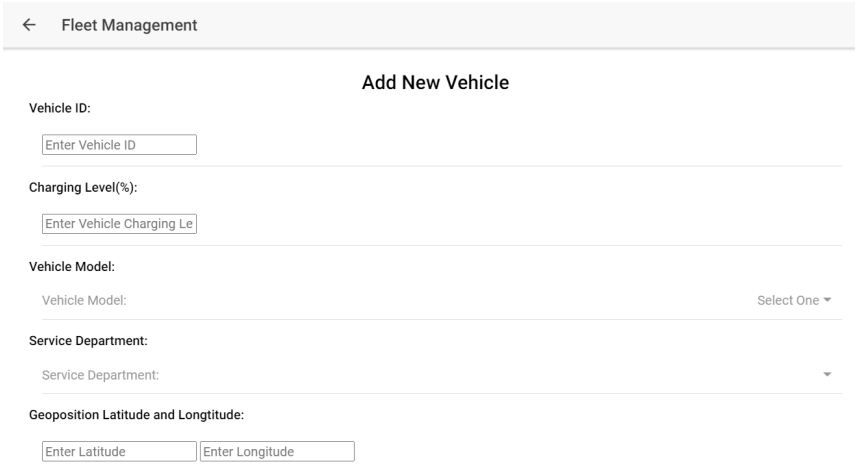
Figure 18: Code Snippet: Ranking vehicle

4.1.5 Fleet Management

Two main functions are introduced in this page, view and edit fleet information. The application will record the current fleet containing all vehicle models and service departments when the **View** icon is clicked. This list is sorted by the charging level order. Managers can either update the charging level and geo-position of the vehicle or delete this vehicle manually. Besides, managers can also add more vehicles to the fleet by filling in new vehicle details and click **Add** icon.



(a) Current Fleet List



(b) Add New Vehicle

Figure 19: Fleet Management Page

In order to update fleet information at the back-end database, the function showed in Figure 20 was used. A constraint is added to the query, which extracts the required vehicle ID that needs to be updated through '**equalTo**' method. The new information is uploaded to overwrite the original ones for each variable respectively. In this example, Variable **VehicleID** and **VehicleType** has been rewritten.

```
142  freshtask(i){
143    // front-end update data
144    let Tasks = Parse.Object.extend('Task');//go to 'Task' class
145    let tasks = new Parse.Query(Tasks);
146      tasks.get(this.taskinfo[i].object)    //update this task according to its object id
147      .then((player)=>{
148        player.set('VehicleID',this.taskinfo[i].vehicleid); // update 'VehicleID' by setting new vehicelid
149        player.set('VehicleType',this.taskinfo[i].vehicletype);//update 'VehicleType' by setting new vehiceltype
150        player.save(); //save updates
151      });
152  }
```

Figure 20: Code Snippet: Update vehicle information

In addition, by clicking the **delete** button, the vehicle can also be removed from the fleet. Figure 21 explains how to achieve this implementation. Instead of using **set** property above, property **destroy** was called to delete this object from the database.

```
163  async destroyfleet(i){
164    //delete data
165    let Fleets = Parse.Object.extend('FleetT');
166    let fleets = new Parse.Query(Fleets);
167    fleets.equalTo("VehicleID", this.fleetinfo[i].vehicleid);
168    const result = await fleets.find()
169    console.log('ID: '+result[0].id);
170    await fleets.get(result[0].id)
171    .then((player)=>{
172      player.destroy();
173    });
174  }
```

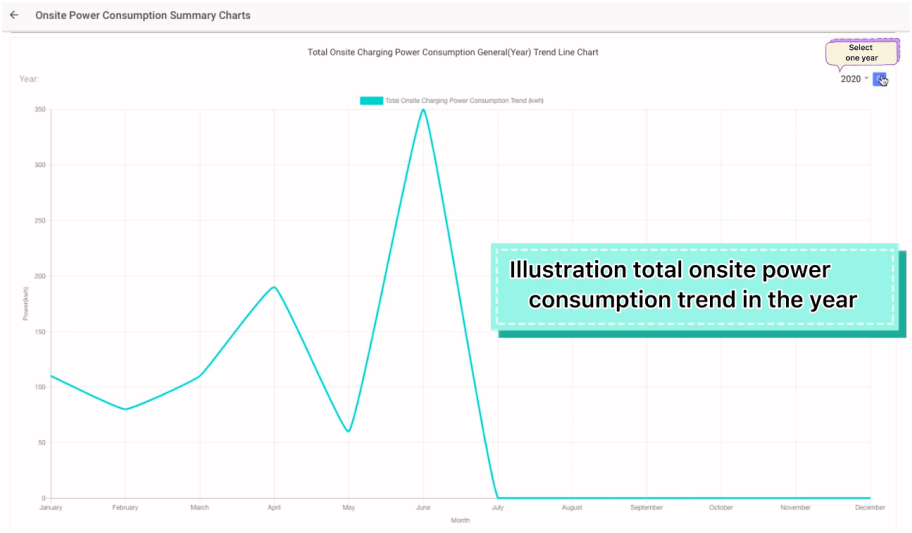
Figure 21: Code Snippet: Delete vehicle information

4.1.6 View Power Consumption

The primary purpose of this page is to provide onsite power consumption summaries to the manager. The maximum power consumed by each service department on a particular day is illustrated by a bar chart Figure 22a. A pie chart Figure 22b displays which service department consume the most power during a selected month. Figure 22c offers an annual trend graph, which compares the entire fleet's power consumption. Consequently, the manager can plan to expand the fleet of a specific service department based on its power demand.

(a) Power Consumption by Service Department in Day

(b) Power Consumption by Service Department in Month



(c) Power Consumption Trend in Year

Figure 22: Onsite Power Consumption Page

4.2 Employee Interface

Employee side application is designed for employees to check in task details to database, obtain driving navigation information as well as receive necessary notifications. With the help of this application, employees can complete and track task processes in a paperless and efficient method. The employee application home page is shown in Figure 23.

Figure 23: Employee Home Page

4.2.1 Employee Profile

Similar to the manager application, there is a profile page for employees as well. Employees can review and edit their contact and security information which was originally saved at registration. Figure 24 shows details of the profile page. Those records are also accessible directly through back-end database.

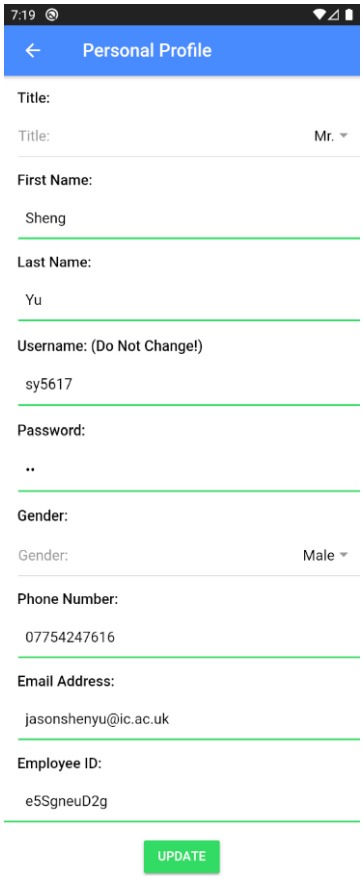


Figure 24: Employee Profile Page

The employee accounts used for the demo purpose is given in Table 3:

Username	Password
sy5617	ic
hg123	ic
da666	ic
ja777	ic
if888	ic

Table 3: Employee Application Account Details

Please either use the above login information to login from the employee application or sign up new employee accounts.

4.2.2 Map View of Charging Points

This page displays all the charging sites in Greenwich area (Figure 25). Green pin indicates that all the connectors are available, whereas blue pin indicates that some connectors are occupied. When a pin is clicked, the info window illustrates that the total and available numbers of three different connectors. The '**Navigate to here**' button shows the directions from driver's current location to this charging station in Google Maps, thus no need for the driver to manually open another application to search for the route. In addition, charging point availability was assumed to be static.

4.2.3 View Task List

This page lists all the previous and current tasks. Colour-coded was used in the Task Reference to imply the completion status of the task. Task Reference highlighted in red indicates the status

of the task is incomplete, whereas the finished task is illustrated in green. Figure 26 shows a screenshot of this page. Click a Task Reference will then navigate to another page, where all the task details are shown.

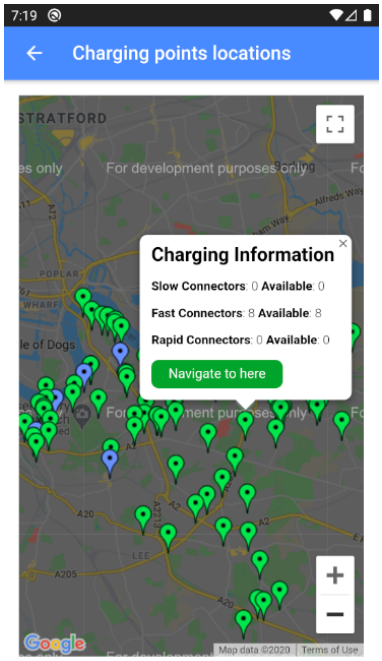


Figure 25: Charging Points in Greenwich

Figure 26: Task List Page

4.2.4 Task Detail

All details of the specific task are displayed in this page. The task information can be checked or deleted at anytime. The employee can tick the '**Complete Check**' box at the bottom of the page which is shown in Figure 27 to update the completion status.

In addition, the '**Navigate your route**' button redirects the app to Google Maps and a planned driving route is shown based on the **Stop** and **Destination** details saved before.

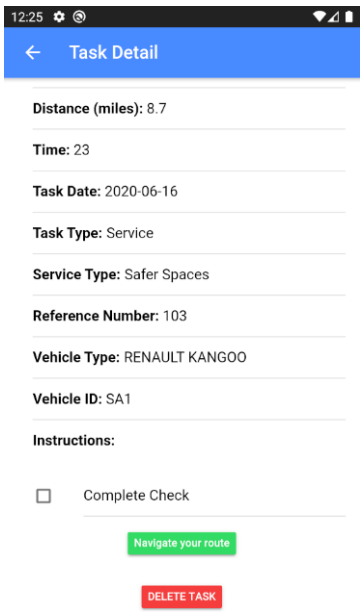


Figure 27: Employee Task Detail Page

4.2.5 Create New Task

This page allows the employee to fill all the details, in order to plan the journey throughout the day. Once a new task is successfully created, it will be added to the top of the task list and wait for a manager to allocate a suitable vehicle.

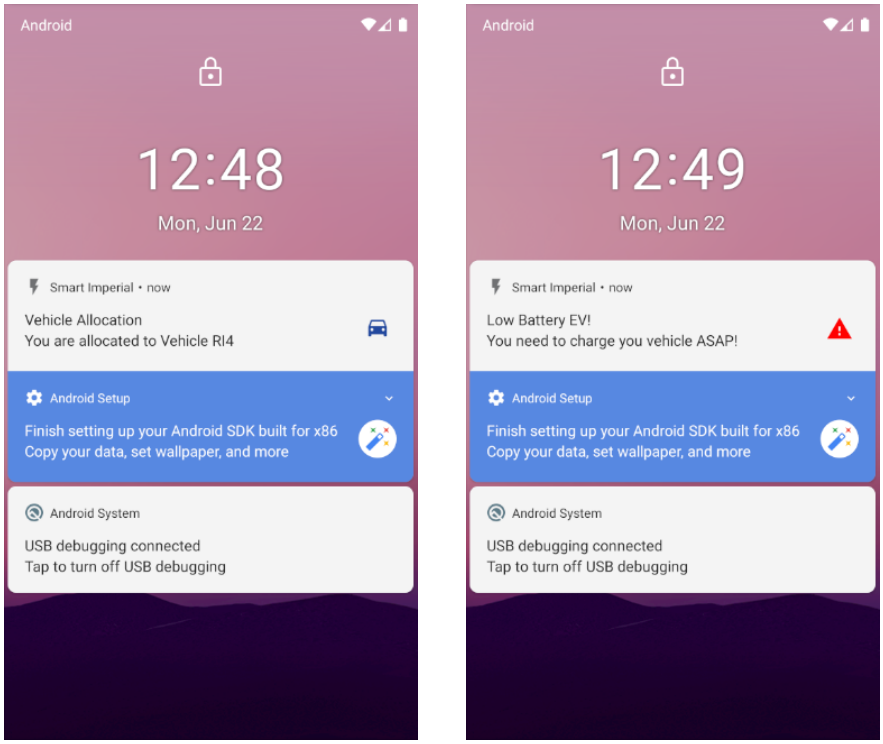
The Code Snippet Figure 28 explains how to create a new object and fill in information at the back-end database. To begin with, an object called **Task** was created by line 96. The next step was to set up all variables and assigned information to them by using **set Parse** property. After saving all data successfully, local variables were required to reset so that they can be used for assigning new classes.

```
96  const Task = Parse.Object.extend("Task"); //create 'Task' object at backend database
97  const task = new Task();
98  //set 16 variables for 'Task' objcet
99  task.set("Username", this.username);
100 task.set("StopPoint", this.stop);
101 task.set("Destination", this.destination);
102 task.set("Distance", this.distance);
103 task.set("EstimatedTime", this.time);
104 task.set("ReferenceNumber", this.number);
105 task.set("Date", this.date);
106 task.set("Day", this.date.slice(8,10));
107 task.set("Month", this.date.slice(5,7));
108 task.set("Year", this.date.slice(0,4));
109 task.set("Complete", this.complete);
110 task.set("VehicleType",this.vehicletype);
111 task.set("VehicleID", this.vehicleid);
112 task.set("TaskType", this.tasktype);
113 task.set("ServiceType", this.servicetype);
114 task.set("Instruction", this.instruction);
115 task.save() //save uploades
116 .then((player) => {
117   //Save succeed:
118   //reset local variables
119   this.username = '';
120   this.stop = '';
121   this.destination = '';
122   this.distance = '';
123   this.time = '';
124   this.number = ' ';
125   this.date = '';
126   this.vehicletype = '';
127   this.vehicleid = '';
128   this.complete = '';
129   this.tasktype = '';
130   this.servicetype = '';
131   this.instruction = '';
132   alert('New task is created successfully!');
133   this.navCtrl.setRoot('HomePage');
134 }, (error) => {
135   // Save fails
136   alert('Failed to create new object, with error code: ' + error.message);
137 });
```

Figure 28: Code Snippet: Create Task

4.2.6 Push Notifications

When a vehicle is assigned by the manager, the employee will receive a notification which is displayed in Figure 29a. Furthermore, when the battery level of the vehicle is running below 30%, an alarm notification (Figure 29b) will be immediately pushed to alert the employee. By clicking the notification bar, it will then navigate to the nearest charging site on Google Maps to ensure that the vehicle has enough battery to continue the journey.



(a) A vehicle has been assigned to the task (b) A vehicle has low power

Figure 29: Notifications of Employee Application

To make sure that a notification will be triggered once the data is updated in the back-end, **Parse Live Query** is introduced in this case. It allows to subscribe to a query that we are interested in. Once subscribed, the server will notify clients whenever an object that match the query is changed in real-time. Figure 30 illustrates the implementation of this method. The variable '*subscription*' is initialised to subscribe the '*Taskquery*'. The '*subsciption.on*' function indicates that whenever the query is updated, implement the rest of the actions inside this function.

Figure 31 shows the push notification function. A cordova plugin has been used, which creates the object cordova.plugins.notification.local. The '*schedule*' function schedules a notification when the device is ready. Line 224-225 means that it will navigate to the Home Page of the application when the notification bar is clicked.

```
269   async RetrieveVehicle(){
270
271     var Task = Parse.Object.extend('Task');
272     var Taskquery = new Parse.Query(Task);
273     let subscription = Taskquery.subscribe();
274
275     Taskquery.equalTo('Username',this.username);
276     Taskquery.equalTo('Date',this.date);
277
278     subscription.on('update', (object) => {
279       // console.log('UPDATED');
280       this.vehicle = object.get('VehicleID');
281       // console.log("id: is ",this.vehicle);
282       this.PushNotificationAllocate();
283       subscription.unsubscribe();
284       this.RetrieveID(this.date);
285     });
286
287   }
```

Figure 30: Code Snippet:Live Query

```
PushNotificationAllocate(){
  this.localnotification.schedule({
    id: 2,
    title: 'Vehicle Allocation',
    text: 'You are allocated to Vehicle '+this.vehicle,
    foreground: true,
    icon: 'res://car',
    smallIcon: 'res://ic_stat_flash_on',
  });
  this.localnotification.on('click').subscribe(notification => {
    this.navCtrl.push('HomePage');
  });
}
```

Figure 31: Code Snippet:Push Notification

4.3 Back-end Database

This sections explains the details and management of the back-end in the Back4App.

The application is named as **DemoApp** which is displayed by Figure 32. **User** class contains key information of all manager and employee users while **Employee** and **Manager** classes save registration records. **LoginHistory** class records login history every time when a user login to the system either from manager or employee application. In addition, **ChargingPoint**, **FleetT**, **OBD** and **Site** classes save hardware information. **Task** class records details of all tasks. Current data stored at the database is for demo purpose only. The client can update in the future according to the development of local infrastructure development.

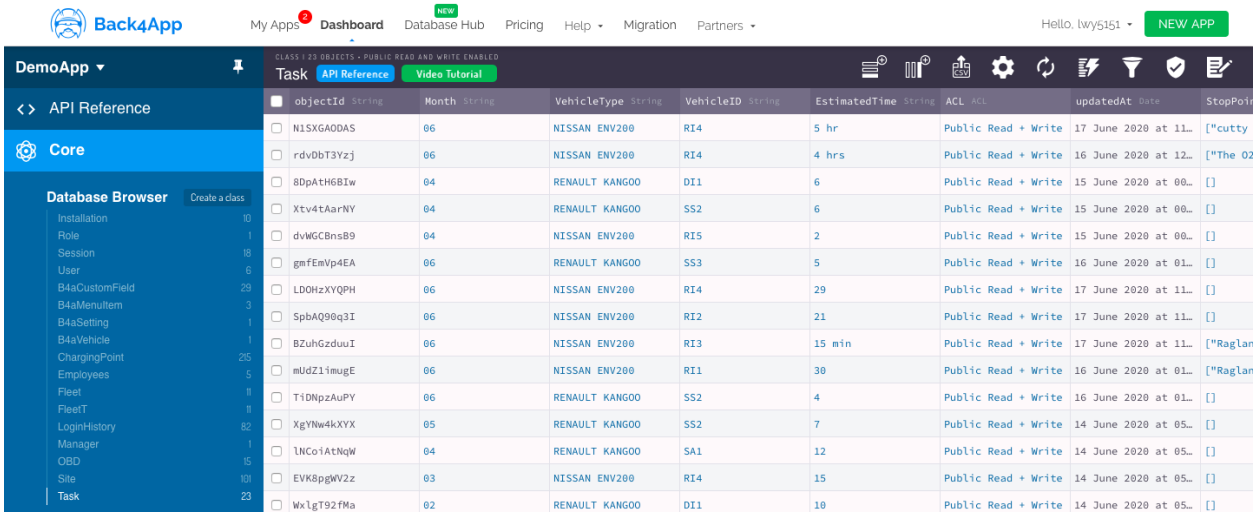


Figure 32: Back-end Database Dashboard

The database is currently under the account of:

Email: lwy5151@sina.cn
Password: 123456

This database is also able to be fully transferred to client's account. The database owner can include more collaborators at the **App Settings** tab by adding email addresses. To develop other applications based on this database, relevant keys of the database can be found at **Serve Settings-Core Settings** tab.

4.4 Discussions on Ethical Consequences and Sustainability

4.4.1 Introduction

Given that the app is tailored towards the management of electric vehicles for different individuals, organisations and governments, the use of electric vehicles is supported by this app. As well as providing fleet management of electric vehicles, the app also allows for real time tracking of the electric vehicles and other on-boarding diagnostics capabilities. Therefore, this section focuses on sustainable and ethical consequences of applying the app in real-life environment.

4.4.2 Advantages

Carbon emission reduced
As a fact, all electric vehicles produce zero direct carbon emissions as combustion engines in conventional cars are replaced by batteries. From the European Environment Agency, it is clear that transport is responsible for nearly 30% of the EU's total CO2 emissions and 72% of that comes from road transportation [4]. With the general consensus now to reduce CO2 emissions, the EU has set a goal to reduce emissions from transport by 60% by the year 2050. These carbon emissions adversely affect the climate in the long run. With the app directly encouraging and supporting the use of electric vehicles, the app will support this reduction in carbon emissions.

Reduction in noise pollution
One of the major causes of the noise pollution on the road are from cars. Combustion engine vehicles have a lot of noise produced from its operation. In the case of electric vehicles, this noise is greatly reduced as combustion engines are replaced by batteries. Batteries produce little or no noise when compared to combustion engines. As the app directly encourages and supports the use of these electric vehicles, the app supports the reduction in environmental noise pollution.

Reduction in production of harmful tailpipe pollutants
Tailpipe emissions are one of the biggest reasons for the adverse effects on the climate today. The tailpipe emissions on a typical combustion engine car includes nitrogen oxides, hydrocarbons, sulphur dioxide, ozone etc. Each of these gases will cause adverse effects on the environment including greenhouse effect, smog, acid rain and in worst cases even cause respiratory illness in individuals [5]. With the app directly encouraging and supporting the use of electric vehicles which do not produce these pollutants, the app will support this reduction in the production tailpipe pollutants.

Lower operating and maintenance costs and higher efficiency
According to a 2018 study by the University of Michigan's Transportation Research Institute, it is seen that nonelectric vehicles will cost more than twice as much to operate and maintain than electric vehicles. Electric vehicles in the US will cost on average 485 dollars per year while fuel-powered vehicles will cost on average 1117 dollars a year[6]. With the app directly encouraging and supporting the use of electric vehicles, the app will allow for more efficient use of vehicles through lowering maintenance and operating costs.

Many new EVs are manufactured using renewable materials
Although this is not the case for all electric vehicles, many new models for electric vehicles including the ones used by the first client – the Nissan cars used by the borough of Greenwich are made using recyclable materials including the seats, doors, panels and dashboards [7]. Furthermore, the batteries used to power electric vehicles are very recyclable as they can be used after it worn out in other applications. As the app directly encourages and supports the use of these electric vehicles, the use of renewable materials which are beneficial to the society is encouraged.

4.4.3 Disadvantages

Tracking drivers
The app allows for the direct tracking of location and other car information like battery level through the on-boarding diagnostics equipment in the car. Tracking the drivers could pose a serious mental demoralisation to the drivers involved as they can feel like there is a lack of trust in the organisation towards them. This could also lead to resentment from the drivers towards to the organisation adopting the real time tracking capabilities of our app.

Jobs will be lost during EV production
The app supports the use of electric vehicles. As the demand for electric vehicles increases, factory workers would need to learn new skills to cater for the production of electric vehicles. Unfortunately, there will be jobs in mechanical and automotive engineering that will be lost as the assembly of electric vehicles require fewer components so it can be automated [8]. These changes however will allow for openings in our technical roles involving the vehicles so it is not completely negative.

Long recharging times can cause road blockages and traffic
The app supports the use of electric vehicles and provides capabilities to find charging points and offer optimized driving route planning. Electric vehicles take longer to recharge compared to the fuelling of combustion engine vehicles. With most of these charging stations being on the side of the roads, the EVs will most often take up space on the road and lead to more congestion on the road and in worst cases even lead to traffic jams. However, the app tackles this by finding optimized driving routes tailored towards distance left to location and charge level in the electric vehicles.

4.4.4 Conclusion

From the research on the various ethical consequences of the app, it is clear that the app completely promotes the use of electric vehicles leading to the support in reduction of pollutants, greenhouse gases, noise pollution and other factors that adversely affects the environment. Furthermore, the possible ethical negatives incurred in the use of the app which includes loss of jobs due to a change in skills needed and tracking drivers that can adversely affect employee's morale was considered [9]. However, whilst looking into these negatives the app was modified in a way that mitigates this. Hence, it can be concluded that the app has a lot of primary and secondary ethical implications that would benefit the society and the app has also been able to mitigate the possible ethical negatives involved with using it.

5 Evaluation

In this section, those ideas that have arisen throughout the project design, but were not fully realised due to time constraints, would be covered. This also serves as a check list for possible future improvements that could be brought to live by users of the app.

5.1 Management-side app

5.1.1 Full Google Maps

Throughout the whole app development process, our group has primarily used the limited version of Google maps, which, as the name suggests, only entails its restricted functions. The app user (usually from the employee-side), therefore, was always re-directed to the full version of the maps, once clicking on the corresponding button. The use of the full version of Google Maps in the future would thus ensure that users always stay inside the app, without passing on to other applications.

5.1.2 Live connection to EVs

In order to implement an efficient vehicle allocation algorithm, real-time data from electric vehicles should be recorded. As already mentioned, the device used for this purpose is called an 'On-board diagnostic' (or simply OBD). The final choice of OBD scanners works by establishing a Bluetooth connection with a mobile phone, which in its turn passes all the data to the back-end-server. More information on OBD scanners is given below in section 5.3.

5.1.3 More advanced vehicle allocation algorithm

Currently, our vehicle allocation algorithm only takes two factors into account - the starting charging level of a vehicle and total charge consumption in certain task list. Although working well for a small-number of vehicles (as of now), this assumptions are very rough for a large fleet. More data should therefore be taken into account one the fleet is expanded - for example, journey time to a charging point and total charging time.

5.1.4 Alerts for on-site charging shortage

In the current model, the total number of on-site charging points is more than double that of electric vehicles. This thus ensures that every vehicle could be found a charging point overnight. However, this proportion could be easily changed in the future, as new vehicles are still bought up. It would therefore be useful to inform managers in cases when there are more cars waiting to be charged than there are charging points available.

5.2 Employee-side app

5.2.1 An introduction for employees

After having the last meeting with DG Cities, the team was recommended to set up an introductory video or a guide for employees in their first use of the application. This is for the purpose of simplifying the adaptation for transferring to the new platform, and thus should entail a description for all the main aspects of the app.

5.2.2 Obtaining real-time information from charging points

At the moment, all information about the charging points is obtained from a static Excel sheet. In the future, it would be useful to take live information into account as well - for example, whether a specific point is broken, or whether it's already occupied by another driver. One possible solution is using Open Charging Point Protocol (OCPP). This is an application protocol that supports EV charging points and central management system to communicate with each other.[10] Over these years, OCPP has been expanded and became popular in more than 50 countries. A diagram which explains the concept of the OCPP is shown in Figure 33. With the help of this technology, the charging stations are able to transfer the data to the back-end; thus, the employee side application can access the information from the back-end and display real-time data in the '*Charging Point*' page. Although the charging stations are yet not OCPP-compliant in the Greenwich Borough, the

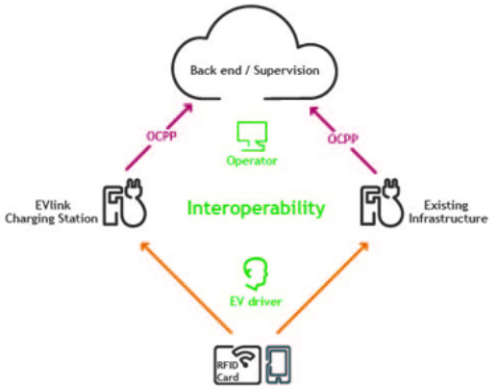


Figure 33: OCPP[12]

use of OCPP (or equivalent) is required for new charging points in the UK from 2019 to meet the *Electric Vehicle home charge Scheme requirements*.[11] In the long run, this application would become a trend and be widely used, hence this proposal can be further developed.

5.2.3 Drivers are given more information on charging

Although the app mostly focuses on task creation and completion rather than charging, it could still relatively beneficial to facilitate the process of car charging for drivers. For the future expansion, the app can therefore entail more information about charging process - such as any tasks to be completed nearby and how much time would charging take.

5.2.4 Simplify the task creation process

The current version of the app makes employees enter all the details about every task that they create (which is usually about 8-9 spaces). This can be a quite tedious process for drivers, and in theory could be optimised by either making the some less essential fields optional, or by implementing an algorithm which suggests answers for drivers from previous entries.

5.3 Obtaining real-time information from on-boarding diagnostics (OBD) scanners

5.3.1 Introduction

In order to obtain real time information on the position of the vehicle and the charge left in the vehicle some sort of telematic device would need to be implemented in the car. The chosen telematics device would be an On-Boarding Diagnostics (OBD) scanner and there are several options when it comes to implementing this telematic device. After research, Bluetooth, Wi-Fi and Cellular were the only available methods of information transmission for the telematic devices. Due to the current unprecedented situation – COVID-19, it was not possible to implement the telematics devices in person. Hence, this section would explore what methods of data transmission to prioritize and how they can be implement in the telematics device in the future.

5.3.2 Categorized by methods of data transmission

Wi-Fi Connection
After considering this method of transmission, it was decided that this would not be suitable for the application. The reason being that once the driver (employee app interface) is connected via WIFI to the OBD scanner, their mobile network connection is automatically disconnected. Hence, real time data from the car cannot be transferred to the management side via the back-end (Back4App Cloud) and so, this method will be disregarded. Figure 34 shows the flowchart for an OBD device with Wi-Fi connectivity.

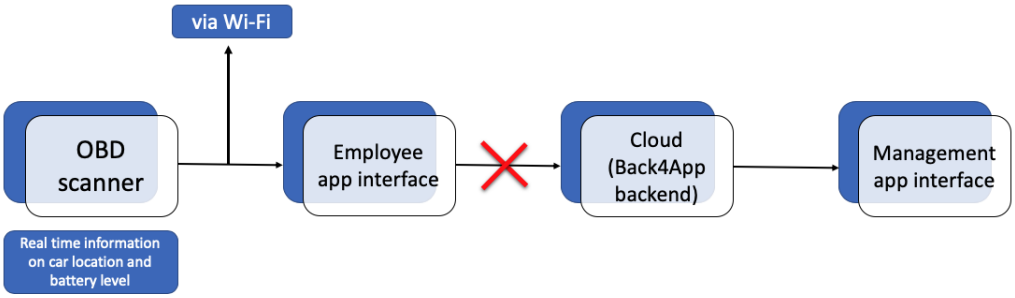


Figure 34: Flowchart for OBD with Wi-Fi connection

Bluetooth Connection
This method of data transmission will be suitable for the application. In order to receive the real time information, the driver (employee app interface) will pair via Bluetooth with the OBD scanner, then the employee app interface will obtain the real time information on the vehicle's location and charge level. This information is then sent to the back-end cloud which then processes it through the algorithm to send real time information to the managers (management app interface). This is shown in Figure 35. This method was not implemented as the devices could not be physically placed in the vehicles for testing due to the virus situation. The proposed workflow for the OBD scanning feature will be:

1. Connect to the OBD2 scanner via Bluetooth: This will be done by the driver (employee-side interface) outside of the app
2. Initialize the OBD2 scanner with inbuilt AT commands: Attention commands are needed to turn on the OBD scanner. This will be sent by the app.
3. Use the Parameter Identification (PID) codes to obtain real time information on chosen parameters

Parameter identification (PID) are codes that can be sent to the OBD scanners to retrieve information. The codes relevant to the app's operation are

- PID Code - 01 2F - To retrieve battery percentage level
- PID Code - 01 33 - To retrieve GPS position

Java API *java.io.inputstream* and *java.io.outputstream* will be used to obtain raw data and send data to the OBD scanner. The proposed pseudo-code is detailed below.

```
newString command //to store PID codes to be OBD input
newString result //to store OBD output

public void run(InputStream in, OutputStream out)
    sendCommand(out) //function to send out PID code to OBD scanner
    readResult(in) //function to read string output from OBD scanner

public void sendCommand(OutputStream out)
    out.write((command+"\r").getBytes()
    //writes to output stream, adds a new line between characters in string
    //then converts to
    out.flush()

public readResult(InputStream in)
    //loop will end when input stream is a null value
    //i.e byte value is zero in the loop, add in.read() values to result
```

```
while (((b = (byte) in.read() != 0){
    c = (char) b
    result.append(b)
```

The code will then be interpreted real time in the app using the algorithm to obtain real time values of battery percentage level and GPS position. These values are then sent to the back-end.

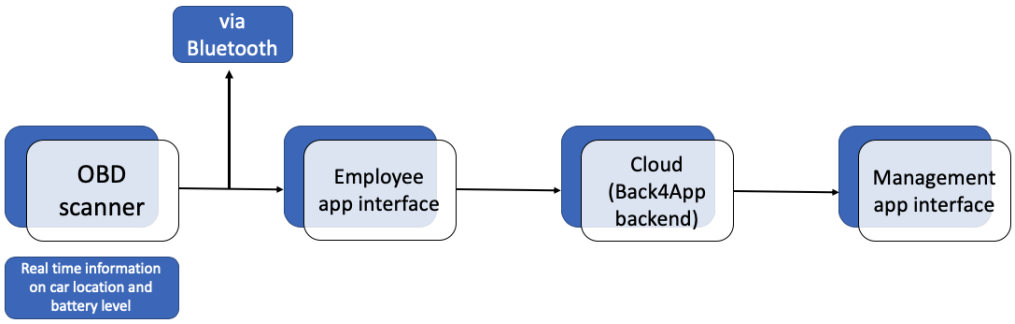


Figure 35: Flowchart for OBD with Bluetooth connection

Cellular Connection

This method of data transmission for the OBD scanners is also considered. It will be suitable to the application as it can provide real time information on the vehicle directly to the back-end, so there is no need for the driver to connect to the OBD scanner. See Figure 36 for flowchart of an OBD scanner with cellular connectivity. For cellular connectivity, a list of companies have been compiled. Zubie and Munic were chosen to be top priority. However, this method was not implemented because at the time, a local host was used for the testing the app and the companies involved need a stable URL in order to be able to implement their products. Also, real time test data is not available on their website for testing and with the virus situation, it is not feasible to implement the device in person in order to get real data for testing. Moreover, a monthly payment is needed to use the cellular OBD scanners. However, this method proves very promising as there is a lot of online resource code provided by the companies (after buying the product) to help with setting up the commands for the OBD scanner.

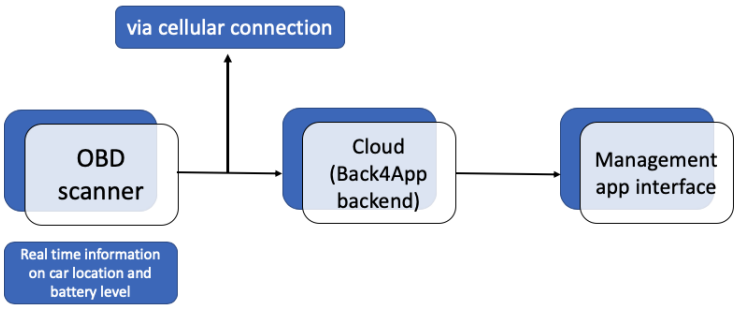


Figure 36: Flowchart for OBD with Cellular connection

5.3.3 Conclusion

In summary, in order to implement the real time data retrieval from the vehicles as a future improvement, the app will consider using Bluetooth or cellular as a means of data transmission. At the moment, the Bluetooth method is the most feasible as a pseudo-code has been developed for future implementation and it is far cheaper than the cellular connectivity option.

6 Overall conclusion

To conclude, the app has been developed to the stage of a working prototype, in which all parts are working together as expected, and management-side app is connected with the employee-side. All the features listed in the initial specification were fully realised, and even some additional tools were suggested and developed by the group. Usage of app development tools, such as Ionic and Angular, alongside with TypeScript and HTML programming languages has enabled to build an effective yet robust solution. The future phase of the project should be its testings with real-life EVs and their date, as well as the apps' implementation into Royal Borough of Greenwich fleet operation.

References

[1] Ionic Framework Resources. What is hybrid app development? https://ionicframework.com/resources/articles/what-is-hybrid-app-development.

[2] Microsoft Xamarin Docs. What is xamarin? https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin.

[3] NGRX. @ngrx/store. https://ngrx.io/guide/store.

[4] European Parliament. Co2 emissions from cars: facts and figures. https://www.europarl.europa.eu/news/en/headlines/society/20190313STO31218/co2-emissions-from-cars-facts-and-figures-infographics.

[5] Climate Change Connection. Tailpipe emissions. https://climatechangeconnection.org/emissions/tailpipe-emissions/.

[6] Energy Sage. Costs and benefits of electric cars vs. conventional vehicles. https://www.energysage.com/electric-vehicles/costs-and-benefits-evs/evs-vs-fossil-fuel-vehicles/.

[7] Admiral. The environmental pros and cons of electric cars. https://www.admiral.com/magazine/guides/motor/the-environmental-pros-and-cons-of-electric-cars.

[8] Autovista Group. Severe job cuts looming as germany moves to ev production. https://autovistagroup.com/news-and-insights/severe-job-cuts-looming-germany-moves-ev-production.

[9] Wikipedia. Environmental aspects of the electric car. https://en.wikipedia.org/wiki/Environmental_aspects_of_the_electric_car#Advantages_and_disadvantages.

[10] Greenlots. Open vs. closed charging stations: Advantages and disadvantages. https://greenlots.com/wp-content/uploads/2018/08/open-standards_white-paper_05.pdf.

[11] Wikipedia. Open charging point protocol. https://en.wikipedia.org/wiki/Open_Charge_Point_Protocol.

[12] Schneider Electric. Open charge point protocol : connecting ev charging stations to central systems. https://blog.se.com/electric-vehicle/2015/05/05/open-charge-point-protocol-connecting-ev-charging-stations-central-systems.

A Appendix

A.1 Project Brief

Task

•Design and create an app/software to be used by council employees, to optimise the way they use and are allocated electric vehicles.

- NB: vehicles can be exchanged within groups of services, however not across groups. Example: drivers for Repairs & Investment can be allocated any of the R&I Nissan e-NV vehicles listed above, but not any of the Renault Kangoos as those are allocated to other services. Drivers of the Street Services team can be allocated any of the three Renault Kangoo vehicles that are allocated to Street Services only.

•App should have two interfaces:
- employee side (tasks, journeys, charger locations)
- management side (overview of vehicles, onsite chargepoints' status, journeys, potentially tracking vehicles on a map live)

•Assume there are 11 drivers (1 for each vehicle) who get allocated a vehicle within their Service, but not always the same vehicle. Assume jobs get allocated in the morning of each day.

- Initially will need to add in hypothetical information for each driver, like the following examples: [Text Wrapping Break] Service X - Driver XA - 3 stops during the day, 2 - 1 - 3hrs of tasks and in total 50km travelled. [Text Wrapping Break] Service X - Driver XB - 3 stops during the day, 1 - 1 - 1hrs of tasks and in total 20km traveled. [Text Wrapping Break] Service Y - Driver YA - 1 stop (4 hours), total 15km travelled.
- Second phase, team can link real-world data streams from council's job allocation system.

•Team will need to consider vehicle charge levels, charging on-site and available charging points in the area.

•Team can consider various devices that plug into the on-board vehicle system (telematics); choose and procure one to use for the project.

•The app should allocate vehicles based on morning charge level and expected distance to travel for jobs. [Text Wrapping Break]
•The app should also notify the vehicle's user whether there are any available charging points near the daily jobs and if it is advisable to get a charge.

•Potential to build in reassessment of charge level (thus recommendation for charging) throughout the day (e.g. if additional ad-hoc tasks allocated, caught in traffic etc.)

•Pilot with several employees – does it work? User friendly? Perceived benefits?

A.2 Sources of Code

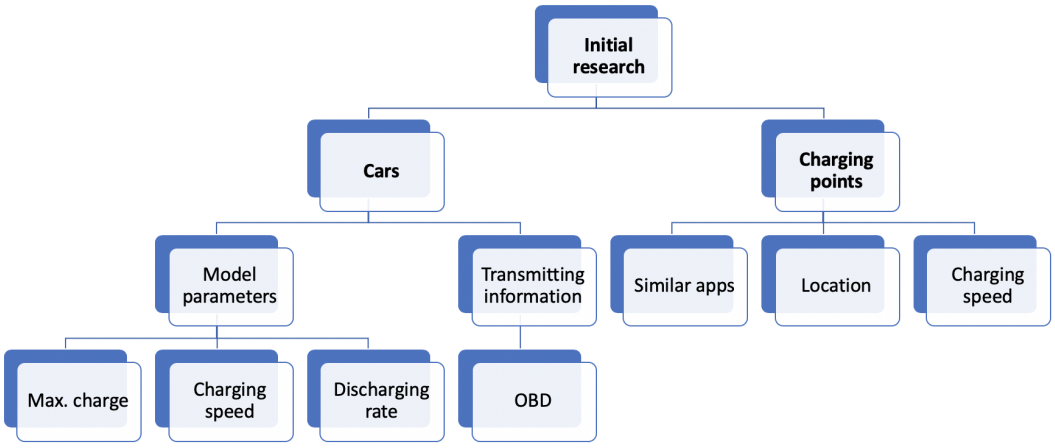
In this part, there are two tree diagrams illustrating the organization of codes for each application. Each page of an application has one specific folder which was marked as(·) below. Each folder furthermore has four files supporting this page, incluing **".ts"** setting functions, **".module.ts"** setting routes, **".scss"** setting layout styles and **".html"** setting layout to define one page. Other folders are assisting documentations and environment setups. The connection between front-end and back-end was achieved by setting **Parse KEY** at **∗app** folder. Please refer to Github links provided below to access the full code.

Management Application: https://github.com/lengwe/Smart_Imperial_Management
Employee Application: https://github.com/Sheeeng222/Smart_Imperial_Employee

- Management App
  - async
  - resources
  - typings
  - pages
    - app
    - assets
    - theme
    - pages
      - home
      - login
      - profile
      - register
      - store
      - maps
      - task
      - ranking
      - fleet
      - power

- Employee App
  - async
  - resources
  - typings
  - pages
    - app
    - assets
    - theme
    - pages
      - home
      - login
      - profile
      - register
      - store
      - map
      - createtask
      - tasklist
      - taskdetail

A.3 Week 2 Research Structure



A.4 Week 2 Clarifying Questions

Week 2 Clarifying Questions

Industrial Supervisor

1. Is license for the vehicles?
2. Are the drivers limited by lockdown?
    a. If limited, how much and what steps should we consider?
3. What is the largest distance that the driver can travel?
    a. Can you classify this by types of services?
4. Can you please send the available charging points in the area?
5. Can the cars be charged overnight?
6. Can you classify charging time for each type of car?
7. REAL-TIME system available?
8. Smart and non-smart chargers?
9. Do we need to plan the route for the driver?

Academic Supervisor

1. Leaflet
    a. Can we divide it as follows?
        i. Part 1: Background introduction – how the situation is, what is the problem?
        ii. Part 2: How does our project solve the project – using more electric vehicles to be more efficient we create an app etc.
    b. Can we make up some numbers to support the leaflet?
2. Should we include telematics in our solution?
    a. If yes, how should we approach it?

A.5 App development breakdown

	Programming language	Specific device used	Back-end tool	Front-end tool	Useful recources (e.g. online tutorials)	Tasks to complete	Notes
Management-side	**JavaScript**	Computers - website	Name: **Back4App** Login details: Individual details	Name: **Visual studio (VScode** - need to install) + **Ionic** (extension) Login details: Individual details		Login Screen Vehicle - map view Tasks list Charging projection Fleet & charging management	Red - very important Black - quite important Grey - not important
Employee-side	**JavaScript**	**??? (waiting on DG Cities) - not important for now**	Name: **Back4App** Login details: Individual details	Name: **Visual studio (VScode** - need to install) + **Ionic** (extension) Login details: Individual details		Current tasks management Charging: journey + time Car information	
Github folder - Front-end	Login details	Branches					
Added manually	Individual details						