

D1 Report.

Word Count: 6188

Cale Clark (H00296909)

Kyle Dick (H00301592)

Andrew McBain (H00317910)

Jake Paterson (H00323303)

Scott Valentine (H00301415)

Alexander Wickham (H00324206)

Contents

Background & Motivation	4
The Current Solution	4
Reliability Concerns	5
Privacy Concerns.....	6
Accessibility Concerns.....	6
Proof of Concept.....	6
Zapmap	6
ChargePlace Scotland.....	7
Requirements	9
Research & Requirements Decisions	9
Functional Requirements.....	9
Non-Functional Requirements	11
Task Specification	12
Task A - Creation of Front End for System Clients	12
Description.....	12
Task B - Creation of Front End for System Administrators	13
Description.....	13
Task C - Creation of Map View for End User and Admin Sites	14
Description.....	14
Task D – Database Creation and Management	15
Description.....	15
Task E - Backend Account System.....	17
Description.....	17
Task F - Backend Queue System.....	18
Task G - Backend Administrative System.....	19
Description.....	19
Task H - Hosting and Deployment	20
Description.....	20
Technical Risk Table	21
Software Integration Plan	24
Table of Sub-tasks.....	26
Gantt Chart.....	27
Requirement Traceability Matrix	27
Table of Responsibilities.....	29
Appendix A: References	30

References.....	30
Appendix B: Tables.....	31
Table of Figures.....	31
Table of Tables.....	31

Background & Motivation

This document describes the design and technical specifications for a system that would aid electric vehicle users in reliably locating available charge points. The scope of this system is concerned specifically with the charge points located on Heriot-Watt's Hermiston campus.

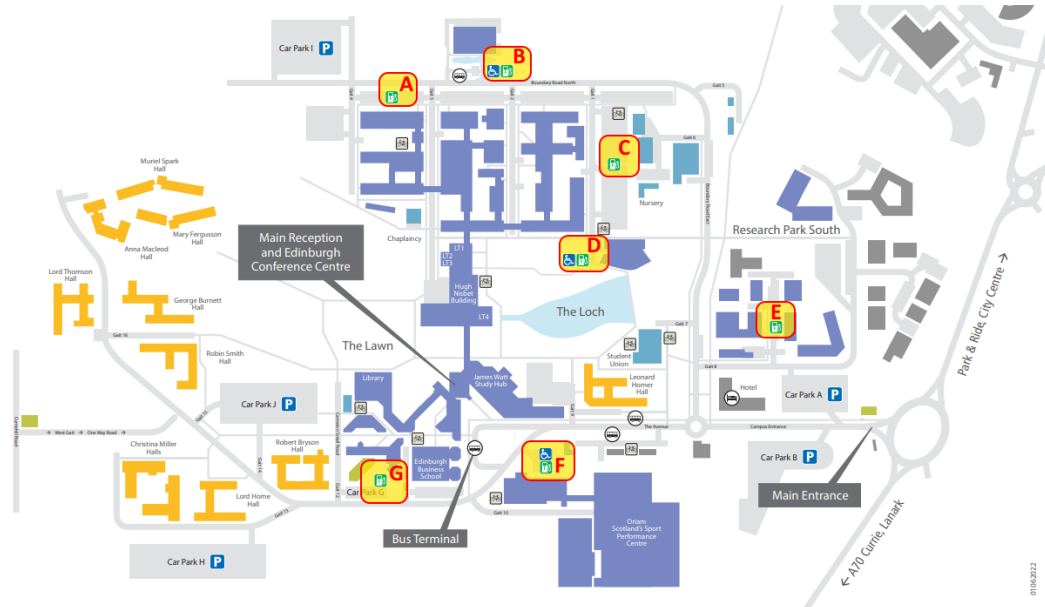


Figure 1: A map of Heriot-Watt's Hermiston Campus, charge point locations highlighted.

Heriot-Watt's Hermiston campus has several locations where visitors may park their electric vehicles and utilise the charging points for the duration of their visit. The issue encountered by many of these users however is that the demand for these electric vehicle charge points exceeds the existing supply. The initial solution to this problem would be the installation of new charge points however this has its own barriers relating to cost of installation, spacing and construction of infrastructure which would allow for the chargers to be built.

The solution this project aims for is to design and implement a system which would better support the flow of electric vehicle users requiring use of the charge points. This system would help electric vehicle users in reliably finding a charge point on campus when required and provide a tool that aids in the process of space allocation to better manage the shared use of the spaces.

The Current Solution

A basic solution to the problems faced by the electric vehicle users on campus has been attempted. The feedback received from users of this system has been used to guide the design decisions within this document.

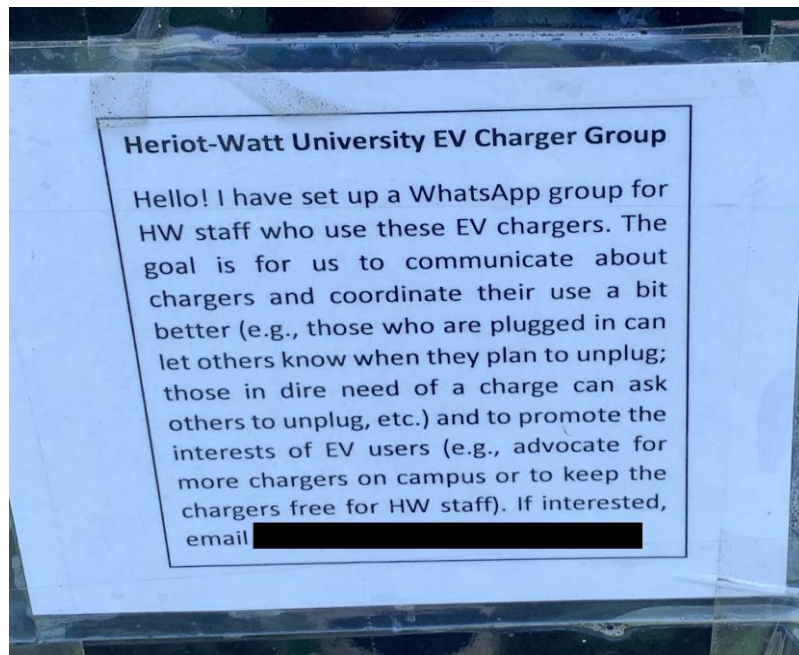


Figure 2: A notice for the current system found attached to one of the charge points.

The current system utilises a publicly available group chat on the messaging application WhatsApp. The goal of this current system is to allow electric vehicle users a channel of communication for managing the use of the campus charging points. The primary use of this system by its users is to request access to a charging point on campus that may be currently in use by a member of the group. This system currently relies on a set of assumptions by its users when in use as the group chat has no direct control over the physical access to the charge points.

The purpose of this existing system is to provide a tool that will aid the electric vehicle users in the self-governance of campus charging point access. Through evaluation of this current system and feedback from its users gathered from interviews and discussion several key areas of concern were identified. These areas of concern are related to problems self-reported by the users of the current system and should inform the requirements of the improved system. The areas of concern that were found to be of high importance to users are the reliability of the current system, the respect for privacy of its users and the accessibility of the system.

Reliability Concerns

The WhatsApp solution currently implemented is exclusively a manual process that relies on the cooperation of its users to achieve success in reporting the status of the charge points. This manual factor has negative implications on the reliability of the solutions function due to the assumptions that must be made for it to operate smoothly.

A concern that was central to the issues faced by users of the WhatsApp solution is the unreliability of information that travels through the system. A main purpose of the solution was to provide a channel for users to stay updated on the status of the charging points on campus however due to the manual nature of the communication it is expected that the status of a charging point may change unbeknownst to its users.

The feature in an improved solution that would have the greatest positive effect on the reliability on the system would be the automation of the campus charging point status reporting. This method of automation could implement a queue structure which would allow for users to arrange a time for their vehicle to occupy a charging point, this would have the advantage of ensuring that the system accurately reflects the real-world status of the charge points. This improvement would additionally put less responsibility on the users to be vigilant in their monitoring of the WhatsApp solution for update requests and can instead rely on the automated solution to keep users updated.

Privacy Concerns

The privacy of users is a factor in the concerns of some users of the current system. It is required to gain access to the WhatsApp system that a user provide their personal phone number as a unique identifier attached to their messages. As the current WhatsApp solution utilises a publicly accessible group the ability to identify an individual by their personal phone number is a major privacy concern.

An improved solution should aim to minimize the amount of identifiable information is linked to a user of the system. Though personal information can be omitted from the system it should be noted that the system will be unable to protect its users from an individual knowing the location of a vehicle from first-hand knowledge as such there is still a degree of risk to privacy that is beyond the scope of what this solution can allow for.

Accessibility Concerns

Discussions with users of the WhatsApp system revealed complications that arise from members of the public travelling onto campus to utilise the charging points. Use of the physical chargers by outside parties is less likely to be reported on the WhatsApp system.

There can be no guarantee that access to the charging points can be restricted to only verified users and as such it was decided through discussion with users that the best course of action when designing an improved solution that the system remain publicly accessible.

Proof of Concept

Addressing the concerns of the WhatsApp system users would require the design and implementation of a system that ideally operates as a booking manager for the physical charging point. Several applications exist currently which aim to achieve similar goals, some of which the users of the WhatsApp system have familiarity with. The feedback received from users of the WhatsApp system who are also familiar with these external applications was factored into the design decisions described in later sections of this document.

Zapmap

Zapmap is an application available within web browsers and as a mobile application that provides users an interactive map interface for locating available charging point (Zapmap).

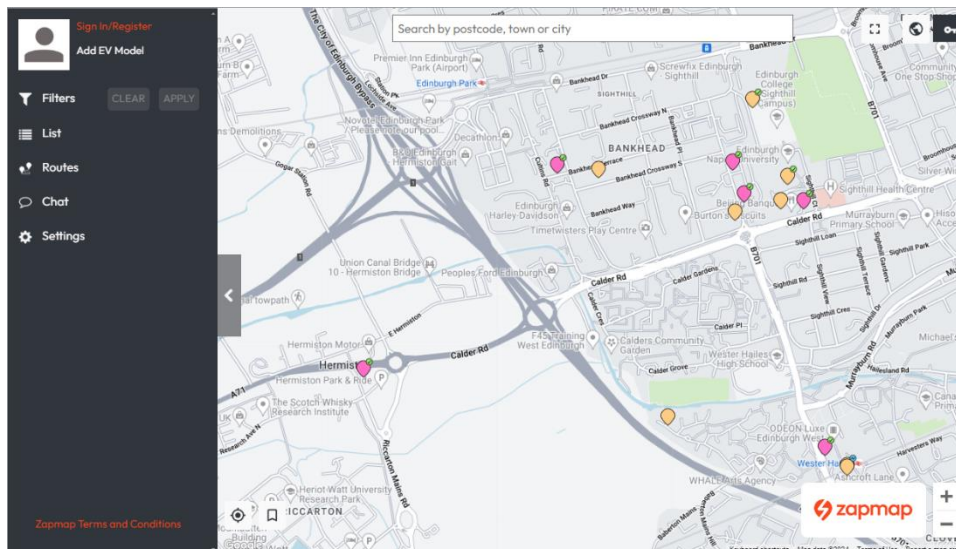


Figure 3: Zapmap as viewed from a web browser.

The scope of the Zapmap application is far greater than the system this project aims to achieve. If implemented, an interactive map interface would be limited to only monitor the charging point present on Heriot-Watt's Hermiston campus. A user interface of this nature would have the benefit of providing users directions to find the available charging point however would risk overcomplicating a simple booking process.

ChargePlace Scotland

Chargeplace Scotland is a similar application to Zapmap as it also provides an interactive map user interface for viewing charging points. ChargePlace Scotland also includes a feature that allows for the individual charging point status at each zone to be viewed (ChargePlace Scotland).



Charge Point Information		
50937 Edinburgh		
Type: AC	Number of connectors: 2	
Status: Available	[12 Feb 2024 17:13:03] Check latest status	
Connector 1  Type 2 Power output 22kW <div>● Available</div>		Address Westerhailes Healthy Living Centre, Edinburgh Edinburgh Westerhailes Healthy Living Centre EH14 3JF GB Price 50p per kWh.
Connector 2  Type 2 Power output 22kW <div>○ In use</div>		Directions <div>Driving directions Open in what3words</div> Additional Info

Figure 4: ChargePlace Scotland information menu for a charge point location near Hermiston campus.

The charging points on Heriot-Watt's Hermiston campus can be found in seven different locations spread throughout the grounds. At each of these locations users would be

benefitted by a system that integrates an information panel like ChargePlace that provides on demand updates related to the charger status. This is a factor that is particularly relevant to the chargers found on campus as each location differs in the model of charger used which can affect the maximum charging speeds each location offers.

Requirements

Research & Requirements Decisions

During the research stage of this project, the team identified that newer models of EV Charging Points implemented a protocol with which key charging information can be communicated across systems. The team identified this as a potential solution to updating the queuing system with what chargers were in use and which were available. However, after looking into this further the team identified that the time taken to gain access to the protocol and the sheer amount of the protocol that would have to be implemented would push this feature out of scope for the time the team has to complete the project. The team has identified it as an important part to investigate and attempt to implement in future development.

Another major discussion point for the team was the integration of car manufacturers' APIs into our system. The idea here was to allow users to link their cars to the system in order to receive far more accurate data relating to their charge levels. After researching this idea, the team realised that this would be seriously out of scope for the project. The sheer amount of personal data required to access these APIs on a specific vehicle is far too sensitive for the team to consider handling and potentially storing. The team therefore took the decision to decide against implementing this functionality.

After discussing with the client and other potential users, the team discovered that there was a desire for there to be a reduction in the number of steps necessary for a user to make use of the system. This helped inform a few major design decisions such as the idea of a password-less sign in feature. By reducing the amount of information, they are required to input each time they use the system, the team hopes this will improve the overall efficiency of the process for the user.

The following section contains tables stating what we have decided are the key requirements for the project. We have used the MoSCoW method to prioritise the importance of each requirement to the project.

Functional Requirements

FR #	Requirement	MoSCoW Priority
FR 1	The user must be able to access the website from on campus	M
FR 2	The user should be able to scan a QR code to navigate to the website	S
FR 3	The users could be able to scan an RFID tag to navigate to the website	C
FR 4	The user must be able to create an account using their email address	M
FR 5	The user should be able to verify their account by email	S

FR 6	The user must be able to delete their account	M
FR 7	The user must be able to log in to the system	M
FR 8	The user must be able to log out of the system	M
FR 9	The user should be logged out after a certain amount of time	S
FR 10	The system could remove unused accounts	C
FR 11	The user must be able to view the status of charging points on campus	M
FR 12	The system should show a map of the campus and where charger points are located	S
FR 13	The system won't get the live status from a charging point API	W
FR 14	The user won't be able to connect the API of their electric vehicle to the system	W
FR 15	The system must allow users to enter a queue for a specific charging location.	M
FR 16	The user must be able to view their queue position	M
FR 17	The user must be able to leave a queue	M
FR 18	The user should get a notification when a charging location that they are queuing for is available	S
FR 19	The user must be able to check into a specific charging place	M
FR 20	The user must be able to check out of a specific charging place	M
FR 21	The user should be able to report a charger that has the incorrect status (e.g. free when in use)	S
FR 22	Administrator users should be able to edit the permission level of non-administrator users	S
FR 23	The system must allow admin users to add, remove and update charging locations in the system	M
FR 24	The system must allow admin users to add, remove and	M

	update individual chargers in the system	
FR 25	The system could allow admin users to add the location of each individual space to the map	C

Table 1: Functional Requirements Table

Non-Functional Requirements

NFR #	Requirement	MoSCoW Priority
NFR 1	The system must be usable on mobile.	M
NFR 2	The system must be usable on desktop.	M
NFR 3	The system must meet the WCAG 2 AA accessibility guidelines.	M
NFR 4	The system could accommodate a range of languages.	C
NFR 5	The system should accommodate different screen sizes.	S
NFR 6	The system should be hosted on the university system instead of a cloud provider.	S
NFR 7	The system should be portable to allow for easy migration	S
NFR 8	The system could use less than 1 gigabyte of RAM and less than 1 CPU (while under a realistic workload).	C
NFR 9	The system should be configurable from a single Docker-Compose YAML file	S
NFR 10	The user should be able to log into the system while giving only their email address.	S
NFR 11	The system should have password-less login using email authorisation.	S
NFR 12	The user must not be able to see the data of other users (e.g. email address)	M

Table 2: Non-Functional Requirements Table

Task Specification

Task A - Creation of Front End for System Clients

Description

This task will involve the creation of a front end for our system that will be intended for use by the system's clients. This portion of the front end will contain a few main pages, the first of these is the user login-page/homepage which will be the first page the user sees upon loading the web app and it will allow the user to log into our system with an email-only login. The second page will be a booking page which will allow users to see which chargers are available and either book a charger or queue for one if none are available. The third page is a QR-Code/RFID landing page. The fourth page tracks the queue of users waiting to use a charging point. The fifth page is a profile page for the user of the system to manage their account.

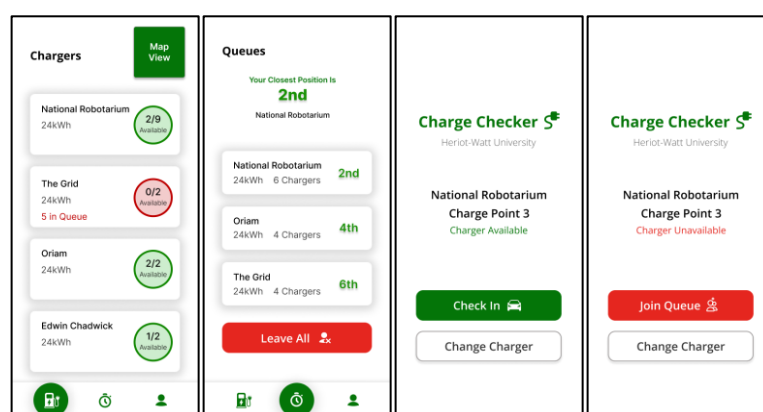


Figure 5: Dashboard Concept

We will use the following colour scheme for our system as green is generally seen as an eco-friendly colour which matches the subject of electric vehicles. All foreground/background colour combinations meet WCAG AA Guidelines while still being visually pleasing.



Figure 6: Accessibility Colour Scheme

due to this, this task is dependent on the creation of a database as well as the development of the account and queueing backend systems, as these are required features that users will be able to interact with on this frontend system. This task is also related to the success of the deployment and finding appropriate hosting, as otherwise, this task will not have a place to be hosted and used by users.

Task B - Creation of Front End for System Administrators

Description

This task will involve the creation of a front end for our system that will be intended for use by the system's administrators. There will be two pages in this system for the administrators to use the first will be a dashboard page allowing the administrator to adjust the charging tracking system as needed, such as adding a new charger area or a new area entirely. The second page will be where admin users will be able to add new admin users.

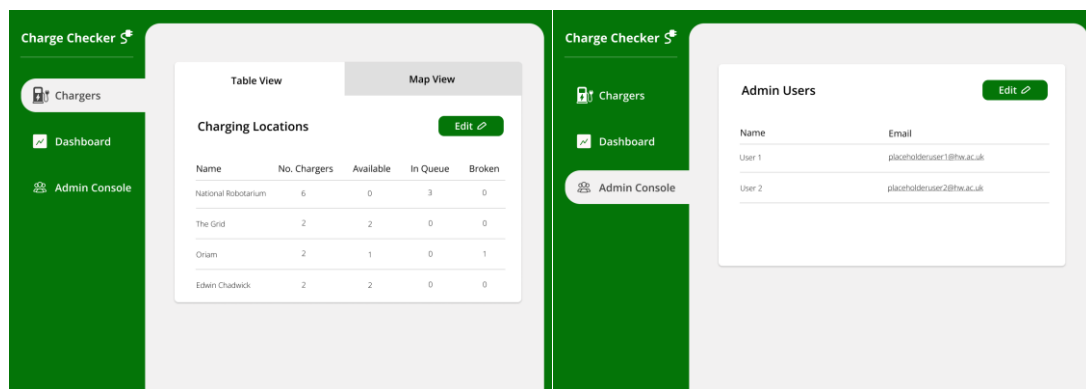


Figure 7: Concept of Charger Table View and Admin Console

The same colour scheme will be used for the admin console as user facing site, except the main body background colour will use a light grey to differentiate between the background colour of the content. The green accent colour will also be used for the menu and border.

#F2F1F1

Body Background Colour

Used as background colour for the main window of admin console

Figure 8: Colour Code for Admin Console

This task is linked to several others. It is dependent on the Database creation and management as without an appropriate database the administrator will not be able to update information that users need to know, such as how many charging points exist in an area. It is also linked to the Creation of a front-end for clients, as without a client-side frontend there will be no users for the administration to update information for. Additionally, this task is connected to the administrative backend system, as this frontend will simply allow an admin to make changes involving this administrative backend system. Finally, this task is also related to hosting and deployment as this, like client-side frontend must be hosted online to be used by administrators.

Task C - Creation of Map View for End User and Admin Sites

Description

This task will involve the creation of an interactive map which will show the charging locations and provide users with information such as charger availability and charging speeds. To allow admin users to update/add charging locations, an interactive map where admins can set the locations of chargers must also be made.

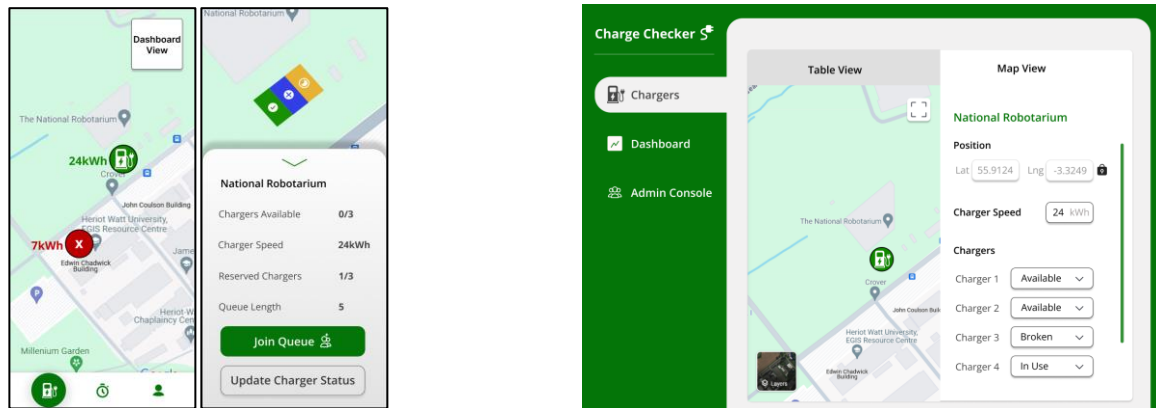


Figure 9: Map View Concept

As we plan to use the Google Maps API, the colour scheme will be the default for Google Maps.

As this task is featured as a subsection on both the client and admin frontend pages, this task is firstly reliant on both. However, this also means that this task inherits the same dependencies as both, including a functional database, a backend account system, and a backend administrative system, and finally that the hosting and deployment of the system is successful.

Task D – Database Creation and Management

Description

This task involves the creation of the database required for our system, where the information will be stored. There will be 5 separate tables will be created, 4 of which are the main booking system, with the fifth table containing the key values allowing it to be changed without accessing the code so that it could be easily maintained. This includes maximum time that one can be in the queue, the maximum amount of time they should be using the charger and the regularity the front end will check the queue.

The user table contains information relating the information to the user, including the status of the user which would state whether it is inactive, queuing, pending or charging, whether they are an admin and what time they started queuing there is also an ID which is used as the primary key as that the email cannot be accessed by other users. There is also the charger ID if they user is currently charging their vehicle and when they started queuing for a charger. Finally, there is a permission level to see if the user is an admin and whether they should have access to admin side of the website.

The location contains the name of location, the type of the chargers at the location and the latitude and longitude of the location.

The charging point table contains the status of the charger which would be a list being either InUse, Free and OutOfOrder, the type of charger, an ID for the charger and the ID of the location it is at. This is because it separates the ability to queue at different locations without referring directly to the charger and over-loading the user.

The Queue table is used to create entries within the queue and the entry stamp is created so the placement of each user in the queue can be derived from the timestamp. It is more resource efficient to do this because you do not need to change the information until the user entry is removed from the queue.

Database Schema

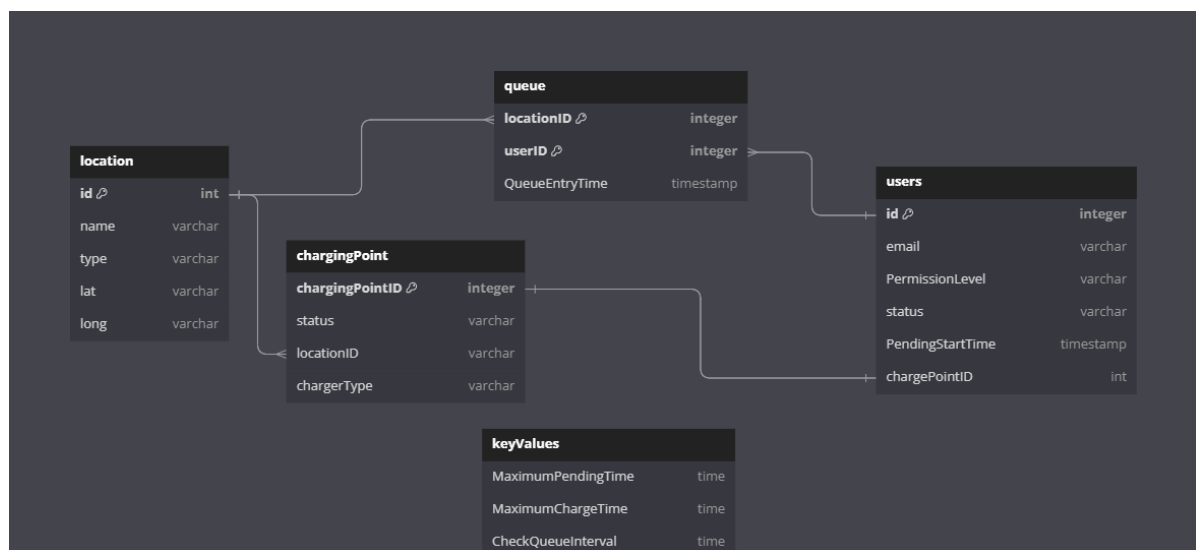


Figure 10: Database Relational Diagram

The database requires a space in a server to host before it can create but there are no other dependencies it has outside of having the space to store. However, every part of the backend relies on the database to feed the information back and forth as all the relevant information is stored in the database. This requires the database to be properly set up before going ahead with testing and running the backend.

Task E - Backend Account System

Description

This task involves the development of an account system that allows a user to create, verify and delete an account, as well as logging in and out. This account should only require their email address, which should then be used for password-less sign in. The API specification is shown below. The system also must check the last time that each user used the system (by looking at the pending start time column in the user table) to determine if their account is inactive. Inactive accounts may be deleted.

Name	Required Data	Output	Action	Type
CreateUser	Email		Add a new row to the User table in the database. Send a verification email	POST
VerifyUser	Email, Token		Update row in the database to have "verified" as true	PATCH
LoginCodeRequest	Email		Generate single use authentication code and send an email with it to the user	GET
Login	Email, AuthenticationCode	JWT	Check if authentication code is valid, if so, generate a JSON web token and send it to the user	POST
Logout	Email, JWT		Set the user's JWT as invalid, update the database accordingly	GET
DeleteUser	Email, JWT		Remove the corresponding row from the User table	GET
GetUserData	JWT	User.Email, User.PermissionLevel, User.Status, User.PendingStartTime, User.ChargingPoint	Get data about the logged in user from the database and send it to the front end	GET

Table 3: Account Function Table

Task F - Backend Queue System

Description

Developing a system that allows the users to queue for a spot. This will allow the front end to request data on charger status and queue lengths, join and leave queues, as well as checking in and out of spaces. The backend automatically updates the status of users that are in the pending state for longer than the maximum pending time. Automatically update the status of users that have been checked into a space for over the maximum check in time, as they have most likely left without checking out.

Name	Required Data	Output	Action	Type
GetChargerData	JWT	ChargingPoint.chargingPointID, ChargingPoint.status, ChargingPoint.location, ChargingPoint.chargerType, List of QueueLengths	Calculate queue length by selecting from the database. Send queue length and charging point data to the user	GET
JoinQueue	JWT, List of Locations	List of QueuePositions	Add rows to the QueueData table, one for each of the locations. Compute the lengths of the different queues and send this data to the user	POST
LeaveQueue	JWT, List of Locations		Remove rows from the QueueData table that have the user at the specified queue locations	DELETE
CheckQueue	JWT	List of QueueLengths, ChargingPointID, MaximumWaitTime	Calculate queue lengths and send them to the user. If at least one of the queues is of length 0, send the ID of the charging point that is free. Send the maximum time that the user can wait before checking in	GET
CheckIn	JWT, ChargingPointID		Update the status in the User table to "charging" and update the status in the ChargingPointTable to "InUse"	PATCH
CheckOut	JWT		Update the status in the User table to "Inactive" and update the status in the ChargingPointTable to "free"	PATCH

Table 4: Queue Function Table

This task is dependent on hosting and deployment, the database as well as the account system backend.

Task G - Backend Administrative System

Description

Developing the backend of the system that administrators can use. This will allow administrators to add, remove and update charging locations and individual charging points. It will also allow administrators to view other administrator users, as well as edit the permission level of non-administrators.

Name	Required Data	Output	Action	Type
SetPermissionLevel	JWT, UserID		Alter the other user's permission level accordingly	PATCH
GetAdminUsers	JWT	List of users	Send a list of users where PermissionLevel is admin	GET
ClearQueue	JWT, Location		Delete from the QueueData table where the location is equal to a specified location	DELETE
UpdateLocation	JWT, Name, Lat, Long		Check if location exists, if it does update it, otherwise add a new one	POST
UpdateChargingPoint	JWT, ID, Status, Location, Type, Lat, Long		Check if the charging point exists, if it does update it, otherwise add a new one	POST
DeleteLocation	JWT, Name		Delete a location. This should cascade to the ChargingPoint table	DELETE
DeleteChargingPoint	JWT, ID		Delete a row from the charging point table accordingly	DELETE

Table 5: Admin Function Table

This task is dependent on hosting and deployment, the database as well as the account system backend.

Task H - Hosting and Deployment

Description

This task involves finding suitable hosting, either the university webserver or a third-party cloud provider (e.g. Azure or AWS), then deploying and managing the application on the chosen platform. It is important that the hosting platform is easily changeable at any time (due to the risks stated in the risk section below), so we have opted to containerize our system using Docker. This will most likely come in the form of two containers: one for the NodeJS application and another for the React backend. These containers, as well as any others that we may add, should be configurable using a single Docker-Compose YAML file. This task also involves ensuring that the different components of the system (NodeJS application, React backend, MySQL database) can communicate with each other correctly, as well as with external users.

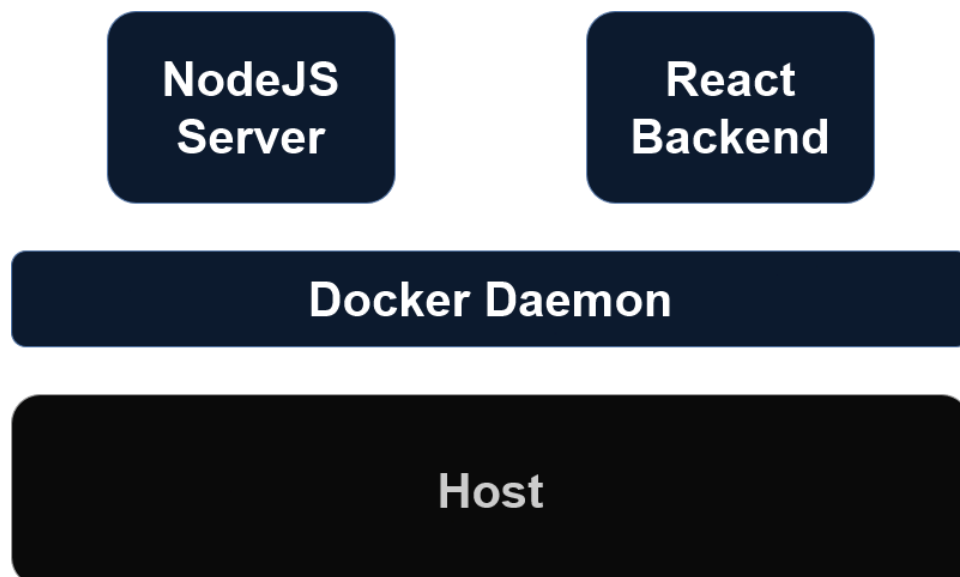


Figure 11: Hosting Structure Diagram

The testing of the database connection is dependent on the database being at least partially complete, and the testing of the connection to external users requires the React frontend and the NodeJS backend to be at least partially complete.

Technical Risk Table

ID	Description	Category	Likelihood of occurrence	Impact of Risk	Severity	Owner	Mitigation
1	The Centos 7 system is updated during development as it is approaching end of life.	Back-End	Medium	High	Medium	Cale	Depending on the down time, we can either attempt to redeploy on the new system or use a public cloud provider (e.g. Azure or AWS).
2	Administrators of the hosting system are unavailable for an extended period		Medium	High	High	Cale	We can host our system using a public cloud provider.
3	University server cannot provide performance required by the system	Back-End	Low	High	Medium	Cale	A virtual machine from a public cloud provider can be used in this scenario instead of sharing resources with other systems on the university server
4	We cannot host the system on a university hosting platform meaning that personal data must be hosted on a 3 rd party system	Back-End	Low	Medium	Medium	Kyle	We can ensure that email addresses are the only personal data that is stored and make sure this is communicated to users of the system
5	MySQL database solution is not deployed correctly on	Back-End	Low	Low	Low	Cale	We can run a containerised MySQL (or alternative database) version

	the university system						of the system. If we are not using Docker (due to risks stated below), we can install a database system directly on the machine that we are using for hosting
6	There are issues with using Docker on the university system (e.g. because Docker commands require root permission)	Back-End	Medium	Medium	Medium	Cale	We can choose to either not use Docker, or to host our system on a public cloud provider such as Azure or AWS
7	No Docker container registry to use for hosting our containers	Back-End	High	Low	Low	Cale	Containers can be built, compressed into files then copied over SSH into the webserver where they can be run
8	Password-less sign-in implementation fails	General	Low	High	High	Kyle	In this scenario, the team would investigate implementing a different authentication method such as OAuth2 or similar. This would likely impact the design ethos identified to reduce access hurdles for users
9	QR code access is not implemented	General	Low	Medium	Medium	Jake	In this scenario, the system would have to fall back on the minimum access through URLs. This would not be the ideal

							scenario for new users gaining access to the system for the first time however regular users may not be impacted too badly if the system URL is saved as a regular link. This solution would still provide an access point for users.
10	We cannot send notifications through mobile web browsers	Front-End	Medium	Medium	Medium	Andrew	In this scenario, we will have to rely on email-based notifications. As the system will require users to provide an email to create their accounts, the notifications can be sent through this instead
11	Google maps API fails to integrate with the system	Front-End	Low	Medium	Low	Scott	<p>This can be mitigated by using an alternative design with a table view that indicates free spaces and details their locations</p> <p>Alternatively, we can utilise an external, open-source map service to preserve the feature</p>

Table 6: Technical Risk Table

Software Integration Plan

The tasks that will be running on the hosting platform at each given stage are shown below.

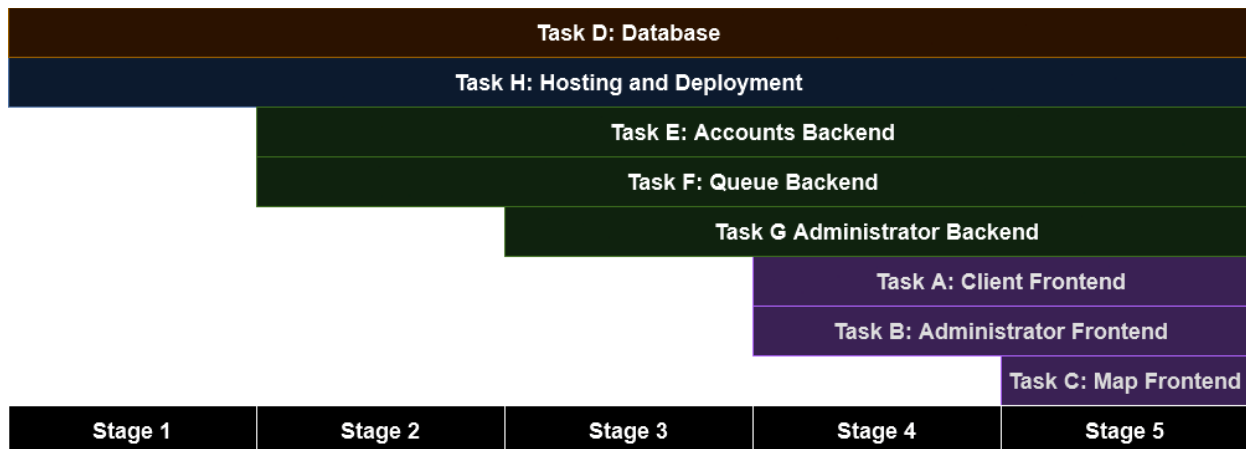


Figure 12: Software Integration Stage Breakdown

We have split our Software Integration Plan into 5 main stages for testing. Stage 1 will ensure that both the database creation and our hosting and deployment is successful as all other features of the system are built on top of these tasks, as the database will contain all relevant data such as charging points and accounts, and the hosting and deployment task will ensure we have an environment to integrate and test all our systems. Once we ensure these are working as intended, we will begin to integrate the rest of the backend features, these being the account/login system and the system that will eventually allow users to queue for and use charging points. We will then add the administrative backend features, this is added last as certain admin features require the successful implementation of the accounts and queueing system, such as changing user permissions. The final 2 stages will see the implementation and testing of all front-end features, with the client and administrator front-ends being completed and integrated first. Finally, we will add and test the Map systems for client and admin frontends, this is added last as it is a subsystem of both tasks and therefore requires them to be functional to test.

A list of the software and systems that we intend to use is shown below.

Software/System	Use
NodeJS	Server runtime
React	Front-end
TailwindCSS	Front-end CSS
Google Maps API	Front-end map view
Docker	Deployment
Docker compose plugin	Deployment configuration and control
University hosted MySQL	Database
University Webserver	Hosting
ExpressJS (NPM Package)	API framework
Prisma (NPM Package)	Database ORM

Jsonwebtoken (NPM Package)	Json web token generation
EmailJS (NPM Package)	Automated email system
Passwordless (NPM Package)	Passwordless sign in
React-QR-Code (NPM Package)	Generating QR codes

Table 7: Software Dependencies Table

Table of Sub-tasks

Task ID	Description
A1	Create login page, user-facing chargers dashboard, and QR code/RFID landing page
A2	Create queue tracking page, charger info menu and profile page
AIT	Integration and testing for task A
B1	Create admin dashboard view for modifying chargers and manage admin user page
BIT	Integration and testing for task B
C1	Create map with charger locations shown with icons
C2	Create map to allow admin users to set the coordinates of the charging locations
D	Implement database schema
E1	Create a login system
E2	Create an account creation system
E3	Create the ability to fetch information about a user/ removing information
EIT	Integration and testing for task E
F1	Get charger data
F2	Join and leave queues
F3	Check in and out of charging points
FIT	Integration and testing for task F
G1	Update, add and delete locations and charging points
G2	View and set permission levels of other admin users, as well as clearing queues
GIT	Integration and testing for task G
H1	Initial Docker test on webserver to ensure that Docker itself is running properly
H2	Ensure that containers can connect to database
H3	Ensure that containers can connect to external users
H4	Deploy system for final testing

Table 8: Stage Allocated Tasks Table

Gantt Chart

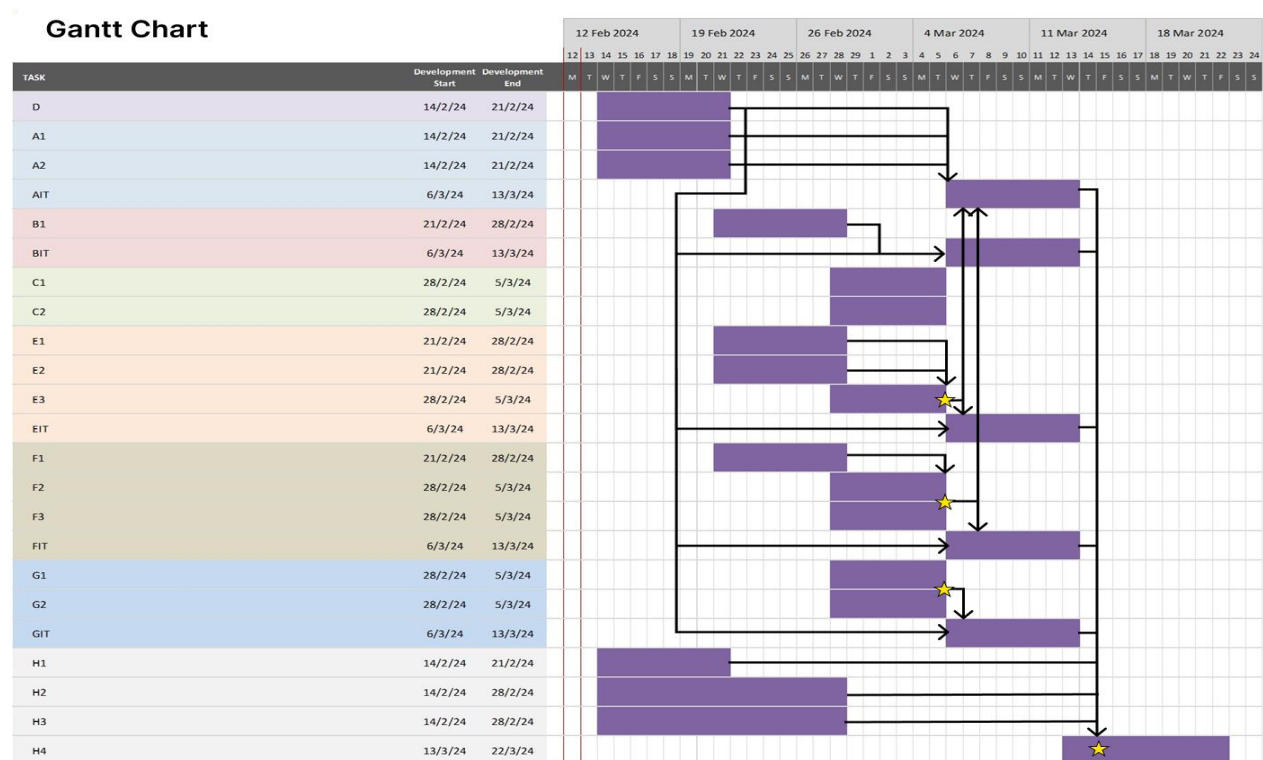


Figure 13: Organisational Gantt Chart

Requirement Traceability Matrix

Please note that integration and testing sub-tasks are not included in the requirement traceability matrix because they include the same requirements as the overall task.

Task ID	Description	Requirements Implemented
A1	Create login page, user-facing chargers' dashboard, and QR code/RFID landing page	FR1, FR2, FR3, FR4, FR7, FR8, FR11, NFR1, NFR2, NFR3, NFR5,
A2	Create queue tracking page, charger info menu and profile page	FR6, FR15, FR16, FR17, FR19, FR20, FR21, NFR4, NFR12
B	Create admin dashboard view for modifying chargers and manage admin users page	FR11, FR22, FR23, FR24, FR25, NFR1, NFR2, NFR3, NFR5
C1	Create map with charger locations shown with icons	FR11, FR12
C2	Create map to allow admin users to set the coordinates of the charging locations	FR23, FR24, FR25
D	Implement database schema	FR 4, FR 6, FR 7, FR 8, FR10, F11, F15, FR 16, FR 17, FR18, FR19, FR20, FR21, FR22, FR23, FR24, FR25, NFR 6, NFR 10, NFR11, NFR12
E1	Create a login system	FR5, FR6, FR7, FR8, NFR11
E2	Create an account creation system	FR4, NFR12

E3	Create the ability to fetch information about a user/ removing information	FR9, FR10
F1	Get charger data	FR12,
F2	Join and leave queues	FR11, FR15, FR16, FR17, FR18
F3	Check in and out of charging points	FR19, FR20, FR21
G1	Update, add and delete locations and charging points	FR23, FR24, FR25
G2	View and set permission levels of other admin users, as well as clearing queues	FR22,
H1	Initial Docker test on webserver to ensure that Docker itself is running properly	NFR6
H2	Ensure that containers can connect to database	NFR8
H3	Ensure that containers can connect to external users	FR1
H4	Deploy system for integration testing	NFR7, NFR8, NFR9

Table 9: Requirements Traceability Table

	A1	A2	B	C1	C2	D	E1	E2	E3	F1	F2	F3	G1	G2	H1	H2	H3	H4
FR1																		
FR2																		
FR3																		
FR4																		
FR5																		
FR6																		
FR7																		
FR8																		
FR 9																		
FR 10																		
FR 11																		
FR 12																		
FR 13																		
FR 14																		
FR 15																		
FR 16																		
FR 17																		
FR 18																		
FR 19																		
FR 20																		
FR 21																		
FR 22																		
FR 23																		
FR 24																		
FR 25																		
NFR1																		
NFR 2																		

NFR 3																	
NFR 4																	
NFR 5																	
NFR 6																	
NFR 7																	
NFR 8																	
NFR 9																	
NFR 10																	
NFR 11																	
NFR 12																	

Table 10: Requirements Traceability Matrix

Table of Responsibilities

Task	Person Responsible
A: Client frontend	Scott
B: Administration frontend	Jake
C: Map frontend	Kyle
D: Database	Xander
E: Accounts backend	Kyle
F: Queue backend	Andrew
G: Administration backend	Xander
H: Hosting	Cale

Table 11: Table of Responsibilities

Appendix A: References

References

ChargePlace Scotland. n.d. 12 February 2024. <<https://chargeplacescotland.org/>>.

Zapmap. n.d. 12 February 2024. <<https://www.zap-map.com/>>.

Appendix B: Tables

Table of Figures

Figure 1: A map of Heriot-Watt's Hermiston Campus, charge point locations highlighted.....	4
Figure 2: A notice for the current system found attached to one of the charge points.....	5
Figure 3: Zapmap as viewed from a web browser.	7
Figure 4: ChargePlace Scotland information menu for a charge point location near Hermiston campus.....	7
Figure 5: Dashboard Concept	12
Figure 6: Accessibility Colour Scheme	12
Figure 7: Concept of Charger Table View and Admin Console	13
Figure 8: Colour Code for Admin Console	13
Figure 9: Map View Concept.....	14
Figure 10: Database Relational Diagram	15
Figure 11: Hosting Structure Diagram	20
Figure 12: Software Integration Stage Breakdown	24
Figure 13: Organisational Gantt Chart.....	27

Table of Tables

Table 1: Functional Requirements Table.....	11
Table 2: Non-Functional Requirements Table	11
Table 3: Account Function Table	17
Table 4: Queue Function Table	18
Table 5: Admin Function Table	19
Table 6: Technical Risk Table	23
Table 7: Software Dependencies Table.....	25
Table 8: Stage Allocated Tasks Table	26
Table 9: Requirements Traceability Table	28
Table 10: Requirements Traceability Matrix	29
Table : Table of Responsibilities	29