

Bus Routing Optimisation



Freddie Nicholson

AQA A-Level Computer Science (7517) NEA

2020-2021

Candidate No. : 4118

Centre No. : 62403

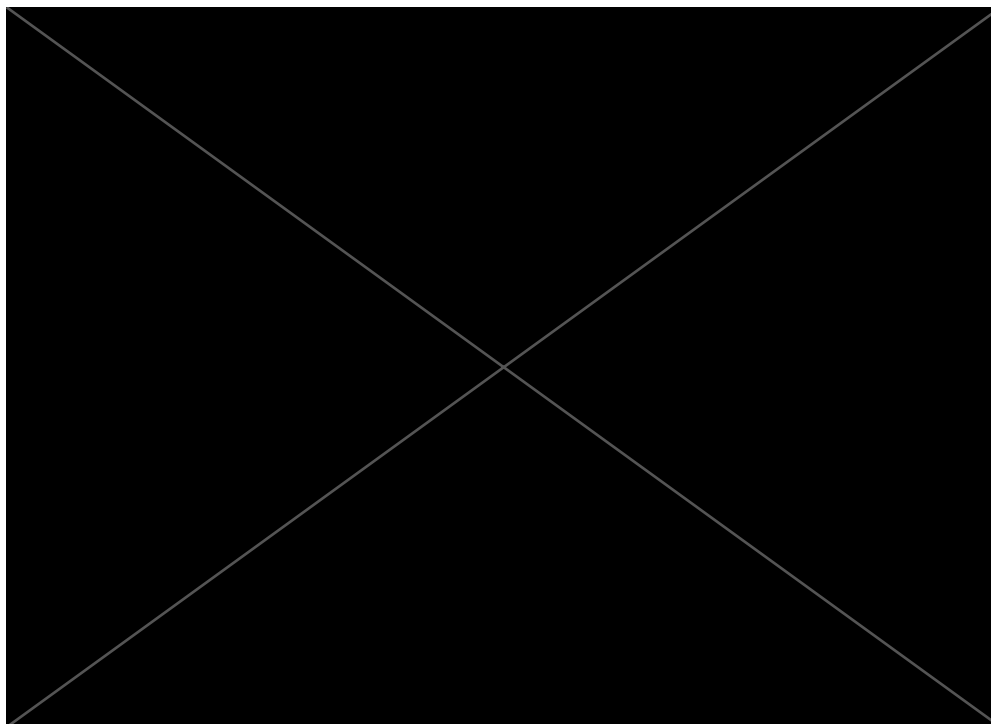
Table of contents

Analysis	4
Introduction	4
The Problem	5
Introducing Potential Users	6
Systems & Infrastructure already in place	8
Existing Context Solutions	10
Objectives	12
Documented Design	14
Structural Diagrams	14
Database Design	19
HCL Design Ideas	23
Algorithms	25
OOP	31
Data Structures	34
Data Flow Diagrams	36
File Structures	37
Technical Solution	38
Reading from OpenStreetMap	38
OOP Representation OSM	41
Route Finding using OR Tools in combination with OSRM	48
Flask Implementation	59
__init__.py - Flask Initialization File	63
auth.py - Authentication / Authorisation Manager	66
db.py - Database Manager	69
imgdispatch.py - Image Asset Manager	71
management.py - Management Module	72
nav.py - Navigation Menu	91
routing.py - Routing Module	94

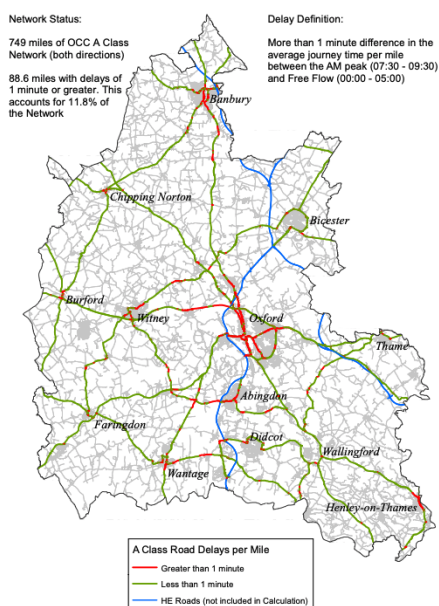
security.py - Security Module	97
user.py - User Module	100
script.js - JavaScript Implementations	101
search.js - JavaScript Implementations	104
editroute.html - Embedded Script - JavaScript Implementations	108
Testing	117
Video	117
Development Issues	119
Evaluation	123
Objectives	123
Independent Feedback	135
Reflections & Conclusion	136
Appendix	137

Introduction

Traffic is a growing problem in many cities. Since 2004, traffic network management of local areas has been passed onto local authorities to manage congestion under the Traffic Management Act 2004. Many councils are now beginning to become concerned about the number of commuters at peak hours during the day causing them to bring in greater legislation on how they should be run. It's not just the road networks that struggle as well, for logistical reasons it is becoming much more difficult to get into work or school for many commuters.



Oxfordshire Trafficmaster A Class Road Network 2018
AM Weekday Peak Analysis Term Time Only (07:30 - 09:30)



¹ From the heatmap shown above, that indicates the most densely populated areas that pupils travel to school from, we can see the trends in school travel. The most densely populated area is clearly the city center as naturally there are going to be a significant number of local pupils within close proximity to the school. There is an even sparse spread of households across the surrounding area. Pupils commute into school by varying methods of transport which adds pressure onto the surrounding traffic system. As In recent years, Oxfordshire County Council have become much more proactive in managing traffic flow around Oxford and keeping an eye on daily commuter patterns and bottlenecks around the city. Shown to the right is a map detailing the traffic congestion points around the Oxfordshire area, there is a clear distinction between Oxford and the surrounding area.

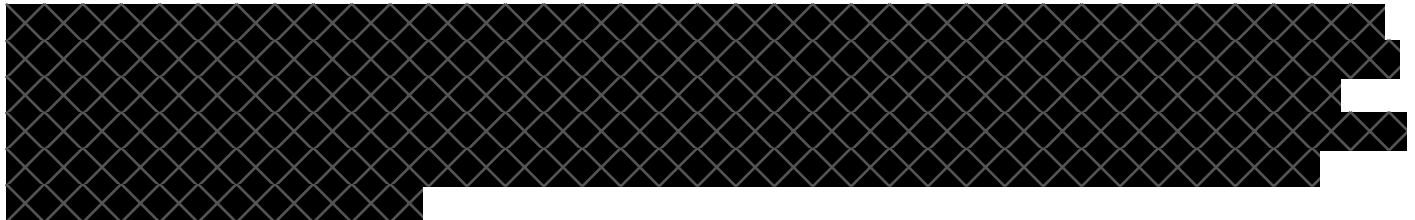
¹ <https://www.oxfordshire.gov.uk/residents/roads-and-transport/traffic/transport-monitoring>

The trend is ever increasing and showing no sign of slowing down. The statistics below shows the increase in traffic on the network within a period of 5 years.

Road class	2014-18	2017-18
A	3.7%	-0.3%
B	2.1%	-0.6%

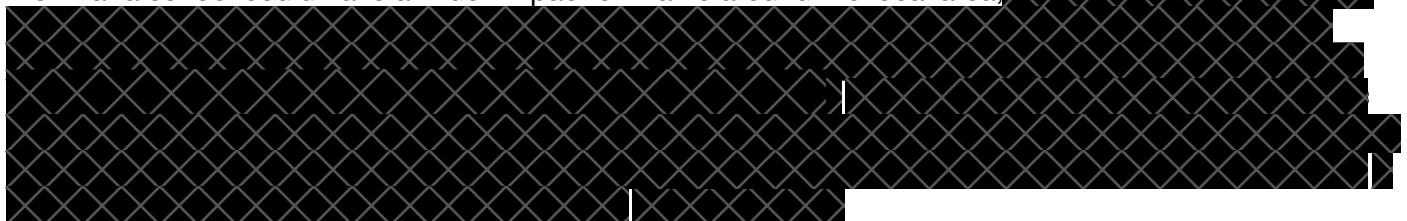
This has resulted in certain road closures and the introduction of new traffic management initiatives, an example of this is the introduction of 462 Automatic Traffic Counting stations that all feed into a network to help inform the council and build a greater understanding of traffic movement around the Oxfordshire area. Other methods of data collection also include manual collection and the use of ANPR cameras that are able to pick up a car's number plate using computer vision and find out more detailed information such as how far a car has travelled in a journey and times as well.

In September 2019, Oxfordshire County Council also proposed the suggestion of a workplace parking levy². The reasoning behind the scheme is the large number of workplace parking spaces compared to public parking which visitors can use leading to public parking spaces becoming hard to find at peak times. Additionally, the hope is that as private parking spaces encourage the journey into work, a levy will hopefully help raise funds to improve transport around the local area.

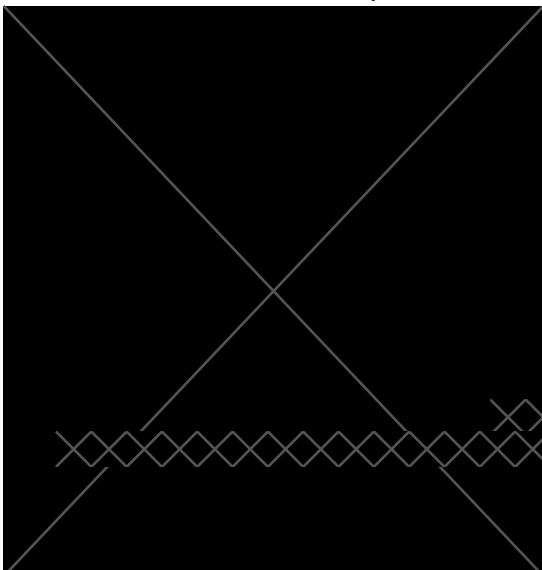


The Problem

Traffic in Oxford is on the rise and the council are trying their best to sort out bottlenecks logistically but fundamentally there are a limited number of roads for vehicles to travel on. Although it may not seem at first that a school could have a wider impact on traffic around the local area,



These buses have to be run on a route system, with routes being decided beforehand to include as many areas of Oxfordshire as possible.



The problem with a route system is that there are going to be suboptimal stops. There could be no pupils getting on at one stop one year but because pupils have used the stop previously, it cannot simply be removed.

[commit-to-cutting-traffic-congestion-and-improving-public-transport-in-oxf](#)

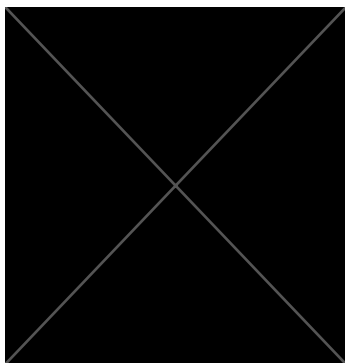
Alongside this, route timings are purely based on limited constraints and do not take into account daily traffic or any kind of data that a commercial application such as Google Maps would use. This leads to time being spent by school staff, finding out where the buses are and what time they will arrive.

There is also no current logical grouping system that decides when and where pupils will get picked up / dropped off. This leads to more time spent by staff working out if a group has been delayed.

These problems and the aforementioned legislation that is being put in place by the council means the Dragon needs to start taking a more strategic approach with its travel into school.

The problem to solve is the suboptimal manual routing system currently implemented by the school bus service provider [REDACTED] I need to come up with a more efficient, optimal maths based algorithm that will be able to find the most optimal stop route for users on a bus route.

[REDACTED]



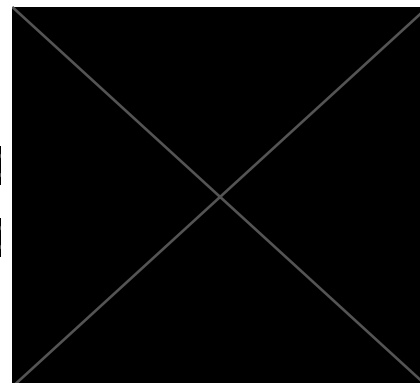
[REDACTED]

[REDACTED]

[REDACTED]

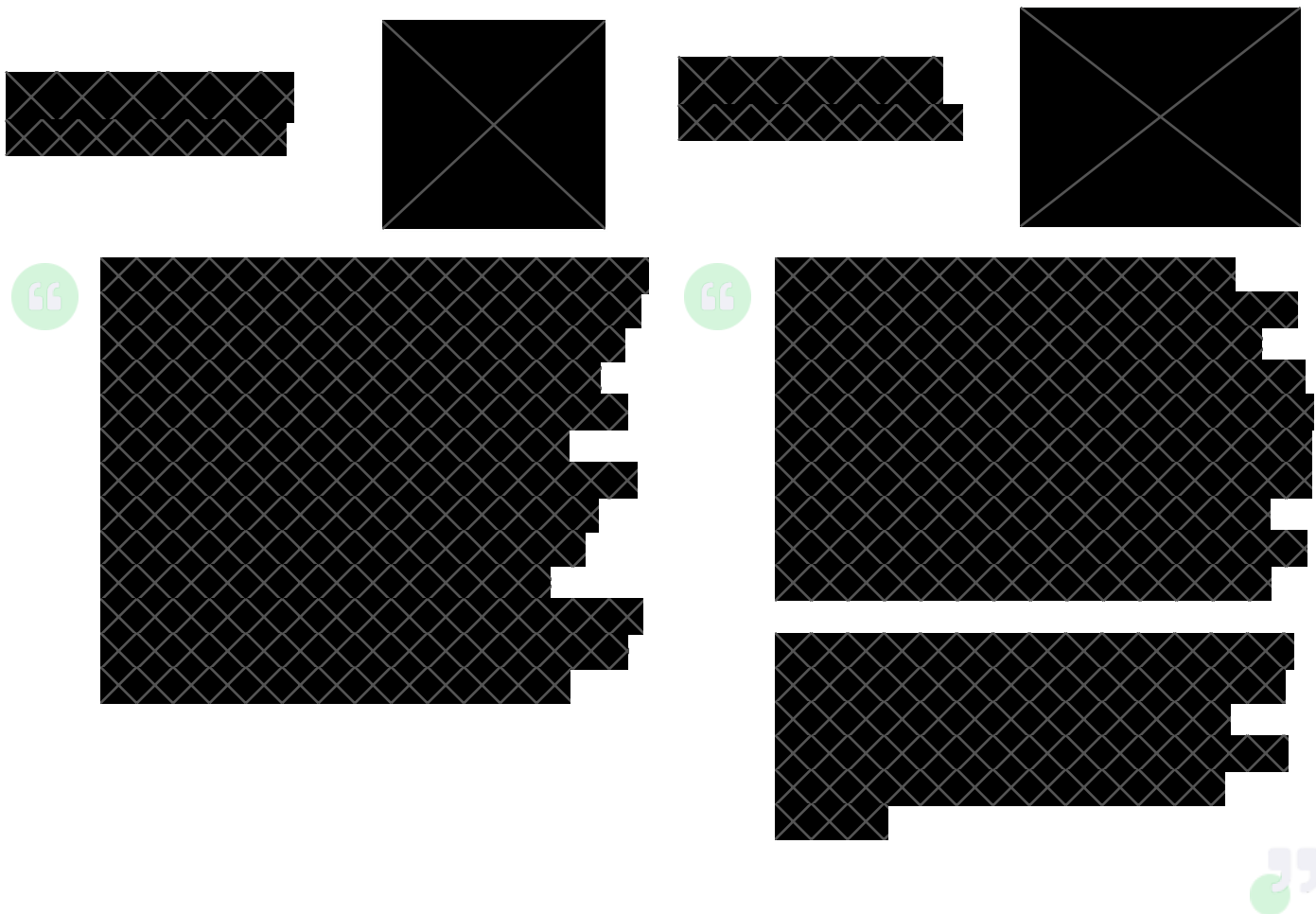
[REDACTED]

[REDACTED]

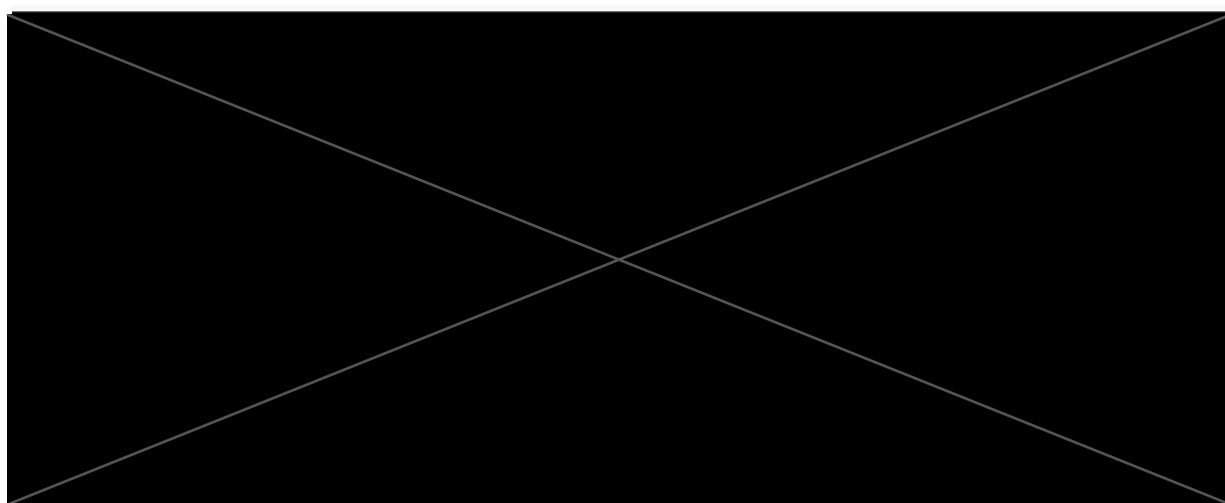



[REDACTED]

Freddie Nicholson Computer Science NEA
Introducing Potential Users – Parents

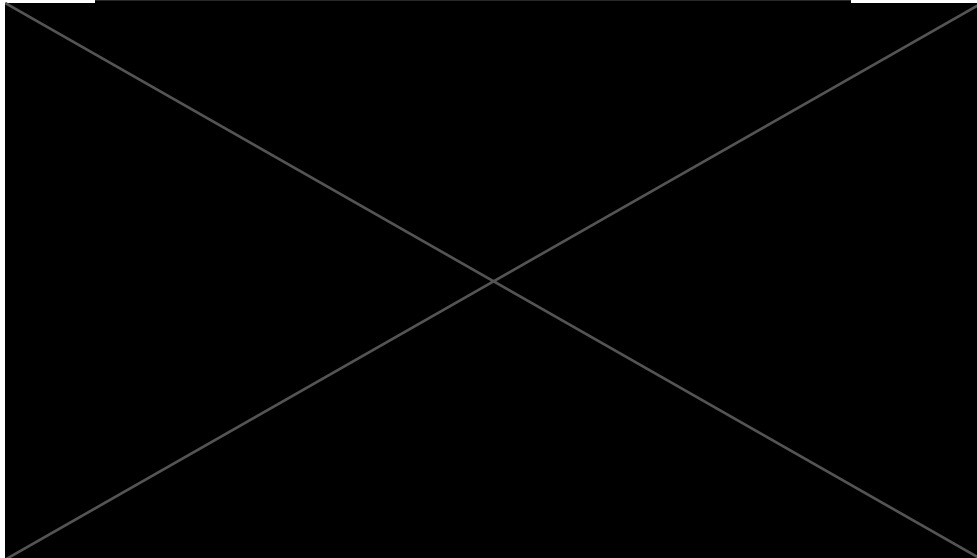


Introducing Potential Users – Bus Company



 official bus service provider.
Parents book using the current interface online which is a classical booking style system using Google Maps and a login system. It requires a lot of manual work to operate as each route needs to be defined in small detail and icons for each school added individually.

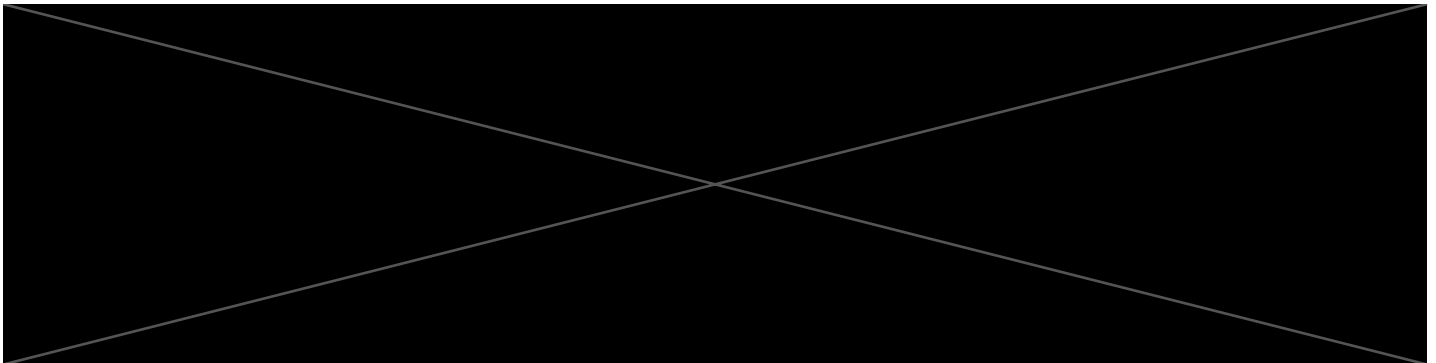
The service is very popular with Day Children and several parents make use of each route pictured on the next page.



XXXX is one of the service providers that would be using the platform. I will need to design the application with this in mind.

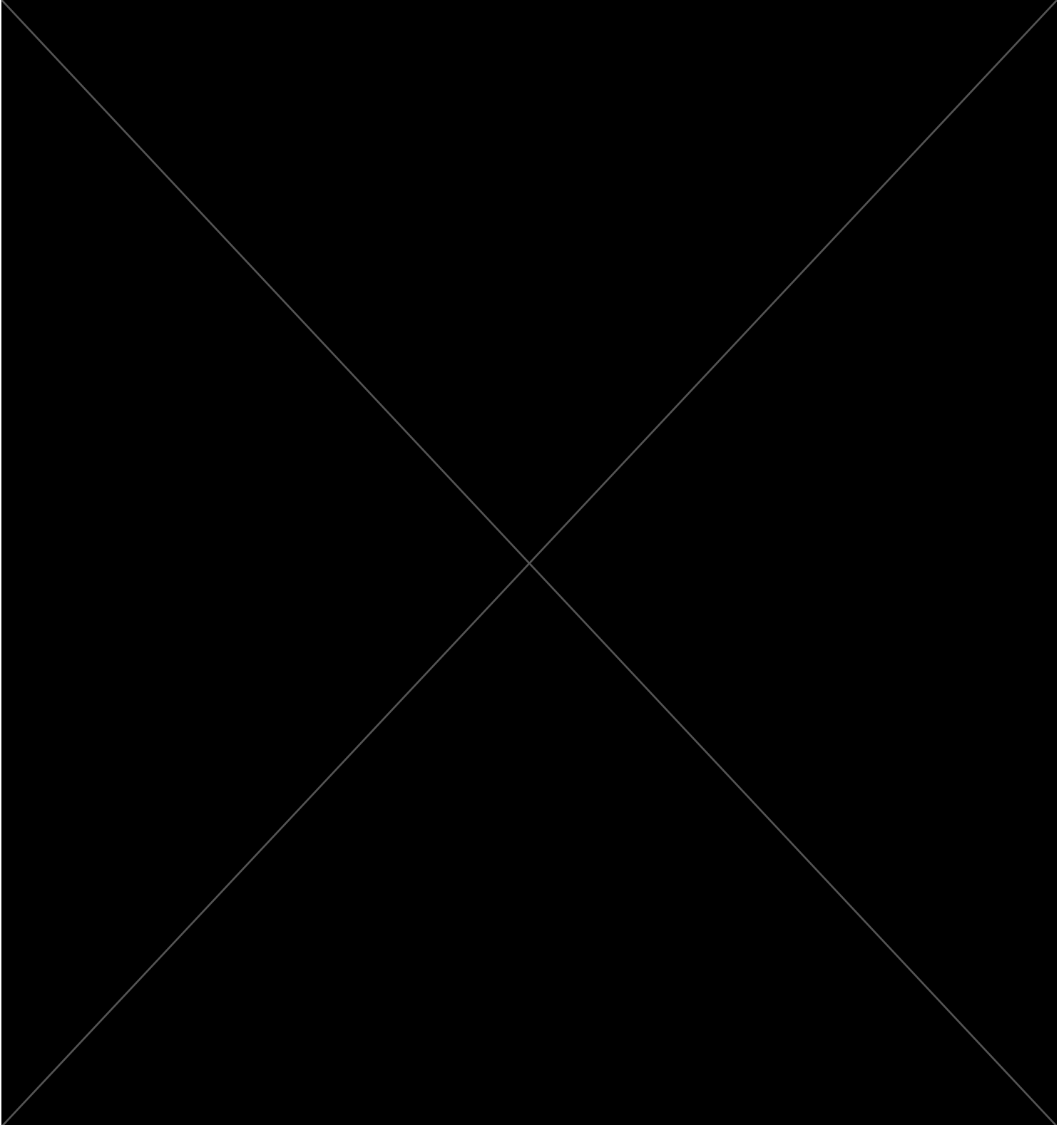
There may be other potential bus providers that the service could be used on such as the XXXX
XXXXXXXXXXXX and therefore ensuring multiple organisations will be able to use the platform would be an advantage if implemented early on.

Infrastructure already in place



Systems already in Place

It is important before I embark on this project that I establish the systems and infrastructure already in place so I know what I could potentially investigate further. It is important to remember that I should also consider the fact that not every school will have the same setup and therefore ensuring that I am making my NEA as flexible as possible in terms of environments it can be run in.

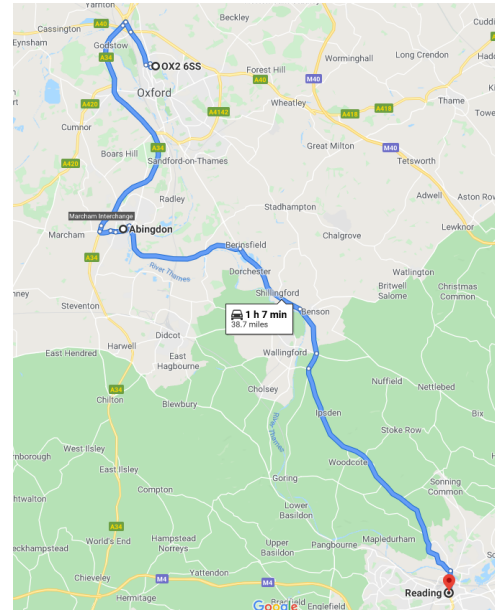




Google Maps is the most popular online navigation system in use today. Launched in 2005, Google maps has since then been gaining more and more users with 1 billion people estimated to use the service every month.

Google Maps has become very complex with many algorithms and external factors taken into account such as data from Google's android operating system and curated infrastructure data from the app Waze. Waze is an app which was taken over by Google in 2013, it relies on user reports to collate its data. 130mn users use Waze each month and most offer map suggestions which could not easily be detected using a passive approach such as the methods that Google Maps use (e.g. unexpected road closures due to road accidents).

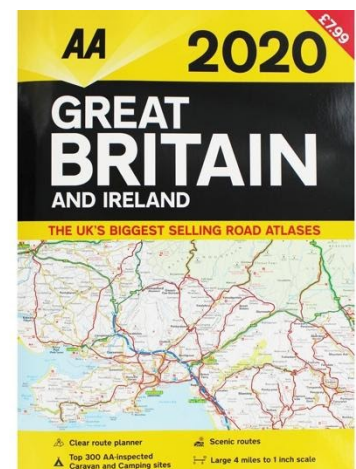
Google Maps would be useful for my project in particular as it has a highly sophisticated codebase that allows it to be the market leader in navigation. However, it charges through the use of an API and if my project to scale this would be a factor to consider. Additionally, the max number of stops that can be placed in a trip is 9. This is a fundamental limitation and would mean that Google Maps is not suitable for my project.



Road Atlas

Road Atlases area. Staple of the past, appearing in most cars across the country. They used to be the default method of navigation in the UK before any kind of electronic equivalent was invented. Even when technology such as satnavs were introduced many people still stuck to paper-based materials as the costs meant that for most people it was not worth it.

One advantage to paper-based maps is their reliability. There is no risk of there being no signal or a battery running out, however with 99% coverage of mobile service in the UK and most new cars coming with USB charging ports as standard, mobile services appear more appealing.



Routific is one of the main navigational routing services for businesses that do not have internal systems.

It offers much more advanced capabilities in terms of optimisation and routing than the previous two examples with scheduled timings, fleet management and other constraint optimisation features.

It features applications for all users including operators, drivers and 'customers' in most cases, allowing for a streamlined



Freddie Nicholson Computer Science NEA
communication between everyone improving the overall service
as a whole.

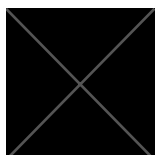
Routific is designed specifically for the ecommerce delivery
industry and therefore would not offer some of the more finer
detail that the Dragon might require.



Mapbox is one of the leading map data software
services. Examples of services that use Mapbox include
Facebook, Snapchat, Shopify and Tesla. They offer a
great variety of mapping services including:

- Advanced 3D mapping in Mapbox Studio
- Augmented Reality Navigation
- Automotive Learning
- Powerful routing engines for navigation problems.

Mapbox is clearly a great contender to a solution
however it does do a lot of the algorithmic work for you
which I would quite like to do myself to gain more out of
the project.



Talking to [REDACTED] Director of IT

After evaluating the problem area and doing some research myself I decided to talk with the main
supervisor of my project, [REDACTED] My discussion with him is outlined below.



What are the current problems to do with traffic management
around the school?

*There is significant morning traffic around the school during the term and the travel times are increasing.
This is only likely to get worse over the next year when the planned changes to the Marston Ferry Road
and others around the Oxford ring road are brought in. The roads around the [REDACTED] are completely
clogged in the mornings, made worse by parents stopping to drop their children off. This leads to
inconsiderate and sometimes dangerous driving.*



What solutions have you been looking into in order to resolve these
issues?

The council has increased the number of no stopping zones and also increased the number of wardens who are on patrol during the mornings and afternoons. Although these are useful and helpful, increasing the use of the bus service is the principal means of addressing the traffic problems we face.



What would you look for in software that would be designed to resolve this problem?

Making the journey to Oxford and the processes around it as efficient as possible. The routes must be simple to book and administer for each family, produce registers for the pupils on the bus before the journey and at each drop-off destination, simple handling of any changes to bookings to help people who administer the system to produce an accurate list of charges for parents.



I will make sure to introduce the possibility of providing the features the client requests within the implementation of my technical solution. A segmented system will need to be used to ensure each application area is appropriately addressed.

Exploring Potential Solutions

1. Google Routing API

One solution to the bus routing problem could be to utilise the Google Routing API. Google creates an excellent set of mapping tools that offer a highly sophisticated level of customization. However as Google's Map Platform is a commercial product there are significant costs associated with using their tools. Using an API also makes the algorithm design easy to use and therefore may not be appropriate for my computer science project to demonstrate my full understanding.

2. OpenStreetMap + Google Operations Research Tools

Another solution is to use the open source tile map OpenStreetMap that is used by several online platforms. OpenStreetMap is ideal for a student project like mine as all the assets are open source and therefore not restricted in their usage. Google Operations Research Tools are a collection of tools useful for optimisation problems and contain many helper functions. However, although the tools are incredibly powerful, they are very abstract and need careful thought into how they should be linked together with an application.

Proposed Solution

I have chosen to go with solution 2. I feel it will allow me to showcase the skills required for my NEA in a suitable way and offer a challenging project. Google's Routing API may end up with the platform relying on too many external services that could get in the way of my development and additionally may end up with a lower quality project.

Objectives

1. The system should be built around OOP principles
 - 1.1. This should be achieved using Python modules named after each module with a selection of subroutines to match the modules purpose
 - 1.2. Flask will be used to provide the framework for modules using Blueprints

- 1.3. Each application featureset should be combined into a flask blueprint module with all subroutines required for each module inside its own file allowing for easier code readability.
- 1.4. Each module should interface with helper functions such as authentication to verify a user has the required access for a feature.
- 1.5. Modules should be presented graphically to the user within the interface.
2. The system should utilise a database for CRUD operation on records
 - 2.1. The system should be set up to interface with a Microsoft SQL Server database in line with standard educational software.
 - 2.2. An ODBC connection should be used to connect the database to the flask web server using a connection socket for each session.
 - 2.3. The database should be split into multiple tables following 3NF
3. The system should use Google OR Tools
 - 3.1. The Google OR Tools Vehicle Routing functions should be utilised in conjunction with OSRM
 - 3.2. OSRM should be setup on a local server to reduce rate limiting
 - 3.3. Custom functions will need to be written in order to perform operations including:
 - 3.3.1. Building the distance matrix for the solver
 - 3.3.2. Transforming the solution to a readable format for the frontend interface
 - 3.3.3. Visualising data within Matplotlib to check the system is operating correctly
4. The system should use a web interface
 - 4.1. The user interface should be modern and follow up to date web design principles
 - 4.2. Input fields for objects should utilise a 'quick search' feature allowing for easy lookup
5. Development Model
 - 5.1. A testing system should be developed using an OOP model for developing the optimisation algorithm
 - 5.2. Matplotlib should be used alongside OpenStreetMap line data using filters to only select the required drivable paths.
 - 5.3. Testing environment should be integrated with OR Tools to visualise the routing algorithm
6. Login system
 - 6.1. A login system should be present to prevent unauthorised access to the site
 - 6.2. A authentication decorator should be present to allow for restricted access to certain pages
 - 6.3. The user should be able to be identified within the system for functions such as listing children
 - 6.4. User should be logout of the system
 - 6.5. User should stay logged in using sessions to prevent redirection to login page
7. Routing
 - 7.1. Ability to create a route
 - 7.1.1. Ability to plot points using stop markers and interactive map
 - 7.1.2. Ability to calculate quickest optimal route between nodes using distance matrix
 - 7.1.3. Ability to view overall route
 - 7.2. Ability to destroy route
8. Management
 - 8.1. Ability to CRUD organisations using unique identifiers
 - 8.2. Ability to add users to an organisation using an interlinking table.
 - 8.3. Organisations should be able to list all users within it.
 - 8.4. Ability to add a user to the platform. Details for the user to be able to login should also come under this object.
 - 8.5. Ability to add a vehicle to the platform. Vehicle details should also be collected from the DVLA using their JSON API.
 - 8.6. Image Asset Management for storing pictures within the system
9. Profile

Freddie Nicholson Computer Science NEA

- 9.1. Ability to view logged in user profile
- 10. Parental Interface
 - 10.1. Ability to book a route as a parent for their own children
 - 10.2. Ability to view current bookings
 - 10.3. Ability to view parent profile

Summary

The project I am taking on is complex but manageable. I will need to make sure I keep on time with deadlines and keep an open narrative with the client and potential users.

Documented Design

In order to solve the problem set out in the Analysis section. I will need to ensure that careful planning goes into the system design and considerations. In order to do this I will produce a section on the Documented Design of the system I end up using.

The proposed system is a particularly complex business model that will need to utilise many different complex program solutions. I will have to carefully consider what languages I use to solve a particular task. I will also need to consider how I will manage application data and plan in advance how models will interact with one another.

In order to analyse the use cases of the software I design and the functions that need to be implemented. I will use the standard of using structural diagrams to convey system design.

System overview

The system will work through the usage of a flask web app frontend with a python backend. User requests from the website will be sent to the various python flask modules in the background and requests will be served back with the necessary data.

Some python modules will need specialist libraries or tools such as Google OR Tools or OSRM. This allows for greater flexibility in the development of my application as I will be using the same language I use for my Computer Science course (Python) in conjunction with technologies I am already familiar with such as HTML5.

A database will also be set up in the background to store user data. This will be running on a virtual machine. As most school software is built on Microsoft SQL Server it makes sense to design my application within this environment.

Users will login to the restricted site and be able to access the functionality they need such as booking routes, calculating routes, adding users etc...

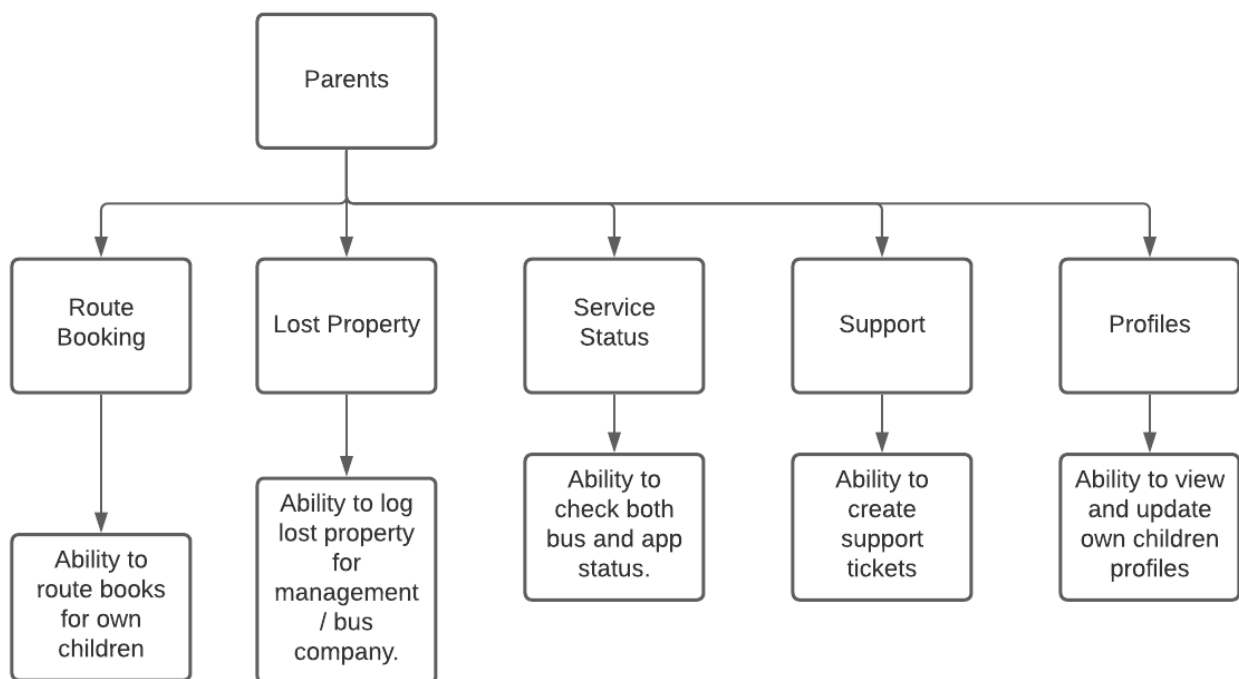
The system will sometimes need to call external APIs using JSON such as the DVLA to retrieve vehicle details. Additionally on the frontend the application will need to retrieve data from the backend dynamically and asynchronously which means internal APIs will need to be developed within flask too.

The system will be made up of 7 main modules that the user can interact with. Some should be restricted to prevent unauthorised changes. Each module will need specialised implementations to ensure they achieve the required functionality for example the routing module will need access to the OR Tools and OSRM implementation whilst the management module will need a CMS implementation.

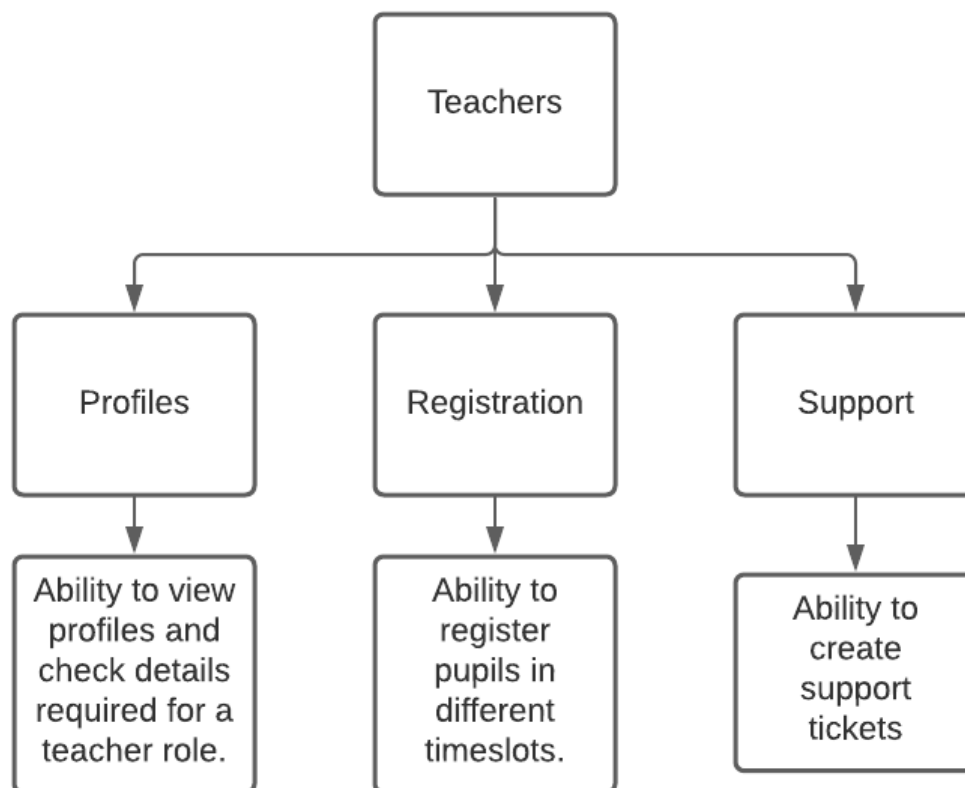
The system is complex however with careful consideration to the design documentation, a well designed NEA will follow

The first diagram I will show is the representation of the 'parent' structural diagram for system design. This system is made up of five modules:

Route Booking	<p>One of the most utilised modules of the system. Potentially hundreds of individuals will interact with the subsystems to do with the booking of routes and therefore it is important that ease of use and security is put in front of all other priorities during its design.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • The booking of a route for authorised children • The cancellation of a route booking • The ability to specify whether the route booking is recurring or a one off booking.
Lost Property	<p>Items are lost on buses frequently, standard procedures before the use of management software was the use of email to staff. This system has obvious issues in the way it can be utilised. These include unresponsiveness due to emails being missed during the working day, communication issues such as what bus the child was on and which driver would have been present.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Report creation for a lost item on a bus • Status updates to a report including more information or whether the item has been successfully found.
Service Status	<p>Ensuring that buses are running in adverse conditions or in mitigating circumstances is crucial to parents in times of need. Service Statuses should also be given to the various application components as well.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • The checking of a bus status, whether it will be running on a certain day or in the near future. • Overview of what application features are functioning properly.
Support	<p>Allowing parents to contact external staff who they may not be familiar with to seek help in a compliant way is important for many schools.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Lodging of support tickets in order to seek assistance from bus driver, management staff or school. • Ability to update tickets with information. • Ability to close tickets once issues are resolved.
Profiles	<p>Allowing parents to easily access information on themselves and child users linked to them is crucially important.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Ability to access personal profiles including users linked to the account in question. • Ability to update personal profiles information. • Update security details and other personal application features.



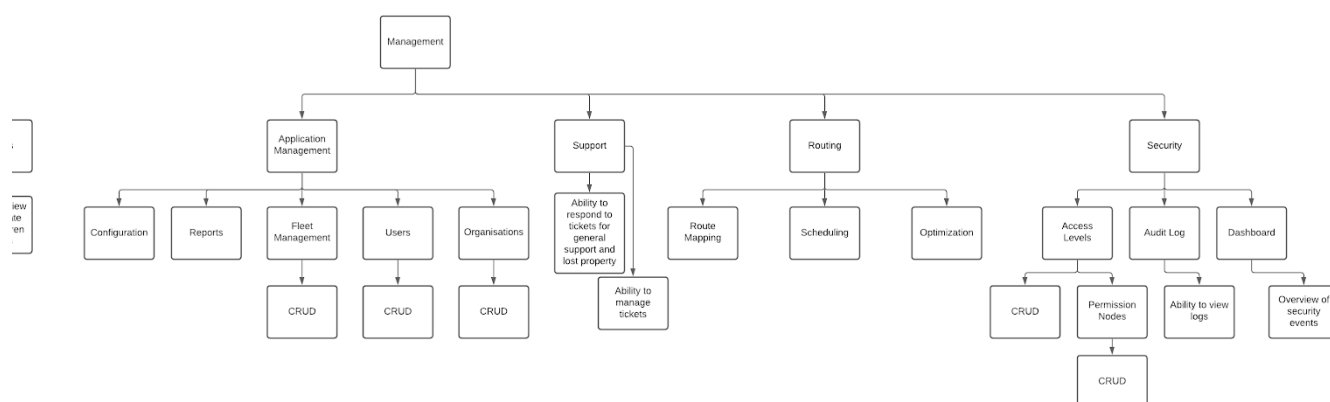
Profiles	<p>Teachers will require a more privileged access to parent and pupil data in order to fulfill their jobs in ensuring children are getting on the right buses and helping when queries around the service arise.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • The lookup of profiles with restriction to listing of users and irrelevant data shown. • The updating of basic details such as what route a child travels on.
Registration	<p>Careful auditing of children within a school is a government requirement and therefore it is crucial that this is taken very seriously. Teachers will need to record each transaction on and off a bus during different time slots during the day and it is crucial that there is no misunderstanding to how this module functions.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Registration of pupils in a clear and concise way. • Auditing of past registration of pupils. • Pupil registration history
Support	<p>Similar to the way in which parents launch support requests, teachers should also be able to see parents at the schools support requests for information when needed.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Escalation of priority of support tickets • Ability to add update to support tickets • Ability to close support tickets

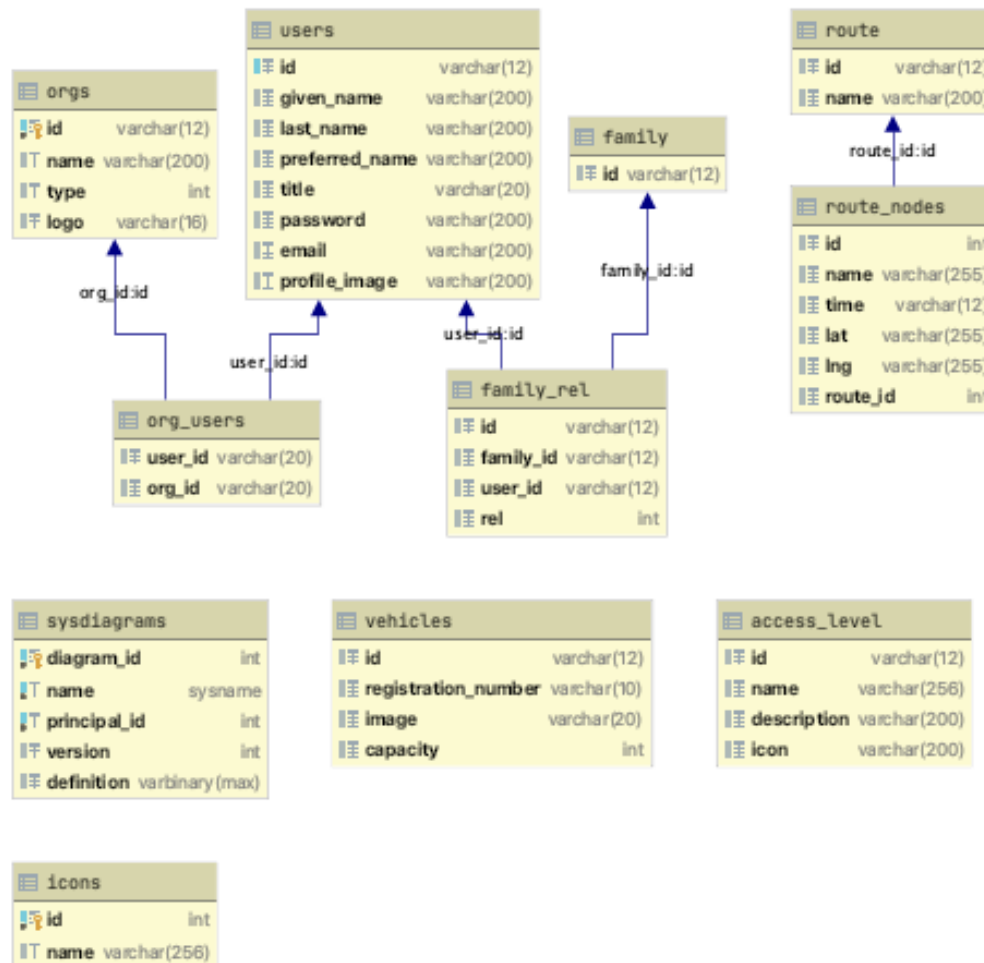


Management Structural Diagram

Application Management	<p>Utilities for application management. As an organisation administrator or a site administrator, tools need to be made available in order to allow for easy updates to the system relating to users, organisations, vehicles and other configuration menus.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • CRUD operations on Users • CRUD operations on Organisations • CRUD operations on Reports • CRUD operations on Vehicles (Fleet Management) • CRUD operations on Configuration
Support	<p>Application managers may need to respond to technical support tickets to do with the website and its use. Organisation managers may want to see the overall support issues occurring within their organisation and address them through additional training for staff or advice in weekly newsletters.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Ability to respond to tickets for general technical support and lost property • Ability to manage tickets across the platform.
Routing	<p>Organisation administrators and site administrators ability to be able to create, update, update, and destroy operating routes.</p>

	<p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Scheduling of routes for certain time slots. • Capacity management in order to ensure safe numbers on each route. • Ability to view bus route members and granular details such as driver and number plate down to each timeslot.
Security	<p>Ensuring users only have access to what they require on the system is crucial to ensuring compliance with legislation relating to Data Protection and for ensuring appropriate usage of the system.</p> <p>Processes that may be undertaken:</p> <ul style="list-style-type: none"> • Ability to access an audit log for activity that has occurred on the platform. • Ability to create access levels and define 'permission nodes' within them that allow or deny user actions. • Mission control style overview of all security activity within the platform displayed in an easy to read dashboard.

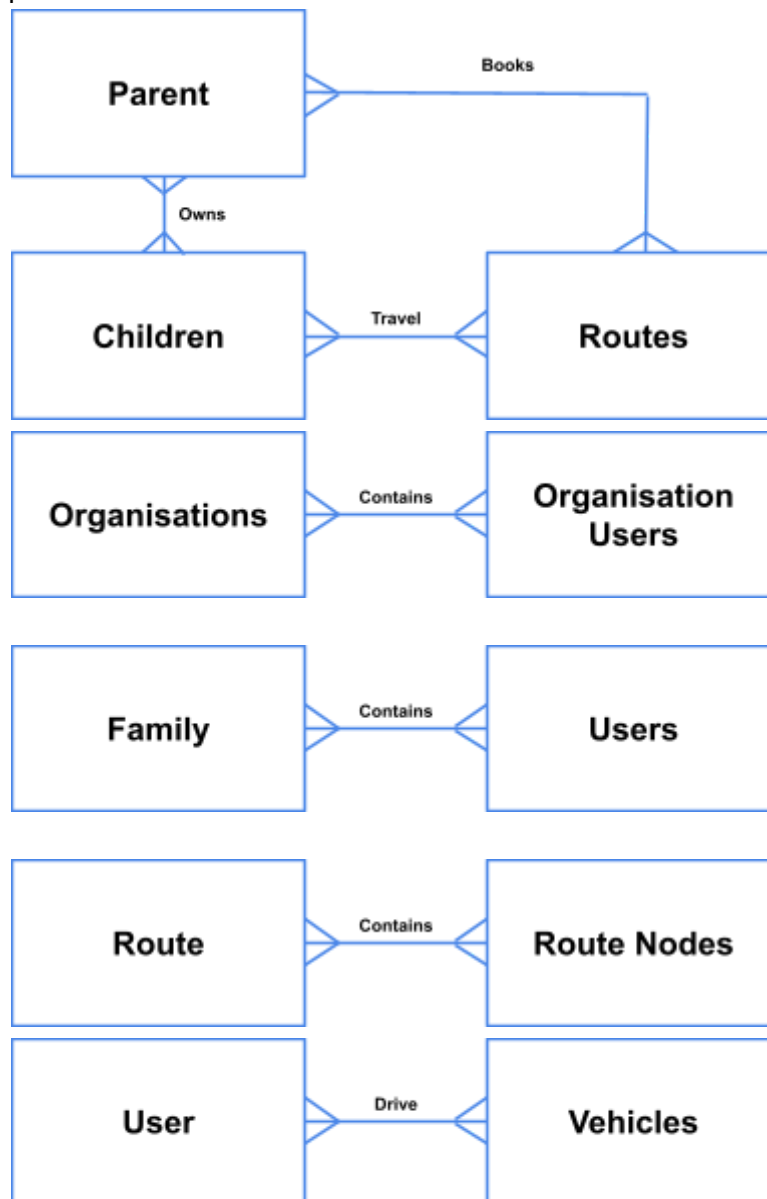




The above diagram shows the database schema. The database will be used to store all application data and to link each object together. The queries that will be used to perform the complex tasks in the application are discussed in the Technical section.

Entity-relationship diagrams

To help show the relationship between different objects (entities) I have created some entity relationship diagrams. These will help show me how to group objects together within the application and come up with JOIN statements for linking tables within the technical solution



Tables

users	
The users table is used to store information about users within the application. Details for the users login are also stored here as well.	
Column	Description
id PK	User's unique 12 digit ID number
given_name	User given name(s)
last_name	User last name
preferred_name	User preferred name
title	User title
password	User password

email	User email
profile_image	User profile picture name within uploads/ userprofilepictures directory

family

The family table is a record of all families within the system. It simply stores a list of IDs to indicate to the application how many families are present and which ID numbers are already taken.

Column	Description
id PK	Family unique 12 digit ID number

orgs

The orgs table is used for storing information about organisations within the application. It is referenced within the org_users table.

Column	Description
id PK	Organisation unique 12 digit ID number
name	Organisation name
type	Organisation type
logo	Organisation logo

org_users

The org_users linking table is used for storing information about organisation users. It links each user to an organisation. A user can be a member of multiple organisations or none at all.

Column	Description
user_id PK	Reference User unique 12 digit ID number
org_id PK	Reference Organisation unique 12 digit ID number

family_rel

The family_rel linking table is used for defining the relationship within a family of a user.

Column	Description
id PK	Family relationship unique 12 digit ID number
family_id	Reference Family ID number
user_id	Reference User ID number
rel	The relationship type, 1 for parent - 0 for child

route_nodes	
The route_nodes table is used for storing information about route nodes within a route.	
Column	Description
id PK	Route Node unique 12 digit ID number
name	Route Stop Name
time	Route Stop Time
lat	Route Stop Latitude
long	Route Stop Longitude
route_id	Reference route unique 12 digit ID number

vehicles	
The vehicles table is used for storing information about vehicles within the application.	
Column	Description
id PK	Vehicle unique 12 digit ID number
registration_number	Vehicle registration number
image	Vehicle image name
capacity	Vehicle seating capacity

access_level	
The access level table is used for storing information about access levels within the application for security purposes.	
Column	Description
id PK	Access Level unique 12 digit ID number
name	Access Level name
description	Access Level description
icon	Access Level icon

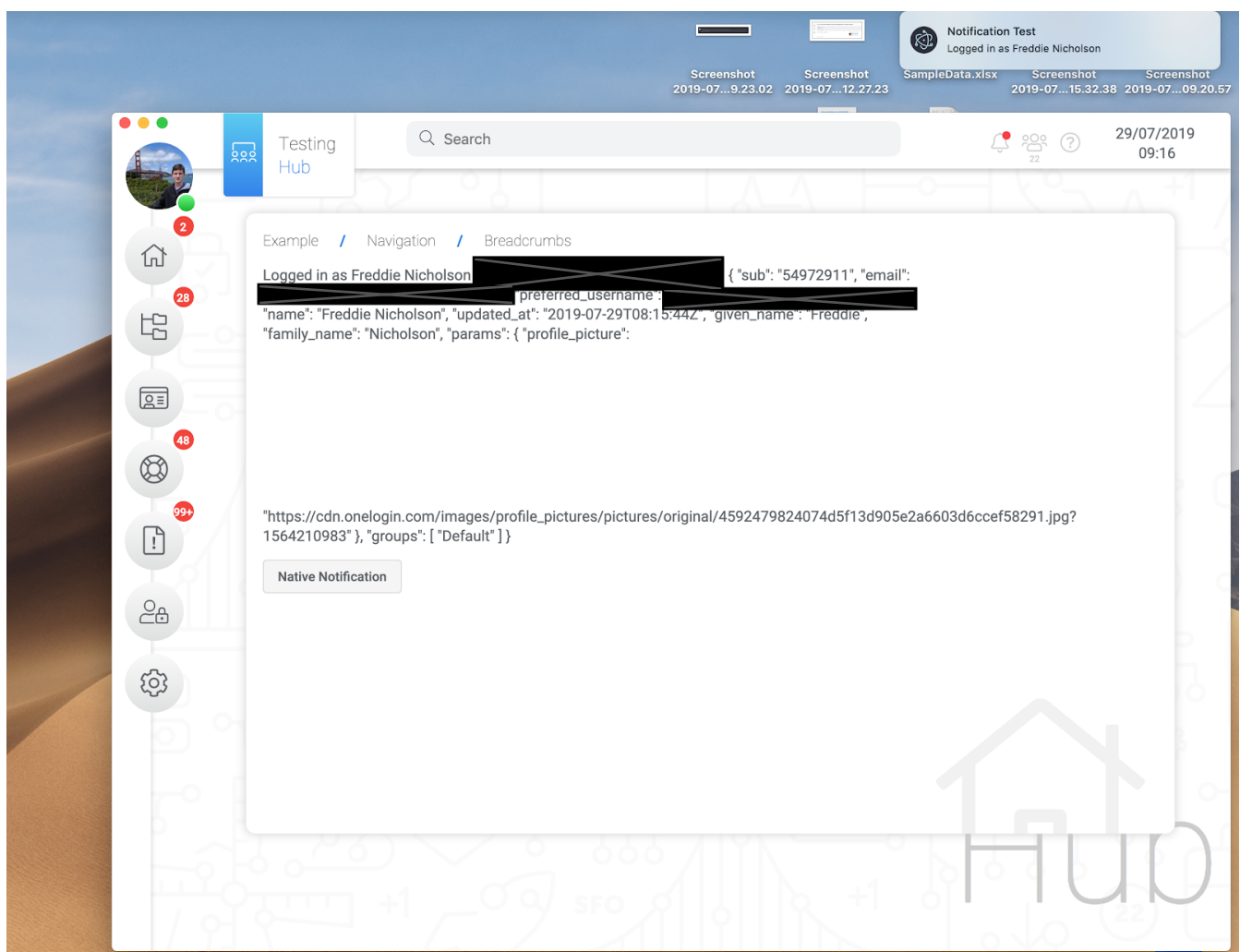
icons	
The icons table is used for storing information about icons	
Column	Description
id PK	Icon unique ID number

name	Icon name
------	-----------

route	
The route table is used for storing information about routes within the application	
Column	Description
id PK	Route unique 12 digit ID number
name	Route name

HCI Design Ideas

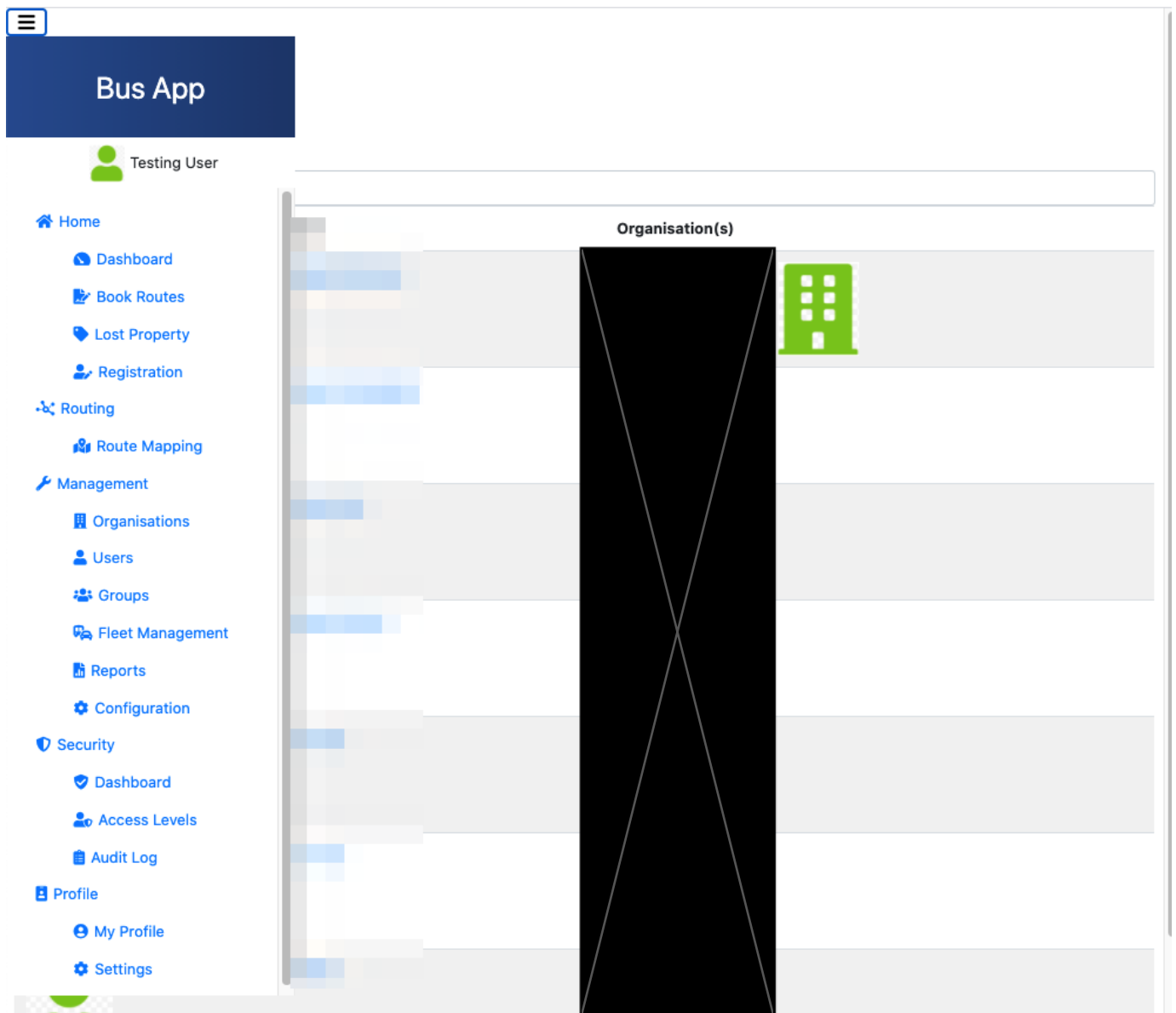
To help design the application interface, I have developed some designs using tools such as Electron in conjunction with HTML & CSS and design software such as Adobe Xd for prototyping.



This application interface I designed in Electron, an app that allows you to run a webpage as a native app within the operating system allowing for features such as notifications.

The navigation on the left hand side is made up of modules that can be clicked. Once clicked submodules appear in the main central area that can be selected and actions performed. There are

additional indicators such as users online, notifications and the date and time in the right hand corner. There are also notification indicators on each icon.

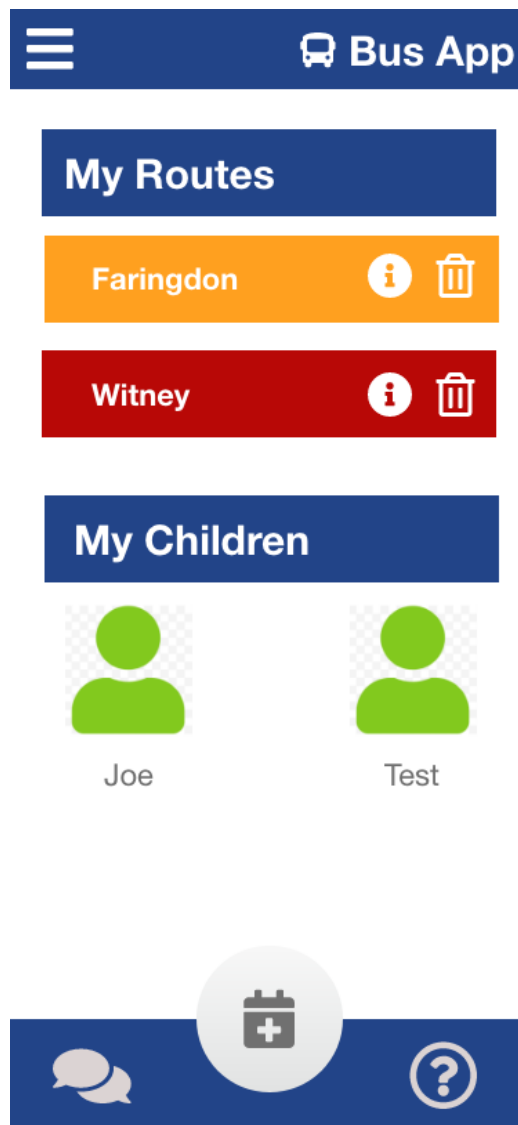


This application design was made using HTML, CSS and JavaScript. It consists of a navigation menu on the left hand side made up of each module. Underneath each module the submodules are defined.

The navigation menu dynamically updates allowing for each page to be consistent. Additionally further code can be added to adjust the navigation menu for a user's permission level to ensure they have the correct access to each application feature.

Bootstrap tables are used in the Design to offer a consistent interface for the user which is both intuitive and accessible. Font Awesome icons are used throughout to give a consistent icon format.

The user navigation menu contains dynamic content adjusted to the user with their profile picture and Preferred Name + Last Name. The user can also click the user indicator to open an additional context menu. The navigation menu is also operated by a 'hamburger' icon allowing it to slide in and out of view as needed. This design was the final design used within the application as it is very time efficient which is critical for my NEA. It is also very easy to make quick changes.



This design was made using Adobe Xd using a mobile device template.

It would be used in conjunction with a website similar to that shown beforehand. Many management applications and educational products that release often start as a website and then end up as a PWA (Progressive Web Apps) on Mobile. Whilst PWAs are great cost wise, they often lack in quality and have several issues such as a slow interface, unintuitive accessibility features and in general are a bodge of a solution.

End users, especially parents, often find apps are much more convenient compared to websites. They offer unique features such as notifications and are also readily accessible when a parent is on the move and needs to make a quick booking in a hurry.

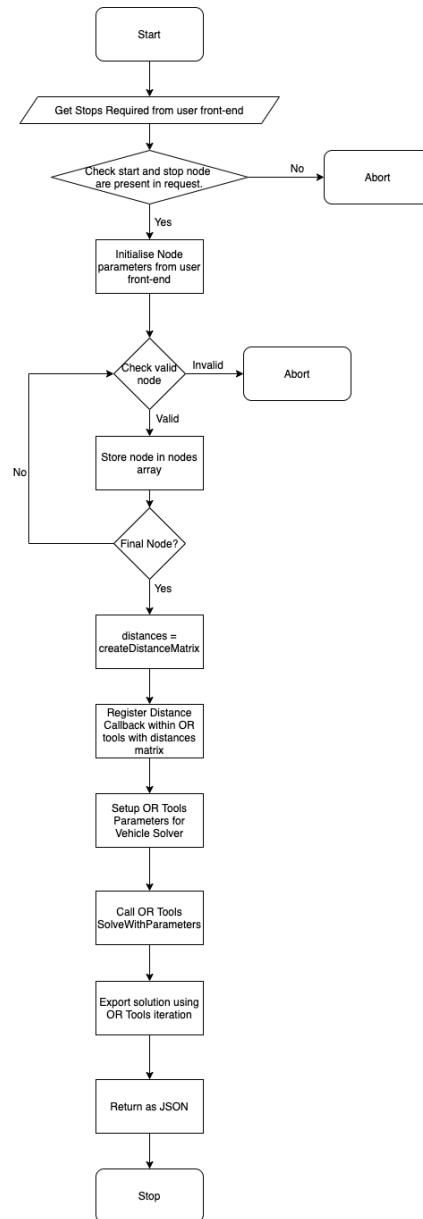
Although the development of a mobile app is out of the scope of my NEA, if I were to go on to turn the project into a commercial product. A native app written with ease of use in mind would be a must.

Algorithms

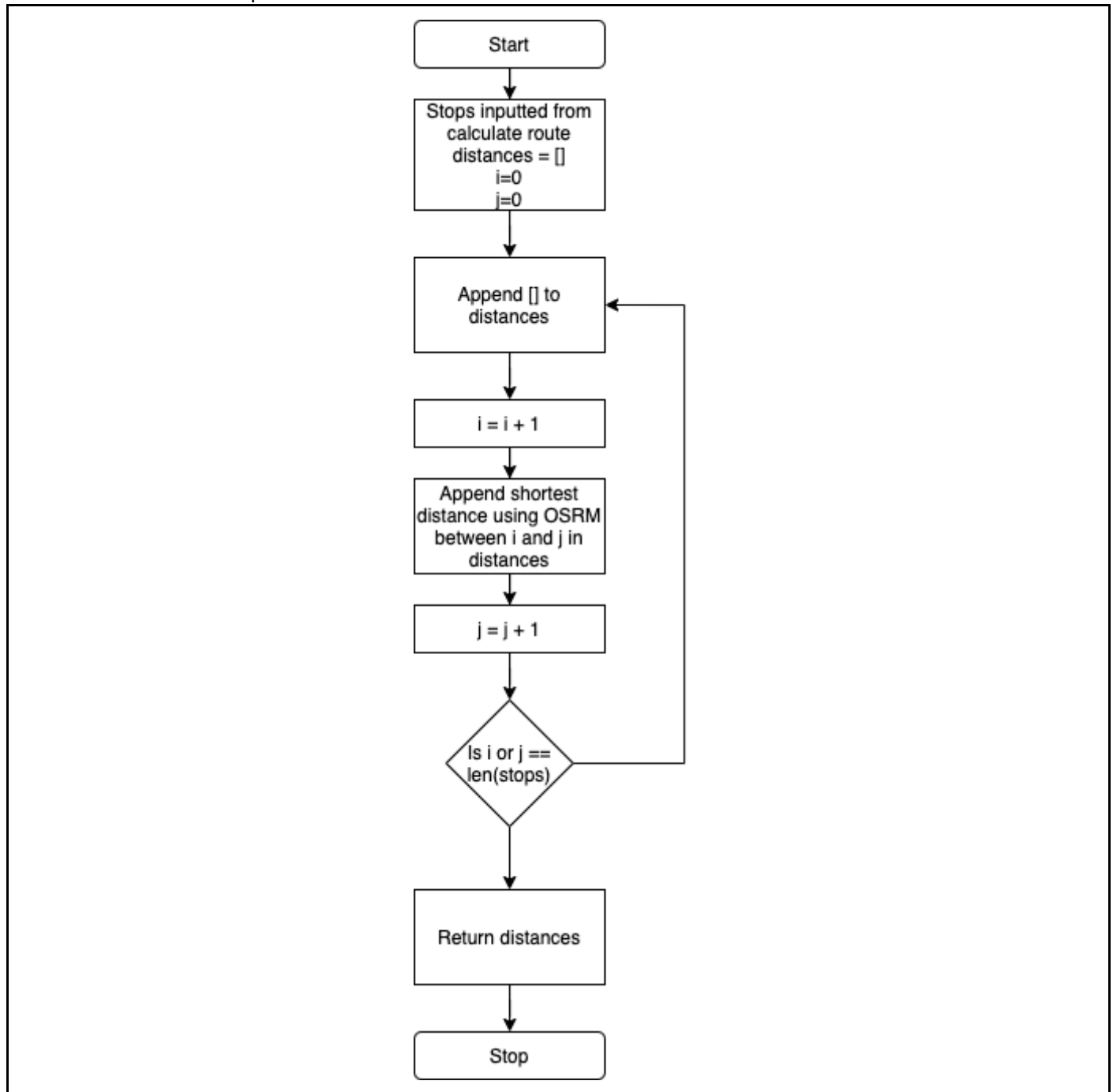
I will use several complex algorithms within my application which I will need to carefully consider. To help with the technical implementation of these algorithms I will set out how some of these will work below. SQL is shown within the Technical Solution section.

Routing Algorithm

The main routing algorithm used within the application is abstractly shown in the flowchart below.



The technical implementation of this flowchart is explained in the technical solution.



This is the createDistanceMatrix function seen within the main routing algorithm. It is also explained in further detail within the technical solution section.

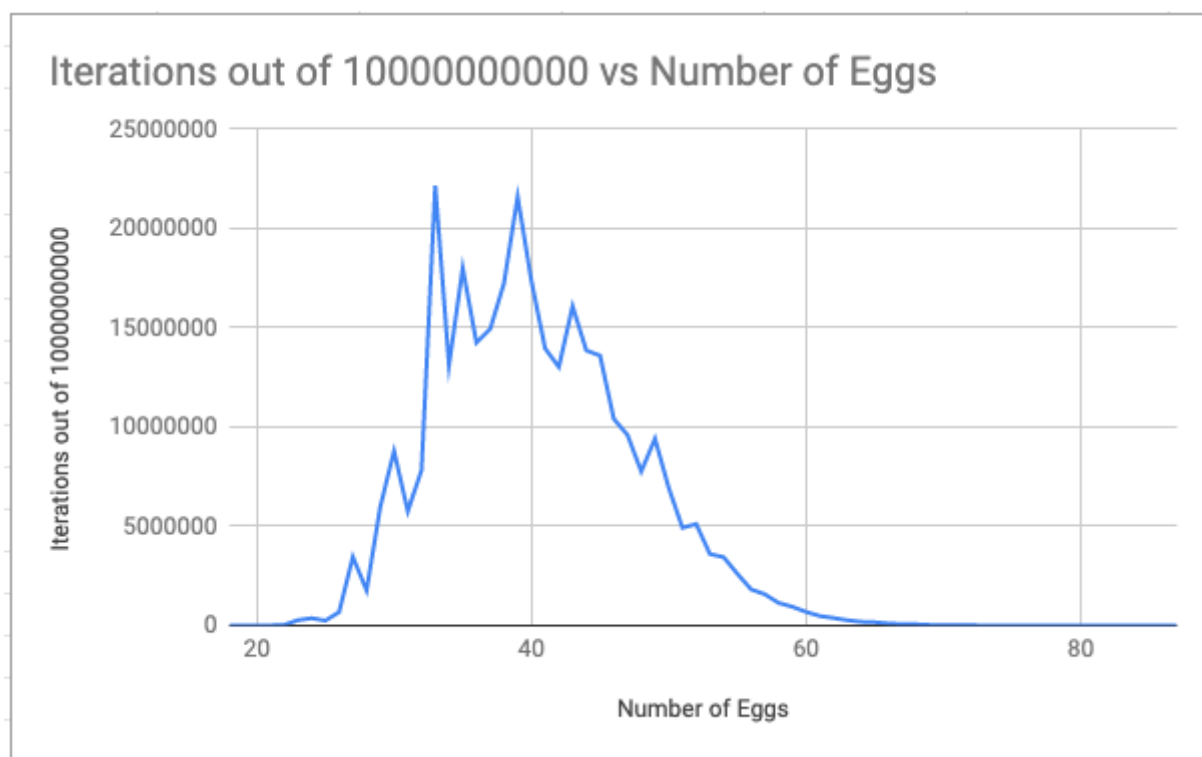
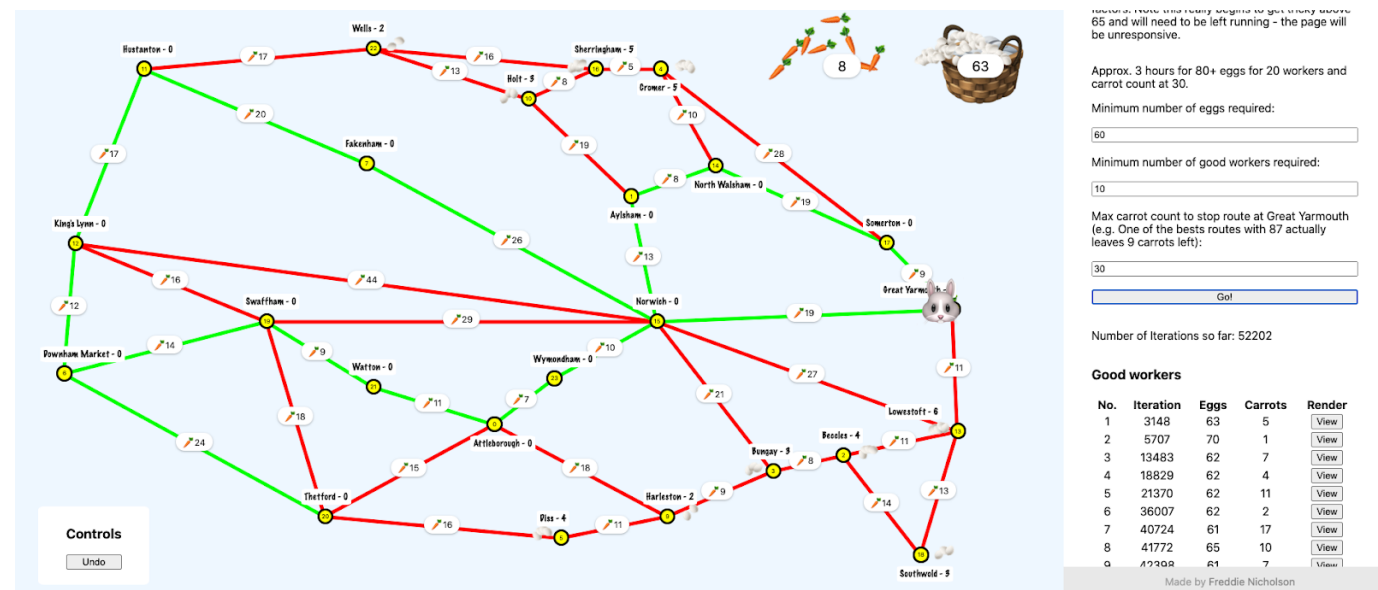
Routing Algorithm Random Alternative (Egg Challenge)

Over the Easter holidays of 2020, I attempted to solve a problem we tackled in a Maths lesson.

The problem was set in Norfolk with a cluster of towns. There were two main constraints: the number of carrots (miles) to minimise and the number of eggs to maximise.

At the point of time when writing the web app, I was not acquainted with the knowledge to solve this problem in any discrete way and therefore instead utilised essentially a random number generator to choose a random path and calculate the results at the end. If the path matched a certain criteria then the route would be displayed.

As the computer performed the calculations very quickly I could just type the low egg counts and get an instant response. For higher egg counts it took exponentially got longer to the point of taking a day. However this still achieved the end goal of finding out the max egg count possible, 87.



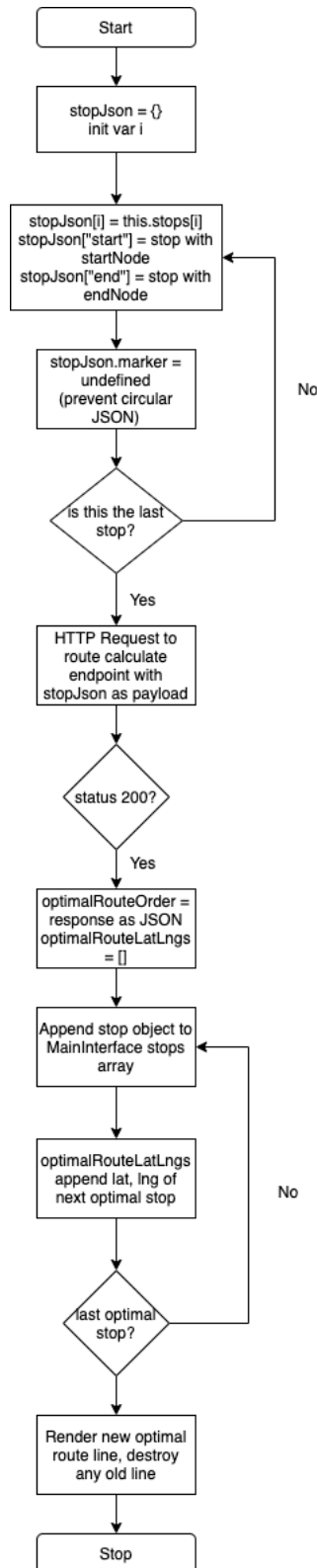
The above chart shows the distribution of eggs collected when the randomisation was used. It came out as a normally distributed curve.

This approach, although it worked for this limited case, is very impractical in the real world. It is not acceptable to have a route calculation taking a day to calculate and randomisation is netrousily bad within algorithmic decision making as there is uncertainty in how long it will take.

Therefore, randomisation is definitely not the approach to take in the implementation of my system.

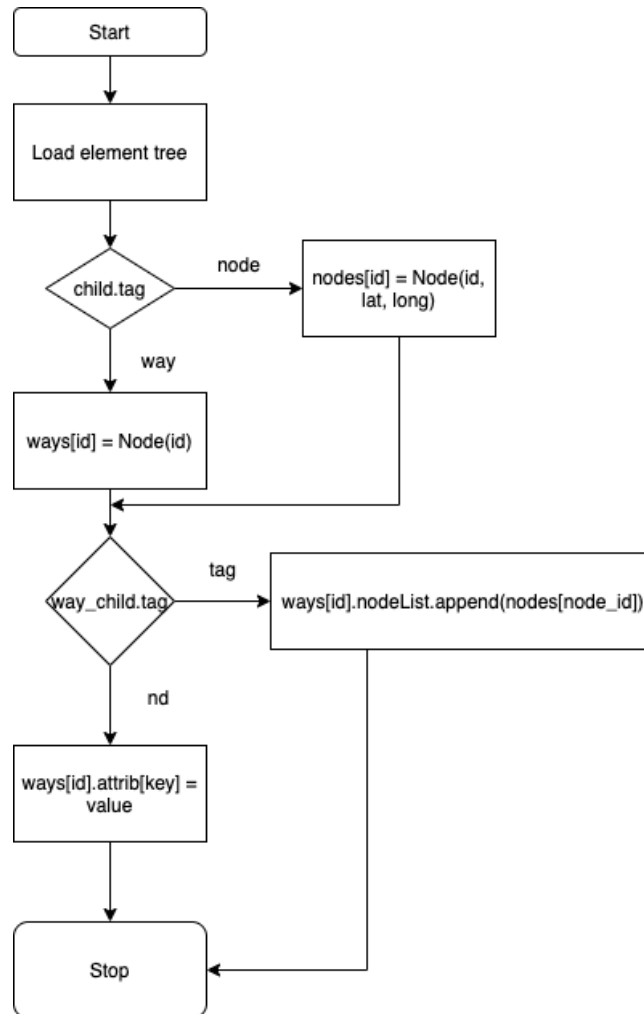
Routing Frontend List Algorithm utilising Request and Response

This algorithm is used for displaying the optimal route line within the interactive frontend route plotter that is shown within the technical solution.



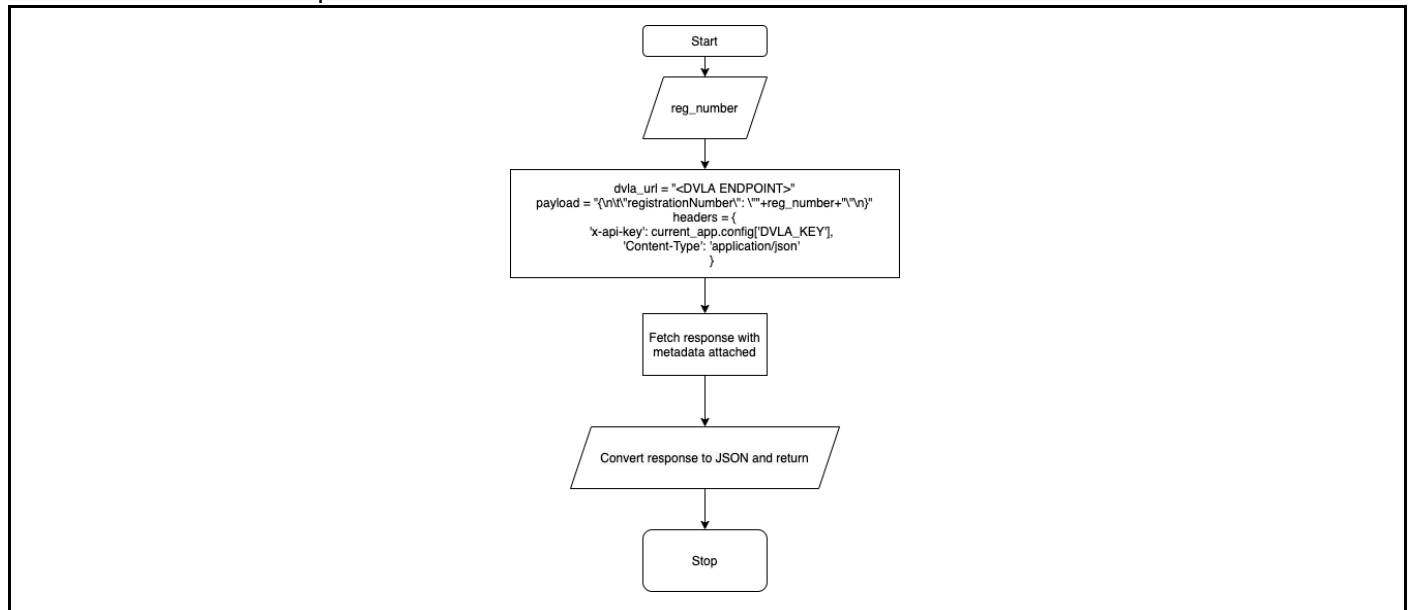
Dynamic Generation of Objects from OpenStreetMap

This algorithm loads the element tree from the OpenStreetMap OSM file into the object orientated python format for testing.



Parameterised Web Service DVLA API for parsing JSON vehicle details

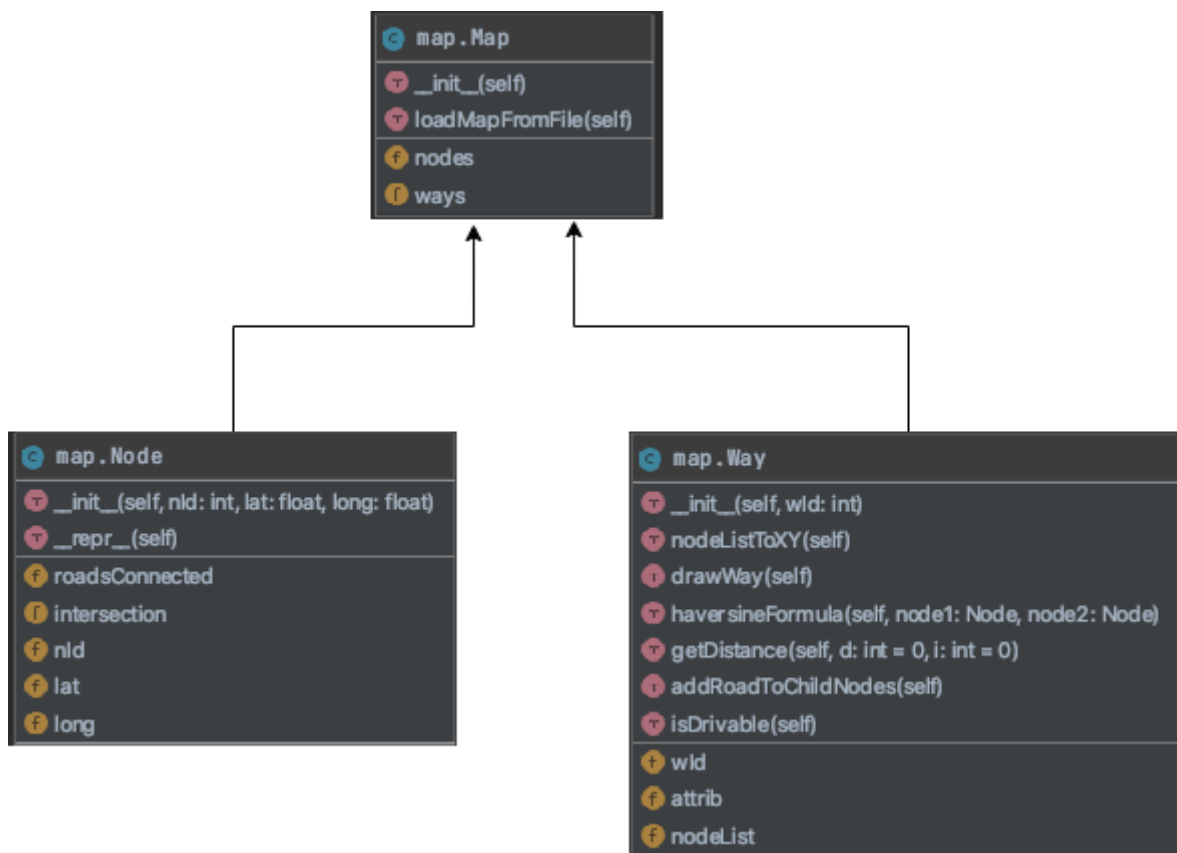
This algorithm fetches vehicle registration details from the DVLA using JSON. Its technical implementation can be found within the technical solution.



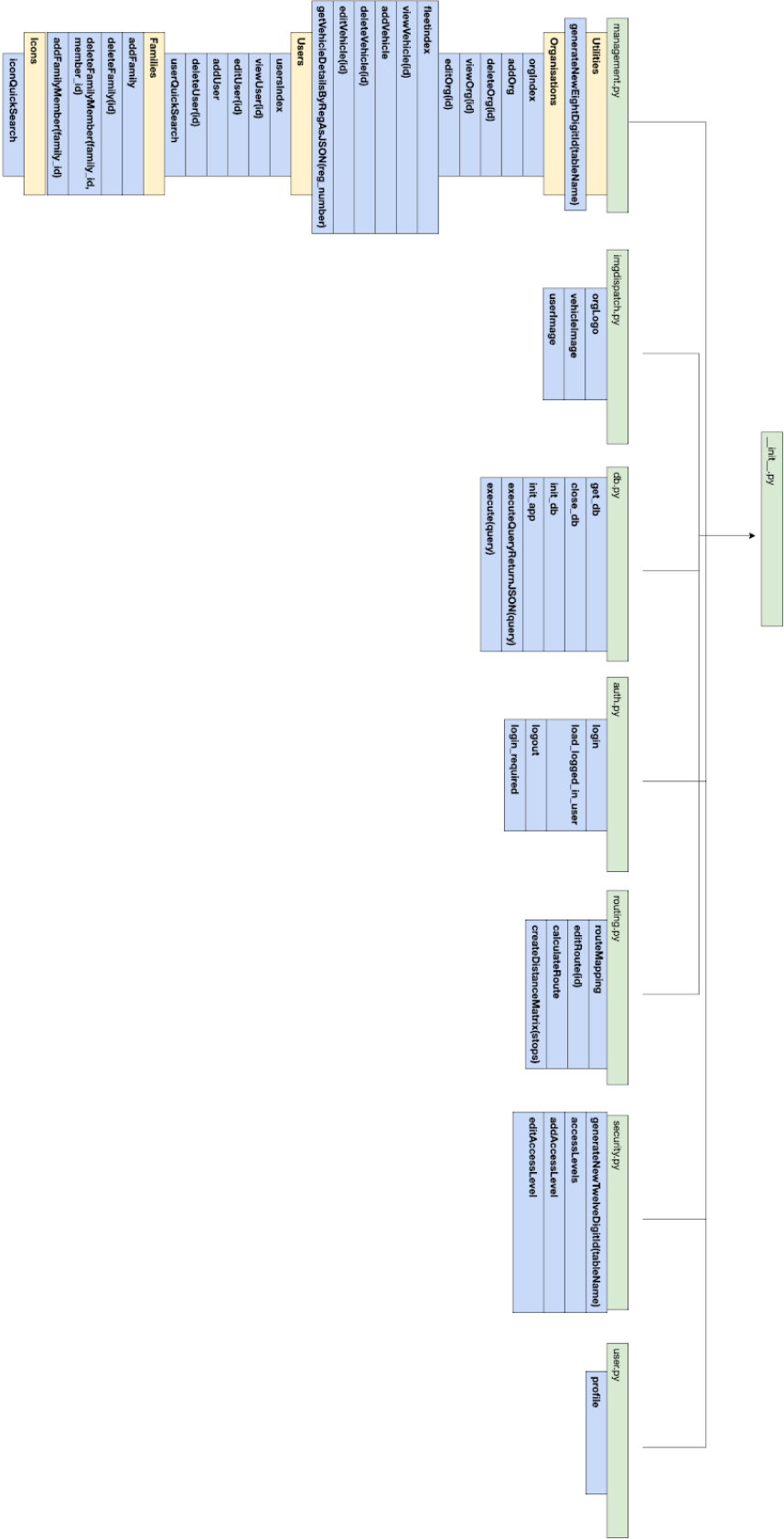
OOP

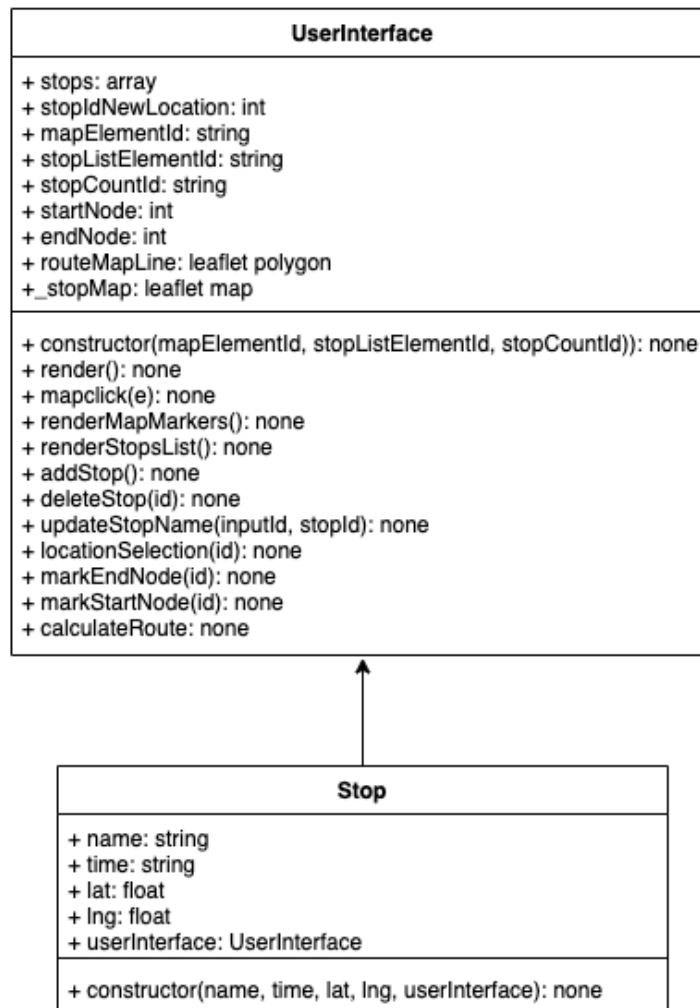
Object Orientated Programming (OOP) was used throughout the project in both the initial testing phase and the final development phase with Flask and Javascript.

Map Testing



Flask Web App
Flasks representation utilises ‘blueprints’.

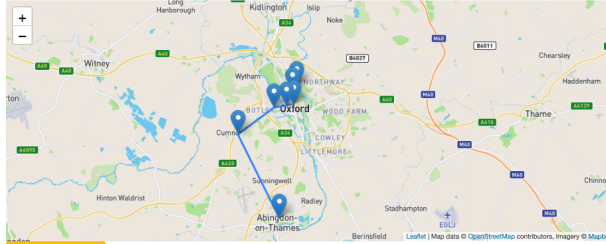




Examples of Data structures

In this section, I will look over some of the data structures that I will utilise within my application. Choosing the right data structure is essential for implementing a flexible system that can be easily added onto.

Graphs



A set of nodes can be determined to be a graph, I will need to write my own custom implementation in order to be able to achieve a readable route that can be presented to the user.

It is important that I am able to tell which order the nodes are placed in and therefore it seems a list style system would be most appropriate

Single-dimensional arrays

Within the application, there will be several examples of list usage

Quickly and easily storing and retrieving a list of nodes is essential within my application and a list object seems the best object to do this. Lists also come with extra functionality allowing for greater flexibility such as removing nodes, duplicating nodes and adding nodes as quickly as possible. You can also use the selector queries in Python allowing for advanced manipulation.

For the JavaScript frontend, some features are going to require lists. For example, any quicksearch is going to need to store the displayed results in order to present them to the user. This also makes sense with the ordered nature of lists.

Dictionaries

Dictionaries will be most frequently used as JSON within the web application. Examples of applications where this may be required is for backend - frontend communications in order to display to a user specified content such as a list of nodes on a map.

External API Services such as DVLA present their data in the standard JSON format and therefore this can be converted to a dictionary for usage within other parts of the application.

Multi-dimensional arrays

Multi-dimensional arrays will need to be utilised for the storage of Matrices of nodes. Each matrix entry needs a column and a row and therefore two nested arrays are required i, j . This allows for quick iteration and calculations to be performed.

Records (Database)

All data for the Content Management System part of the application are going to need to be stored within the Database. The database is running Microsoft SQL Server which follows standard SQL syntax. Additionally, Microsoft SQL Server allows for the additional functionality such as returning a query as a JSON object to be readily converted to a dictionary within Python making operations simpler.

Classes

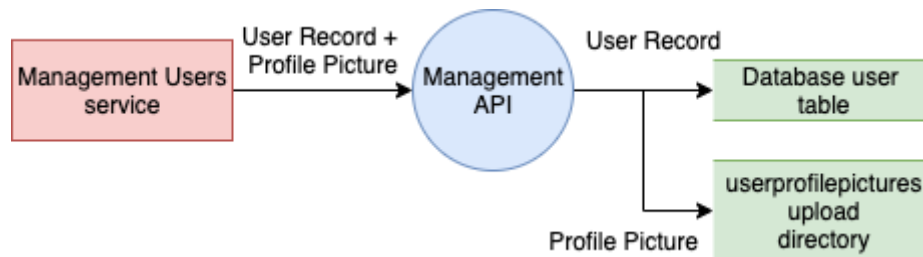
Classes are used within the testing phase of the application to build an abstract model of the OpenStreetMap and perform quick operations to test the algorithm. Objects can also have their own methods allowing for better code readability.

Classes are used on the frontend to render the interface properly. An example could be the route mapping page which utilises a UserInterface object which handles the rendering of the UserInterface. In addition to the UserInterface there is a stop object which can be appended to the UserInterface stops array. The UserInterface then has a render method allowing for easy display of the stops within the interface.

Drawing a data flow diagram will help depict the input / output nature of my system and the processing that will go on behind the scenes for common workflows.

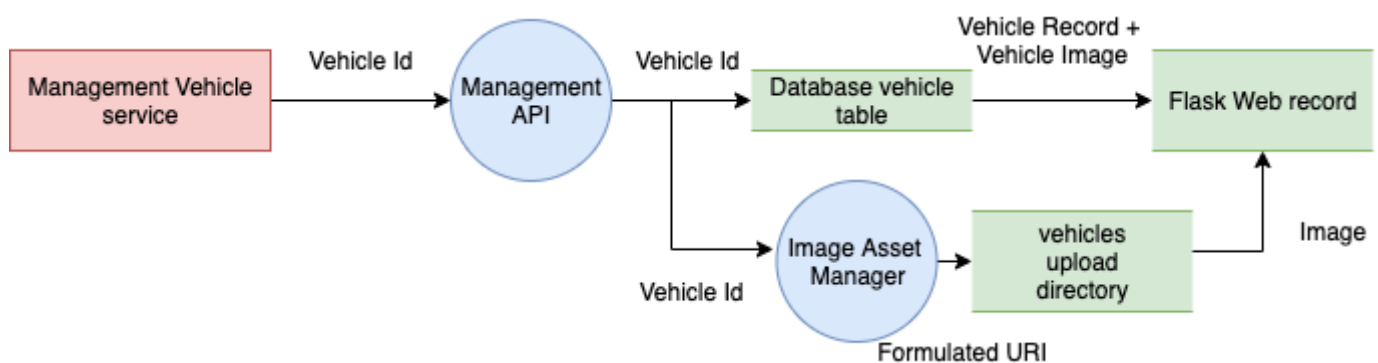
Registering a user

This is the data flow diagram for registering a new user within the management section of the application.



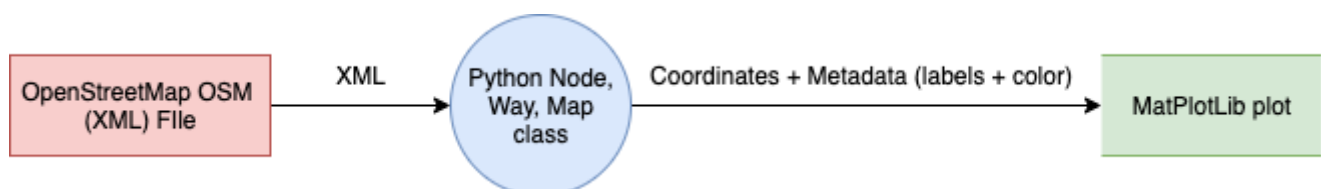
Looking up a vehicle

This is the data flow diagram for retrieving a vehicle within the management section of the application.



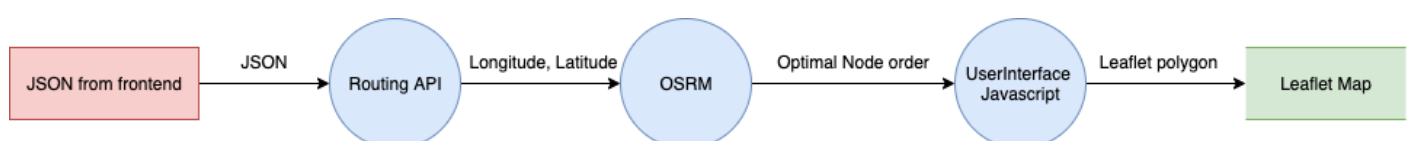
Loading an OSM file

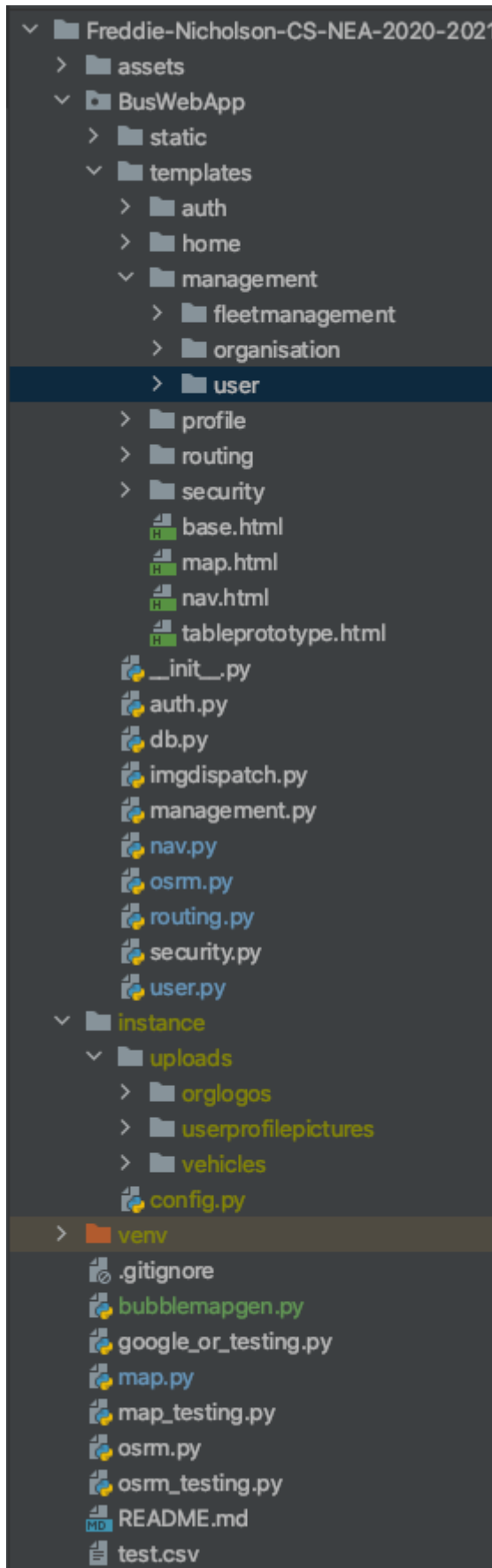
This data flow diagram depicts the OpenStreetMap loading of ways and nodes into a Matplotlib scatter plot.



Calculating a Route

This data flow diagram shows the flow of data when calculating a route within the frontend application.





This is the file structure of my flask web app alongside the testing code used for the initial phase of the project.

All the Flask web app code is stored within the BusWebApp directory.

The BusWebApp folder consists of 2 folders, static and templates. The static folder is for storing static content that can be readily returned to the end user e.g. icons. The templates folder contains all of the flask HTML templates that render within the application.

The rest of the BusWebApp Directory contains each module required for the application.

A level up from the BusWebApp directory is the flask instance folder which contains the configuration (API keys, SQL connection details) and the instance file upload folders:

- orglogos for Organisation Logos within the application
- userprofilepictures for User Profile Pictures within the application
- vehicles for Vehicle Images within the application

The virtual environment for the application is stored within venv.


The rest of the python files were used during testing for designing the algorithm to solve the problem.

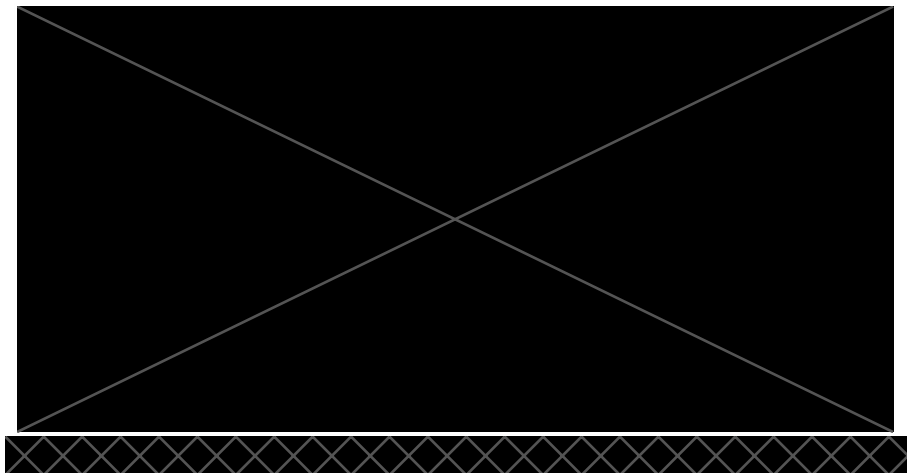
The primary objective of my coursework is to optimise the route taken by buses based on predefined criteria. Before I start developing the entire system it makes sense to write a draft of the final algorithm that will be used to determine routes within the application.

Reading from OpenStreetMap

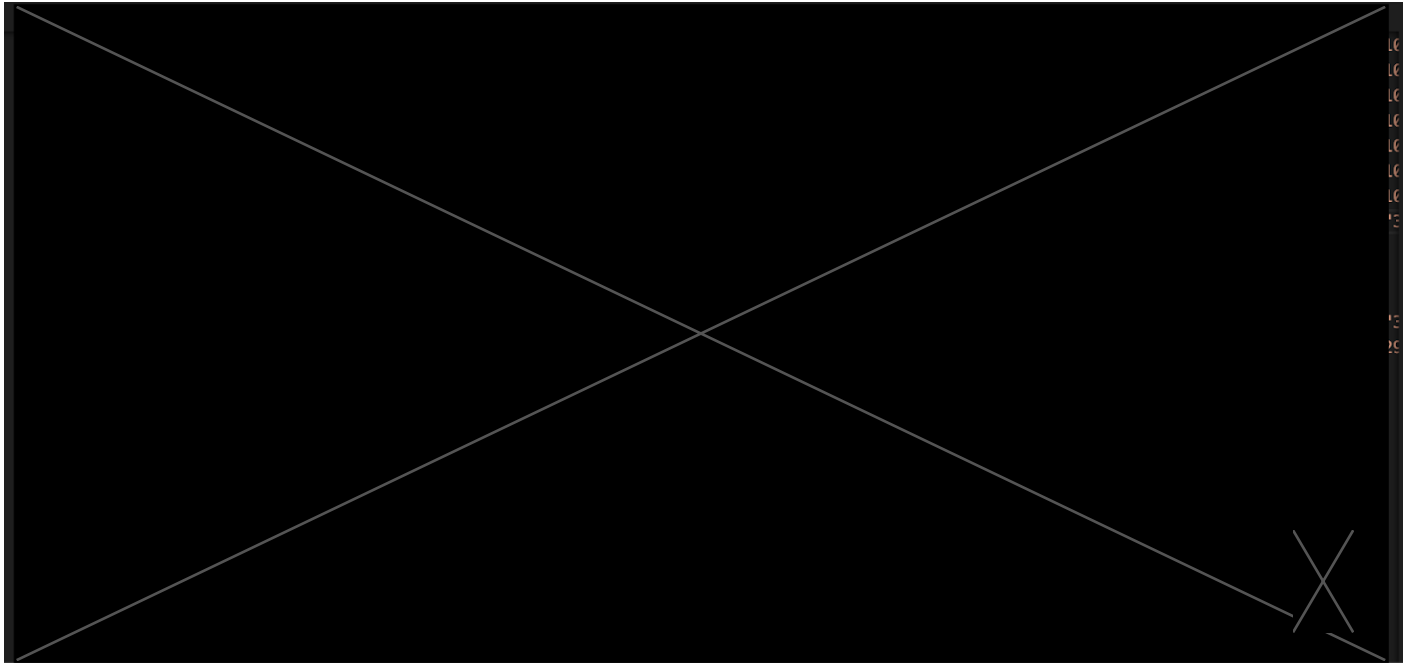
In order to begin writing the algorithm to route vehicles around the area, I will need to gather data that describes the roads around the School. A manual approach would be intractable with one person and inaccurate. Therefore the use of an external mapping provider will need to occur. One of the most popular commercially available platforms is Google Maps. The Google Maps APIs are arguably the most widely available for any kind of mapping application. However, they are a commercial product and often cost money for enterprise logistical features such as what I will be using. Additionally the Google Maps APIs perform a lot of the underlying logic for you and therefore my ability to customize my project down to the fine detail would be limited.

OpenStreetMap is a more attractive alternative. It is a community project worked on by several map editors around the world over several years to map the planet. The project is open source meaning that anyone can contribute. Most of the code and data held by the company is freely available for anyone to use which is ideal for my educational usage. OpenStreetMap is also used by the incredibly popular developer service MapBox which powers many mapping applications and therefore there is a huge developer community behind it in case I run into any major issues when utilizing the data provided. The map editors are also often meticulous in providing data and therefore there is a huge amount of data that I can utilise in the development of my application.

 It is familiar to any other mapping service and shows the various types of roads in a clear manner and shows finer detail such as pitches, water features and ground type using easy to understand colour coding without requiring an explanation to the user.



However in order to write an algorithm to find the quickest route between nodes in the surrounding area for testing, I will need to extract this data in a machine readable format. Helpfully the OpenStreetMap website pictured above offers the feature to download the map data in focus as an OSM (XML) file, pictured below.



Each XML element has a tag name used to represent the abstract map element in OSM. There are also additional attributes to represent specific features of each element. One of the core element attributes is its Unique ID. The Unique ID is used by other OSM elements such as a way to build up a feature, for example the road feature shown above is made of several 'nd' (node) tags. The road then has additional 'tag' elements used to represent a key-value pair e.g. key 'cycleway' value 'share_busway' used to indicate the cyclists share their cycleway with a bus lane. There are also useful admin attributes such as time updated, changeset IDs and usernames.

Those that are familiar with graphs in terms of decision problem solving will quickly see that there are some clear links. Nodes are similar to graph nodes and ways can be used to determine edges between them. We will explore how we can utilise this in the development of the testing.

Here is an OpenStreetMap query for nearby features for a miniature railway, a unique attraction with several elements being used.

Search

Where is it?

Go

Go

Query Features

Nearby features

Tree #5564350521 1

Footpath #9133619 2

Building #427579140 3

Miniature Rail #706400695 4

Enclosing features

Park Cotteslowe & Sunnymed Park

City Boundary Oxford

County Boundary Oxfordshire

Ceremonial Oxfordshire

Traditional Oxfordshire

Statistical Berkshire, Buckinghamshire and Oxfordshire

Police Thames Valley

Region Boundary South East

Electoral Boundary South East

Way: 706400695

Version #2

Sorting out cutswall park

Edited 9 months ago by randomhacks ·

Changeset #83773902

Tags

railway

miniature

Nodes

► 45 nodes

Looking at the railway in further detail we can see it has a 'railway' tag assigned to it with the 'miniature' value. Although it may seem clear to us as an end user that this is not a operating railway line, it is important that our mapping engine has some context based features like this. Otherwise, for example, in development of a railway management application, a developer may accidentally take into account unique edge cases such as the railway pictured above!

Object-oriented programming for Representing Map for Testing

The Object-oriented programming paradigm makes the most sense for my testing as OpenStreetMap already uses abstract representations for its own elements which I can recycle in the code of my testing.

Node Class

```
class Node:
    """
    A node in openstreetmap. Latitude and Longitude are stored as well as whether the node is a
    junction (intersection).
    """
    def __init__(self, nId: int, lat: float, long: float):
        self.nId = nId
        self.lat = lat
        self.long = long
        self.roadsConnected = []
        self.intersection = False

    def __repr__(self):
        """
        Returns a representation of the node in string form with its unique ID.
        :return:
        """
        return f'Node({self.nId})'
```

Node Class

The node class, as it is so fundamental, doesn't require much processing but has five main attributes:

nId	Node ID	The way ID (an integer) is the same as the OSM representation. It is used throughout the program to refer to this specific node.
lat	Node Latitude	The node's latitude (a float).
long	Node Longitude	The node's longitude (a float).
roadsConnected	Roads Connected	An array of way objects where the node lies.
intersection	Intersection Boolean	Whether the node is connected to two or more ways as a boolean.

and two functions:

__init__	Initialising function for assigning constructor variables to object.
-----------------	--

<code>__repr__</code>	Returns a basic representation for debugging purposes. e.g. Node(23313213)
-----------------------	--

Way Class

```

class Way:
    """
    A way in openstreetmap consisting of a list of nodes. Various attributes are also stored with
    metadata about the way.
    """
    def __init__(self, wId: int):
        self.wId = wId
        self.nodeList = []
        self.attrib = {}

    def nodeListToXY(self):
        """
        Returns both a list of X and Y coordinates for nodes within the way specified.
        :return:
        """
        mapX = []
        mapY = []
        for i in range(len(self.nodeList)):
            node = self.nodeList[i]
            mapX.append(node.lat)
            mapY.append(node.long)
        return mapX, mapY

    def drawWay(self):
        """
        Plots an image of the way on a matplotlib chart.
        :return:
        """
        mapX, mapY = self.nodeListToXY()
        #plt.scatter(mapX, mapY)
        mapLabel = self.wId
        if 'name' in self.attrib.keys():
            mapLabel = self.attrib['name']
        plt.plot(mapX, mapY, label=mapLabel, color="black")

    def haversineFormula(self, node1: Node, node2: Node):
        """
        Uses the Haversine formula to find the distance between node1 and node2 in km.
        :param node1:
        :param node2:
        :return:
        """
        lat1 = math.radians(node1.lat)
        lat2 = math.radians(node2.lat)
        long1 = math.radians(node1.long)
        long2 = math.radians(node2.long)

        distanceLong = long1 - long2
        distanceLat = lat2 - lat1
        return (2 * math.asin(math.sqrt(math.sin(distanceLat/2)**2 + math.cos(lat1) * math.cos(lat2) *
        math.sin(distanceLong/2)**2)))*6371

    def getDistance(self, d: int = 0, i: int = 0):
        """
        Recursive function to find the total distance of a way in km.
        :param d:
        :param i:
        :return:
        """
        if (i+1 < len(self.nodeList)):
            d+=self.haversineFormula(self.nodeList[i], self.nodeList[i + 1])
            return self.getDistance(d, i+1)
        else:
            print(f'End of Way {i} nodes')
            return d

    def addRoadToChildNodes(self):
        """
        For each node belonging to a way, each node will have this way appended to its own
        roadsConnected list.
        Allows for us to see junctions (nodes with more than one road connected)
        :return:
        """
        for node in self.nodeList:
            if(self.isDrivable()):
                node.roadsConnected.append(self)

    def isDrivable(self):
        """
        Returns a boolean to indicate whether a route is drivable by car / bus
        :return:
        """
        drivable_highway = ['motorway', 'trunk', 'primary', 'secondary', 'tertiary', 'unclassified',
        'residential']
        if 'highway' in self.attrib:
            for highway_opt in drivable_highway:
                if(highway_opt == self.attrib['highway']):
                    return True
            return False

    def __repr__(self):
        """
        Returns a representation of the way in string form with its unique ID.
        :return:
        """
        return f'Way({self.wId})'

```

Way Class

The way class has three main attributes:

wld	Way Identifier	The way ID (an integer) is the same as the OSM representation. It is used throughout the program to refer to this specific way.
nodeList	Node listing	It makes sense to have a node list (an array) when processing on this small of a scale and in order to plot the way in matplotlib for testing.
attrib	Attributes	Listing the attributes from the xml file for the way in a more python friendly way (using a dictionary).

and eight main functions:

__init__	Initialising function for assigning constructor variables to object.
nodeListToXY	Returns a tuple for numpy / matplotlib that contains in chronological order the way's child nodes latitude and longitude. This allows for easy plotting to help visualise data.
drawWay	Using nodeListToXY plots each point on the way to form a line. A label is then added to be shown in the plot legend.
haversineFormula	<p>Uses the Haversine formula to work out the shortest 'distance' between two longitudes and latitudes. Sailors used to use haversine tables similar to log tables allowing them to perform the calculations themselves.</p> $\text{Haversine formula: } a = \sin^2(\Delta\phi/2) + \cos \phi_1 \cdot \cos \phi_2 \cdot \sin^2(\Delta\lambda/2)$ $c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a})$ $d = R \cdot c$ <p>where ϕ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371km);</p> <p>Taken from: https://www.movable-type.co.uk/scripts/latlong.html</p> <p>I have represented the formula in Python as a function.</p> <p>The function is used to find the shortest 'distance' between two points. As the earth is a sphere, using the latitude and longitude is inaccurate at small distances and for mission critical mapping applications such as the program I will be designing it is important to ensure precision.</p>
getDistance	<p>A recursive function that utilises the haversineFormula to find the total distance of way.</p> <p>The function recursively gets each adjacent pair of nodes within a way and calculates the distance between them incrementing the distance d each time.</p> <p>The function is recursively called until it has reached the final node in the ways nodeList. Once this has occurred the function returns the distance d.</p>
addRoadToChildNodes	For each node within the way nodeList, append this way to the roadsConnected list allowing for easy calculation of when a junction occurs.
isDrivable	Calculates whether a way is drivable using the 'highway' OSM tag.

	Roads with the tag 'motorway', 'trunk', 'primary', 'secondary', 'tertiary', 'unclassified' and 'residential' are considered drivable. If the road contains this tag then a boolean indicating True is returned otherwise False is returned.
__repr__	Returns a basic representation for debugging purposes. e.g. Way(841432931)

Map Class

```

class Map:
    """
    Openstreetmap representation of a real world map. Nodes and Way objects are the main source of
    information for the project.
    """

    def __init__(self):
        self.nodes = {}
        self.ways = {}
        self.loadMapFromFile()

    def loadMapFromFile(self):
        """
        Loads an openstreetmap xml map from file.
        :return:
        """
        with open('assets/OX26SS_SmallOpenStreetMap.osm') as map_file:
            map_xml_root = ET.fromstring(map_file.read())
            for child in map_xml_root:
                if(child.tag == 'node'):
                    #print(child.tag, child.attrib)
                    lat = (float(child.attrib['lat']))
                    long = (float(child.attrib['lon']))
                    nId = int(child.attrib['id'])
                    self.nodes[nId] = Node(nId, lat, long)
                if(child.tag == 'way'):
                    wId = int(child.attrib['id'])
                    self.ways[wId] = Way(wId)
                    for way_child in child:
                        if(way_child.tag == 'tag'):
                            wAttrKey = way_child.attrib['k']
                            wAttrValue = way_child.attrib['v']
                            self.ways[wId].attrib[wAttrKey] = wAttrValue
                        if(way_child.tag == 'nd'):
                            node_id = int(way_child.attrib['ref'])
                            self.ways[wId].nodeList.append(self.nodes[node_id])
                    self.ways[wId].addRoadToChildNodes()

```

The map class has two main attributes:

nodes	Nodes	The collection of nodes held within the map.
ways	Ways	The collection of ways held within the map.

and two functions:

__init__	Initialising function for assigning constructor variables to object.
loadMapFrom File	<p>Reads the map OSM file (XML) using Python xml library ElementTree as ET into the abstract OOP format used in the rest of the testing program.</p> <p>Uses a python 'with' clause in conjunction with 'open' to open and close the file when processing is finished in a tidy manner.</p> <p>Using a 'for' loop to incrementally evaluate each child within the xml file a check is performed to determine whether the tag is a node or way using two if loops.</p>

If the tag is a node, latitude and longitude are extracted as floats and the node ID as an integer. These are then passed to the local nodes collection as a new node with attributes assigned.

If the tag is a way, the way id is extracted and passed into the local ways collection with the corresponding ID. For each child a second loop is used to incrementally evaluate each tag to determine if it is either a 'tag' (attribute) or a node. If a tag is found a new key-value pair is added to the attributes list within the way with the corresponding id. If a node is present it is appended to the way nodeList.

After this processing is complete, each node belonging to a way will have a check performed using addRoadToChildNodes to replicate the road across several nodes. This allows for us to see junctions (nodes with more than one road connected).

Functions for OOP Testing

After drafting my object orientated approach, I wrote some code to test whether the features were being loaded correctly. To do this, I chose the Oxford University Parks as my testing landmark as it was in focus of my testing map downloaded from OpenStreetMap.

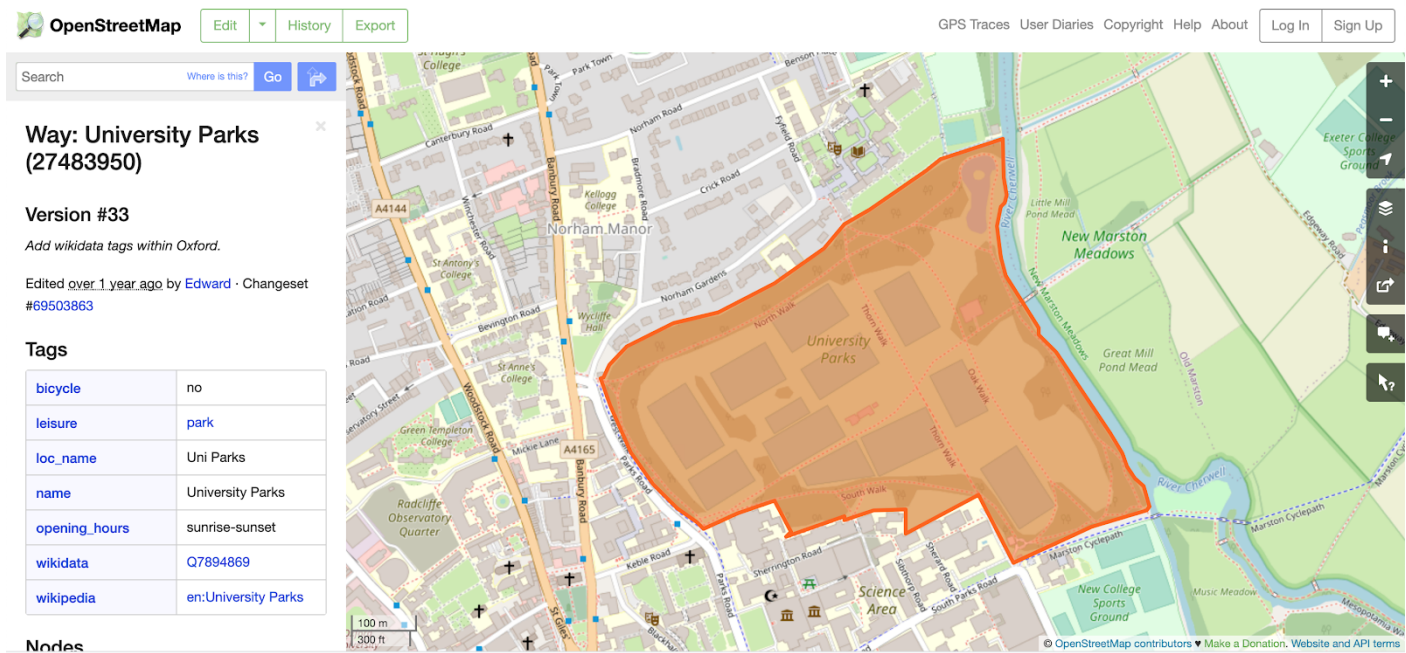
I used the query feature on the OpenStreetMap website to gather IDs of prominent landmarks around the park and used the drawWay function to display them using matplotlib alongside a legend labelling each one with the name specified in the OSM xml file.

```

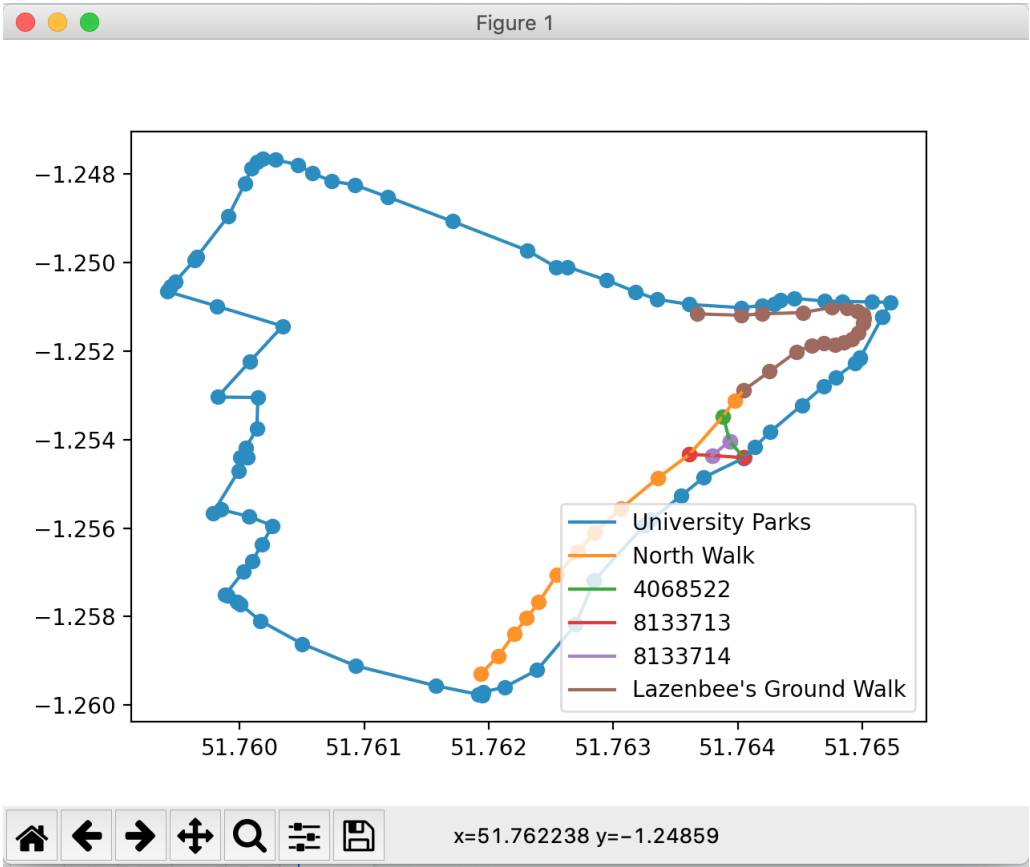
mainMap = Map( )
mainMap.ways[27483950].drawWay( ) #University Parks
mainMap.ways[3487836].drawWay( ) #North Walk
mainMap.ways[4068522].drawWay( ) #Uni Parks entrance walkway 1
mainMap.ways[8133713].drawWay( ) #Uni Parks entrance walkway 2
mainMap.ways[8133714].drawWay( ) #Uni parks entrance walkway join
mainMap.ways[4068495].drawWay( ) #Lazenbee's Ground Walk
plt.legend( )
plt.show( )

```

Testing Code



University Parks Area boundary (27483950)



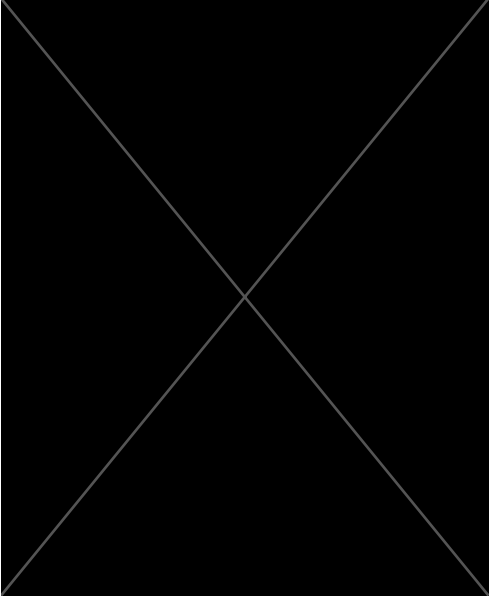
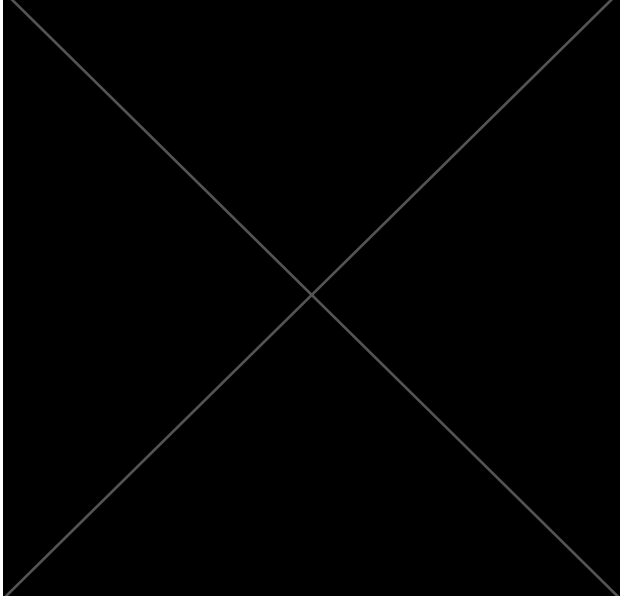
Landmarks shown in matplotlib as point polygons and lines

I also tested the attribute gathering feature by printing each way's attributes. For example, below you can see the University Parks boundary has the opening hours during daytime hours. You can also see a wikidata reference that points to the wikipedia service allowing users to find out more about a feature. Other data such as the fact you can't ride a bicycle in the park is also shown. These are all very useful.

```
{
  "bicycle": "no",
  "leisure": "park",
  "loc_name": "Uni Parks",
  "name": "University Parks",
  "opening_hours": "sunrise-sunset",
  "wikidata": "Q7894869",
  "wikipedia": "en:University Parks"
},
{
  "bicycle": "no",
  "foot": "permissive",
  "highway": "footway",
  "lit": "no",
  "name": "North Walk",
  "note": "Access not 24/7, see park opening times",
  "surface": "fine_gravel"
},
{
  "bicycle": "no",
  "foot": "permissive",
  "highway": "footway",
  "lit": "no",
  "surface": "fine_gravel"
},
{
  "bicycle": "no",
  "foot": "permissive",
  "highway": "footway",
  "lit": "no",
  "surface": "fine_gravel"
},
{
  "bicycle": "no",
  "foot": "permissive",
  "highway": "footway",
  "lit": "no",
  "name": "Lazenbee's Ground Walk",
  "note": "Access not 24/7. See park opening times",
  "surface": "fine_gravel"
}
```

Route Finding using OR Tools in combination with OSRM

Now that I have managed to represent our map data in an abstract way that I am able to interact with using Python's OOP using classes, I can begin to work on the actual routing algorithm that I will use within my application.

	
A	B
<p>What is the shortest route I can take to get from point A to point B using a predefined set of drivable roads?</p>	<p>What is the most optimal route of stops in order to minimize mileage and therefore most likely time taken to get to a stop?</p> <p>NB: Picture is an abstracted graph not showing the actual roads taken.</p>

Both are extremely complex tasks that have been worked on by computer scientists for decades. Most commercial solvers only provide a certain level of optimisation that is just about sure rather than having a 100% certainty.

I initially planned to build a graph that represents the nodes in the surrounding area and perform the necessary calculations however I soon realised that I was out of my depth. The commercial solvers have thousands of people working on optimisations and often use much lower level languages for efficiency such as C++. In initial development of my draft solution I also ran into initial barriers with memory and processing as this type of problem solving often requires a huge amount of data to be stored in memory for processing.

I decided to use two helper libraries to help me process my data in a reasonable timeframe for my coursework.

Problem A: OSRM

In order to approach problem A, I decided to use an well known open source library piece of software called OSRM. The great part about the project being open source is the fact that I can override any API limits by installing a local route finder myself.

However even though OSRM has a huge community of programmers behind it and it has become highly optimised in order to determine routes for at least the UK, the system requirements are high.

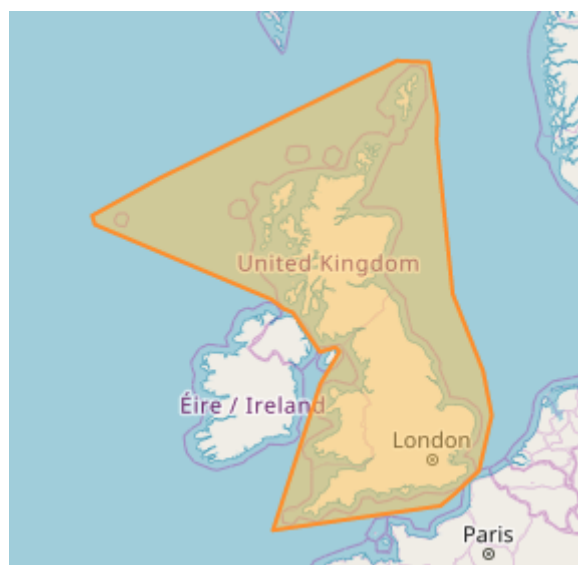
```

PROD-OSRM
drwxr-xr-x  5 fnicholson fnicholson      4096 Oct 14 08:45 ..
-rw-rw-r--  1 fnicholson fnicholson 2024184454 Oct 12 01:39 great-britain-latest.osm.bz2
-rw-rw-r--  1 fnicholson fnicholson 1018114560 Oct 15 09:30 great-britain-latest.osrm
-rw-rw-r--  1 fnicholson fnicholson  94842368 Oct 15 09:32 great-britain-latest.osrm.cnbg
-rw-rw-r--  1 fnicholson fnicholson  94842368 Oct 15 09:32 great-britain-latest.osrm.cnbg_to_ebg
-rw-rw-r--  1 fnicholson fnicholson   69632 Oct 24 09:28 great-britain-latest.osrm.datasource_names
-rw-rw-r--  1 fnicholson fnicholson  489528832 Oct 15 09:41 great-britain-latest.osrm.ebg
-rw-rw-r--  1 fnicholson fnicholson  139107328 Oct 15 09:41 great-britain-latest.osrm.ebg_nodes
-rw-rw-r--  1 fnicholson fnicholson  142790144 Oct 15 09:40 great-britain-latest.osrm.edges
-rw-rw-r--  1 fnicholson fnicholson  135828992 Oct 15 09:40 great-britain-latest.osrm.enw
-rwxr-xr-x  1 fnicholson fnicholson  417871540 Oct 15 09:41 great-britain-latest.osrm.fileIndex
-rw-rw-r--  1 fnicholson fnicholson  480222720 Oct 15 09:40 great-britain-latest.osrm.geometry
-rw-rw-r--  1 fnicholson fnicholson 1000220160 Oct 24 11:02 great-britain-latest.osrm.hsgr
-rw-rw-r--  1 fnicholson fnicholson   80700416 Oct 15 09:40 great-britain-latest.osrm.icd
-rw-rw-r--  1 fnicholson fnicholson    5120 Oct 15 09:35 great-britain-latest.osrm.maneuver_overrides
-rw-rw-r--  1 fnicholson fnicholson   6901760 Oct 15 09:30 great-britain-latest.osrm.names
-rw-rw-r--  1 fnicholson fnicholson  244077568 Oct 15 09:32 great-britain-latest.osrm.nbg_nodes
-rw-rw-r--  1 fnicholson fnicholson    6144 Oct 15 09:30 great-britain-latest.osrm.properties
-rw-rw-r--  1 fnicholson fnicholson  1670656 Oct 15 09:41 great-britain-latest.osrm.ramIndex
-rw-rw-r--  1 fnicholson fnicholson   4096 Oct 15 09:35 great-britain-latest.osrm.restrictions
-rw-rw-r--  1 fnicholson fnicholson   3584 Oct 15 08:48 great-britain-latest.osrm.timestamp
-rw-rw-r--  1 fnicholson fnicholson   5632 Oct 15 09:40 great-britain-latest.osrm.tld
-rw-rw-r--  1 fnicholson fnicholson   9216 Oct 15 09:40 great-britain-latest.osrm.tls
-rw-rw-r--  1 fnicholson fnicholson  40797184 Oct 15 09:35 great-britain-latest.osrm.turn_duration_penalties
-rw-rw-r--  1 fnicholson fnicholson 244763136 Oct 15 09:35 great-britain-latest.osrm.turn_penalties_index
-rw-rw-r--  1 fnicholson fnicholson  40797184 Oct 15 09:35 great-britain-latest.osrm.turn_weight_penalties
drwxrwxrwx  1 fnicholson fnicholson    25 Oct 14 08:46 lib -> osrm-backend/profiles/lib
drwxrwxrwx 21 fnicholson fnicholson   4096 Oct 14 08:10 osrm-backend
drwxrwxrwx  1 fnicholson fnicholson    29 Oct 14 08:45 profile.lua -> osrm-backend/profiles/car.lua
-rw-rw-r--  1 fnicholson fnicholson    28 Oct 14 08:45 .stxxl
fnicholson@osrm:~/osrm$ _

```

OSRM Folder with routing compiled data sources with file sizes in bytes

The final file size of the compiled data source for the UK (`great-britain-latest.osrm`) is about 1GB. The original uncompressed OSM XML for Great Britain is 2GB. The uncompressed OSM XML for the entirety of the planet in OSM is 98GB in size.



Compiled data source routing area

Although over time with the development of storage and technology 2GB may not seem a great deal. For purely XML based data with thousands of nodes all needing to be processed it requires very efficient finding and memory management.

In order to run a local install of OSRM for the entire planet the OSRM say for the car profile (in order to calculate routes for road vehicles) I would roughly require:

- around 175 GB of RAM for pre-processing
- around 280 GB of STXXL disk space

STXXL stands for Standard Template Library for Extra Large Data Sets. It is an efficient storage mechanism used by the OSRM routing engine.

Unfortunately I don't have a machine quite that powerful therefore I decided to utilise the subregion of the planet file provided for the entirety of the great britain. It is unlikely that I will be requiring anything else for educational purposes.

Nonetheless, I still need a large amount of processing power in order to run OSRM locally. Therefore I utilised my 'homelab' setup to set up a 'virtual machine' to run the processing.



Hardware	
Manufacturer	Dell Inc.
Model	PowerEdge R710
CPU	8 CPUs x Intel(R) Xeon(R) CPU X5550 @ 2.67GHz
Memory	31.98 GB

Actual machine operating specifications - 10 year old server designed for high power processing.

Hardware Configuration	
CPU	1 vCPUs
Memory	12 GB
Hard disk 1	16 GB
USB controller	USB 2.0
Network adapter 1	LAN (Connected)
Video card	16 MB
CD/DVD drive 1	ISO [WD Red Disk #0] ubuntu
Others	Additional Hardware

Virtual Machine Specifications scaled down from recommended 175GB memory to 12GB to suit the smaller data source.

Once setup I ran a few test API calls using an API debugger with some sample test coordinates taken from the sample data seen before in the document. Everything functioned as expected according to the provided documentation.

I then needed to write a simple python function / mini module that I could use in other parts of the application in order to find the shortest route using OSRM.



```
import requests
import map

def getShortestRoute(n1, n2):
    osrm_request = f'http://osrm:5000/route/v1/driving/{n1[1]},{n1[0]};{n2[1]},{n2[0]}'
    r = requests.get(osrm_request)
    return(r.json())
```

getShortestRoute	<p>Two input nodes longitude latitude pairs are given.</p> <p>The two nodes are then passed to the OSRM server with a driving profile (only routes that are drivable by a land vehicle are permitted).</p> <p>The function then returns the response as JSON for the calling function to process the data provided by OSRM.</p>
-------------------------	---

An example of the a JSON response from the local OSRM server is shown below:

URL	http://osrm:5000/route/v1/driving/-1.2592877447605135,51.75516718114736;-1.2546528875827792,51.7677575155314
------------	--

Response

```

{
  "code": "Ok",
  "routes": [
    {
      "geometry": "cm{zHp}tF_NxCaHQ{L|@kUrBsL`Cm@qFaG}Z",
      "legs": [
        {
          "steps": [
            ],
          "summary": "",
          "weight": 220.3,
          "duration": 220.3,
          "distance": 1763.4
        }
      ],
      "weight_name": "routability",
      "weight": 220.3,
      "duration": 220.3,
      "distance": 1763.4
    }
  ],
  "waypoints": [
    {
      "hint": "l54AgHxkToAFAAAAADwAAAAAAAAAAAAAAL-  
Z7QFGJI0EAAAAAAAAAAUAAAAAAAAAAAAAAAAAADgCgAA5cjs_9K4FQPoy0z_n7gVAwAAZxC27p47",
      "distance": 5.678223,
      "name": "",
      "location": [
        -1.259291,
        51.755218
      ]
    },
    {
      "hint": "M-  
AAgELgAIAAAAAAHwAAAJwAAAAqAAAAAAAAABgksEFLQt5CpfrvQQAAAAAfAAAAAnAAAACoAADgCgAAadbs__zsFQMD2-  
z_zukVAwQAnwG27p47",
      "distance": 121.713023,
      "name": "Bardwell Road",
      "location": [
        -1.255831,
        51.768572
      ]
    }
  ]
}

```

Unusually OSRM expects an input in the form of longitude, latitude rather than latitude, longitude. Each pair is separated by a semi colon and the longitude, latitude is separated by a comma. The response indicates that the request was valid using the 'code' key. The main response data we are interested in is the 'routes' object that contains the metrics we require for our processing using OR Tools. As you can see from the response given above, the duration is estimated to be 220.3 float seconds and the distance as a float is 1763.4 meters.

Running on a local server also offers performance benefits when querying OSRM. Aside from the request rate limiting that OSRM put in place on the public service, the connection speed is greatly increased. I am running tests on my laptop connected via wired ethernet directly to the same network that the server is operating on and therefore there is near gigabit speeds and very low latency between the two systems. Additionally when deployed the application will most likely be run in the same data center leading to even greater performance.

Time taken building a Distance Matrix with 5 nodes using Public OSRM service	Time taken building a Distance Matrix with 5 nodes using onsite server
21.24s	1.56s

These results show that running a local server gives performance up to 12x faster than a public counterpart. This could perhaps be due to the smaller dataset needing to be processed within the OSRM engine but I believe this is unlikely.

After writing my small helper OSRM module I imported it into a larger section of code that I used to solve problem B.

The main library that I used to help solve this problem was Google's excellent open source project 'Google Optimization Tools'. The tools provided within the open source package are often highly sought after commercially and the fact that it is freely available for anyone to use in their projects is incredibly helpful.

The library has a very broad scope for being designed for solving combinatorial optimization problems. However for my coursework, I am only looking at the Vehicular Routing functions that the library provides.

However, like many of these tools in order to be highly efficient there are very specific input data criteria that need to be adhered to and I will need to adjust the data I have already gathered in order to allow OR tools to help me solve the vehicular routing problems I raise. It should also be noted that the library does not aim to offer a perfect solution, just a route which is to a high degree of professional certainty one of the most optimal available.

This code is adapted in the final program but the following version was used to build the testing figures used later on in this section hence some of the variable names being rough and function names not being as self documenting as they should be.

```
import map
import matplotlib.pyplot as plt
import osrm

mainMap = map.Map()
#print(mainMap.graph.graphNodes[7836189542].adjacentNodes)
for way in mainMap.ways:
    if(mainMap.ways[way].isDrivable() and way != 3063659):
        mainMap.ways[way].drawWay()

n1 = {0: (51.769153, -1.256153), 1: (51.770139, -1.258175), 2: (51.766748, -1.258154), 3: (51.763016, -1.261844), 4: (51.771787, -1.262616)}
def optimiseRouteBetweenNodes(nodeCoordList):
    for nodeCoord in nodeCoordList:
        plt.scatter(nodeCoordList[nodeCoord][0], nodeCoordList[nodeCoord][1], color='#27D507', s=100)
        plt.annotate(nodeCoord, (nodeCoordList[nodeCoord][0], nodeCoordList[nodeCoord][1]),
            color='#FF69B4', size=20)
def doMatrix():
    distances = []
    for i in n1:
        distances.append([])
        for j in n1:
            distances[i].append(osrm.getShortestRoute(n1[i], n1[j])['routes'][0]['distance'])
            plt.plot([n1[i][0], n1[j][0]], [n1[i][1], n1[j][1]], color="red")
            xMidpoint = (n1[i][0] + n1[j][0])/2
            yMidpoint = (n1[i][1] + n1[j][1])/2
            # print(xMidpoint)
            shortestDistance = osrm.getShortestRoute(n1[i], n1[j])['routes'][0]['distance']
            plt.annotate(shortestDistance, (xMidpoint, yMidpoint), color='#00FFFF', size=15)
            # print(i, j, shortestDistance)
    return distances

if __name__ == "__main__":
    doMatrix()
    optimiseRouteBetweenNodes(n1)
    plt.show()
```

doMatrix

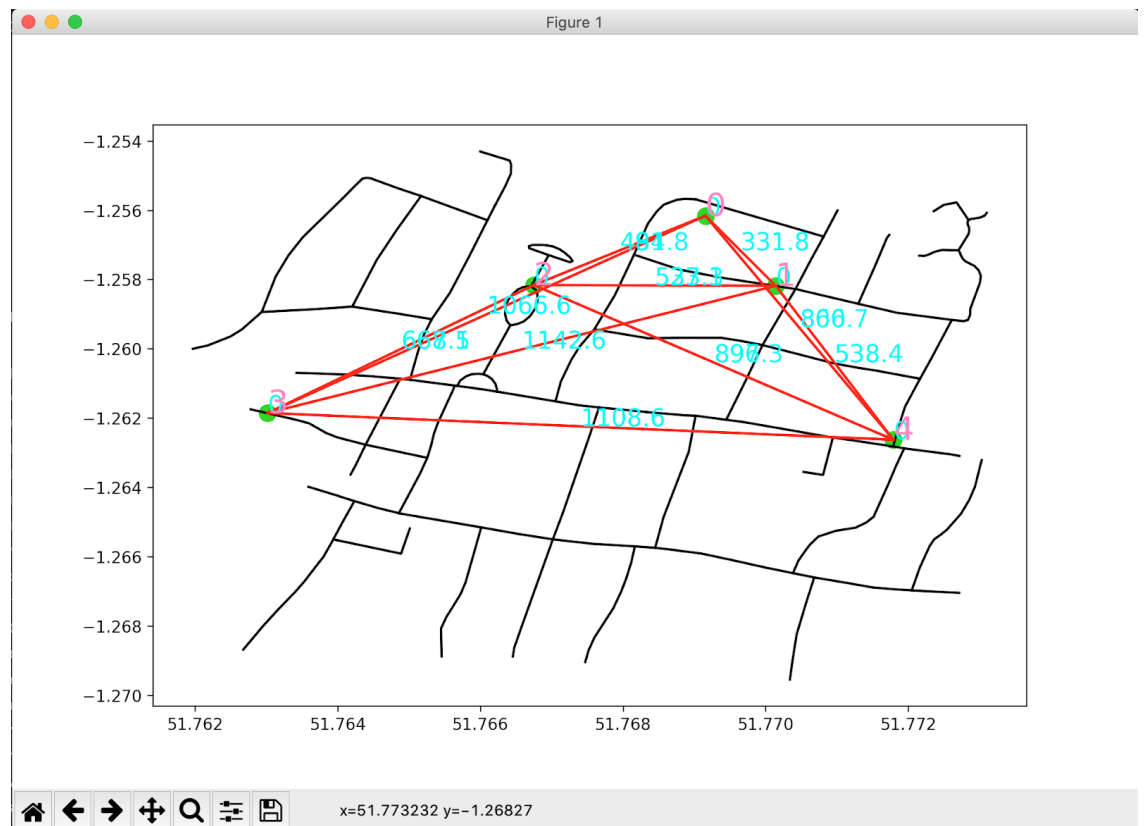
An advanced matrix operation forming a multi dimensional array using a nested for loop with variables i,j . I initially attempted to use the excellent Python itertools library combinations function however I needed to create a new array for each node in the y axis therefore I used the traditional nested approach instead.

For each node, a query is made to OSRM to calculate the distance between every other node provided building a distance matrix. This allows OR tools to quickly retrieve the distances between two nodes and perform the optimisation using the CP-SAT solver that OR tools utilises for most of its processing. Each distance is calculated using the **getShortestRoute** function shown beforehand.

An example of a distance matrix produced by the function is shown below:

	0	1	2	3	4
0	0	331.8	484.8	1067	860.7
1	331.8	0	527.1	1143	538.4
2	491	533.3	0	667.1	896
3	1067	1143	669	0	1109
4	806	538.4	897.3	1108	0

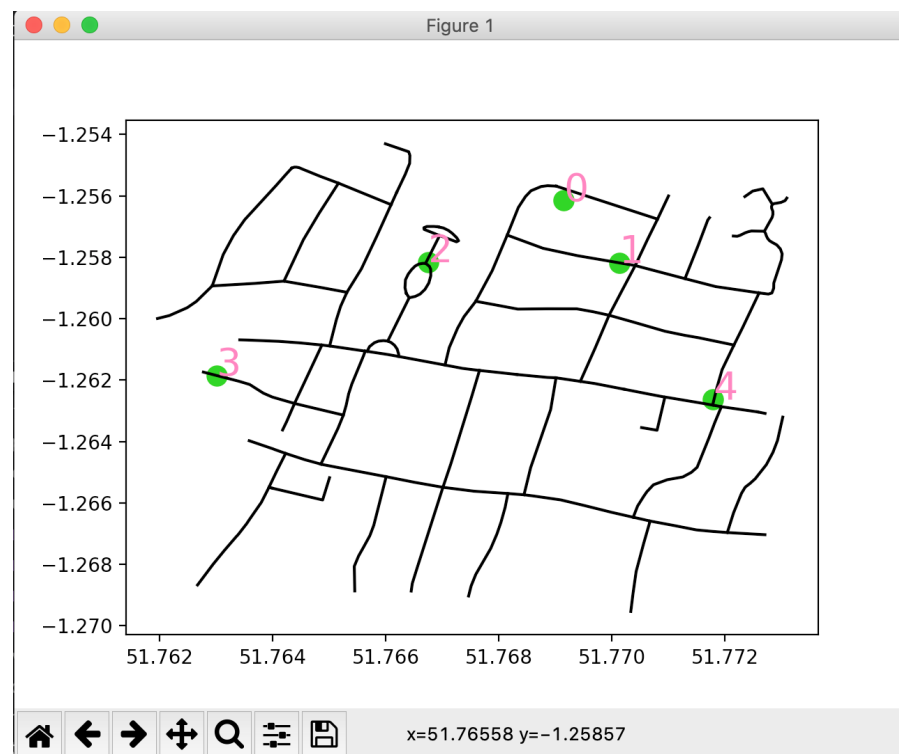
The function also graphically shows the processing and distances shown below allowing me to check that the functions I had programmed so far were operating as intended.



optimiseRoute BetweenNodes

This function prior to my approach using libraries contained my manual approach at trying to achieve what OSRM provided much more effectively hence it's deceptive name.

Now the function simply plots each pick up node as a green point on a plot with its number as an annotation allowing me to debug.



```

import map_testing
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

distances = map_testing.doMatrix()
print(distances)

def distance_callback(i, j):
    iR = manager.IndexToNode(i)
    jR = manager.IndexToNode(j)
    return distances[iR][jR]

manager = pywrapcp.RoutingIndexManager(len(distances), 1, 0)
routing = pywrapcp.RoutingModel(manager)

transit_callback_index = routing.RegisterTransitCallback(distance_callback)
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

solution = routing.SolveWithParameters(search_parameters)

def print_solution(manager, routing, solution):
    print('Objective: {} m'.format(solution.ObjectiveValue()))
    index = routing.Start(0)
    plan_output = 'Route for vehicle:\n'
    route_distance = 0
    while not routing.IsEnd(index):
        plan_output += f' {manager.IndexToNode(index)} -> '
        previous_index = index
        index = solution.Value(routing.NextVar(index))
        route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
    plan_output += f' {manager.IndexToNode(index)}\n'
    print(plan_output)

def main():
    if solution:
        print_solution(manager, routing, solution)

if __name__ == '__main__':
    main()

```

Following the excellent documentation provided by OR Tools

(<https://developers.google.com/optimization/routing/vrp>) I adapted the example given to see whether my processing so far had worked successfully. **NB:** This was simply for testing purposes and the adapted code is shown later on in the project as this does not achieve immediately the desired objective.

<p>distance_callback</p>	<p>This is a google OR tools specific function that needs to exist in order to be passed to the transit callback index that is in turn passed to the ArcCostEvaluator feature of the routing engine (used to calculate the cost of a path of nodes).</p> <p>The manager IndexToNode function is simply returning the index in the routing model of that particular node that is passed through the callback, this is then passed to the distances matrix in order to retrieve the correct calculated distance from OSRM.</p>
	<p>We then set up the OR Tools parameters in order to configure it for a test use case.</p> <p>The following features are defined:</p> <ul style="list-style-type: none"> ● manager - A routing index manager with the number of nodes, number of vehicles and starting node. As the routing solver uses variable indices internally making it hard for me to manage directly as a node can link to many other variables. Therefore OR Tools provides a class to make the management of indexes much easier. ● routing - An abstract routing model utilised by OR Tools taking into account the routing index manager. ● transit_callback_index - Registers the transit callback taking the distance callback as the function to call when performing the optimization which is set as the arc cost evaluator. ● search_parameters - Using the quick solution as provided by default by OR Tools. This won't achieve in all cases the most optimal outcome but will achieve a reasonably optimal outcome. ● solution - Calculates the solution using the routing model and the first solution heuristic defined in the search_parameters.
<p>print_solution</p>	<p>Prints the solution provided by OR Tools. Also displays the result graphically.</p> <p>Using a 'while' loop in conjunction with the routing.IsEnd function that continues until the optimal route recall is finished. The starting index is determined beforehand using the routing.Start function. On each loop, the index is retrieved from the manager and appended to a string output pictured in the terminal below. The next index is then found by passing through to the route engine using the NextVar function that returns the value for the next node. The route distance is also incremented by using the GetArcCostForVehicle function that utilises the distance callback to increase the distance.</p> <p>The final result is printed and shown graphically below.</p> <div data-bbox="454 1534 1380 2049"> <pre> [[0, 331.8, 484.8, 1066.6, 860.7], [331.8, 0, 527.1, 1142.6, 538.4], [491, 533.3, 0, 667.1, 896], [1066.6, 1142.6, 668.5, 0, 1108.6], [806, 538.4, 897.3, 1108.6, 0]] Objective: 3128 m Route for vehicles: 0 -> 2 -> 3 -> 4 -> 1 -> 0 [0, 2, 3, 4, 1, 0] </pre> </div>

One critical feature that is missing from this example is a start and end node used to determine where the route will stop and start. Currently the OR Tools example code is configured to do a delivery driver round-robin style path returning to a central depot which for the objective of a school bus service is not what we want to achieve.

We will return to the implementation of the outcome of this testing when reviewing the flask web application that was made for the end user.

Flask Introduction



Flask is a web framework written in Python. It does not rely on any other libraries and therefore technically has the title of a microframework. This also means that there are less tools for features that other web frameworks might offer requiring the user to do much of this themselves. Another popular framework that I considered was Django however due to the unique application and the abstraction that Django places using it's model system for SQL whilst not being completely transparent to the user what queries are being run, I decided for educational purposes to go with the more manual framework.

There are four main concepts in Flask that I utilised in the development of the web application:

Name	Modules	Blueprints	Templates	Static Files
Description	Standard python files that can be used by the flask system in the standard python module manner.	A sub-module but a specific flask term for an extension to a flask web application.	A form of static file that is primarily HTML, the syntax understood by web browsers, that is returned to a users webpage. Support for liquid templating allowing for dynamic variables.	Static files that are used by the application in various ways, primarily designed for the front end files. Dynamically generated files such as profile pictures also fall under this category.
Examples	<ul style="list-style-type: none">• Core configuration and initialisation files• Extension of features using custom libraries	<ul style="list-style-type: none">• Defining new urls for a certain sub section of a website.• Implement ation of application specific	<ul style="list-style-type: none">• Front end website pages that are presented to the user.• Dynamic CMS using liquid templating	<ul style="list-style-type: none">• Photos• Fonts• Stylesheets• JavaScript• Icons

functions.








variables.

Implementation of Flask

In order to keep my code organised, I utilised Flask's blueprinting system. When using a modular system it is important to have a clear understanding of what classes a subsection. Since I had completed my documented design beforehand, I knew what primary application modules I expected to use and therefore I made a blueprint (or module) for each subsection of the system.

Front-end naming	Filename	Description
Flask initialization file	<code>__init__.py</code>	Each flask application needs to have an initialisation file so flask understands the context of the application. This is that file and contains all blueprint links and other core application specific configuration.
Authentication / Authorisation Manager	<code>auth.py</code>	Most applications on the internet have a form of Authentication and Authorisation. The two are not the same; authentication is surrounding the concept about identifying an individual is who they say they are. Authorisation is the process of checking whether a user should be allowed to perform a specific action which is passed onto the security section of the application.
Database Manager	<code>db.py</code>	Other web frameworks often do database management behind the scenes however flask does not offer such an easy option. Therefore, adapted from official flask documentation on the topic, I wrote my own manager to connect to my Microsoft SQL Server back-end. There are some excellent libraries such as 'sqlalchemy' that offer SQL modelling similar to that seen in Django however this again abstracts the core SQL commands away from the programmer and for educational purposes I have decided to run all queries in raw SQL.

Image Asset Manager	imgdispatch.py	Many images used for different purposes will need to be stored within the application ranging from user profile pictures to vehicle photographs. The image asset manager will be able to check if a user should have access to a certain image and return it to the required template as required.
<p>Application management functions for users, orgs...</p> <ul style="list-style-type: none"> 🔧 Management 🏢 Organisations 👤 Users 👥 Groups 🚚 Fleet Management 📊 Reports ⚙️ Configuration 	management.py	This module will be the primary module used by a system administrator. It contains CRUD operations for much of the core application data. Additional complex business model features that are invisible to the user are also present such as family grouping with relationships.
Navigation Menu	nav.py	The navigation menu in the application has been designed to be dynamically generated on the fly rather than being a static form as different users will have access to different applications and it makes sense to hide applications that are not relevant to a user. Additionally I can reference icons and permissions in a concise dictionary format.
OSRM module as previously seen in testing adapted for flask	osrm.py	This module is similar to that seen before in testing beforehand. It is the barrier between the OSRM back-end run on a server and my flask application.
<p>Flask adapted implementation of routing seen beforehand in testing</p> <ul style="list-style-type: none"> 🗺️ Routing 📍 Route Mapping 📅 Schedule 	routing.py	This module controls all the templates for the routing management system. It also contains all the adapted functions related to the routing seen beforehand in testing and new client-server json functions

		allowing for easy manipulation within javascript as this module is primarily utilised on the front-end by the user dynamically and a static webpage format would not be as effective.
<p>Security specific functions such as role-based access control (RBAC)</p> <ul style="list-style-type: none">  Security <ul style="list-style-type: none">  Dashboard  Access Levels  Audit Log 	<code>security.py</code>	An extension to the authentication / authorisation module. With role-based access control and a permissions system being used to identify whether a user should be allowed access to a certain feature. Additional security features such as audit logs are also implemented for use across the application.
<p>Personal user functions</p> <ul style="list-style-type: none">  Profile <ul style="list-style-type: none">  My Profile  Settings 	<code>user.py</code>	Functions that only the user themselves would need. Such as being able to view your own profile, update password and other personal account management functions.

Freddie Nicholson Computer Science NEA
Flask initialization file

```
import os

from flask import Flask, render_template, session, g
from flask.views import View

class Table:
    def __init__(self, title, columns, endpoint, buttons, filters, searchPresent, paginatorPresent,
script):
        self.title = title
        self.endpoint = endpoint
        self.columns = columns
        self.buttons = buttons
        self.filters = filters
        self.searchPresent = searchPresent
        self.paginatorPresent = paginatorPresent
        self.script = script

class TableButton:
    def __init__(self, text, action, bType):
        self.text = text
        self.action = action
        self.bType = bType

class TableFilter:
    def __init__(self, name, options):
        self.name = name
        self.options = options

def create_app(test_config=None):
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_pyfile('config.py')
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    from . import nav
    app.register_blueprint(nav.bp)

    from . import imgdispatch
    app.register_blueprint(imgdispatch.bp, url_prefix='/img')

    from . import management
    app.register_blueprint(management.bp, url_prefix='/management')

    from . import auth
    app.register_blueprint(auth.bp, url_prefix='/auth')

    from . import user
    app.register_blueprint(user.bp, url_prefix='/user')

    from . import security
    app.register_blueprint(security.bp, url_prefix='/security')

    from . import routing
    app.register_blueprint(routing.bp, url_prefix='/routing')

    from . import db
    with app.app_context():
        db.init_db()

    @app.route('/profile')
    @auth.login_required
    def profile():
        return render_template('profile.html', nav=nav.nav)

    UPLOAD_FOLDER = os.path.join(app.instance_path, 'uploads')
    app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

    @app.route('/')
    @auth.login_required
    def index():
        return render_template('home/home.html', nav=nav.nav, uid=session['user_id'])

    @app.template_filter('num_plate')
    def num_plate(num_plate):
        """Formats a Number Plate"""
        return f'{num_plate[:4]} {num_plate[4:]}'

    return app
```

Freddie Nicholson Computer Science NEA

The initialization file contains the `create_app` function that is run by flask when performing the 'flask run' command. It returns the app object that flask can interpret and use to satisfy a request. In my development environment I had to specify to flask the environment variables in my virtual environment so that it could recognise where the initialisation file / app folder is present.

```
fnicholson@Freddies-MacBook-Pro Freddie-Nicholson-CS-NEA-2020-2021 % export FLASK_APP="BusWebApp"
fnicholson@Freddies-MacBook-Pro Freddie-Nicholson-CS-NEA-2020-2021 % export FLASK_ENV="development"
fnicholson@Freddies-MacBook-Pro Freddie-Nicholson-CS-NEA-2020-2021 % flask run
* Serving Flask app "BusWebApp" (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 204-331-066
```

'BusWebApp' is the folder present and is defined as the `FLASK_APP`. Flask looks inside the BusWebApp folder for an `__init.py__` file and retrieves the app context. `FLASK_ENV` being set to development simply means flask offers some helpful debug tools such as a live console on each webpage when an error occurs. This is only used for development purposes as otherwise the system would be very insecure!

During the app context creation, Flask loads the app configuration from 'config.py'.



```
SECRET_KEY = 'dev'
FLASK_APP = 'BusWebApp'
FLASK_ENV = 'DEVELOPMENT'

SQL_SERVER_URL = 'DRIVER={ODBC Driver 17 for SQL
Server};SERVER=tcp:mssql;DATABASE=BusWebApp;UID=BusWebApp;PWD=<REDACTED>'
DVLA_KEY = '<REDACTED>'
```

This file contains potentially sensitive information that is 'gitignored' in my git configuration file so I do not accidentally release API keys or database connection details to the public domain. Throughout the entire design process of the system I have ensured that all sensitive personal information is held on the SQL server only and none is stored within the application code itself. This follows the UK's Data Protection Act 2018 and GDPR requirement of 'storage limitation'. Storing data in a database is a common practice that most companies follow as it helps ensure they are complying with the 'accountability' principle as all data is held in one place and most likely audited.

The `SECRET_KEY` is an internal variable used by flask for encryption functions such as that used for the hashing of passwords in the database. The `SQL_SERVER_URL` is the pyodbc library url allowing for easy access to the database in one simple line. The server in the instance is hosted on the same machine that the OSRM instance is running on with a local DNS hostname of mssql. I created a user on the SQL server specific to web application queries with access only to relevant databases, as is best practice, named BusWebApp. I also defined the government DVLA API key as this is another variable that should be held cautiously.

I then import all the various blueprints from other modules. I also initialize the DB manager in the app context to keep the connection for each session. During development, the page redirects to a very basic user profile showing the User ID that is logged in. Most function / endpoints have a decorator attached called `auth.login_required`. This is to indicate that the endpoint should only be accessed by registered users, I will expand on this within the authentication section. With development of the application these decorators were modified slightly for role-based access control.

Flask has the ability to add 'filters' to the application. These are used within liquid templating.

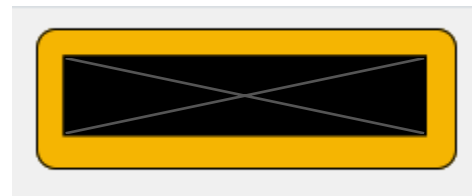
The filter shown above as num_plate takes in a number plate as recorded in the database as a single string and adds the correct spacing to show to the end user. I achieve this by using Python's string slicing mechanisms to concatenate the first four characters of the string, a space and the final 3 characters of the string. An example of the templating language that would be used to achieve this is shown below.

```
<td><a class="numberplate" href="view/{{vehicle.id}}/">{{vehicle.registration_number|num_plate}}</a>
</td>
```

Input

registration_number
GL10ECC

Output



```
from flask import Blueprint, render_template, request, current_app, redirect, flash, session, g
from . import nav
from . import db
from werkzeug.security import check_password_hash, generate_password_hash
import functools

bp = Blueprint('auth', __name__)

@bp.route('/login/', methods=['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template('/auth/login.html', nav=nav.nav)
    if request.method == 'POST':
        err = None
        email = request.form['email']
        user = db.executeQueryReturnJSON(f'SELECT * FROM users WHERE email = \'{email}\'' FOR JSON
AUTO;')
        if user == {}:
            err = 'Incorrect username.'
        else:
            user = user[0]
            if not check_password_hash(user['password'], request.form['password']):
                err = 'Incorrect password.'

        if err is None:
            session.clear()
            session['user_id'] = user['id']
            return redirect('/')

        flash(err, 'danger')
        return redirect('/auth/login')

@bp.before_app_request
def load_logged_in_user():
    user_id = session.get('user_id')

    if user_id is None:
        g.user = None
    else:
        try:
            g.user = db.executeQueryReturnJSON(f'SELECT * FROM users WHERE id = \'{user_id}\'' FOR JSON
AUTO;')[0]
        except:
            print('err in load logged in user')

@bp.route('/logout')
def logout():
    session.clear()
    return redirect('/')



def login_required(view):
    @functools.wraps(view)
    def wrapped_view(**kwargs):
        if g.user is None:
            return redirect('/auth/login')

        return view(**kwargs)

    return wrapped_view
```

The auth.py contains the authentication logic for the application. There are four main functions / endpoints:

<p>login</p>	<p>Function to check the user is who they say they are. If a 'GET' request is sent by the browser then a login template is returned.</p> <div data-bbox="432 367 1482 537"> <p>Login</p> <p>User Email:</p> <input type="text"/> <p>User Password:</p> <input type="password"/> <p>Login</p> </div> <p>If a 'POST' response is received (the login form has been filled in and the user has submitted their form entries) the logic to check the user begins. An error variable, err, is set up to catch any errors present to the user. Using flask's request.form object I retrieve the email and pass into the SQL query directly. It should be noted that to prevent SQL injection this should be escaped. However, for the practicality of testing during development I have got the logic to work first. Using the database manager JSON response function I list all users with that name, if a user does not exist then an error is lodged as an incorrect username. If the user does exist, the function goes on to retrieve the first user (there should be no user with the same name). I then use Flask's werkzeug.security tool check_password_hash that uses the aforementioned config secret key to compare the two hashes in the database.</p> <div data-bbox="432 1066 1544 1164"> <p>password</p> <p>pbkdf2:sha256:150000\$Q4Gg74iV\$...</p> </div> <p>If they do not match, then an error is lodged as an incorrect password. In most web applications, they do not actually tell you whether the username or password was specifically correct as this could lead to exploitation instead a generic combination error is returned instead. E.g. you could work out whether john.doe@contoso.com has access to a sensitive internal application. However for debugging and educational purposes this is helpful.</p> <p>If no error is present in the processing of the function, then the current session is cleared and replaced with the new user id and the user is redirected to the home page. Otherwise the error is flashed on the main login screen with a redirect back to the same template above.</p>
<p>load_logged_in_user</p>	<p>This function loads all the user data into a session variable used by the flask back-end for this particular request. Flask does much of the work required for ensuring the variables are linked to the correct requests.</p> <p>I first get the user id from the flask session, if the user id is none then I set the flask g variable, an application context variable that survives for the lifetime of the request, for the user to none. Otherwise, I perform an SQL query using the db manager library to retrieve all users with that id. I then select the first entry as there will be only one record if there is a match. If there is any error decoding the JSON then either the user does not exist or there is another error with the session and the g.user is not updated and an error is logged.</p>

	<p>The following fields are returned into the g.user object:</p> <pre> id varchar(12) given_name varchar(200) last_name varchar(200) preferred_name varchar(200) title varchar(20) password varchar(200) email varchar(200) profile_image varchar(200) </pre> <p>This allows me to render user specific content within templates, for example the logged in user indicator within the navbar:</p> <div> <div>  Testing User <ul style="list-style-type: none"> My Profile Change Password Switch User Logout </div> <div> <h3>Profile</h3> <p>Please contact your organisation administrator to update your details.</p> <table> <tr> <td>Name</td> <td>Testing (Testing) User</td> </tr> <tr> <td>Email</td> <td>testing.user@appdevteam.com</td> </tr> </table>  </div> </div>	Name	Testing (Testing) User	Email	testing.user@appdevteam.com
Name	Testing (Testing) User				
Email	testing.user@appdevteam.com				
logout	<p>Logs the user out of the system by clearing the flask session and redirecting to the home page which will in turn redirect to the login page.</p>				
login_required	<p>A decorator view to be added to relevant endpoints that will check whether g.user is present and if not a redirect back to the login page will occur to prevent unauthenticated access. If the user does exist the wrapped view (original page) is returned.</p>				


```
import pyodbc
from flask import current_app, g
import json

def get_db():
    if 'db' not in g:
        g.db = pyodbc.connect(current_app.config['SQL_SERVER_URL'])
    return g.db

def close_db(e=None):
    db = g.pop('db', None)

    if db is not None:
        db.close()

def init_db():
    db = get_db()

def init_app():
    app.teardown_appcontext(close_db)

def executeQueryReturnJSON(query):
    db = get_db()
    cursor = db.cursor()
    cursor.execute(query)
    retJson = ''
    for row in cursor:
        retJson += str(row[0])
    if (retJson is not ''):
        return json.loads(retJson)
    else:
        return {}

def execute(query):
    db = get_db()
    cursor = db.cursor()
    cursor.execute(query)
    db.commit()
    return True
```

The db.py file contains all the database handling for the application. Requests are forwarded from other subsections of the application in order for a standard format to be used for all queries. It is adapted for MS SQL from the official documentation from Flask.

There are six main functions used within the database manager:

get_db	<p>Gets the flask global context variable for the db. If it does not exist then a new one is created for the request using pyodbc. ODBC is a standard driver and stands for Open Database Connectivity. It is popular in consumer software such as Microsoft Excel and Access for advanced projects. pyodbc is a library bringing the standard into python.</p> <p>The server URL is imported from the flask configuration file for security.</p>
close_db	<p>Pops the db from the flask global context variable and closes the connection using pyodbc.</p>
init_db	<p>Initialises the database using the get_db function. This function is run within the app context as specified in the app initialisation file.</p>
init_app	<p>Registers an appcontext teardown. This means the function will be called when the application context ends. Also when a request context is popped.</p> <p>The database connection will therefore be closed (using close_db) whenever a user ends their request.</p>
executeQueryReturnJSON(query)	<p>Fetches the database object and creates a cursor.</p> <p>Executes the SQL statement and loops over each row given in the response. Normally this function will be called with an SQL statement using MS SQL's JSON formatting such as 'FOR JSON AUTO;'. MS SQL by default has a defined limit on a column's data length (8000 characters) and therefore longer JSON responses will return multiple rows hence the need for an iterator to loop over and build a string response. The string response is then parsed by the python json library json.loads and returned to the function caller. If no data is present then an empty dictionary is returned.</p>
execute(query)	<p>Fetches the database object and creates a cursor.</p> <p>Simply executes the given command within the database and returns True to acknowledge that the action has been performed. The changes are committed. Useful when no response is required and the interaction is a simple SQL statement.</p>

```
from flask import send_from_directory, Blueprint, current_app
import os

bp = Blueprint('img', __name__)

@bp.route('/org/<filename>')
@auth.login_required
def orgLogo(filename):
    return send_from_directory(os.path.join(current_app.config['UPLOAD_FOLDER'], 'orglogos'), filename)






@bp.route('/vehicle/<filename>')
@auth.login_required
def vehicleImage(filename):
    return send_from_directory(os.path.join(current_app.config['UPLOAD_FOLDER'], 'vehicles'), filename)

@bp.route('/user/<filename>')
@auth.login_required
def userImage(filename):
    return send_from_directory(os.path.join(current_app.config['UPLOAD_FOLDER'],
'usercontent/profilepictures'), filename)
```

The imgdispatch.py image asset manager is the ‘middleman’ between the secure storage folders held within the storage provider and the end user. During development the images were stored on my local machine in a static folder however in production it would be most likely that a cloud storage platform would be utilised such as Google’s Cloud Storage or Amazon Web Services S3 bucket.

orgLogo	Sends the org logo file from the orglogos folder within the uploads directory in the storage provider.
vehicleImage	Sends the vehicle image file from the orglogos folder within the uploads directory in the storage provider.
userImage	Sends the user profile picture file from the orglogos folder within the uploads directory in the storage provider.

The application management module management.py contains all the logic relating to application management including the following:

Object	Description	Operations
Organisations 	Multiple organisations / schools may be using the same platform server. It makes sense to implement some logic in order to differentiate between schools and bus companies both for a data protection standpoint and to make management of the platform easier.	<ul style="list-style-type: none"> • Create new organisations • List all organisations • Add an organisation • Delete an organisation • View an organisation • Edit an organisation
Vehicles 	Buses will need to be recorded on the system in some capacity. Vehicles will need to be linked to routes in certain schedules. For ease of use for a bus company management features such as checking a vehicle's DVLA record is also implemented.	<ul style="list-style-type: none"> • List all vehicles • View a vehicle • Add a vehicle • Delete a vehicle • Edit a vehicle • Retrieve a vehicle's details from the DVLA
Users 	Users are at the core of the application, not necessarily application users but also indirect users of the service such as children travelling on a route.	<ul style="list-style-type: none"> • List all users • View a user • Edit a user • Add a user • Delete a user • Perform a quick search of all users
Families 	Families are not directly shown as a CMS item that can be created by the administrator however they are an important logical feature of the application for parents and therefore this object ties into the user profiles. Families have two relations, children and parents / guardians.	<ul style="list-style-type: none"> • Create a family • Disband a family • Add a family member • Delete a family member
Icons 	Throughout the application a standard icon set provided by Font Awesome is used. This is a simple module used for the listing of icons in the interface.	<ul style="list-style-type: none"> • Perform a quick search of all icons

Due to the size of the module I will evaluate each section individually. A full copy is available in the document appendix.

Utilities





```

def generateNewEightDigitId(tableName):
    testId = random.randint(111111111111, 999999999999)
    respJson = db.executeQueryReturnJSON(f'SELECT * FROM {tableName} WHERE id = \'{testId}\'' FOR JSON
    AUTO;')
    if(respJson):
        return generateNewEightDigitId(tableName)
    else:
        return testId
        
```

generateNewEightDigitId(tableName)	<p>A utility function used to generate a unique twelve digit ID, matching the client's MIS, iSAMS, ID system.</p> <p>It is a recursive function, a random 12 digit number is generated and it checks to see if any entries exist in the given table provided as a parameter.</p> <p>As there are a huge number of combinations of digits with a twelve digit number it is very unlikely that an id will match at a small scale and therefore the function will simply return the number. However if there is a match the system will recursively call the function again until it has found an ID which is not in use.</p>
---	--

Organisations



```

@bp.route('/organisations/')
@auth.login_required
def orgindex():
    orgsJSON = db.executeQueryReturnJSON('SELECT * FROM orgs FOR JSON AUTO;')
    return render_template('/management/organisation/organisations.html', nav=nav.nav, orgs=orgsJSON)

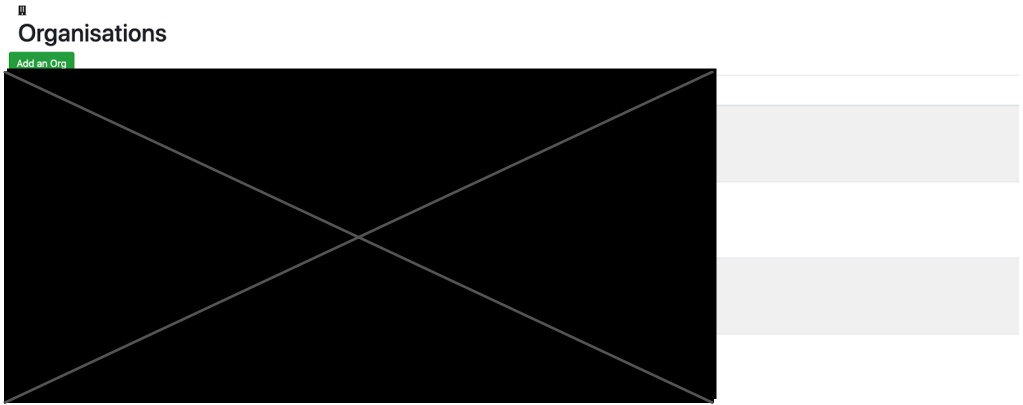
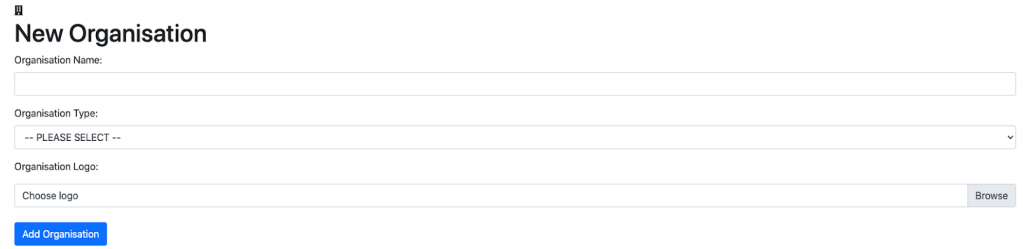
@bp.route('/organisations/add/', methods=['GET', 'POST'])
@auth.login_required
def addOrg():
    if(request.method == 'GET'):
        return render_template('/management/organisation/addorganisation.html', nav=nav.nav)
    if(request.method == 'POST'):
        newId = generateNewEightDigitId('orgs')
        if request.files['logo']:
            filename = secure_filename(str(newId)+'.'+request.files['logo'].filename.split('.')[-1])
            request.files['logo'].save(os.path.join(current_app.config['UPLOAD_FOLDER'], 'orglogos',
filename))
        else:
            filename = 'default.png'
        db.execute(f'INSERT INTO orgs (id, type, logo, name) VALUES ({newId}\',
{request.form["type"]}, \'{filename}\', \'{request.form["name"]}\')')
        flash(f'Organisation {request.form["name"]} ({newId}) added.', 'success')
        return redirect(f'/management/organisations/view/{newId}')

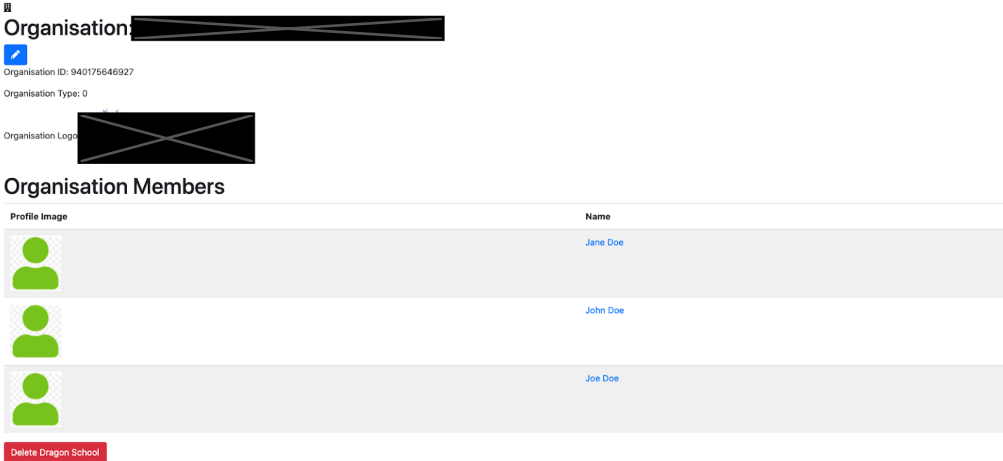

@bp.route('/organisations/delete/<int:id>/')
@auth.login_required
def deleteOrg(id):
    orgsJSON = db.execute(f'DELETE FROM orgs WHERE id = \'{id}\';')
    flash(f'Organisation ({id}) deleted.', 'success')
    return redirect('/management/organisations')

@bp.route('/organisations/view/<int:id>/')
@auth.login_required
def viewOrg(id):
    orgJSON = db.executeQueryReturnJSON(f'SELECT * FROM orgs WHERE id = \'{id}\' FOR JSON AUTO;')
    orgMembersJSON = db.executeQueryReturnJSON(f'SELECT * FROM org_users INNER JOIN users ON
org_users.org_id = users.id WHERE org_id = \'{id}\' FOR JSON AUTO;')
    return render_template('/management/organisation/vieworganisation.html', nav=nav.nav,
org=orgJSON[0], orgMembers=orgMembersJSON)

@bp.route('/organisations/edit/<int:id>/', methods=['GET', 'POST'])
@auth.login_required
def editOrg(id):
    if(request.method == 'GET'):
        orgJSON = db.executeQueryReturnJSON(f'SELECT * FROM orgs WHERE id = \'{id}\' FOR JSON AUTO;')
        return render_template('/management/organisation/editorganisation.html', nav=nav.nav,
org=orgJSON[0])
    if(request.method == 'POST'):
        if request.files['logo']:
            filename = secure_filename(request.form['id']+'.'+request.files['logo'].filename.split('.')
[-1])
            request.files['logo'].save(os.path.join(current_app.config['UPLOAD_FOLDER'], 'orglogos',
filename))
        else:
            filename = 'default.png'
        db.execute(f'UPDATE orgs SET id = \'{request.form["id"]}\', type = {request.form["type"]}, logo
= \'{filename}\', name = \'{request.form["name"]}\' WHERE id = \'{id}\';')
        flash(f'Organisation {request.form["name"]} ({request.form["id"]}) updated.', 'success')
        return redirect(f'/management/organisations/view/{request.form["id"]}')

```

orgIndex	<p>Queries the organisations table using the db manager library to retrieve a JSON representation of every organisation.</p> <p>Renders a template using the orgsJSON.</p> 
addOrg	<p>If the browser sends a 'GET' request a template is returned for the user to fill in:</p>  <p>If the browser sends a 'POST' request the following actions occur:</p> <ul style="list-style-type: none"> • A new 12 digit id is generated using generateNewEightDigitId('orgs') • A check is performed to see if the user provided an organisation logo using the file input in the html. If one is present, the filename is securely processed (preventing unrestricted file upload vulnerabilities) and saved into the 'orglogos' uploads folder. If no image is present, 'default.png' is used instead which is simply the font awesome icon for an organisation. • <code>'INSERT INTO orgs (id, type, logo, name) VALUES (\'{newId}\', {request.form["type"]}, \'{filename}\', \'{request.form["name"]}\')'</code> SQL is executed inserting the form data and newly generated filename into the orgs table. • A redirect is performed to the organisation view page.
deleteOrg(id)	<p><code>'DELETE FROM orgs WHERE id = \'{id}\''</code> SQL is executed, deleting the organisation from the table if it exists.</p> <p>A message is flashed confirming the action occurred when the user is redirected to the organisations listing page.</p>
viewOrg(id)	<p><code>'SELECT * FROM orgs WHERE id = \'{id}\''</code> FOR JSON AUTO; SQL is executed retrieving all organisations with the ID (only one) and stored as orgJSON to be passed to the template.</p> <p><code>'SELECT * FROM org_users INNER JOIN users ON org_users.user_id = users.id WHERE org_id = \'{id}\''</code> FOR JSON AUTO; SQL is executed in addition. An inner join is used in</p>

	<p>conjunction with the org_users table that contains all organisation-user relations to get all users within that organisation. This is then passed to the template to create the 'organisation members' section of the webpage.</p> <p>The rendered template is then returned:</p> 
<p>editOrg(id)</p>	<p>If a 'GET' request is sent by the browser an SQL statement is run to retrieve the organisation with the corresponding ID from the GET request url and convert to JSON for rendering inside the template that is returned to the user:</p>  <p>If a 'POST' request is sent by the browser the following actions occur:</p> <ul style="list-style-type: none"> • A check is performed on the request files object to check whether the user uploaded a file. If a file is present it is securely renamed and saved within the 'orglogos' folder within the uploads folder. If no file was provided, default.png is used instead. • <code>'UPDATE orgs SET id = \''{request.form["id"]}\', type = {request.form["type"]}, logo = \''{filename}\', name = \''{request.form["name"]}\', WHERE id = \''{id}\''</code> SQL is executed updating the organisation(s) with the id provided with the data provided in the request form data field. • A redirect with a flash message highlighting the change is provided to the individual organisation view page.

Vehicles (Fleet Management)



```

@bp.route('/fleetmanagement/')
@auth.login_required
def fleetindex():
    vehiclesJSON = db.executeQueryReturnJSON('SELECT * FROM vehicles FOR JSON AUTO;')
    return render_template('/management/fleetmanagement/fleetmanagement.html', nav=nav.nav,
        vehicles=vehiclesJSON)

@bp.route('/fleetmanagement/view/<int:id>/')
@auth.login_required
def viewVehicle(id):
    vehicleJSON = db.executeQueryReturnJSON(f'SELECT * FROM vehicles WHERE id = \'{id}\'' FOR JSON
        AUTO;')
    vehicleJSON = vehicleJSON[0]

    vehicleJSON['details'] = getVehicleDetailsByRegAsJSON(vehicleJSON['registration_number'])
    return render_template('/management/fleetmanagement/viewvehicle.html', nav=nav.nav,
        vehicle=vehicleJSON)

@bp.route('/fleetmanagement/add/', methods=['GET', 'POST'])
@auth.login_required
def addVehicle():
    if(request.method == 'GET'):
        return render_template('/management/fleetmanagement/addvehicle.html', nav=nav.nav)
    if(request.method == 'POST'):
        newId = generateNewEightDigitId('vehicles')
        if request.files['image']:
            filename = secure_filename(str(newId)+'.'+request.files['image'].filename.split('.')[-1])
            request.files['image'].save(os.path.join(current_app.config['UPLOAD_FOLDER'], 'vehicles',
                filename))
        else:
            filename = 'default.png'
            numplate = request.form["registrationNumber"].replace(' ', '')
            db.execute(f'INSERT INTO vehicles (id, registration_number, image, capacity) VALUES
                (\'{newId}\', \'{numplate}\', \'{filename}\', \'{request.form["capacity"]}\')')
            flash(f'Vehicle {request.form["registrationNumber"]} ({newId}) added.', 'success')
            return redirect(f'/management/fleetmanagement/view/{newId}')



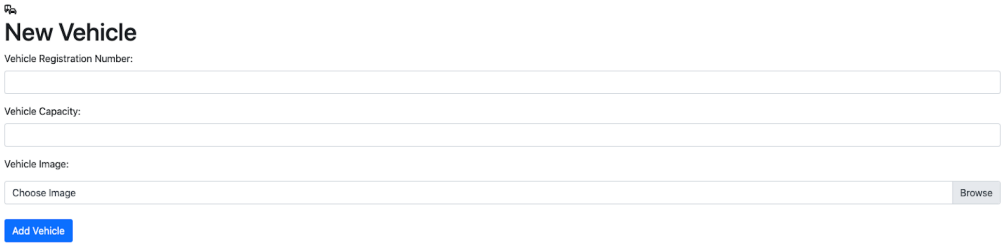
@bp.route('/fleetmanagement/delete/<int:id>/')
@auth.login_required
def deleteVehicle(id):
    orgsJSON = db.execute(f'DELETE FROM vehicles WHERE id = \'{id}\';')
    flash(f'Vehicle ({id}) deleted.', 'success')
    return redirect('/management/fleetmanagement')

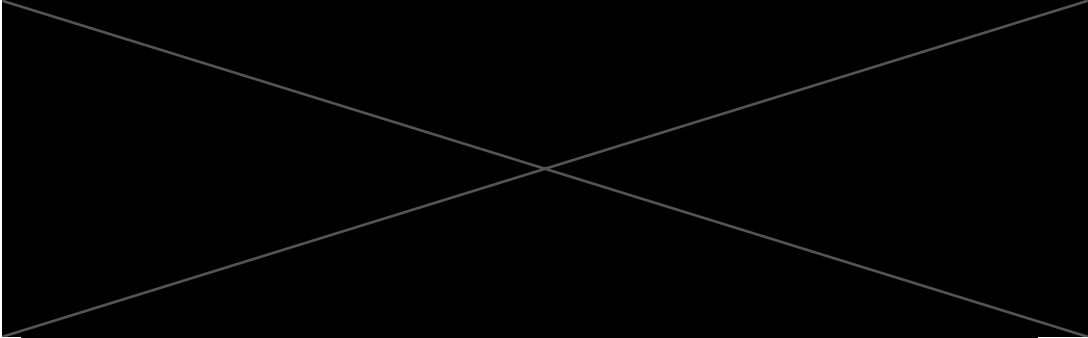
@bp.route('/fleetmanagement/edit/<int:id>/', methods=['GET', 'POST'])
@auth.login_required
def editVehicle(id):
    if(request.method == 'GET'):
        vehicleJSON = db.executeQueryReturnJSON(f'SELECT * FROM vehicles WHERE id = \'{id}\'' FOR JSON
            AUTO;')
        return render_template('/management/fleetmanagement/editvehicle.html', nav=nav.nav,
            vehicle=vehicleJSON[0])
    if(request.method == 'POST'):
        if request.files['image']:
            filename =
                secure_filename(request.form['id']+'.'+request.files['image'].filename.split('.')[-1])
            request.files['image'].save(os.path.join(current_app.config['UPLOAD_FOLDER'], 'vehicles',
                filename))
        else:
            filename = 'default.png'
            db.execute(f'UPDATE vehicles SET id = \'{request.form["id"]}\', registration_number =
                \'{request.form["registration_number"]}\', image = \'{filename}\', capacity =
                \'{request.form["capacity"]}\'' WHERE id = \'{id}\')')
            flash(f'Vehicle {request.form["registration_number"]} ({id}) updated.', 'success')
            return redirect(f'/management/fleetmanagement/view/{request.form["id"]}')

def getVehicleDetailsByRegAsJSON(reg_number):
    dvla_url = "https://driver-vehicle-licensing.api.gov.uk/vehicle-enquiry/v1/vehicles"
    payload = "{\n\t\t\"registrationNumber\": \"\"+reg_number+\"\"\n}"
    headers = {
        'x-api-key': current_app.config['DVLA_KEY'],
        'Content-Type': 'application/json'
    }

    response = requests.request("POST", dvla_url, headers=headers, data=payload)
    return(response.json())

```


fleetindex	<p>A SQL query is run retrieving all the vehicles within the database. The JSON is passed to the template to be rendered:</p> 
viewVehicle(id)	<p>A SQL query is run to retrieve the vehicle matching that ID. In addition to the database record, the vehicle's DVLA record is requested using <code>getVehicleDetailsByRegAsJSON</code> using the number plate from the database. These details are appended to the object under the 'details' key. The object is passed to the template:</p> 
addVehicle	<p>If a 'GET' request is sent by the browser a form is returned as a template to the browser:</p>  <p>If a 'POST' request is sent by the browser the following actions occur:</p> <ul style="list-style-type: none"> • A new 12 digit ID is generated using the aforementioned <code>generateNewEightDigitId</code> function. • A check is performed to see whether a user provided a vehicle photograph within the form. If an image was provided, the necessary security checks are performed and the file is saved into the uploads folder. If no file is attached, 'default.png' is used instead. • If any spaces are present in the number plate, they are removed to

	<p>ensure a standard format within the database table.</p> <ul style="list-style-type: none"> • 'INSERT INTO vehicles (id, registration_number, image, capacity) VALUES (\'{newId}\', \'{numplate}\', \'{filename}\', \'{request.form["capacity"]}\')' SQL is executed using the db manager module to insert the new entry into the vehicle table. • A redirection occurs to the new vehicle view page and a message is flashed to indicate success.
deleteVehicle(id)	<p>The vehicle is deleted from the database using an SQL query via the db manager execute function. The user is redirected to the vehicle listing page with a message to indicate the operation was successful.</p>
editVehicle(id)	<p>If a 'GET' request is sent by the browser then a query is run to retrieve the vehicle details specified in the GET request and passed to a template:</p>  <p>If a 'POST' request is sent by the browser then the following actions occur:</p> <ul style="list-style-type: none"> • A check is performed to see whether the user attached an image. If an image was attached in the request data then the security checks occur and the file is saved to the 'vehicles' upload folder. If none is present 'default.png' is used instead. • 'UPDATE vehicles SET id = \'{request.form["id"]}\', registration_number = \'{request.form["registration_number"]}\', image = \'{filename}\', capacity = \'{request.form["capacity"]}\'' WHERE id = \'{id}\'' SQL is performed updating all the vehicle with the corresponding record ID with the request form data. • The user is redirected to the individual vehicle view page and a message is flashed to confirm the action.
getVehicleDetailsByRegAsJSON(reg_number)	<p>This uses an external API Service run by the DVLA. It is tightly controlled and requires an API key. The API key I was provided limits my usage to 10 queries a second, this is more than enough for educational and development purposes.</p> <p>A request is formed using the Python requests library. A payload is formed with the registration number as a parameter. Headers are appended with the API key from the app configuration file and the MIME content type is set to JSON.</p> <p>The request is sent to the dvla_url and the response is returned to the calling</p>

function as a JSON object with all parameters provided by the DVLA library.

The DVLA Vehicle Enquiry service API returns the following attributes as available (from official documentation):

- registrationNumber - Registration number of the vehicle
- taxStatus - Tax status of the vehicle
- taxDueDate - Date of tax liability, Used in calculating licence information presented to user
- artEndDate - Additional Rate of Tax End Date, format: YYYY-MM-DD
- motStatus - MOT Status of the vehicle
- motExpiryDate - Mot Expiry Date
- make - Vehicle make
- monthOfFirstDvlaRegistration - Month of First DVLA Registration
- monthOfFirstRegistration - Month of First Registration
- yearOfManufacture - Year of Manufacture
- engineCapacity - Engine capacity in cubic centimetres
- co2Emissions - Carbon Dioxide emissions in grams per kilometre
- fuelType - Fuel type (Method of Propulsion)
- markedForExport - True only if vehicle has been export marked
- colour - Vehicle colour
- typeApproval - Vehicle Type Approval Category
- wheelplan - Vehicle wheel plan
- revenueWeight - Revenue weight in kilograms
- realDrivingEmissions - Real Driving Emissions value
- dateOfLastV5CIssued - Date of last V5C issued
- euroStatus - Euro Status (Dealer / Customer Provided (new vehicles))

Users


```

@bp.route('/users/')
@auth.login_required
def usersIndex():
    usersJSON = db.executeQueryReturnJSON('SELECT * FROM users LEFT JOIN org_users ON
    org_users.user_id=users.id LEFT JOIN orgs ON orgs.id=org_users.org_id FOR JSON AUTO;')
    return render_template('/management/user/users.html', nav=nav.nav, users=usersJSON)

@bp.route('/users/view/<int:id>/')
@auth.login_required
def viewUser(id):
    userJSON = db.executeQueryReturnJSON(f'SELECT * FROM users WHERE id = \'{id}\'' FOR JSON AUTO;')
    usersOrgsJSON = db.executeQueryReturnJSON(f'SELECT orgs.id, orgs.name, orgs.type, orgs.logo FROM
    users LEFT JOIN org_users ON org_users.user_id=users.id LEFT JOIN orgs ON orgs.id=org_users.org_id
    WHERE users.id = \'{id}\'' FOR JSON AUTO;')
    checkIfInAnyFamily = db.executeQueryReturnJSON(f'SELECT * FROM family_rel WHERE user_id = \'{id}\''
    FOR JSON AUTO;')
    if(checkIfInAnyFamily):
        userFamilyId = db.executeQueryReturnJSON(f'SELECT family_id FROM family_rel WHERE user_id =
        \'{id}\'' FOR JSON AUTO;')[0]['family_id']
        userFamilyParentJSON = {"parent": db.executeQueryReturnJSON(f'SELECT users.id,
        users.profile_image, users.preferred_name, users.last_name FROM family_rel JOIN users ON
        user_id=users.id WHERE family_id = \'{userFamilyId}\'' AND rel = 1 FOR JSON AUTO;')}
        userFamilyChildJSON = {"children": db.executeQueryReturnJSON(f'SELECT users.id,
        users.profile_image, users.preferred_name, users.last_name FROM family_rel JOIN users ON
        user_id=users.id WHERE family_id = \'{userFamilyId}\'' AND rel = 0 FOR JSON AUTO;')}
        userFamily = {"id": userFamilyId, "parent": {}, "children": {}}
        list(map(userFamily.update, [userFamilyParentJSON, userFamilyChildJSON]))
        print(userFamily)
    else:
        userFamily = False
        return render_template('/management/user/viewuser.html', nav=nav.nav, user=userJSON[0],
        usersOrgs=usersOrgsJSON, userFamily=userFamily)

@bp.route('/users/edit/<int:id>', methods=['GET', 'POST'])
@auth.login_required
def editUser(id):
    if(request.method == 'GET'):
        userJSON = db.executeQueryReturnJSON(f'SELECT * FROM users WHERE id = \'{id}\'' FOR JSON AUTO;')
        usersOrgsJSON = db.executeQueryReturnJSON(f'SELECT orgs.id, orgs.name, orgs.type, orgs.logo
        FROM users LEFT JOIN org_users ON org_users.user_id=users.id LEFT JOIN orgs ON orgs.id=org_users.org_id
        WHERE users.id = \'{id}\'' FOR JSON AUTO;')
        userOrgs = ""
        if(usersOrgsJSON[0]):
            for org in usersOrgsJSON:
                userOrgs += f'{org["id"]}\n'
            return render_template('/management/user/edituser.html', nav=nav.nav, user=userJSON[0],
            usersOrgs=userOrgs)
        if(request.method == 'POST'):
            if request.files['image']:
                filename =
                secure_filename(request.form['id']+'.'+request.files['image'].filename.split('.')[-1])
                request.files['image'].save(os.path.join(current_app.config['UPLOAD_FOLDER'],
                'userprofilepictures', filename))
            else:
                filename = 'default.png'
            db.execute(f'UPDATE users SET email = \'{request.form["email"]}\', given_name =
            \'{request.form["given_name"]}\', last_name = \'{request.form["last_name"]}\', password =
            \'{generate_password_hash(request.form["password"])}\', preferred_name =
            \'{request.form["preferred_name"]}\', title = \'{request.form["title"]}\', id =
            \'{request.form["id"]}\', profile_image = \'{filename}\'' WHERE id = \'{id}\''')
            db.execute(f'DELETE FROM org_users where user_id = \'{id}\''')
            for userOrg in request.form["organisations"].split("\r\n"):
                if(userOrg):
                    db.execute(f'INSERT INTO org_users (user_id, org_id) VALUES
                    (\'{request.form["id"]}\', \'{userOrg}\')')
            return redirect(f'/management/users/view/{request.form["id"]}')

@bp.route('/users/add/', methods=['GET', 'POST'])
@auth.login_required
def addUser():
    if(request.method == 'GET'):
        return render_template('/management/user/adduser.html', nav=nav.nav)
    if(request.method == 'POST'):
        newId = generateNewEightDigitId('users')
        if request.files['image']:
            filename = secure_filename(newId+'.'+request.files['image'].filename.split('.')[-1])
            request.files['image'].save(os.path.join(current_app.config['UPLOAD_FOLDER'],
            'userprofilepictures', filename))
        else:
            filename = 'default.png'
        db.execute(f'INSERT INTO users (email, given_name, last_name, password, preferred_name, title,
        id, profile_image) VALUES (\'{request.form["email"]}\', \'{request.form["given_name"]}\',
        \'{request.form["last_name"]}\', \'{generate_password_hash(request.form["password"])}\',
        \'{request.form["preferred_name"]}\', \'{request.form["title"]}\', \'{newId}\', \'{filename}\')')
        db.execute(f'DELETE FROM org_users where user_id = \'{newId}\''')
        for userOrg in request.form["organisations"].split("\r\n"):
            if(userOrg):
                db.execute(f'INSERT INTO org_users (user_id, org_id) VALUES
                (\'{newId}\', \'{userOrg}\')')
        return redirect(f'/management/users/view/{newId}')

@bp.route('/users/delete/<int:id>/')
@auth.login_required
def deleteUser(id):
    orgsJSON = db.execute(f'DELETE FROM users WHERE id = \'{id}\''')
    flash(f'User ({id}) deleted.', 'success')
    return redirect('/management/users/')

@bp.route('/users/quicksearch')
@auth.login_required
def userQuickSearch():
    query = request.args.get('query')
    resultJSON = db.executeQueryReturnJSON(f'SELECT id, preferred_name, last_name, profile_image FROM
    users WHERE preferred_name LIKE \'{query}%\' OR last_name LIKE \'{query}%\' OR given_name LIKE \'{
    query}%\' OR id LIKE \'{query}%\' OR CONCAT(preferred_name, \', \', last_name) LIKE \'{query}%\' OR
    CONCAT(given_name, \', \', last_name) LIKE \'{query}%\' FOR JSON AUTO;')
    return jsonify(resultJSON)

```

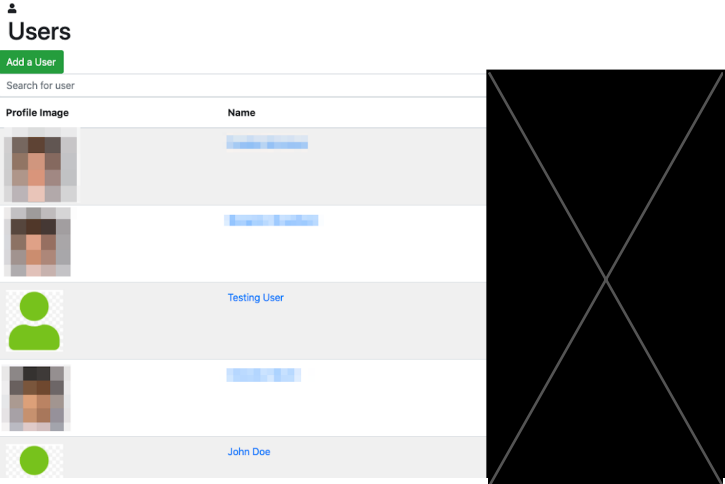
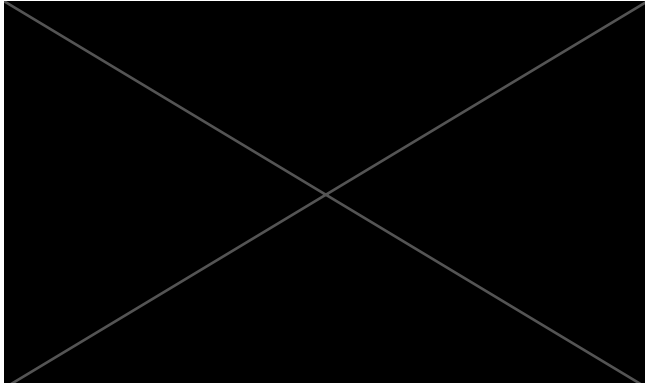
usersIndex

Executes the following SQL statement on the org_users table:

```

'SELECT * FROM users LEFT JOIN org_users ON
org_users.user_id=users.id LEFT JOIN orgs ON

```

	<p><code>orgs.id=org_users.org_id FOR JSON AUTO;</code> and passes the JSON response to a template. A left join is used on the orgs table to extract all organisations that the user belongs to for display within the template:</p> 
<p>viewUser(id)</p>	<p>If a 'GET' request is received with a user ID the user profile is requested using an SQL query and passed to the template.</p> <p>In addition to the user details retrieval an additional check is performed to see if the user is a member of any family using the family_rel table. If the user is in a family then an additional set of queries is performed to build a family object for the user gathering the parents and children within the family.</p> <p>An example of the parent query for a user family is <code>'SELECT users.id, users.profile_image, users.preferred_name, users.last_name FROM family_rel JOIN users ON user_id=users.id WHERE family_id = \''{userFamilyId}\'' AND rel = 1 FOR JSON AUTO;'</code> where the users are selected from the family_rel table and a JOIN is used to find any parents (rel = 1 for parent rel = 0 for child) who are in the same family.</p> <p>Both queries are run and the result is mapped into the userFamily object using the python 'map' function and the python dictionary 'update function'.</p> <p>The user family is then passed to the template to be rendered:</p> 
<p>editUser(id)</p>	<p>If the request from the browser is a 'GET' request a lookup for the requested user id is performed using SQL. The users organisations are also retrieved in a similar manner to that described in viewUser. These are then passed to the user edit template:</p>

- A new unique twelve digit ID is generated for the user for the users

	<p>table using generateNewEightDigitId</p> <ul style="list-style-type: none"> • A check is performed to see if the user attached a profile picture within the request form data. If a photo is present, security checks are performed and the photo is saved into 'userprofilepictures' within the static 'uploads' folder. • Similar to editUser, the org_users table is cleared where any matching user_id is present. • Organisations are appended to the user in the org_users table using a SQL statement for each line entry within the text area. • A redirect is performed to the new user view page.
deleteUser(id)	<p>An SQL statement is performed to remove the corresponding user from the database with the corresponding user ID. A redirect is performed and a message is flashed to the user to indicate the operation was successful.</p>
userQuickSearch	<p>The 'GET' request 'query' is extracted using the request.args.get function within flask retrieving the URL GET parameters (after the ? mark).</p> <p>The following SQL query is used to perform the search from the query: 'SELECT id, preferred_name, last_name, profile_image FROM users WHERE preferred_name LIKE \'%{query}%\' OR last_name LIKE \'%{query}%\' OR given_name LIKE \'%{query}%\' OR id LIKE \'%{query}%\' OR CONCAT(preferred_name, \' \', last_name) LIKE \'%{query}%\' OR CONCAT(given_name, \' \', last_name) LIKE \'%{query}%\' FOR JSON AUTO;'</p> <p>The 'LIKE' keyword is used to perform a pattern matching technique on various fields including the concatenation of the first name, last name field so queries like 'John D' still return results. The '%' symbol indicates a wildcard that will match anything beyond / before the query.</p> <p>An example of the JSON response when performing a query 'doe' using a 'GET request':</p> <pre>http://localhost:5000/management/users/quicksearch?query=doe</pre> <pre>[{ "id": "452134765609", "last_name": "Doe", "preferred_name": "John", "profile_image": "default.png" }, { "id": "608041185341", "last_name": "Doe", "preferred_name": "Jane", "profile_image": "default.png" },]</pre>

```
{  
  "id": "571915266587",  
  "last_name": "Doe",  
  "preferred_name": "Joe",  
  "profile_image": "default.png"  
}
```

More about the front-end implementation of this function is provided within the JavaScript section.

Families



```

@bp.route('/family/add/', methods=['GET'])
@auth.login_required
def addFamily():
    if(request.method == 'GET'):
        if(request.args.get('parent')):
            checkIfInAnyFamily = db.executeQueryReturnJSON(f'SELECT * FROM family_rel WHERE user_id = \'{request.args.get("parent")}\'' FOR JSON AUTO;')
            checkIfUser = db.executeQueryReturnJSON(f'SELECT id FROM users WHERE id = \'{request.args.get("parent")}\'' FOR JSON AUTO;')
            if(checkIfInAnyFamily or not checkIfUser):
                return Response(status=400)
            else:
                familyId = generateNewEightDigitId('family')
                family_relId = generateNewEightDigitId('family_rel')
                db.execute(f'INSERT INTO family (id) VALUES (\'{familyId}\')')
                db.execute(f'INSERT INTO family_rel (id, family_id, user_id, rel) VALUES (\'{family_relId}\', \'{familyId}\', \'{request.args.get("parent")}\', 1)')


                return redirect(f'/management/users/view/{request.args.get("parent")}')
        else:
            return Response(status=404)

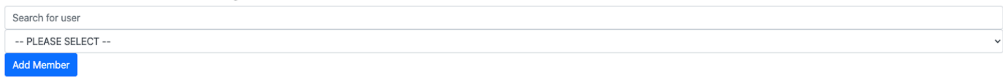
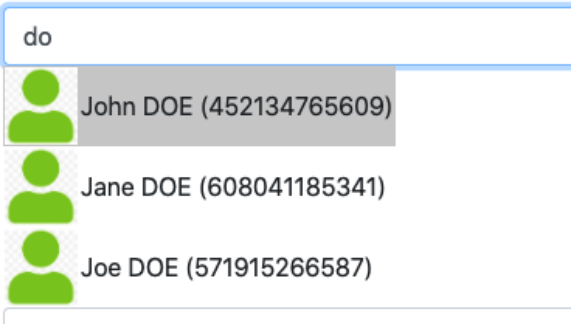
@bp.route('/family/delete/<int:id>/')
@auth.login_required
def deleteFamily(id):
    db.execute(f'DELETE FROM family_rel WHERE family_id = \'{id}\';')
    db.execute(f'DELETE FROM family WHERE id = \'{id}\';')
    flash(f'Family ({id}) deleted.', 'success')
    return redirect(f'/management/users/')

@bp.route('/family/deletemember/<int:family_id>/<int:member_id>/')
@auth.login_required
def deleteFamilyMember(family_id, member_id):
    familyMemberCount = db.executeQueryReturnJSON(f'SELECT * FROM family_rel WHERE family_id = \'{family_id}\'' FOR JSON AUTO;')
    if(len(familyMemberCount) <= 1):
        return redirect(f'/management/family/delete/{family_id}')
    db.execute(f'DELETE FROM family_rel WHERE family_id = \'{family_id}\'' AND user_id = \'{member_id}\';')
    return redirect(f'/management/users/view/{member_id}')

@bp.route('/family/addmember/<int:family_id>', methods=['GET', 'POST'])
@auth.login_required
def addFamilyMember(family_id):
    if(request.method == 'GET'):
        return render_template('/management/user/addmemberfamily.html', nav=nav.nav, familyId=family_id)
    if(request.method == 'POST'):
        newId = generateNewEightDigitId('family_rel')
        user_id = request.form['user_id']
        relType = request.form['type']
        checkIfInSameFamily = db.executeQueryReturnJSON(f'SELECT * FROM family_rel WHERE user_id = \'{user_id}\'' AND family_id = \'{family_id}\'' FOR JSON AUTO;')
        checkIfUser = db.executeQueryReturnJSON(f'SELECT id FROM users WHERE id = \'{user_id}\'' FOR JSON AUTO;')
        if(checkIfInSameFamily or not checkIfUser):
            flash(f'User ({user_id}) already in family ({family_id}) or doesn\'t exist.', 'danger')
            return redirect(f'/management/users/')
        if(relType != '1' and relType != '0'):
            flash(f'Relative Type Invalid.', 'danger')
            return redirect(f'/management/users/')
        family_relId = generateNewEightDigitId('family_rel')
        db.execute(f'INSERT INTO family_rel (id, family_id, user_id, rel) VALUES (\'{family_relId}\', \'{family_id}\', \'{user_id}\', \'{relType}\')')
        return redirect(f'/management/users/view/{user_id}')

```

addFamily	<p>If the request is a 'GET' request then the request arguments are extracted (the parent user ID). A check is performed to ensure this is present otherwise an error response will be returned.</p> <p>A check is performed to check if the parent user is already in any families, a check is also performed to ensure the user ID is within the database. If either of these checks fail an error 404 not found is returned as the parameters are invalid.</p> <p>Two unique identifiers are generated using generateNewEightDigitId in preparation for the insertion.</p> <p>If all checks pass then two SQL statements are executed:</p> <ul style="list-style-type: none"> • <code>'INSERT INTO family (id) VALUES (\'{familyId}\')</code> • <code>'INSERT INTO family_rel (id, family_id, user_id, rel) VALUES (\'{family_relId}\', \'{familyId}\', \'{request.args.get("parent")}\', 1)'</code> <p>These setup the 'scaffolds' for the family to be valid. The family table is updated to indicate a family with that ID is present in the system and the user is added to the family through the family relations table, family_rel. The user is by default classed as the parent as they were the origin of the family.</p> <p>A redirect is performed back to the individual user view page:</p> <p>Family</p> 
deleteFamily(id)	<p>Two SQL operations are performed to delete any record of the corresponding family ID provided in the 'GET' request URL from both the family and family_rel table, disbanding the family:</p> <ul style="list-style-type: none"> • <code>'DELETE FROM family_rel WHERE family_id = \'{id}\';'</code> • <code>'DELETE FROM family WHERE id = \'{id}\';'</code>
deleteFamilyMember(family_id, member_id)	<p>A check is first performed to retrieve the family member count using python's length operator len and the db.executeQueryReturnJSON function. If the count is less than or equal to 1 then the request is redirected to deleteFamily(id) with the family_id passed through which will in turn remove the member from the family and disband the family permanently.</p> <p>If there are more is more than one member within the family the user is simply deleted from the family relationship table, family_rel, using the following SQL statement:</p> <pre>'DELETE FROM family_rel WHERE family_id = \'{family_id}\'</pre> <pre>AND user_id = \'{member_id}\';'</pre>

	<p>The request is redirected to the individual user view page once the operation is complete.</p>
<p>addFamilyMember(family_id)</p>	<p>If the request sent by the client's browser is a 'GET' request then the add family member form template is returned with the corresponding family ID present in the 'GET' request URL:</p> <p>Add Member to Family#939422712308</p>  <p>The form 'user' field is also implementing the quick search feature seen earlier in the users section. Simply typing a user's name allows for easy selection:</p>  <p>If a 'POST' request is sent to the endpoint then the following operations are performed:</p> <ul style="list-style-type: none"> • A new family_rel ID is generated for the new relationship. • The request form data is retrieved and stored as new variables 'user_id' and 'relType' • A check is performed to check whether the proposed user is already in the same family. An additional check is performed to ensure the user exists within the database. If either check fails, a redirection occurs to the users listing page with a message indicating the operation was not successful. • An additional check is performed to ensure the relType (relation type) is valid, either being a 1 (parent) or 0 (child). If the test fails a redirection occurs to the user listing page with an error message to indicate an invalid relation type. • An insertion SQL query is executed to insert the new data into the family_rel table. • A redirection occurs to the added user's view page.

Icons





```

@bp.route('/icons/quicksearch')
@auth.login_required
def iconQuickSearch():
    query = request.args.get('query')
    resultJSON = db.executeQueryReturnJSON(f'SELECT id, name FROM icons WHERE name LIKE \'%{query}%\'
    FOR JSON AUTO;')
    return jsonify(resultJSON)
        
```

iconQuickSearch	<p>Upon receiving a 'GET' request to the icon quick search endpoint the query parameter is extracted from the request.args object.</p> <p>A search is performed using query pattern matching utilising the SQL '%' wildcard operating on the icon name property.</p> <p>A JSON response is returned with the data required to render on the front-end:</p> <div> <p>http://localhost:5000/management/icons/quicksearch?query=book</p> <pre> [{ "id": 8, "name": "address-book" }, { "id": 224, "name": "book" }, { "id": 225, "name": "book-alt" }, { "id": 226, "name": "book-dead" }, ... </pre> </div> <p>More about the front-end implementation of this function is provided within</p>
------------------------	---

Navigation Menu

The navigation menu module contains all the flask code related to navigation within the application. The main example of this being the dynamic navigation bar presented to all users of the application

```

from flask import Blueprint, render_template
from flask.views import View

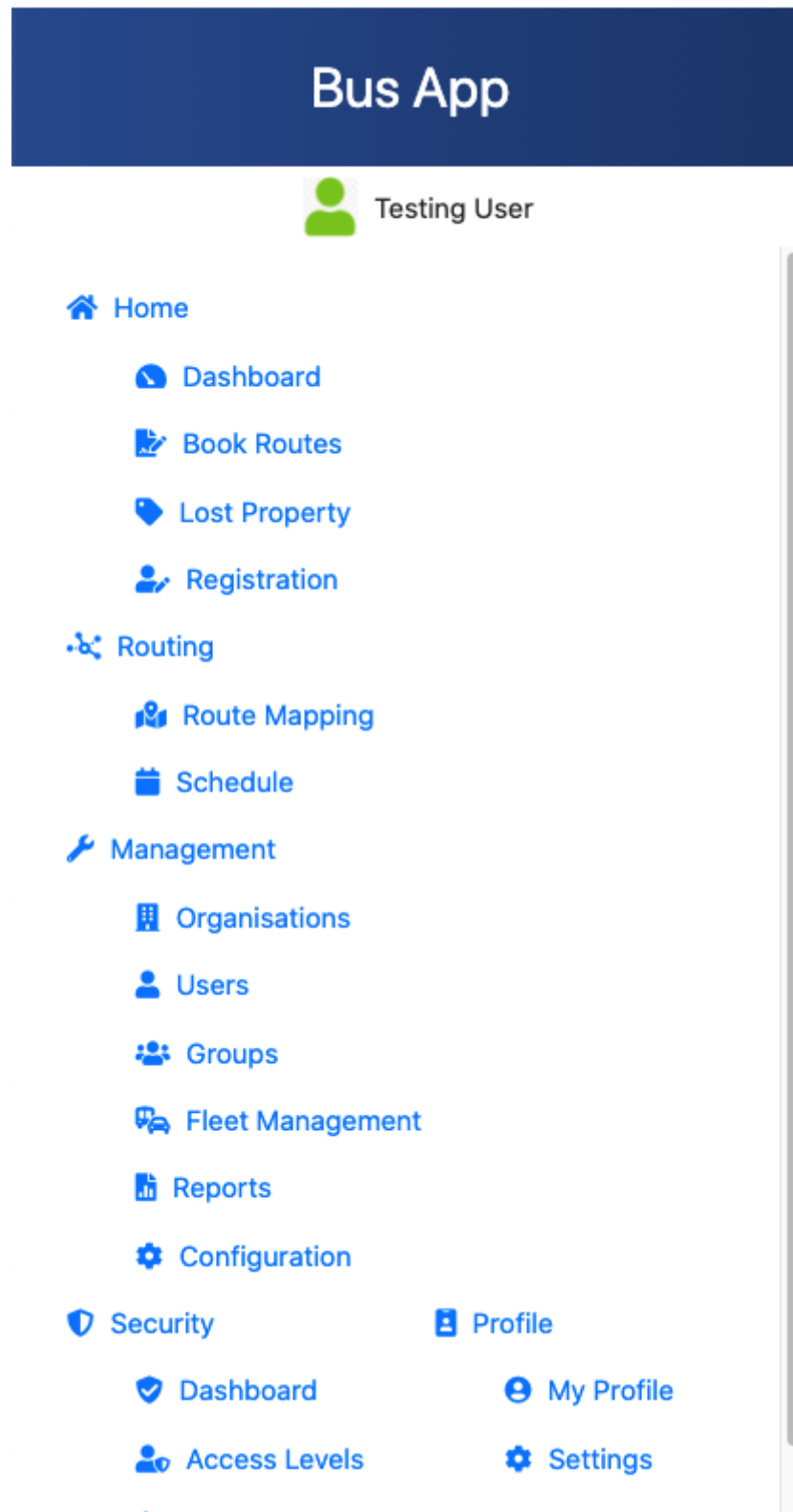
nav = {
    'Home': {
        'perm': 'nav.home',
        'icon': 'fa-home',
        'subLinks': [
            {'displayName': 'Dashboard', 'icon': 'fa-tachometer-slow', 'perm': 'nav.home.dashboard',
            'link': '#'},
            {'displayName': 'Book Routes', 'icon': 'fa-file-signature', 'perm': 'nav.home.bookroutes',
            'link': '#'},
            {'displayName': 'Lost Property', 'icon': 'fa-tag', 'perm': 'nav.home.lostproperty', 'link':
            '#'},
            {'displayName': 'Registration', 'icon': 'fa-user-edit', 'perm': 'nav.home.dashboard',
            'link': '#'},
        ]
    },
    'Routing': {
        'perm': 'nav.routing',
        'icon': 'fa-chart-network',
        'subLinks': [
            {'displayName': 'Route Mapping', 'icon': 'fa-map-marked-alt', 'perm':
            'nav.routing.routemapping', 'link': '/routing/route-mapping'},
            {'displayName': 'Schedule', 'icon': 'fa-calendar', 'perm': 'nav.routing.schedule', 'link':
            '#'},
        ]
    },
    'Management': {
        'perm': 'nav.management',
        'icon': 'fa-wrench',
        'subLinks': [
            {'displayName': 'Organisations', 'icon': 'fa-building', 'perm':
            'nav.management.organisations', 'link': '/management/organisations'},
            {'displayName': 'Users', 'icon': 'fa-user', 'perm': 'nav.management.users', 'link':
            '/management/users'},
            {'displayName': 'Groups', 'icon': 'fa-users', 'perm': 'nav.management.groups', 'link':
            '#'},
            {'displayName': 'Fleet Management', 'icon': 'fa-car-bus', 'perm':
            'nav.management.fleetmanagement', 'link': '/management/fleetmanagement'},
            {'displayName': 'Reports', 'icon': 'fa-file-chart-line', 'perm': 'nav.management.reports',
            'link': '#'},
            {'displayName': 'Configuration', 'icon': 'fa-cog', 'perm': 'nav.management.configuration',
            'link': '#'},
        ]
    },
    'Security': {
        'perm': 'nav.security',
        'icon': 'fa-shield-alt',
        'subLinks': [
            {'displayName': 'Dashboard', 'icon': 'fa-shield-check', 'perm': 'nav.security.dashboard',
            'link': '#'},
            {'displayName': 'Access Levels', 'icon': 'fa-user-shield', 'perm':
            'nav.security.accesslevels', 'link': '/security/access-levels'},
            {'displayName': 'Audit Log', 'icon': 'fa-clipboard-list', 'perm': 'nav.security.auditlog',
            'link': '#'},
        ]
    },
    'Profile': {
        'perm': 'nav.profile',
        'icon': 'fa-id-badge',
        'subLinks': [
            {'displayName': 'My Profile', 'icon': 'fa-user-circle', 'perm': 'nav.profile.profile',
            'link': '/user/profile'},
            {'displayName': 'Settings', 'icon': 'fa-cog', 'perm': 'nav.profile.settings', 'link': '#'},
        ]
    },
}

bp = Blueprint('nav', __name__)

```

The navigation nav object is passed forwards to all requests. The base template is then able to render each node and subnode correctly for the end-user allowing for easy modification without having to update every template.

The object above is formatted to render the template:

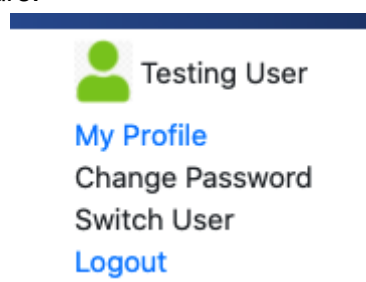


The HTML template used to provide this dynamic generation is shown below:

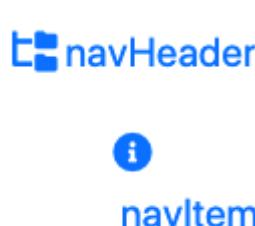

```
{% if g.user %}
<div class="logo d-flex flex-row justify-content-center align-items-center">Bus App</div>
  <div onclick="toggleUserActionMenu()" id="userInfoBar" class="user d-flex flex-row justify-
content-center align-items-center">{{ g.user['preferred_name'] }} {{ g.user['last_name'] }}</div>
  <ul id="userActionMenu" style="display: none;" class="userActionMenu">
    <a href="/user/profile"><li>My Profile</li></a>
    <li>Change Password</li>
    <li>Switch User</li>
    <a href="/auth/logout"><li>Logout</li></a>
  </ul>
  <ul class="nav p-3 navLinks">
    {% for navHeader in nav %}
      <li class="nav-item">
        <a class="nav-link" href="#"><i class="fas {{nav[navHeader].icon}}"></i><span
class="fa-margin">{{navHeader}}</span></a>
        <ul>
          {% for navItem in nav[navHeader].subLinks %}
            <li class="nav-item">
              <a class="nav-link" href="{{navItem.link}}"><i class="fas {{navItem.icon}}">
</i><span class="fa-margin">{{navItem.displayName}}</span></a>
            </li>
          {% endfor %}
        </ul>
      </li>
    {% endfor %}
  </ul>
{% endif %}
```

A check is performed to check if the user is logged in, otherwise the menu will not present the right options. Additionally the application has no real public facing features and therefore does not need to be rendered without an authorised user.

The first chunk of template is used for the user login indicator at the top of the navbar with features such as dynamic username and profile picture:



The next section consists of two nested loops to produce the header-navsublink effect shown in the example below.

	<p>The first loop is used to iterate over each navHeader or application category. The header icon is extracted and dynamically loaded in as a font awesome icon.</p> <p>For each header, another for loop is used to iterate over the navItems of a navHeader subLinks as seen in nav.py. The navItem is then rendered with a link pointing to the corresponding page and an icon with display name as well.</p>
---	--

Freddie Nicholson Computer Science NEA

Routing Module

The Routing Module is the back-end responsibility of handling the processing to define routes within the application.

```
from flask import Blueprint, render_template, request, current_app, redirect, flash, g, session, abort
from . import auth
from . import nav
from . import db
from . import osrm
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
import json

bp = Blueprint('routing', __name__)

@bp.route('/route-mapping/')
@auth.login_required
def routeMapping():
    routesJSON = db.executeQueryReturnJSON(f'SELECT * FROM route FOR JSON AUTO;')
    return render_template('/routing/routes.html', nav=nav.nav, routes=routesJSON)

@bp.route('/route-mapping/add/', methods=['GET', 'POST'])
@auth.login_required
def newRoute():
    if request.method == 'GET':
        return render_template('/routing/newroute.html', nav=nav.nav)

@bp.route('/route-mapping/calculate', methods=['POST'])
@auth.login_required
def calculateRoute():
    requestStops = request.get_json(force=True)
    if('start' not in requestStops.keys() or 'end' not in requestStops.keys()):
        return abort(404)
    nodes = {}
    startNode = requestStops['start']
    endNode = requestStops['end']
    for stop in requestStops:
        if(stop != 'start' and stop != 'end'):
            print(stop)
            if('lat' in requestStops[stop].keys()):
                nodes[int(stop)] = (requestStops[stop]['lat'], requestStops[stop]['lng'])
            else:
                return abort(404)
    print(requestStops)
    distances = createDistanceMatrix(nodes)
    def distance_callback(i, j):
        iR = manager.IndexToNode(i)
        jR = manager.IndexToNode(j)
        return distances[iR][jR]

    manager = pywrapcp.RoutingIndexManager(len(distances), 1, [startNode], [endNode])
    routing = pywrapcp.RoutingModel(manager)

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

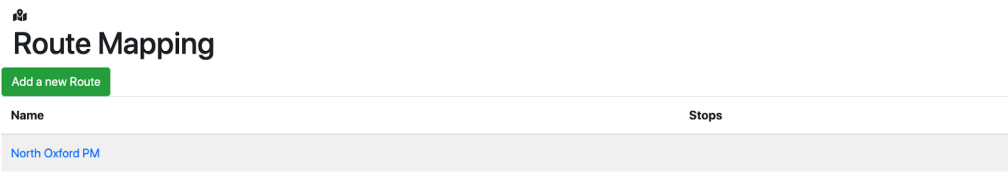
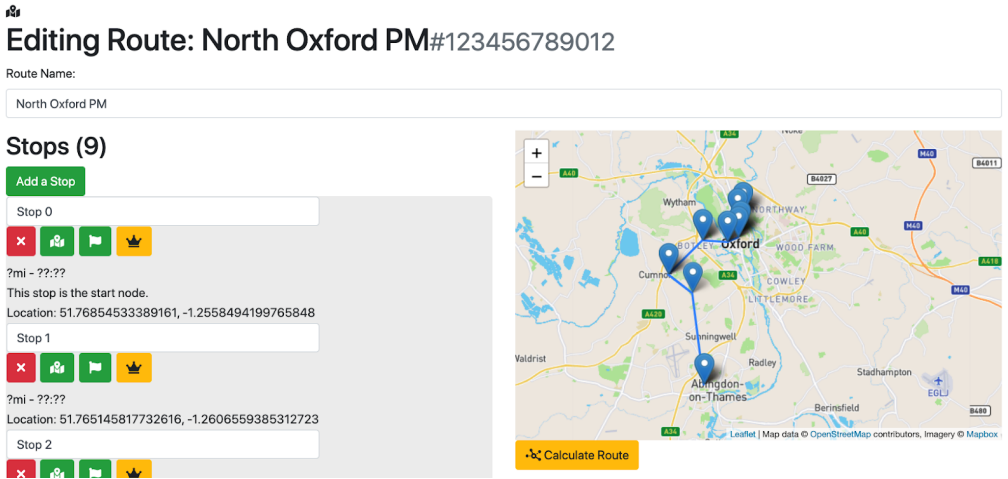
    solution = routing.SolveWithParameters(search_parameters)

    def exportSolution(manager, routing, solution):
        index = routing.Start(0)
        route_nodes = []
        route_distance = 0
        while not routing.IsEnd(index):
            route_nodes.append(manager.IndexToNode(index))
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
            route_nodes.append((manager.IndexToNode(index)))
            print(route_nodes)
        return(route_nodes)

    return json.dumps(exportSolution(manager, routing, solution))

def createDistanceMatrix(stops):
    distances = []
    for i in stops:
        distances.append([])
        for j in stops:
            distances[i].append(osrm.getShortestRoute(stops[i], stops[j])['routes'][0]['distance'])
    return distances

def createTimeMatrix(stops):
    times = []
    for i in stops:
        times.append([])
        for j in stops:
            times[i].append(osrm.getShortestRoute(stops[i], stops[j])['routes'][0]['duration'])
    return times
```

routeMapping	<p>Queries using SQL for all routes within the route table and passes them to the route listing template.</p> 
editRoute(id)	<p>If the request is sent from the browser as a 'GET' request then a routeJSON object will be extracted from the route table using the db manager in conjunction with the specific ID specified in the function parameters. This route is then passed via templating to the template to be rendered within the browser.</p> <p>Most of the dynamic editing logic is performed by the front-end JavaScript functions described in the JavaScript section.</p> 
calculateRoute	<p>Contains sub functions distance_callback and exportSolution</p> <p>The stops set out by the user are extracted from the request payload JSON. A check is performed to ensure that both a start and end node have been specified otherwise an error is returned. The startNode and endNode are then stored as variables for OR Tools.</p> <p>An iterator is used to loop over each stop and ensure each stop has a location marked. If no location is marked, an error is sent to the user. If all stops have a location then a nodes dictionary is formed with each latitude and longitude similar to the n1 variable used in testing.</p> <p>The nodes are then passed to the createDistanceMatrix function that forms the distance matrix as demonstrated before.</p> <p>Much of this function has already been described before in the development of the testing code towards the solution. As OR Tools relies on functions within some solution definitions I have utilised Python's ability to nest functions within one another allowing for new problem solvers to be generated for each unique routing problem.</p>

	<p>In addition, previously the approach defined in the algorithm was to do a delivery round-robin style run around each node. This is not appropriate for a school bus route. Instead, I needed to set up the solver to utilise a 'start' and 'end' node defined by the user. This involved modifying the RoutingIndexManager definition to take into account the user defined startNode and endNode from the user request payload JSON:</p> <pre>manager = pywrapcp.RoutingIndexManager(len(distances), 1, [startNode], [endNode])</pre> <p>The print_solution function has been renamed to exportSolution and adjusted to return the solution to the front end as a JSON object to be processed and displayed to the end user using JavaScript.</p>
<p>createDistanceMatrix(stops)</p>	<p>This is almost identical to the original code seen in the testing scripts, the logic related to plotting for debugging purposes in matplotlib is removed.</p>

Security Module

```

from flask import Blueprint, render_template, request, current_app, redirect, flash, Response, jsonify
from . import nav
from . import db
from werkzeug.utils import secure_filename
from werkzeug.security import check_password_hash, generate_password_hash
from . import auth
import os
import random
import requests
import json

bp = Blueprint('security', __name__)

def generateNewTwelveDigitId(tableName):
    testId = random.randint(111111111111, 999999999999)
    respJson = db.executeQueryReturnJSON(f'SELECT * FROM {tableName} WHERE id = \'{testId}\'' FOR JSON
    AUTO;')
    if(respJson):
        return generateNewTwelveDigitId(tableName)
    else:
        return testId

@bp.route('/access-levels/')
@auth.login_required
def accessLevels():
    accessLevelsJSON = db.executeQueryReturnJSON(f'SELECT * FROM access_level FOR JSON AUTO;')
    return render_template('/security/accesslevels.html', nav=nav.nav,
    accessLevelsJSON=accessLevelsJSON)

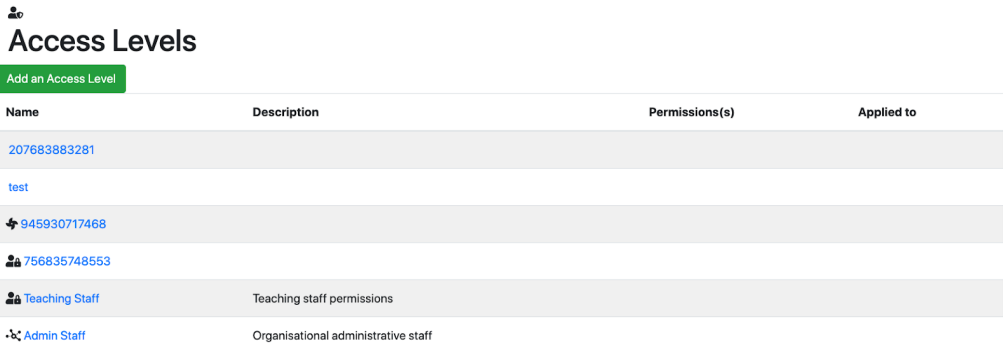
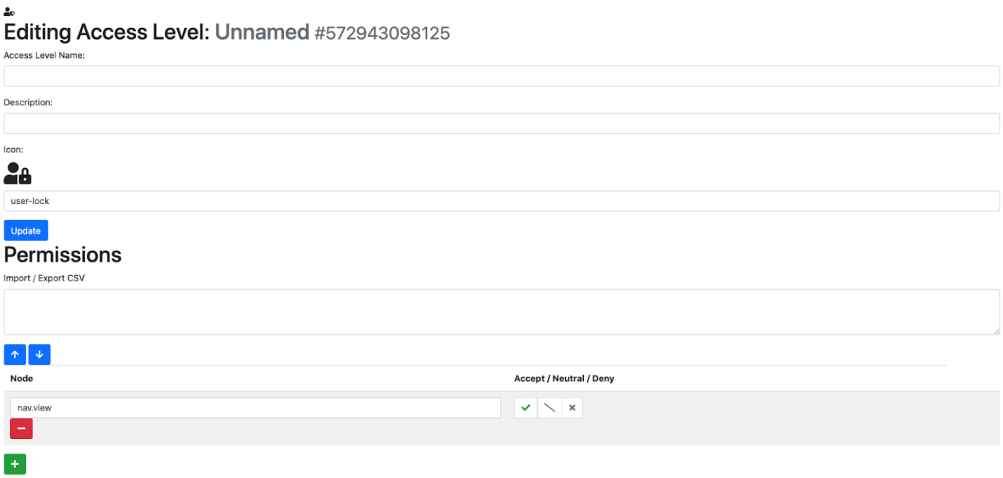
@bp.route('/access-levels/add')
@auth.login_required
def addAccessLevel():
    accessLevelId = generateNewTwelveDigitId('access_level')
    db.execute(f'INSERT INTO access_level (id, icon) VALUES (\'{accessLevelId}\', \'user-lock\')')
    return redirect(f'/security/access-levels/edit/{accessLevelId}')


@bp.route('/access-levels/edit/<int:id>', methods=['GET', 'POST'])
@auth.login_required
def editAccessLevel(id):
    if(request.method == 'GET'):
        accessLevel = db.executeQueryReturnJSON(f'SELECT * FROM access_level WHERE id = \'{id}\'' FOR
    JSON AUTO;')
        return render_template('/security/editaccesslevel.html', nav=nav.nav,
    accessLevel=accessLevel[0])
    if(request.method == 'POST'):
        db.execute(f'UPDATE access_level SET name = \'{request.form["name"]}\', icon =
    \'{request.form["accessLevelIcon"]}\',description = \'{request.form["description"]}\'' WHERE id =
    \'{id}\')')
        flash(f'Access Level {request.form["name"]} ({id}) updated.', 'success')
        return redirect(f'/security/access-levels/')

```

The security module is the back-end representation of all authorisation and audit logging features. There are four main functions:

generateNewTwelveDigitId(tableName)	Seen beforehand in management.py
accessLevels	Endpoint to list all access levels. Executes an SQL query to the access-level table requesting a JSON object

	<p>consisting of all levels.</p> <p>accessLevelsJSON is passed to access level listing template to be rendered by Flask:</p> 
addAccessLevel	<p>A new accessLevelId is generated for the access_level table and stored as a variable.</p> <pre>'INSERT INTO access_level (id, icon) VALUES (\'{accessLevelId}\', \'user-lock\')'</pre> <p>SQL is executed to insert a new default record into the access_level table with placeholder and undefined values.</p> <p>A redirect is then performed to the editAccessLevel function for the corresponding accessLevelId.</p>
editAccessLevel	<p>If a 'GET' request is sent to the endpoint then the selection SQL query for the access_level will be run retrieving the individual record.</p> <p>The individual record will be passed to the template to be rendered with some assistance in the permissions management system using JavaScript:</p>  <p>The icon field also utilises the management module's icon quick search feature allowing the user to quickly lookup icons using keywords:</p>

icon:


address-book (8)

book (224)

book-alt (225)

book-dead (226)

book-heart (227)

book-medical (228)

book-open (229)

book-reader (230)

book-spells (231)

book-user (232)

```
from flask import Blueprint, render_template, request, current_app, redirect, flash, g, session
from . import auth
from . import nav
from . import db

bp = Blueprint('user', __name__)

@bp.route('/profile/')
def profile():
    uId = session['user_id']
    userJSON = db.executeQueryReturnJSON(f'SELECT * FROM users WHERE id = \'{uId}\'' FOR JSON AUTO;')
    return render_template('/profile/profile.html', nav=nav.nav, userProfileData=userJSON[0])
```

This module handles information related to a user personally and utilities to help assist with management of their personal data within the application.

profile

Gets the requester's users ID using the Flask request session object.

Retrieves the userJSON object from the database using the db manager JSON query function with the corresponding user ID.

User object is passed to the template to be rendered in browser:

Profile

Please contact your organisation administrator to update your details.

Name	Testing (Testing) User
Email	testing.user@appdevteam.com



JavaScript is another high-level language that uses similar programming concepts to Python. However the syntax is fundamentally different but can be read similar to Python. It is used within front-end web applications to help offer a dynamic featureset without relying on new page requests to a back-end server.

script.js


General site-wide scripts to fulfill requirements within the application that do not readily fall into another script category.


```
navShowing = false;
function toggleNav() {
    if(navShowing) {
        document.getElementById('sideNav').style.display = 'none';
        navShowing = false;
    } else {
        document.getElementById('sideNav').style.display = 'block';
        navShowing = true;
    }
}

userActionMenuShowing = false;
function toggleUserActionMenu() {
    if(userActionMenuShowing) {
        document.getElementById('userActionMenu').style.display = 'none';
        userActionMenuShowing = false;
    } else {
        document.getElementById('userActionMenu').style.display = 'block';
        userActionMenuShowing = true;
    }
}
```

script.js consists of two main functions that are used site-wide:

toggleNav	<p>navShowing is a global site variable used to indicate whether the side navbar is showing. By default the navbar is hidden hence the value being set to False.</p> <p>Whenever the toggleNav function is called a check is performed to determine whether the sideNav's display styling needs to be set to 'none' or 'block' (shown).</p> <p>On each toggle, the navShowing variable will be updated to reflect the current navbar situation.</p>
------------------	---


The main example of a caller to the toggleNav function is the ‘hamburger’ icon in the corner of each application page. 




Profile

Please contact your organisation administrator to update your profile

Name	Testing (Testi
Email	testing.user@



Bus App

 Testing User

Home

Dashboard

Book Routes

Lost Property

Registration

Routing

Route Mapping

Schedule

Management

Organisations

Users

Groups

Fleet Management

Reports

Configuration

Security

Dashboard


Access Levels


Audit Log

Profile

toggleUserActionMenu

Similar to the logic within toggleNav, when the username field within the side navbar is clicked, additional links related to the personal profile will have their display toggled.

 Testing User

	<div data-bbox="769 159 831 226"></div> <div data-bbox="841 172 1024 217">Testing User</div> <div data-bbox="526 237 676 282">My Profile</div> <div data-bbox="526 280 783 324">Change Password</div> <div data-bbox="526 322 705 365">Switch User</div> <div data-bbox="526 362 638 409">Logout</div>
--	---

Generic scripts relating to searches across the web application.

```
function userQuickSearch(searchInputId, searchResultsId, format) {
    searchInputDOM = document.getElementById(searchInputId)
    query = searchInputDOM.value
    console.log(query)
    if(query) {
        var xmlhttp = new XMLHttpRequest();

        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                updateResults(JSON.parse(this.responseText), searchInputId, searchResultsId, format)
            }
        };
        xmlhttp.open("GET", endpoint(format)+query, true);
        xmlhttp.send();
    } else {
        updateResults([], searchInputId, searchResultsId, format);
    }
}

function updateResults(results, searchInputId, searchResultsId, format) {
    searchResultsDOM = document.getElementById(searchResultsId)
    console.log(searchResultsDOM)
    console.log(searchResultsId)

    var resultHTML = '';
    var i;
    for (i = 0; i < results.length; i++) {
        resultHTML += searchFormat(results[i], format, searchInputId, searchResultsId);
    }

    searchResultsDOM.innerHTML = resultHTML
}

function searchResult(result, searchInputId, searchResultsId) {
    searchInputDOM = document.getElementById(searchInputId)
    searchInputDOM.value = result

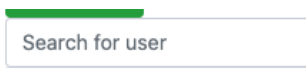
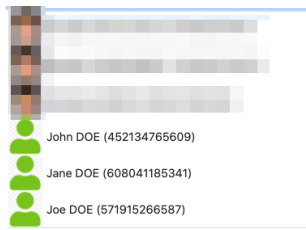
    searchResultsDOM = document.getElementById(searchResultsId)
    searchResultsDOM.innerHTML = ''

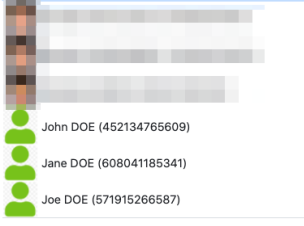
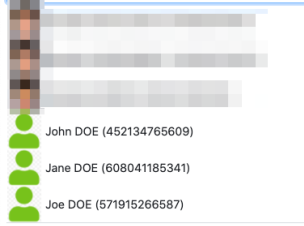
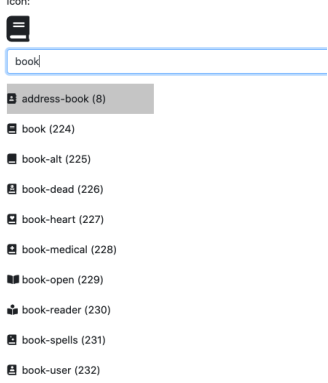
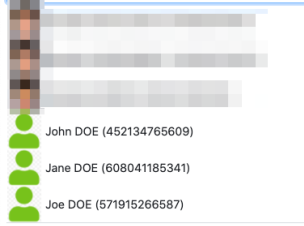
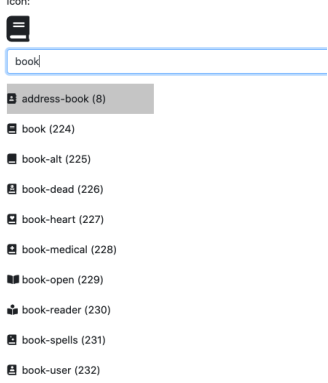
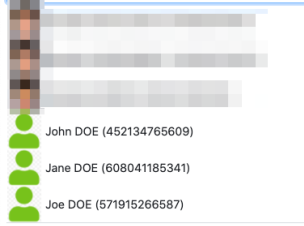
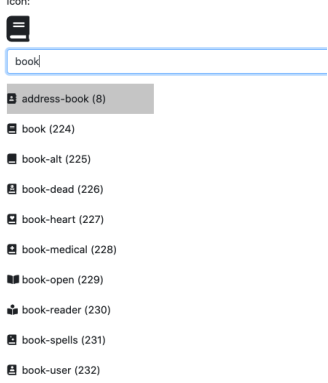
    searchInputDOM.dispatchEvent(new Event('change'));
}

function searchFormat(result, format, searchInputId, searchResultsId) {
    if(format == 'user') {
        return "<tr onclick=\"searchResult('"+result.id+"','"+searchInputId+"','"+searchResultsId+"')\" class=\"user\"><td><img src=\"/img/user/\"+result.profile_image+\"\" height=50></td><td>"+result.preferred_name+" "+(result.last_name).toUpperCase()+" (" +result.id+")</td></tr>";
    }
    if(format == 'icon') {
        return "<tr onclick=\"searchResult('"+result.name+"','"+searchInputId+"','"+searchResultsId+"')\" class=\"user\"><td><i class=\"fas fa-\"+result.name+\"\"></i></td><td>"+result.name+" (" +result.id+")</td></tr>";
    }
}

function endpoint(format) {
    if(format == 'user') {
        return "/management/users/quicksearch?query="
    }
    if(format == 'icon') {
        return "/management/icons/quicksearch?query="
    }
}
```

search.js consists of five main functions that are used site-wide:

userQuickSearch(searchInputId, searchResultsId, format)	Three parameters are passed into the function:		
	searchInputId	The HTML identifier attribute for the search input box	
	searchResultsId	The HTML identifier attribute for the search results table where results from the back-end quick search will be printed.	
	<p>The format parameter is used to indicate what type of search is required for the input box.</p> <p>The searchInputDOM element is retrieved using the document.getElementById function within JavaScript and stored as a local variable for further manipulation.</p> <p>The search query is retrieved using the value property on the searchInputDOM</p> <p>A XMLHttpRequest is used to make an asynchronous request to the server without refreshing the page. The endpoint is retrieved by passing the required format into the endpoint function and appending the query to the end of the search URL as a 'GET' parameter.</p> <p>When a response is received from the server, the results are parsed as JSON and forwarded onto the updateResults function with the additional context specific parameters forwarded too.</p> <p>If the user does not enter any value then the updateResults function will be called with an empty array causing the results table to be cleared preventing a listing of every object.</p>		
updateResults(results, searchInputId, searchResultsId, format)	<p>This function is called to update the results table with the results received from the back-end server.</p> <p>The searchResultsDOM element is stored as a local variable allowing for manipulation using the document.getElementById JavaScript function.</p> <p>A results html string is initialized before a for loop iterating over each result within the results object is completed.</p> <p>Within the iteration loop, for each individual result a searchFormat function is performed to format the HTML in the required way for display to the end user.</p> <p>Once the loop is complete the results table HTML (innerHTML) is updated to</p>		

	<p>reflect the results:</p>  <p>John DOE (452134765609) Jane DOE (608041185341) Joe DOE (571915266587)</p>				
<p>searchResult(result, searchInputId, searchResultsId)</p>	<p>A function that is added as an onclick (function will be called when this specific element is clicked) attribute to each search result.</p> <p>Clears both the searchInputDOM element and the searchResultsDOM element as the search is now complete.</p> <p>Dispatches a change event to the search input box which can be utilised by an event handler in any other JavaScript code related to a page.</p>				
<p>searchFormat(result, format, searchInputId, searchResultsId)</p>	<p>Function used to format the HTML for each result.</p> <p>There are currently two formats:</p> <table border="1"> <tr> <td>User</td><td>  <p>John DOE (452134765609) Jane DOE (608041185341) Joe DOE (571915266587)</p> </td></tr> <tr> <td>Icon</td><td> <p>icon:</p>  <p>address-book (8) book (224) book-alt (225) book-dead (226) book-heart (227) book-medical (228) book-open (229) book-reader (230) book-spells (231) book-user (232)</p> </td></tr> </table> <p>Each format has different requirements such as font rendering, profile pictures and capitalization. These can all be formatted within the searchFormat function allowing for reuse of the core search functions and adaptations being reflected within all searches instead of bespoke functions for each search.</p>	User	 <p>John DOE (452134765609) Jane DOE (608041185341) Joe DOE (571915266587)</p>	Icon	<p>icon:</p>  <p>address-book (8) book (224) book-alt (225) book-dead (226) book-heart (227) book-medical (228) book-open (229) book-reader (230) book-spells (231) book-user (232)</p>
User	 <p>John DOE (452134765609) Jane DOE (608041185341) Joe DOE (571915266587)</p>				
Icon	<p>icon:</p>  <p>address-book (8) book (224) book-alt (225) book-dead (226) book-heart (227) book-medical (228) book-open (229) book-reader (230) book-spells (231) book-user (232)</p>				
<p>endpoint(format)</p>	<p>Returns the endpoint corresponding to search format.</p> <p>There are currently two endpoints:</p> <table border="1"> <tr> <td>User</td><td>/management/users/quicksearch?query=</td></tr> </table>	User	/management/users/quicksearch?query=		
User	/management/users/quicksearch?query=				

	Icon	/management/icons/quicksearch?query=

This script is embedded within the Edit Route template and helps provide a dynamic user interface.

```
class UserInterface {
  constructor(mapElementId, stopListElementId, stopCountId) {
    this.stops = [];
    this.stopIdNewLocation;
    this.mapElementId = mapElementId;
    this.stopListElementId = stopListElementId;
    this.stopCountId = stopCountId;
    this.startNode = undefined;
    this.endNode = undefined;
    this.routeMapLine = undefined;

    this.stopMap = L.map(mapElementId).setView([51.505, -0.09], 13);

    L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}', {
      attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, Imagery &copy; <a href="https://www.mapbox.com/">Mapbox</a>',
      maxZoom: 18,
      id: 'mapbox/streets-v11',
      tileSize: 512,
      zoomOffset: -1,
    }).addTo(this.stopMap);

    this.stopMap.on('click', function(e) {
      MainInterface.mapClick(e);
    });

    render() {
      this.renderStopsList();
      this.renderMapMarkers();
    }

    mapClick(e) {
      if(this.stopIdNewLocation || this.stopIdNewLocation == 0) {
        var clickLocation = e.latlng;
        this.stops[this.stopIdNewLocation].lat = clickLocation.lat;
        this.stops[this.stopIdNewLocation].lng = clickLocation.lng;
        this.stopIdNewLocation = undefined;
        this.render();
      }
    }

    renderMapMarkers() {
      var i;
      for (i = 0; i < this.stops.length; i++) {
        var busStop = this.stops[i];
        if(busStop.marker) {
          this.stopMap.removeLayer(busStop.marker);
        }
        if(busStop.lat) {
          busStop.marker = L.marker([busStop.lat, busStop.lng]).bindTooltip(busStop.name).addTo(this.stopMap);
        }
      }
    }

    renderStopsList() {
      document.getElementById(this.stopCountId).innerHTML = this.stops.length;
      var tableHtml = ''
      var i;
      for (i = 0; i < this.stops.length; i++) {
        var busStop = this.stops[i];
        tableHtml += '<tr><td><h3><input
onchange="MainInterface.updateStopName(\'stopName'+i+'\', '+i+')" name="stopName'+i+'" class="form-control" id="stopName'+i+'" value="'+busStop.name+'"> <button onclick="MainInterface.deleteStop('+i+')" type="button" class="btn btn-danger"><i class="fas fa-times"></i></button> <button type="button" class="btn btn-success"><i onclick="MainInterface.locationSelection('+i+')" class="fas fa-map-marked-alt"></i></button> <button type="button" class="btn btn-success"><i
onlick="MainInterface.markStartNode('+i+')" class="fas fa-flag"></i></button> <button type="button" class="btn btn-warning"><i onclick="MainInterface.markEndNode('+i+')" class="fas fa-crown"></i>
</button></h3></td></tr><td>mi - ??:??</td></tr>'
        if(i == this.stopIdNewLocation) {
          tableHtml += '<tr><td>Click on map to place stop marker.</td></tr>';
        }
        if(i == this.endNode) {
          tableHtml += '<tr><td>This stop is the end node.</td></tr>';
        }
      }
    }
  }
}
```



```
if(i == this.startNode) {
    tableHtml += '<tr><td>This stop is the start node.</td></tr>';
}
if(busStop.lat) {
    tableHtml += '<tr><td>Location: '+busStop.lat+ ', ' + busStop.lng+'</td></tr>';
}
}

document.getElementById(this.stopListElementId).innerHTML = tableHtml;
}

addStop() {
    var newStop = new Stop('Stop '+this.stops.length, undefined, undefined, undefined);
    this.stops.push(newStop);
    this.locationSelection(this.stops.length-1);
}

deleteStop(id) {
    if(this.stops[id].marker) {
        this.stopMap.removeLayer(this.stops[id].marker);
    }
    var newStops = [];
    var i;
    for (i = 0; i < this.stops.length; i++) {
        var busStop = this.stops[i];
        if(i != id) {
            newStops.push(busStop);
        }
    }
    this.stops = newStops;
    this.render();
}

updateStopName(inputId, stopId) {
    var newName = document.getElementById(inputId).value;
    this.stops[stopId].name = newName;
    this.render();
}

locationSelection(id) {
    this.stopIdNewLocation = id;
    this.render();
}
```

```

markStartNode(id) {
    this.startNode = id;
    this.render();
}

calculateRoute() {
    var stopJson = {};
    var i;
    for (i = 0; i < this.stops.length; i++) {
        stopJson[i] = this.stops[i];
        if(i == this.startNode) {
            stopJson["start"] = i
        }
        if(i == this.endNode) {
            stopJson["end"] = i
        }
        stopJson[i].marker = undefined;
    }
    console.log(stopJson)
    var xmlhttp = new XMLHttpRequest();

    xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var optimalRouteOrder = (JSON.parse(this.responseText));

            var optimalRouteLatLngs = [];
            for (i = 0; i < optimalRouteOrder.length; i++) {
                var stopId = parseInt(optimalRouteOrder[i]);
                console.log(stopId)
                console.log(MainInterface.stops[0])
                var stopObject = MainInterface.stops[stopId];

                console.log(parseFloat(stopObject.lat))

                optimalRouteLatLngs.push(new L.LatLng(parseFloat(stopObject.lat),
parseFloat(stopObject.lng)))
            }

            console.log(optimalRouteLatLngs)
            if(MainInterface.routeMapLine) {
                MainInterface.stopMap.removeLayer(MainInterface.routeMapLine);
            }
            MainInterface.routeMapLine = new
L.polyline(optimalRouteLatLngs).addTo(MainInterface.stopMap);

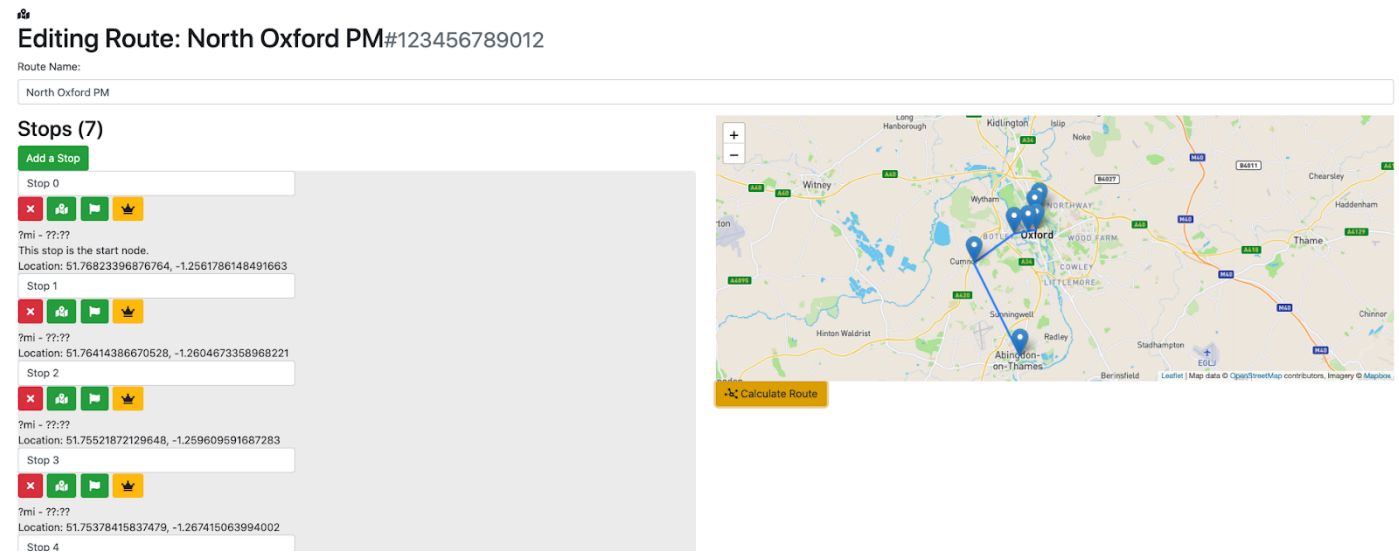
        }
    };
    xmlhttp.open("POST", '/routing/route-mapping/calculate', true);
    xmlhttp.send(JSON.stringify(stopJson));
}

}

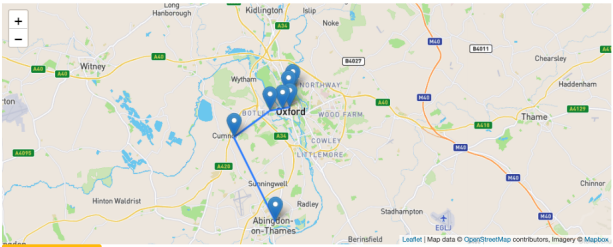
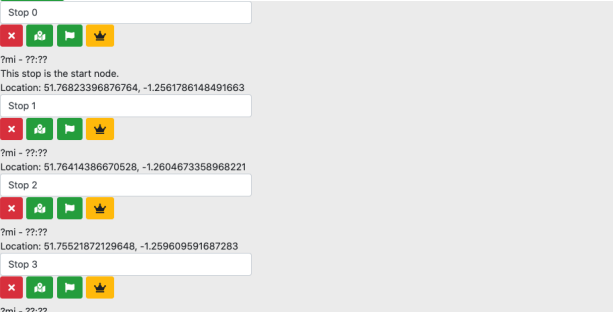
class Stop {
    constructor(name, time, lat, lng, userInterface) {
        this.name = name;
        this.time = time;
        this.lat = lat;
        this.lng = lng;
        this.userInterface = userInterface;
        this.marker;
    }
}

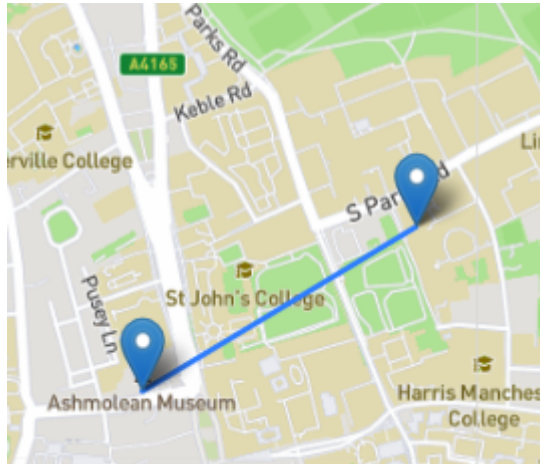

MainInterface = new UserInterface('stopMap', 'stopList', 'stopCount');
```


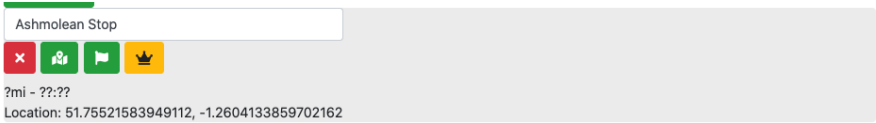
The page when rendered appears as the following:








There are two OOP classes and thirteen sub functions:

UserInterface	constructor(mapElementId, stopListElementId, stopCountId)	Initializes the following variables:	
		stops	An ordered array used to hold Stop objects representing the stops within the route.
		stopIdNewLocation	Integer for the currently selected stop to place a new location marker within the map.
		mapElementId	The map HTML element (a div) identifier to be used for initialization by leaflet (explained below) 
		stopListElementId	The stop list HTML element (a table) identifier used when displaying stop data to the end-user. 
		stopCountId	The stopCount HTML element (a span tag)

		identifier to update the overall stop count for display. Stops (7)
	startNode	The route start node ID.
	endNode	The route end node ID.
	routeMapLine	The map line shown once a route has been calculated. 
	stopMap	The leaflet map object for performing operations on the end-user displayed map.
	<div></div> <p>Leaflet is an open source mapping platform used by many websites. It is incredibly robust and well documented for achieving tasks which are outside the scope of my project.</p> <p>For the 'tilemap' of leaflet I have also used the popular map CDN MapBox. A tilemap is simply the 'puzzle pieces' that make up the map shown to the user.</p> <p>The stopMap element is initialized using Leaflet with the view initialised.</p> <p>A click event handler is added to the stopMap calling the MainInterface object's mapClick event. The event object is passed as a parameter.</p>	
	render	This function calls sub rendering functions that update the view for the user.
	mapClick(e)	A check is performed to check whether the interface object has a stop location to be selected using the stopIdNewLocation variable.

		<p>The click location is gathered using the event handler's event tag latlng attribute.</p> <p>The current stopIdNewLocation is passed to the stops array to select the stop object. The latitude and longitude attributes of this object are then updated to match that of the click location.</p> <p>The stopIdNewLocation is cleared to indicate the stop location has been selected.</p> <p>A render is called to reflect the changes.</p>
	renderMapMarkers	<p>An increment variable i is initialized.</p> <p>A for loop iterates over the stops array for the interface object. A busStop local variable is initialized for the iterator stop.</p> <p>If any bus marker currently exists for the stop, it is removed from the map.</p> <p>If the busStop has a latitude attribute (and therefore a longitude) a new marker is created as a local marker variable for the busStop object. A tooltip is also bound with the stop name.</p> 
	renderStopsList	<p>The stopCount span element is updated to reflect the current stops array length.</p> <p>An iterator and tableHtml variable are initialized to be updated within the following loop.</p> <p>The loop iterates over each stop within the array. A string containing the standard HTML format for a stop is appended to the tableHtml string variable:</p> <pre data-bbox="651 1675 1437 1877"><tr><td><h3><input onchange="MainInterface.updateStopName('\stopName'+i+'\','+i+')" name="stopName'+i+'" class="form-control" id="stopName'+i+'" value="'+busStop.name+'"> <button onclick="MainInterface.deleteStop('+i+')" type="button" class="btn btn-danger"><i class="fas fa- times"></i></button> <button type="button" class="btn btn-success"><i onclick="MainInterface.locationSelection('+i+')" class="fas fa-map-marked-alt"></i></button> <button type="button" class="btn btn-success"><i onclick="MainInterface.markStartNode('+i+')" class="fas fa- flag"></i></button> <button type="button" class="btn btn-warning"><i onclick="MainInterface.markEndNode('+i+')" class="fas fa-crown"></i></button></h3></td></tr><tr><td>?mi - ??:??</td></tr></pre> 

			Stop name input field	On a user input change event, a call is made to the updateStopName function with the stop element ID.
			Delete stop	Calls the deleteStop function with ID as parameter
			Relocate / add location to stop	Calls the locationSelection function with ID as parameter
			Mark stop as start node	Calls the markStartNode function with ID as parameter
			Mark stop as end node	Calls the markEndNode function with ID as parameter
		<p>If the stop in the current iteration is marked as needing to have its location updated (the current stop ID is equal to the stopIdNewLocation variable) then a string is appended to the tableHtml to indicate that this is the case.</p> <p>If the stop in the current iteration is marked as the endNode, a string is appended to indicate this.</p> <p>If the stop in the current iteration is marked as the startNode, a string is appended to indicate this.</p> <p>If the busStop has a latitude attribute (and therefore a longitude) then a string is appended reflecting the stop's current longitude and latitude.</p> <p>Once the loop has completed, the tableHtml is pushed to the stopListElement using its innerHtml attribute.</p>		
	addStop	<p>A new stop object is created with the name 'Stop {{stops.length}}' i.e. for the first stop 'Stop 0'</p> <p>The new stop object is pushed into the interface stops array.</p> <p>A call to the locationSelection function is called with the stopId to allow the user to select the stop's location.</p>		
	deleteStop(id)	If the stop passed in as a parameter has a map marker. The map		

		<p>marker is removed.</p> <p>An iterator variable <i>i</i> and a newStops array is initialized.</p> <p>A loop is performed on the interface stops array. For each stop, a check is performed to see if the stop is the proposed deleted stop, if it is not then it is pushed to the newStops array.</p> <p>The newStops array will not contain the proposed deleted stop.</p> <p>The stops array will be overwritten with the newStops contents.</p> <p>The render function is called to reflect the changes within the interface.</p>
	updateStopName(inputId, stopId)	<p>A newName variable is taken using the inputId's, passed as a parameter, value.</p> <p>The stop is updated within the interface's stops array to reflect the newName provided.</p> <p>The render function is called to reflect the changes within the interface.</p>
	locationSelection(id)	<p>The stopIdNewLocation variable is updated to reflect the chosen stop value.</p> <p>The render function is called to reflect the changes within the interface.</p>
	markEndNode(id)	<p>The interface endNode variable is updated to reflect the chosen stop value.</p> <p>The render function is called to reflect the changes within the interface.</p>
	markStartNode(id)	<p>The interface startNode variable is updated to reflect the chosen stop value.</p> <p>The render function is called to reflect the changes within the interface.</p>
	calculateRoute	<p>A stopJson and iterator variable are initialized.</p> <p>A loop is performed on the interface stop array to append each stop to the stopJson. A check is also performed to see if the node is marked as a start or stop node, if it is then an additional key is added to reflect the ID of the start or stop node. To prevent a circular JSON object (cannot be sent over POST), the stop marker object is cleared in the stopJson.</p> <p>A new XMLHttpRequest object is formed to asynchronously perform a request to the server without refreshing the page. The request is sent to the '/routing/route-mapping/calculate' endpoint as a 'POST' request with the stopJson variable as the</p>

		<p>payload (formatted as a string by the JSON.stringify function within JavaScript).</p> <p>Once a response is received, a check is performed to ensure the request was successful. If the check passes, a new local optimalRouteOrder array is initialised with the response parsed as a JSON object. An optimalRouteLatLngs array is also initialised.</p> <p>A loop is performed over the optimalRouteOrder sent by the back-end flask server. For each stop, the stopId is parsed as an integer and required stopObject extracted from the local interface stops array.</p> <p>The optimalRouteLatLngs array has a new Leaflet LatLng object appended with both the latitude and longitude of the optimal stop as parameters whilst being converted to floats.</p> <p>If a routeMapLine object already exists, it is removed and replaced with a new line with the new optimalRouteLatLngs and appended to the interface map to indicate visually to the user the calculated route.</p>												
Stop	constructor(name, time, lat, lng, userInterface)	<p>Each stop object has some specific attributes that are referenced by the interface:</p> <table><tr><td>name</td><td>The stop name.</td></tr><tr><td>time</td><td>The stop time duration.</td></tr><tr><td>lat</td><td>The stop location latitude</td></tr><tr><td>lng</td><td>The stop location longitude</td></tr><tr><td>userInterface</td><td>The user interface object that this stop belongs to.</td></tr><tr><td>marker</td><td>The stop's Leaflet marker object.</td></tr></table>	name	The stop name.	time	The stop time duration.	lat	The stop location latitude	lng	The stop location longitude	userInterface	The user interface object that this stop belongs to.	marker	The stop's Leaflet marker object.
name	The stop name.													
time	The stop time duration.													
lat	The stop location latitude													
lng	The stop location longitude													
userInterface	The user interface object that this stop belongs to.													
marker	The stop's Leaflet marker object.													

Testing

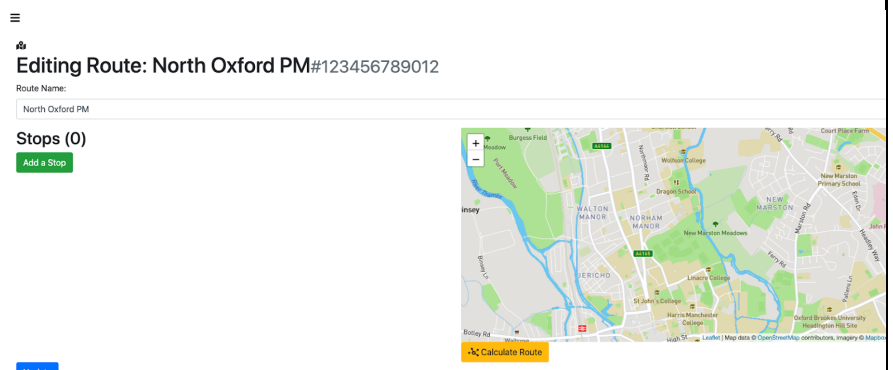
In order to ensure my program is operating correctly, further to the testing previously shown in the technical implementation, I will need to show functionality within the application is working as intended.

I have produced this short video to briefly show some of the features within the application to save multiple pages with images of testing.

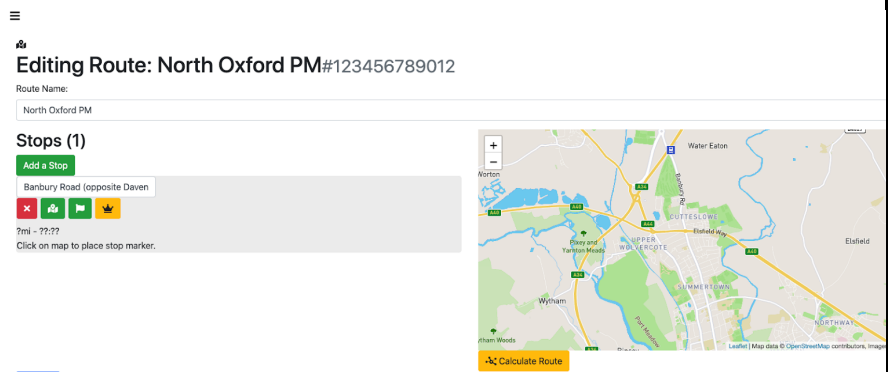
Testing Video

Testing the routing functionality within the web app

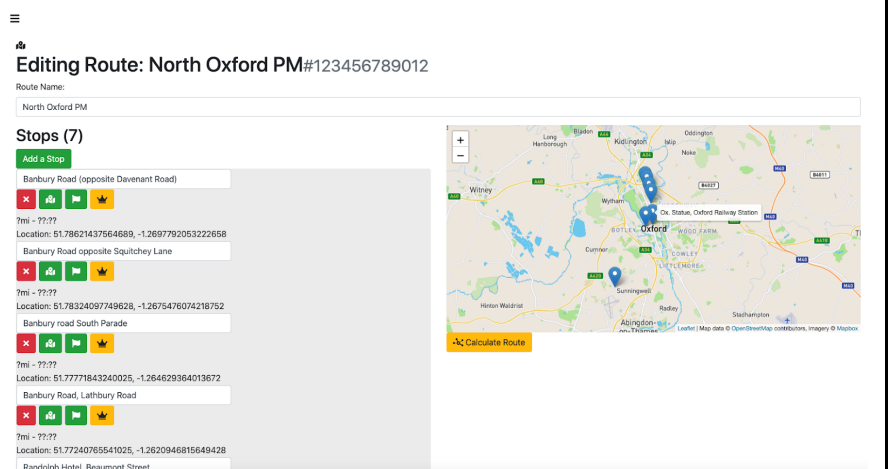
1. A route is created and a name is imputed into the route name field. The user focuses the map on the desired area to place the stops.

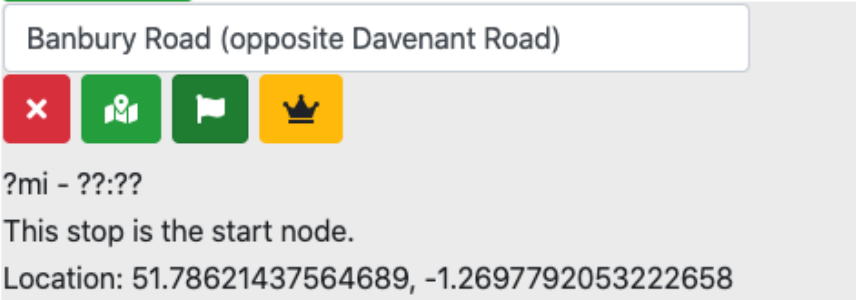
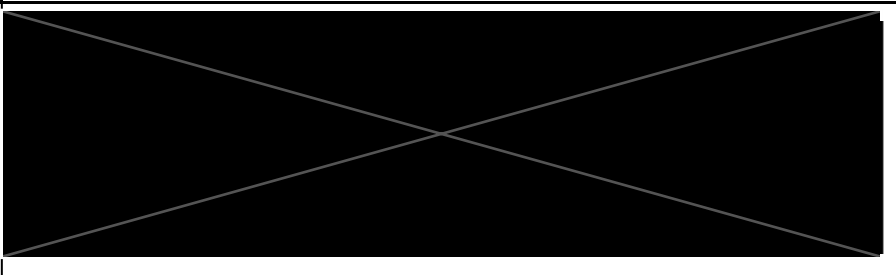
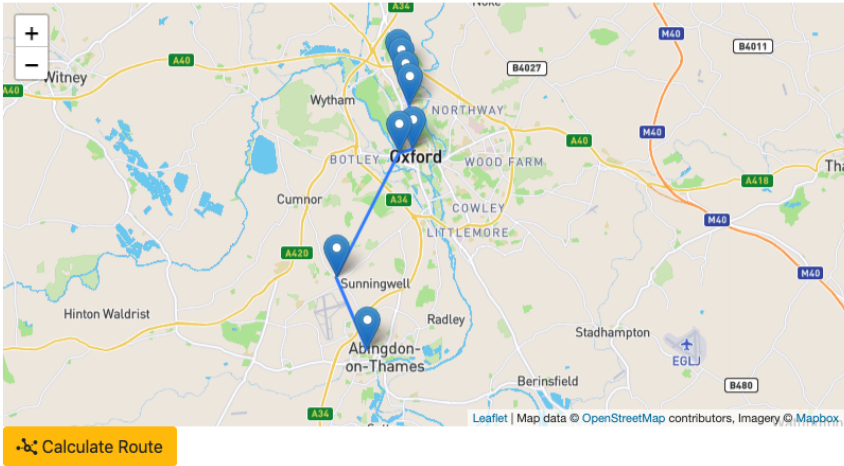


2. The user clicks the 'Add a stop' button that adds a new stop record with the delete stop, replace marker, select start stop, select end stop button.



3. The user continues to add stops increasing the stop counter and renaming each one to show the stop name to the end user.



<p>4. The user hits the start node button to indicate where the route should begin.</p>	
<p>5. The user clicks the end node button to indicate where the route should end.</p>	
<p>6. The user clicks the calculate route button to send the nodes to the backend server to perform the necessary calculations to find the most optimal route. The result is then returned via JSON to be rendered within the leaflet map.</p>	
<p>Conclusion The results show that the system is working as intended.</p>	

I faced a few issues within the development of my application. I have selected a few examples to demonstrate how I resolved issues after testing concluded that a bug was present.

Mapping Matplotlib Issues

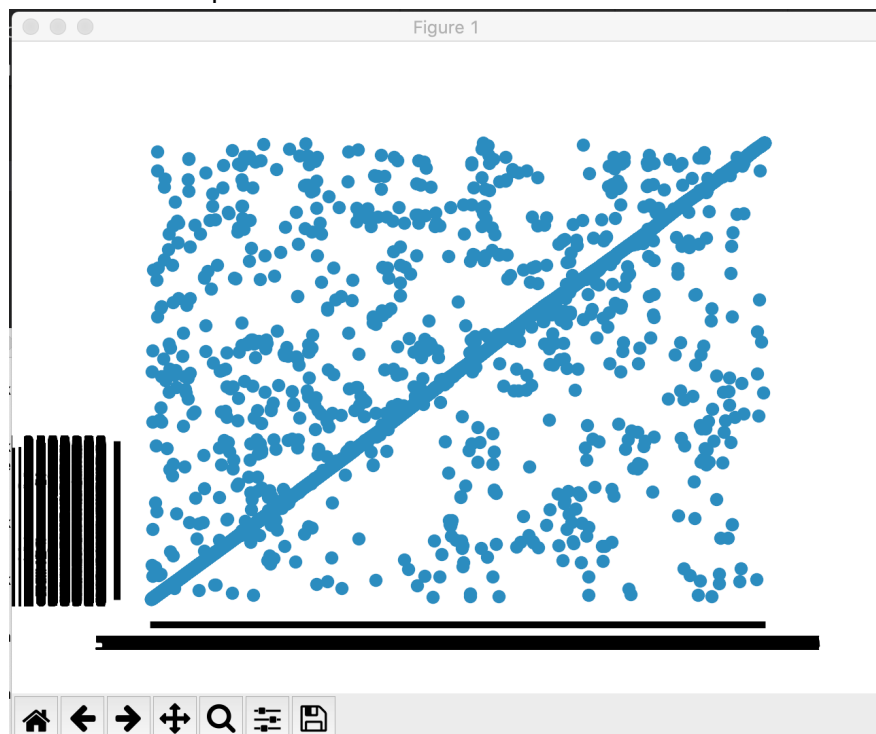
I experience a perplexing problem when attempting to map the nodes within an OSM file in Matplotlib. I first started by loading a XML file using python's XML library. I read the ElementTree as a string and then placed each node into two arrays mapX and mapY to be plotted within a scatter plot within matplotlib using numpy.

```
import xml.etree.ElementTree as ET
import matplotlib.pyplot as plt
import np

with open('map_file.xml') as map_file:
    ''' Reads OpenStreetMap osm xml file and plots longitude and latitude using matplotlib. '''
    map_xml_root = ET.fromstring(map_file.read())
    mapX = []
    mapY = []
    for child in map_xml_root:
        if(child.tag == 'node'):
            mapX.append(child.attrib['lat'])
            mapY.append(child.attrib['lon'])
    np.array([mapX, mapY])

plt.scatter(mapX, mapY)
plt.show()
```

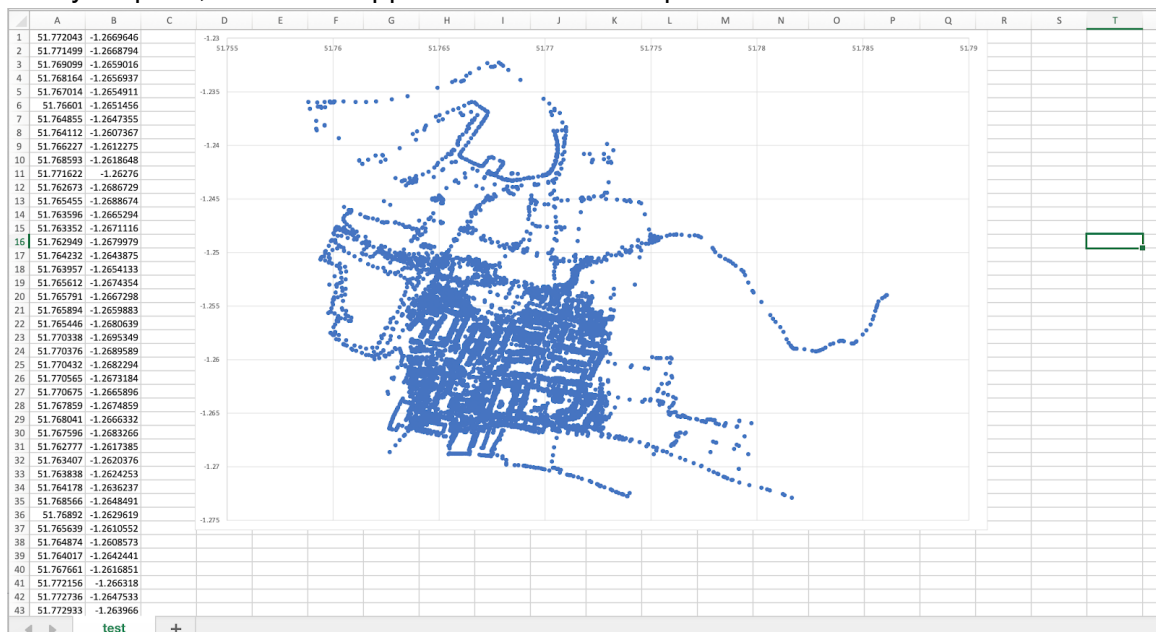
However, instead of achieving the desired result showing a map of Oxford instead a sporadic pattern was shown with a line across the center. This issue seemed very bizarre and I attempted to rewrite the code in a different way to see if it would help however the issue still reoccurred.



I continued to struggle with getting the map to appear correctly within Matplotlib so I decided to see what would happen if I exported the data as a CSV file.

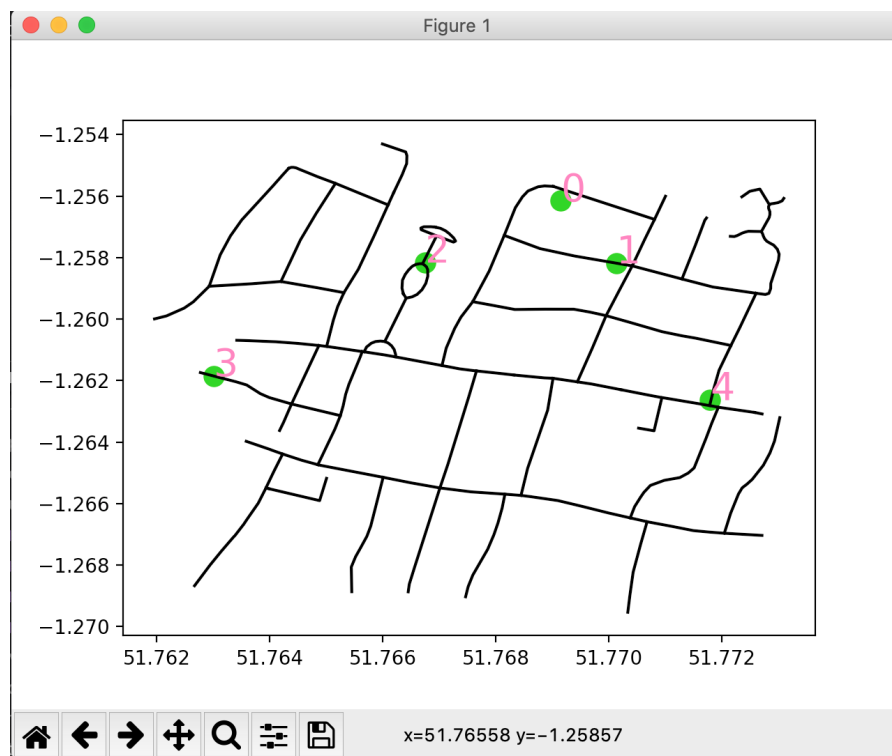
```
...
with open('test.csv', 'w') as file2:
    for i in range(len(mapX)):
        file2.write(f'{mapX[i]}, {mapY[i]}'+'\n')
```

However to my surprise, the data mapped fine as a scatter plot within



After many attempts at trying to fix the problem I soon realised that Matplotlib was reading the values given as literal strings instead of numerical values and therefore was unable to interpret the data given. With a very simple float conversion the issue was fixed!

```
def loadMapFromFile(self):
    with open('assets/OX26SS_SmallOpenStreetMap.osm') as map_file:
        map_xml_root = ET.fromstring(map_file.read())
        for child in map_xml_root:
            if(child.tag == 'node'):
                ''' Variables now updated to explicitly read the xml attribute as a float rather
                than a string as before. '''
                lat = (float(child.attrib['lat']))
                long = (float(child.attrib['lon']))
        ...
```



JSON Malformed

Upon adding a new user during development of the system. I was greeted with an internal server error page from flask.

← → ↻ ⓘ localhost:5000/management/users/

Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

I proceeded to look at the console to see what could be causing the problem.

```
return self.view_functions[rule.endpoint](**req.view_args)
File "/Users/fnicholson/PycharmProjects/Freddie-Nicholson-CS-NEA-2020-2021/BusWebApp/auth.py", line 55, in wrapped_view
return view(**kwargs)
File "/Users/fnicholson/PycharmProjects/Freddie-Nicholson-CS-NEA-2020-2021/BusWebApp/management.py", line 132, in usersIndex
usersJSON = db.executeQueryReturnJSON('SELECT * FROM USERS LEFT JOIN org_users ON org_users.user_id=users.id LEFT JOIN orgs ON orgs.id=org_users.org_id FOR JSON AUTO;')
File "/Users/fnicholson/PycharmProjects/Freddie-Nicholson-CS-NEA-2020-2021/BusWebApp/db.py", line 29, in executeQueryReturnJSON
return json.loads(resp[0])
File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/json/__init__.py", line 357, in loads
return _default_decoder.decode(s)
File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/json/decoder.py", line 337, in decode
obj, end = self.raw_decode(s, idx=_w(s, 0).end())
File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/json/decoder.py", line 353, in raw_decode
obj, end = self.scan_once(s, idx)
json.decoder.JSONDecodeError: Unterminated string starting at: line 1 column 2029 (char 2028)
127.0.0.1 - - [26/Dec/2020 09:00:02] "GET /management/users/ HTTP/1.1" 500 -
```

This console output indicates that the program cannot decode the JSON as it was malformed. This surprised me as I was using Microsoft SQL Server to provide the JSON response. I could not tell immediately what the issue was so I decided the best course of action would be to see what is returned in the query in DataGrip.

	JSON_F52E2B61-18A1-11d1-B105-0...
1	[{"id": "371883267944", "given_name": "F...
2	id": "940175646927", "orgs": [{"id": "940...

I Immediately knew the problem. I had forgotten that SQL often has a hard limit on the length of a piece of data returned, in this case for Microsoft SQL Server, 8000 characters. The densely populated cell of JSON had become too long that SQL Server returned a response with two rows. This was a systematic error present in quite a few locations so it was good that I caught it early on!

The update fortunately due to my OOP approach in flask meant I only had to update a few lines of code in one place.

```
//db.py

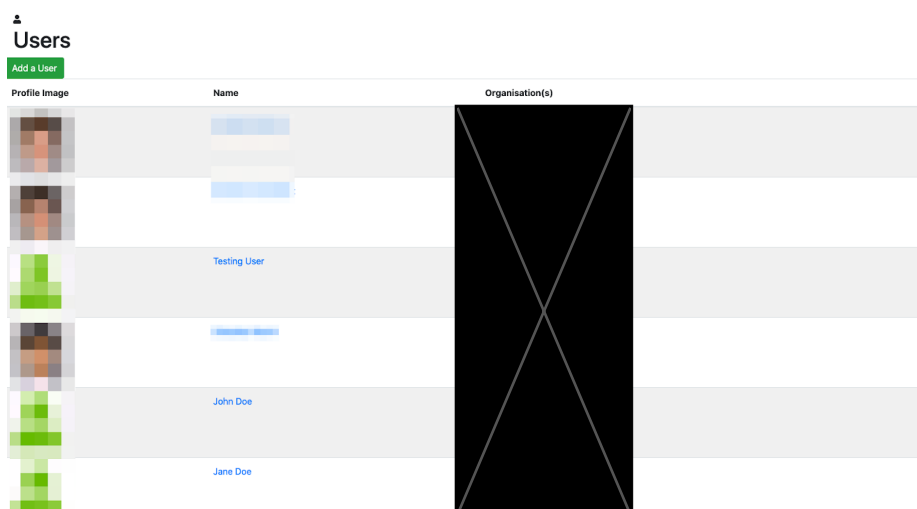
def executeQueryReturnJSON(query):
    db = get_db()
    cursor = db.cursor()
    cursor.execute(query)
    resp = cursor.fetchone()
    if(resp is not None):
        return json.loads(resp[0])
    else:
        return {}
```

Simply rewriting some of the database retrieval logic to load each row into a string and loading that as JSON fixed the problem in all areas where this function was used.

```

//db.py

def executeQueryReturnJSON(query):
    db = get_db()
    cursor = db.cursor()
    cursor.execute(query)
    retJson = ''
    for row in cursor:
        retJson += str(row[0])
    if(retJson is not ''):
        print(retJson)
        return json.loads(retJson)
    else:
        return {}
    
```



This simple change within the **db.py** database helper module shows how my usage of DRY (don't repeat yourself) makes significant changes much easier to make.

Evaluation

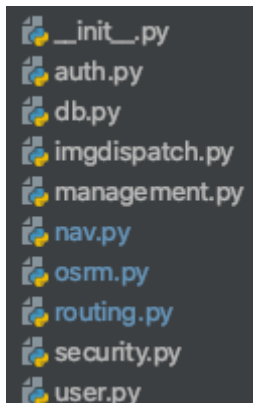
The conclusion of my NEA comes with the evaluation of the system design. Since the documented design section I have realised the large undertaking my project has been and therefore I am now in the position to reflect on what I could have done differently to improve the system and what designs worked well.

I will also talk to the end-users aforementioned at the beginning of the report to see what they think of the prototype system.

Objectives

Identifier	Objective	Met?
1	The system should be built around OOP principles	✓
I managed to achieve this objective throughout the entire program. Each module housed the subroutines required for a specific purpose. This allowed for much easier code readability. One problem I did		

encounter was deciding which modules to place certain subroutines in when they were generic i.e. needing to be used by other modules.



1.1

This should be achieved using Python modules named after each module with a selection of subroutines to match the modules purpose



I achieved this throughout each module with a set of subroutines that all succinctly worked together to perform a common purpose as a module.

```
@bp.route('/organisations/delete/<int:id>/')
@auth.login_required
def deleteOrg(id):
    orgsJSON = db.execute(f'DELETE FROM orgs WHERE id = \'{id}\';')
    flash(f'Organisation ({id}) deleted.', 'success')
    return redirect('/management/organisations')

@bp.route('/organisations/view/<int:id>/')
@auth.login_required
def viewOrg(id):
    orgsJSON = db.executeQueryReturnJSON(f'SELECT * FROM orgs WHERE id = \'{id}\';')
    orgMembersJSON = db.executeQueryReturnJSON(f'SELECT * FROM org_users INNER JOIN users ON org_users.user_id = users.id WHERE org_id = \'{id}\';')
    return render_template('/management/organisation/vieworganisation.html', nav=nav.nav, org=orgsJSON[0], orgMembers=orgMembersJSON)

@bp.route('/organisations/edit/<int:id>/', methods=['GET', 'POST'])
@auth.login_required
def editOrg(id):
    if(request.method == 'GET'):
        orgsJSON = db.executeQueryReturnJSON(f'SELECT * FROM orgs WHERE id = \'{id}\';')
        return render_template('/management/organisation/editorganisation.html', nav=nav.nav, org=orgsJSON[0])
    if(request.method == 'POST'):
        if request.files['logo']:
            filename = secure_filename(request.files['logo'].filename.split('.')[0])
            request.files['logo'].save(os.path.join(current_app.config['UPLOAD_FOLDER'], 'orglogos', filename))
        else:
            filename = 'default.png'
        db.execute(f'UPDATE orgs SET id = \'{request.form["id"]}\', type = {request.form["type"]}, logo = \'{filename}\', name = \'{request.form["name"]}\';')
        flash(f'Organisation {request.form["name"]} ({request.form["id"]}) updated.', 'success')
        return redirect(f'/management/organisations/view/{request.form["id"]}/')
```

1.2

Flask will be used to provide the framework for modules using Blueprints



Each module was registered as a module within the flask initialisation file `__init__.py`.


```

from . import imgdispatch
app.register_blueprint(imgdispatch.bp, url_prefix='/img')

from . import management
app.register_blueprint(management.bp, url_prefix='/management')

from . import auth
app.register_blueprint(auth.bp, url_prefix='/auth')

from . import user
app.register_blueprint(user.bp, url_prefix='/user')

from . import security
app.register_blueprint(security.bp, url_prefix='/security')

from . import routing
app.register_blueprint(routing.bp, url_prefix='/routing')

```

1.3

Each application featureset should be combined into a flask blueprint module with all subroutines required for each module inside its own file allowing for easier code readability.

✓

Each blueprint is registered within its own module file and subroutines are added into the application context.

```

bp = Blueprint('security', __name__)

def generateNewTwelveDigitId(tableName):
    testId = random.randint(111111111111, 999999999999)
    respJson = db.executeQueryReturnJSON(f'SELECT * FROM {tableName} WHERE id = \'{testId}\'' FOR JSON AUTO;')
    if(respJson):
        return generateNewTwelveDigitId(tableName)
    else:
        return testId

@bp.route('/access-levels/')
@auth.login_required
def accessLevels():
    accessLevelsJSON = db.executeQueryReturnJSON(f'SELECT * FROM access_level FOR JSON AUTO;')
    return render_template('/security/accesslevels.html', nav=nav.nav, accessLevelsJSON=accessLevelsJSON)

@bp.route('/access-levels/add')
@auth.login_required
def addAccessLevel():
    accessLevelId = generateNewTwelveDigitId('access_level')
    db.execute(f'INSERT INTO access_level (id, icon) VALUES (\'{accessLevelId}\', \'user-lock\')')
    return redirect(f'/security/access-levels/edit/{accessLevelId}')

```

1.4

Each module should interface with helper functions such as authentication to verify a user has the required access for a feature.

✓

Each subroutine that requires authentication has the **auth.login_required** decorator which only allows the page to be accessed if the user is logged in. If the user is not logged in then a login page is returned.

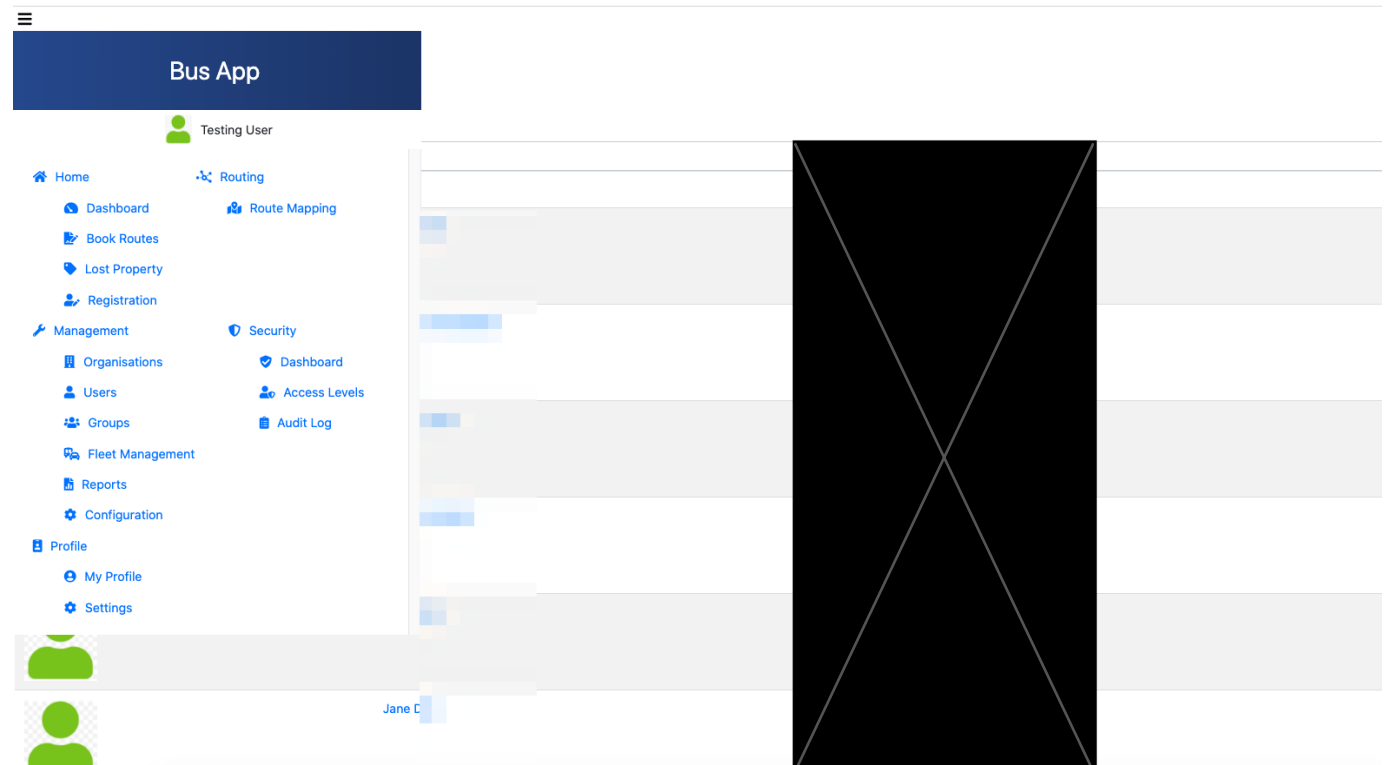
```
@bp.route('/fleetmanagement/add/', methods=['GET', 'POST'])
@auth.login_required
def addVehicle():
```

1.5

Modules should be presented graphically to the user within the interface.



Modules within the application are represented using HTML, Javascript and CSS to the end user. With a navigation bar for each module and interactive elements within them allowing the user to perform certain actions.



2

The system should utilise a database for CRUD operation on records



In most instances, a record is able to be easily created, read, updated and destroyed using the object interface.

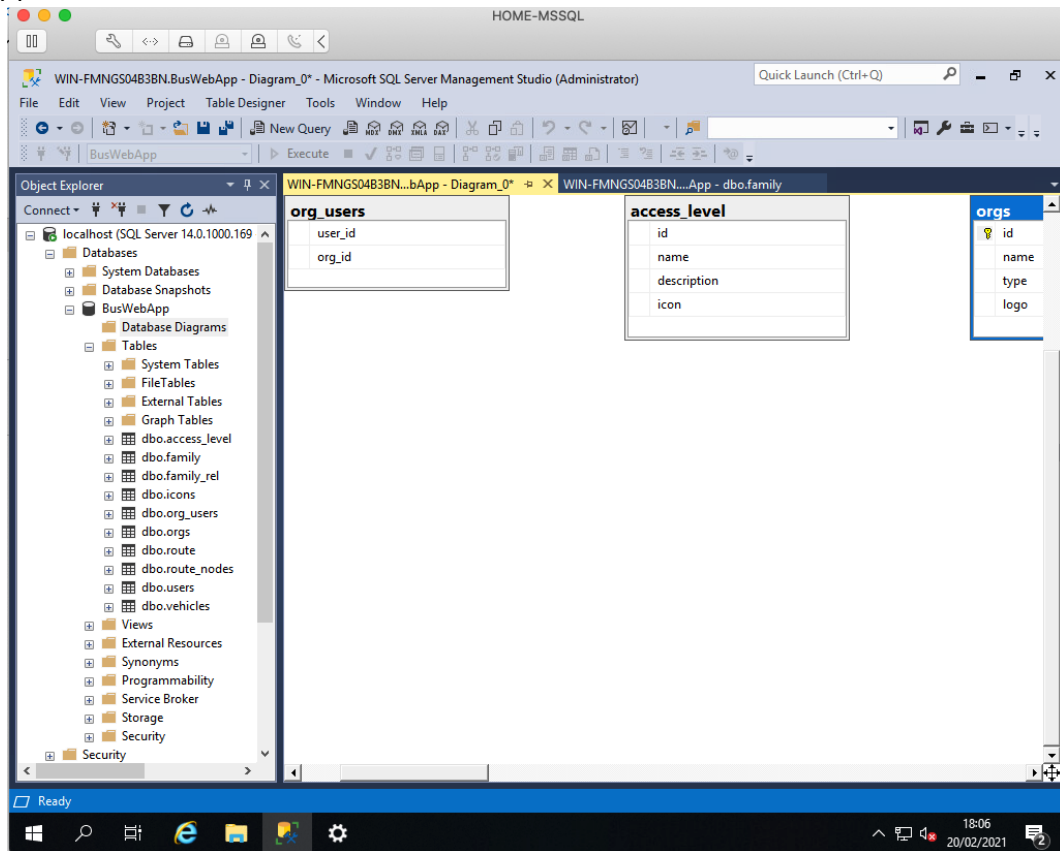


2.1

The system should be set up to interface with a Microsoft SQL Server database in line with standard educational software.



Microsoft SQL Server is running on a Windows Server virtual machine allowing for remote access from a client on the same network. All application data is stored within the database improving security across the entire application.



2.2

An ODBC connection should be used to connect the database to the flask web server using a connection socket for each session.



There is a database management module within the application allowing for easy connection to the database and results to be returned as JSON for easy manipulation within each subroutine.

2.3

The database should be split into multiple tables following 3NF



3rd Normal Form means that each column in each row is only dependent on columns that are part of the primary key. The data in each column in each row is dependent on each row's primary key. Each row has columns that represent only a single element.

An example of a 3NF table within the application is the **family_rel** table:

	id	family_id	user_id	rel
1	510396973957	611417741472	452134765609	1
2	583941078236	611417741472	571915266587	0
3	866963752479	611417741472	608041185341	1

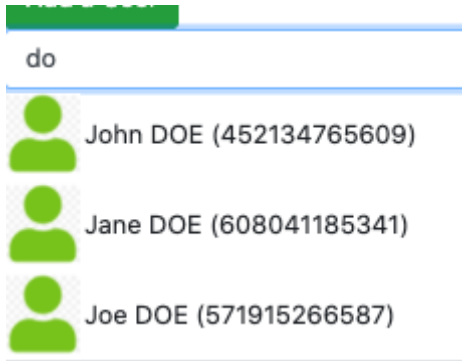
Each column only contains a single element of data. There is a primary key for each record and a ID is given for any foreign key to retrieve the relevant record.

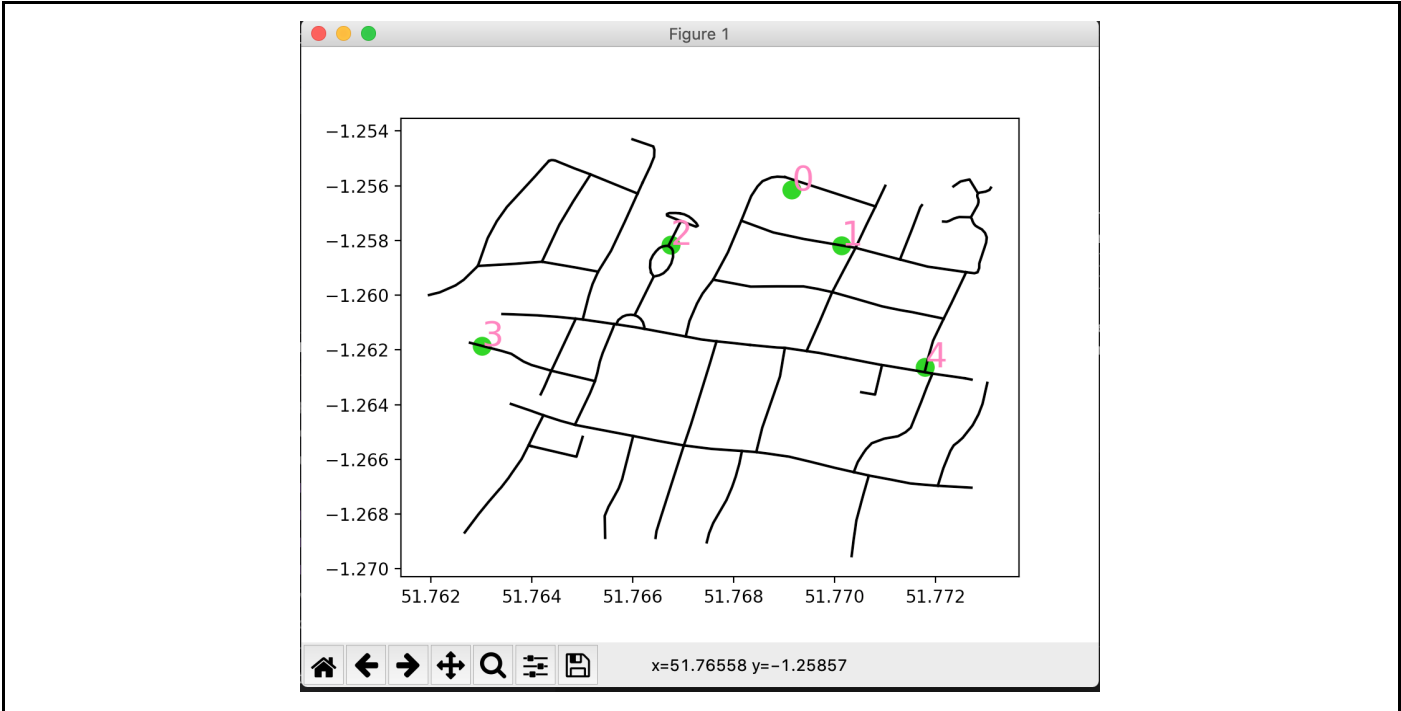
3

The system should use Google OR Tools



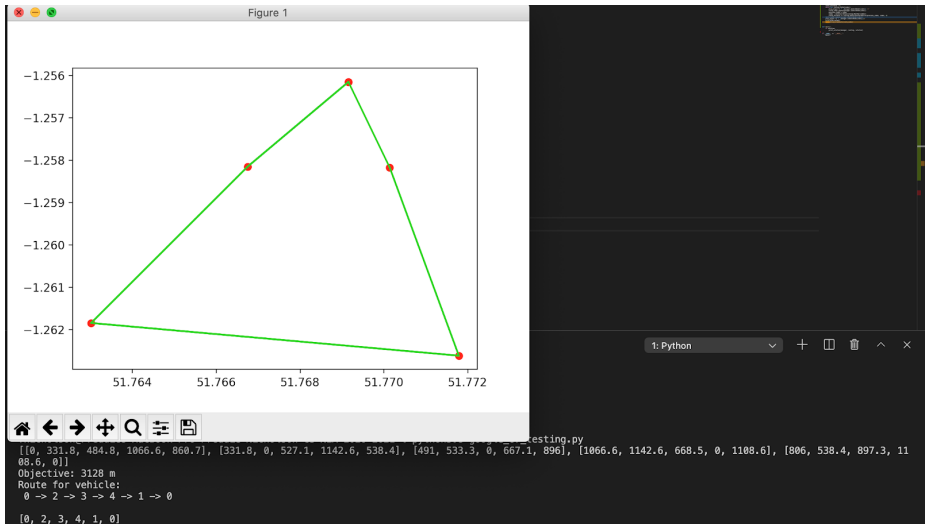
OR Tools was implemented during testing and implemented in the final application which is shown within the Technical development section.		
3.1	The Google OR Tools Vehicle Routing functions should be utilised in conjunction with OSRM	✓
OR Tools was used in conjunction with a distance matrix generated from OSRM. The distance matrix was generated by iterating over a set of nodes to calculate the shortest distance using OSRM. These distances were then fed into the vehicle routing algorithm with OR Tools to produce a final result.		
3.2	OSRM should be setup on a local server to reduce rate limiting	✓
OSRM is set up on a linux virtual machine with the local hostname osrm allowing for quicker queries to the OSRM backend.		
3.3	Custom functions will need to be written in order to perform operations including: <ul style="list-style-type: none"> • Building the distance matrix for the solver • Transforming the solution to a readable format for the frontend interface • Visualising data within Matplotlib to check the system is operating correctly 	✓
The distance matrix generator function works fully as intended using the local OSRM backend. The data is successfully transformed into a human readable format within the leaflet map on the frontend interface. Matplotlib was also used successfully to convey the routing engine.		
4	The system should use a web interface	✓
There is a web interface served by flask. Templates are stored within the project directory and many use liquid templating / JavaScript to serve dynamic content. The pages use bootstrap to help make the styling of the website quicker as I needed to create a lot of pages.		
4.1	The user interface should be modern and follow up to date web design principles	✓
The pages are designed using modern HTML5 standards. Each page also uses custom styling to depict different elements in a way that comes natural to the user. Although I feel I could have done more to improve the styling of each page, unfortunately this would take too much time and I feel I would not gain as much from the project if I was to focus solely in this area. However some of the website I do feel could do with a design improvement however I felt this was beyond the scope of my NEA and would be a further topic of development.		
4.2	Input fields for objects should utilise a 'quick search' feature allowing for easy lookup	✓
There are several instances of a quick search throughout the entire app. When a user types characters into a field a list of matching objects is returned allowing for easy selection without the user using unfriendly ID numbers.		

		
5	Development Model	✓
<p>I successfully programmed a development model to help show the routing algorithm that would be used in the final application. This was achieved using Matplotlib, Google OR Tools, OpenStreetMap and OSRM.</p>		
5.1	A testing system should be developed using an OOP model for developing the optimisation algorithm	✓
<p>Each element of OpenStreetMap was represented as a class within map.py allowing for quick and easy loading of the OSM (XML) file into the application and OOP methods to be used.</p> <pre> class Node: """ A node in openstreetmap. Latitude and Longitude are stored as well as whether the node is a junction (intersection). """ def __init__(self, nId: int, lat: float, long: float): self.nId = nId self.lat = lat self.long = long self.roadsConnected = [] self.intersection = False def __repr__(self): """ Returns a representation of the node in string form with its unique ID. :return: """ return f'Node({self.nId})' class Way: """ A way in openstreetmap consisting of a list of nodes. Various attributes are also stored with metadata about the way. """ def __init__(self, wId: int): </pre>		
5.2	Matplotlib should be used alongside OpenStreetMap line data using filters to only select the required drivable paths.	✓
<p>Matplotlib is used to draw the paths within the plot. Each way is read into a way object. Each way is then rendered with a filter checking if the road is drivable by having the 'highway' flag set to something that is typically drivable allowing for all the roads to be displayed.</p>		



5.3	Testing environment should be integrated with OR Tools to visualise the routing algorithm	✓
-----	---	---

The testing environment used OR Tools in conjunction with OSRM to allow for routes to be displayed within Matplotlib.





6	Login system	✓
---	--------------	---

A login system is present.

6.1	A login system should be present to prevent unauthorised access to the site	✓
-----	---	---

When a user attempts to access the site they are immediately greeted with a login page requesting they sign in with relevant error catching in place such as user not found.

<div><h3>Login</h3><div>User Email:<div></div></div><div>User Password:<div></div></div><div>Login</div></div>						
6.2	A authentication decorator should be present to allow for restricted access to certain pages	✓				
<div>Any protected page visited while a user is not logged in is automatically redirected to the login page.</div> <div><pre>def login_required(view): @functools.wraps(view) def wrapped_view(**kwargs): if g.user is None: return redirect('/auth/login') return view(**kwargs) return wrapped_view</pre></div>						
6.3	The user should be able to be identified within the system for functions such as listing children	✓				
<div>The user is able to be identified and attributes related to them are looked up, for example the profile page that shows the children within their family.</div> <div><div><h3>Profile</h3><p>Please contact your organisation administrator to update your details.</p><table><tr><td>Name</td><td>Testing (Testing) User</td></tr><tr><td>Email</td><td>testing.user@appdevteam.com</td></tr></table><div></div></div><div><h3>My Children</h3><div><div>Children</div><div><div>Joe Doe</div><div></div></div></div></div></div>			Name	Testing (Testing) User	Email	testing.user@appdevteam.com
Name	Testing (Testing) User					
Email	testing.user@appdevteam.com					
6.4	User should be logout of the system	✓				
<div>A logout button is present within the navigation bar which will destroy the users session.</div>						


6.5	User should stay logged in using sessions to prevent redirection to login page	✓
A session is created for the user allowing them to browse the website without being redirected.		
7	Routing	✓
The routing algorithm functions as intended, finding the most optimal route between nodes to travel the shortest journey. To improve this feature, I would also add the functionality to be able to see the exact route rather than an abstract representation between nodes.		
7.1	Ability to create a route <ul style="list-style-type: none"> • Ability to plot points using stop markers and interactive map • Ability to calculate quickest optimal route between nodes using distance matrix • Ability to view overall route 	
Points can be plotted within the interactive mapping view on the new route page. The route can then be calculated using the calculate route function that uses the previously tested algorithm. To improve this feature in a full release I would add the functionality of being able to calculate the times for each stop.		
7.2	Ability to destroy route	
8	Management	✓
The tools required to manage the application are available to the management staff of the application. This includes features related to users, organisations and vehicles.		
8.1	Ability to CRUD organisations using unique identifiers	✓
Each object is given a unique 12 digit identifier to be used throughout the application. Each object, in most cases, can be created, read, updated and destroyed in line with CRUD.		
8.2	Ability to add users to an organisation using an interlinking table	✓
Organisations can have users added via the org_users linking table.		

<table border="1"> <thead> <tr> <th></th> <th>user_id</th> <th>org_id</th> </tr> </thead> <tbody> <tr><td>1</td><td>371883267944</td><td>557998546419</td></tr> <tr><td>2</td><td>371883267944</td><td>894647030600</td></tr> <tr><td>3</td><td>608041185341</td><td>940175646927</td></tr> <tr><td>4</td><td>130363728372</td><td>557998546419</td></tr> <tr><td>5</td><td>143277934379</td><td>11</td></tr> <tr><td>6</td><td>204549501206</td><td>11</td></tr> <tr><td>7</td><td>452134765609</td><td>940175646927</td></tr> <tr><td>8</td><td>571915266587</td><td>940175646927</td></tr> <tr><td>9</td><td>803421460545</td><td>557998546419</td></tr> </tbody> </table>				user_id	org_id	1	371883267944	557998546419	2	371883267944	894647030600	3	608041185341	940175646927	4	130363728372	557998546419	5	143277934379	11	6	204549501206	11	7	452134765609	940175646927	8	571915266587	940175646927	9	803421460545	557998546419
	user_id	org_id																														
1	371883267944	557998546419																														
2	371883267944	894647030600																														
3	608041185341	940175646927																														
4	130363728372	557998546419																														
5	143277934379	11																														
6	204549501206	11																														
7	452134765609	940175646927																														
8	571915266587	940175646927																														
9	803421460545	557998546419																														
8.3	Organisations should be able to list all users within it	✓																														
An organisation can have all its users listed within it by querying the org_users table.																																
8.4	Ability to add a user to the platform. Details for the user to be able to login should also come under this object.	✓																														
A user can be added to the platform and details for login can be added too. An image can be also uploaded for the users profile picture. To improve this area, I would make the user page more interactive with different elements rather than a form of textboxes.																																
8.5	Ability to add a vehicle to the platform. Vehicle details should also be collected from the DVLA using their JSON API	✓																														
Vehicles can be added to the platform by their registration number, specific details such as seating numbers can also be appended. When a user views the vehicle, its details are collected via the JSON API from the DVLA. To improve this feature, I would add the option to set seating plans for the COVID19 period.																																
8.6	Image Asset Management for storing pictures within the system	✓																														
Images are uploaded into specific directories relevant to the intended purpose e.g. profile pictures go to profile pictures directory <i>uploads/userprofilepictures</i> and are named after their 12 digit object ID.																																
9	Profile	✓																														
The user is able to access their profile that displays information about themselves.																																
9.1	Ability to view logged in user profile	✓																														
The logged in user can see all the details pictured below:																																

Profile

Please contact your organisation administrator to update your details.


Name	Testing (Testing) User
Email	testing.user@appdevteam.com



My Children

Children

Joe Doe



To improve this feature, I would add the option to edit a user's own details.

11	Parental Interface	
This is in development stages as it is out of the scope of the main brief at the beginning of the project which was to design the algorithm to find the shortest route alongside the other main features.		
11.1	Ability to book a route as a parent for their own children	
11.2	Ability to view current bookings	
11.3	Ability to view parent profile	

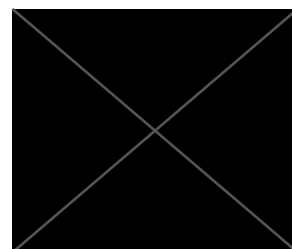
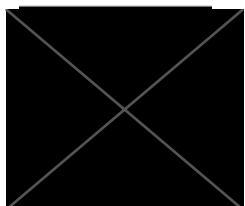
As shown above, I managed to complete each objective to an acceptable standard. There are definitely some areas I would do differently if I knew what I did now but overall the solution came out as originally envisioned.

The main scope of the project was getting the algorithm to calculate the shortest route between nodes to work. This took up the majority of the time with the project and initially I started out attempting to write the algorithm manually until I soon realised I was out of my depth and would need to use some helper tools. If I had known originally the memory issues I would have with a manual approach I would have skipped this altogether and jumped right into using OR Tools which may have allowed more time for the web development side of the application.

The algorithm works as intended possible areas that I would like to develop are a time matrix similar to the distance matrix allowing for me to readily show the stop times for parents to know when to drop off their children. Additionally I still need to pursue the client's original request for adding easy billing options that can be fed into the schools MIS.

The design of the application could do with some improvements too. Although the bootstrap form is a familiar site to many, the usage of a SPA framework such as Google's angular with Google's material framework, similar to the design in the documented design section, would make the interface even better.

Freddie Nicholson Computer Science NEA
Independent Feedback – Parents



[Redacted text]



[Redacted text]

[Redacted text]

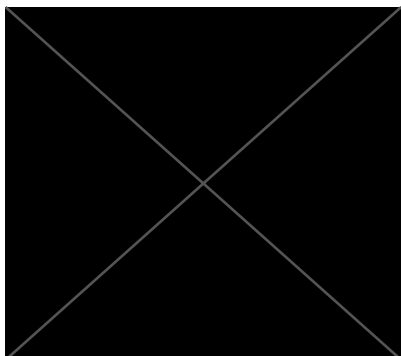
[Redacted text]

[Redacted text]

[Redacted text]



Independent Feedback - Staff

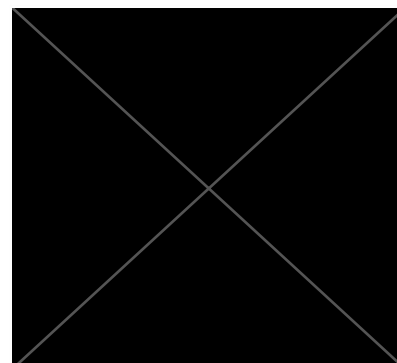


[Redacted text]

[Redacted text]

[Redacted text]

[Redacted text]



Users were generally very positive about the implementation and the proposal for the future implementation of the system. Overall a clear addition that users requested was a mobile app that would allow them to book their children at any time of the day. Another popular feature was that parents should be able to book their children on a one time basis without having to commit to a running schedule. This would be useful as it would allow parents to book last minute bus journeys if they were unable to travel into school or a child was travelling to another persons house with the parents consent being recorded.

From a staff standpoint, both staff commented that the system suited their requirements however some further additions were required if it were to be put into production. One of the fundamental changes that I would need to make from development to production would be the implementation of the

Additionally it was noted that if the system were to be put into production a careful audit would need to be done to ensure that there would be a very unlikely chance of any breach. Clear examples of what this might include might be testing trivial attacks such as SQL injection and then patching any security flaws that are found.

The next stage before introducing the project into production would be a pilot scheme with a limited number of parents and buses to gain real world feedback that could be fed back into the design process to improve the system.

Conclusion

The original aim of the project was to achieve an optimal route for a bus using a set of nodes which has been achieved to a high standard. However there are definitely still some areas that need to be clearly addressed such as the addition of time calculations using a time matrix and the costing feature that is so important to the client.

Moving forward with the project, I will need to do a careful code audit and implement a clear design language that parents can understand. The usage of a more interactive framework and the introduction of a mobile application are clear areas that should be addressed before implementation of the system and these will be my next steps.

In reflection, during the analysis section I believe that I should have obtained a narrower scope for the application as during the technical solution I quickly realised that the project was going to end up being considerably larger than originally expected.

Overall I felt that my NEA had met my original objectives very well and the project has definitely been useful both for evaluating good design principles, managing a complex codebase and other skills that would not be demonstrated in the theory side of the course.