

Route planning application for passenger and freight traffic on the rail network

A Level Computer Science Portfolio

Saen Kazak (Westcliff High School for Boys)
6-27-2020

Problem Analysis

Introduction

Project Objective

Public transport is a necessity and luxury used by many people across the world, every day. Whether assisting the adventurer taking on the British countryside with a camera and an eye for wildlife or accommodating the skilfully seasoned commuter on one of many cold mornings, the use of Britain's rail and bus systems are very much adapted to every kind of person's individual needs and desires. The rail network also carries vast quantities of freight between the country's many docks and ports, often including goods found in our supermarkets, providing much of the food and electronic equipment we use on a daily basis.

The aim of this project is to provide live transport information to the user alongside a path finder for the national rail network (or at least part of it.) The application will be designed to satisfy the needs of a variety of stakeholders with different priorities; this includes leisure travellers and commuters (both of which are collectively referred to as 'regular travellers') as well as rail enthusiasts, rail companies and freight operators (collectively referred to as 'advanced users'). The relevant information will be displayed in a comprehensible format; modelled to be simplistic for those who desire simplicity, but also powerful enough to allow those who may need to use more complex data.

Recognising the problem

Whilst there are already tools available for displaying the information the application will provide, these tools are often difficult to navigate. For more complex and specific data, tools are either restricted to individual transport operators and are therefore far and few between for enthusiasts. For regular travellers, many of the tools that already exist can prove to be largely outdated, complex or incompatible

with certain devices. Whilst advanced users will generally make better use of existing technologies due to higher levels of experience, many of the requirements of such users mean multiple different programs must be used with little cross-compatibility between them. A freight operator requiring rail signalling data **and** rail timetable data for their operations, will, for example, have to fetch the data from many different sources. The software eliminates the need to look in different places for the relevant data.

Features

The application has two main components: the live data display and the pathfinding tool. The live data display will provide visual data regarding rail services, platform numbers, stations, routes, and times. This will be tailored to both regular travellers and advanced users, with a more visual approach taken for regular travellers and a more detailed approach taken for advanced users. The pathfinding tool will allow a user to find the shortest route between two stations, with configurable options in place for advanced users.

In addition, the application will also make use of a user account-based system allowing users to save routes and retrieve previously accessed data more conveniently. It will also function without a user account necessary.

Computational Methods

Abstraction

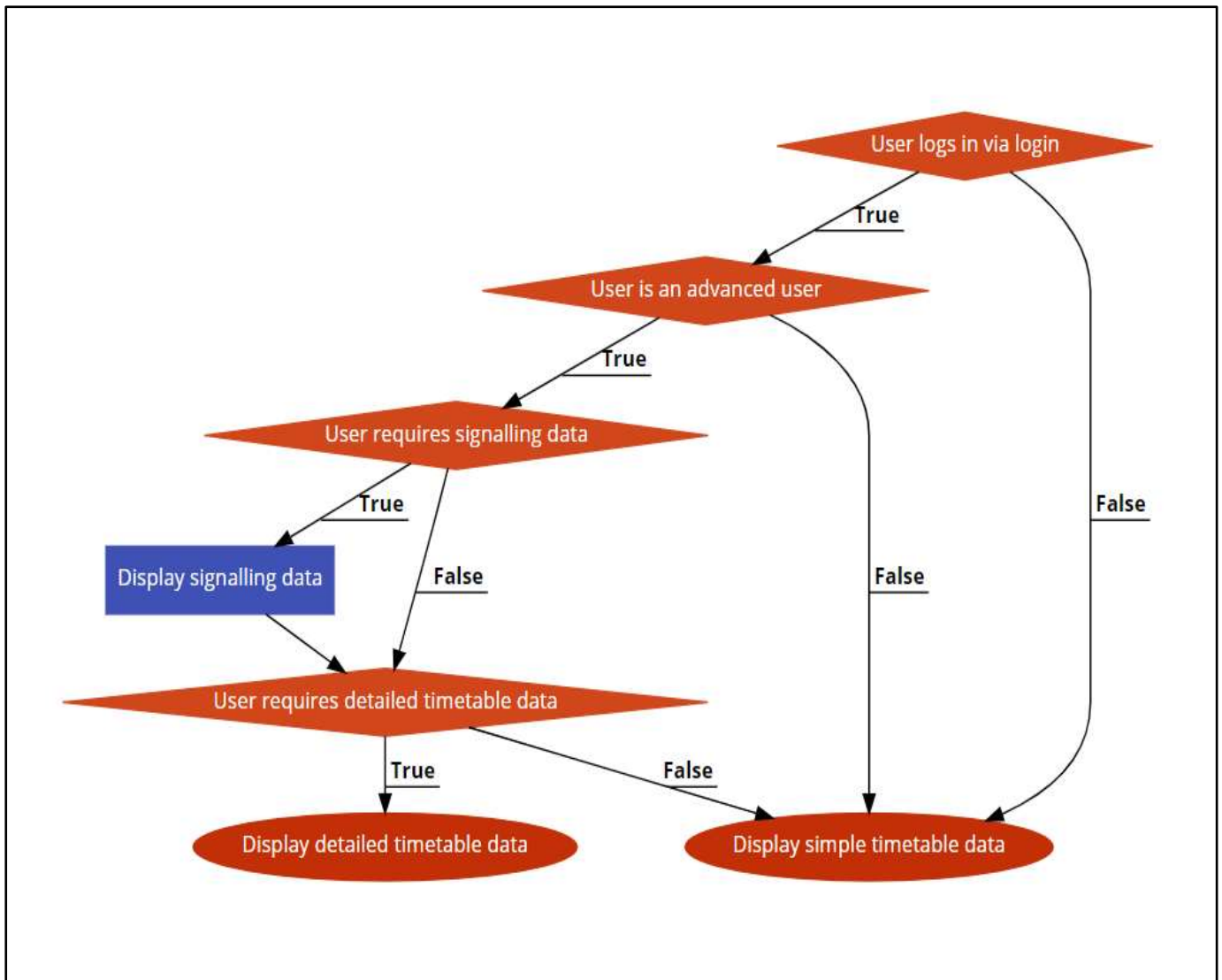


Figure 1: Different data types are omitted from view based on requirements of the user.

The application makes use of abstraction in a number of ways.

The data display can be filtered via a number of different options. This allows relevant data to be displayed and irrelevant data to be omitted. For example, say a regular traveller wanted to find the next train from London Victoria to Orpington. The user can search for trains departing from London Victoria, and then filter out those that do not travel to Orpington, since data regarding any trains that do not

travel to Orpington is considered irrelevant data. This is inherently useful to someone who may be in a rush and does not want to have to manually find the data they want by scrolling through irrelevant data.

In another scenario, say an advanced user stationed with a camera at a specific rail station wanted only to take photographs of trains on a specific platform, and only those operated by a certain company. The necessary data would be filterable so that only the information required is displayed; all other information is discarded. Often in very specific circumstances, due to the nature of the rail network, a significant proportion of the retrieved data is not useful for the user. This makes the process of abstraction ever so crucial for the purposes of the application.

The pathfinder also makes use of abstraction. The program will only output the necessary data; in this case only the shortest route will be displayed to the end user.

Decomposition

The application makes heavy use of decomposition. First and foremost, the components of the app are kept separate: the data display and frontend; the user account system and the pathfinder. Going further, data is separated down further to rail signalling data and rail timetable data. The data is also of course split into different interfaces for those requiring advanced data and those not requiring advanced data. By breaking the application down into bite-sized chunks, different areas can be developed separately and troubleshooting/debugging is kept simple as components can be modified without breaking the rest of the application.

- How can it be ensured that the user account system implemented is secure?
- How can it be ensured that the live data display is comprehensible and easy to follow?
- How can it be ensured that the pathfinder is as efficient as it can be?

'Divide and Conquer'

Whilst the steps required to build the application are complex, it can be seen that by breaking it down into 'bite-size chunks'; focussing on one aspect of development at a time, and *then* bringing in cross-compatibility, once confidence is ensued that all components of the application work independently, the divide and conquer method of solving computational problems is achieved.

Stakeholders

Maintaining Useability Across Different Stakeholders

References are made to 'regular travellers' and 'advanced users' separately here. 'Regular travellers' refers to those who simply want to get from A to B on the rail network and are using the application for that purpose i.e., commuters and ordinary passengers. 'Advanced users' refers to those looking at the data either for 1) their own personal interest, perhaps to track certain trains or make observations or 2) employees of rail and freight companies utilising the provided data to ensure they are running optimally and plan/prepare future services. Of course, regular travellers may have their interests peaked by the advanced data and advanced users may also want to view the simplified formats - therefore these features will be available to all users of the application.

Proposed Stakeholders

The nature of an application such as this one mean that there are many potential stakeholders with a broad range of different desires, interests, and general uses. The application will therefore be tailored to fulfil the needs of a wide range of technical ability among users.

The two main proposed stakeholders are commuters and rail enthusiasts.

Stakeholder Questionnaire

The stakeholder questionnaire is one of the best ways to collect data regarding services demanded by a range of users. The questionnaire was different for the two stakeholder categories and follow-up questions were asked in some cases.

Regular Users: Commuters & Leisure

Questions were presented to two stakeholders in this category, Charles, and Ethan, who represent regular users, desiring simplicity, and convenience. The questions were as follows:

- 1) Do you find it easy to plan your train route when you need to travel?
 - a. If no, why is this the case?
- 2) How do you plan your routes?
- 3) Would you benefit from a more transparent way of planning rail journeys?
- 4) How would you like the application to be operated?
- 5) Have you ever been delayed or found yourself lost because a route planning application (including official route planning tools from train operators) has been confusing, hard to navigate or difficult to comprehend?
- 6) Do you have any further comments / anything else to add?

Question 1 gives an insight into current methods of solving the problem at hand and whether they suit the general public, with further evaluation if the current methods are not satisfactory.

Question 2 attempts to collect data on current methods of solving the problem at hand.

Question 3: 'Transparency' refers to the level of cooperation the user feels exists between designers of route planning apps. To elaborate on this, different rail companies may bias their own services over more efficient or faster services; in simpler terms the question being asked is 'do you feel the current solutions are suited towards you, the user, or a corporation's profits?'

Question 4 attempts to collect data on the preferred platform for an application to be developed, whether this be mobile, desktop PC or another alternative.

Question 5 collects data on the success rate of current solutions and provides a basis for elaboration on how lessons can be learned from existing tools to try and develop a better tool.

Advanced Users: Enthusiasts

Questions were presented to two stakeholders in this category, Sam and Alex, representative of rail enthusiasts desiring a further level of complexity and access to a higher level of data than the regular users. The questions were as follows:

- 1) When planning rail journeys, are you satisfied with the level of detail given by conventional rail planners? (e.g., National rail app, google maps)
 - a. If no, then what would you like to see in a program designed for rail enthusiasts to partake in different aspects of their hobbies?
- 2) How do you plan your routes?
- 3) How would you like the application to be operated?
- 4) Would you benefit from a more transparent way of retrieving station data, such as signalling information, trainset unit numbers, service type, etc.
- 5) You are a hobby photographer on the rail network, and you are looking to capture images of a special steam train that is operating today. Would you benefit from being notified about special events such as this so you can plan and prepare for travel to the right locations in advance?
- 6) Do you have anything else to add?

Question 1 for advanced users is of a similar nature to above adding in an element of detail regarding hobbies and allows for a broader response scope.

Questions 2, 3, and 4 are as above.

Question 5 provides a specific scenario that may be of relevance to some of the interviewees. This allows the collection of specific scenario-based data which can be of benefit to development.

Interview

Name: Ethan (regular traveller)

Do you find it easy to plan your train route when you need to travel?	Answer: I find it moderately easy, but it could be improved
If no, why is this the case?	Ethan does not find it difficult to plan routes.
How do you plan your routes?	Answer: I usually plan my routes through Google Maps
Would you benefit from a more transparent way of planning rail journeys?	Answer: Yes, I would.
How would you like the application to be operated?	Answer: I would like it to be mobile friendly and fast since I rarely plan journeys on a desktop computer and I am often in a rush.
Have you ever been delayed or found yourself lost because a route planning application (including official route planning tools from train operators) has been confusing, hard to navigate or difficult to comprehend?	Answer: Yes, I have been.

Do you have any further comments /
anything else to add?

Answer: I would like planning
software that accounts for new
cancellations and delays, and also
tells me how long I would have to
cross between platforms.

Name: Charles (regular traveller)

Do you find it easy to plan your train route when you need to travel?	Answer: No, I often find it time consuming and confusing.
If no, why is this the case?	Answer: Google Maps can be wildly inaccurate
How do you plan your routes?	Answer: I use Google Maps to find the fastest route and do some extra research on the location/line I am travelling to/on if necessary.
Would you benefit from a more transparent way of planning rail journeys?	Answer: Yes, as I find it hard to trust route planners run by train companies themselves because I think they would be dishonest and favour their services.
How would you like the application to be operated?	Answer: I would like it to work with all different devices and show on my phone screen properly.
Have you ever been delayed or found yourself lost because a route planning application (including official route planning tools from train operators) has been confusing, hard to navigate or difficult to comprehend?	Answer: Yes, google maps is good but does give misinformation and lacks some of the features I want.

Do you have any further comments /
anything else to add?

Answer: I always thought that if a
rail information application
contained maps of individual
station interiors, it would be very
helpful.

Name: Sam (advanced user)

When planning rail journeys, are you satisfied with the level of detail given by conventional rail planners? (e.g. National rail app, google maps)	Answer: Not particularly, I tend to find official tools to be slow and cumbersome.
If no, then what would you like to see in a program designed for rail enthusiasts to partake in different aspects of their hobbies?	I would like to see a wider range of included data and better options for filtering said data to fulfil my personal requirements.
How do you plan your routes?	I use my own knowledge of the rail network in addition to paper maps.
How would you like the application to be operated?	Preferably on a computer. I tend not to use my phone much, but I guess a friendly mobile interface does not have any disadvantages – as long as it still shows all the data I want.
Would you benefit from a more transparent way of retrieving station data, such as signalling information, trainset unit numbers, service type, etc.	Answer: Yes.
You are a hobby photographer on the rail network, and you are looking to capture images of a special steam train that is operating today. Would you benefit from being notified about special events such as this so you can plan and prepare for travel to the right locations in advance?	Answer: Yes, this would be incredibly beneficial to me.

Do you have anything else to add?

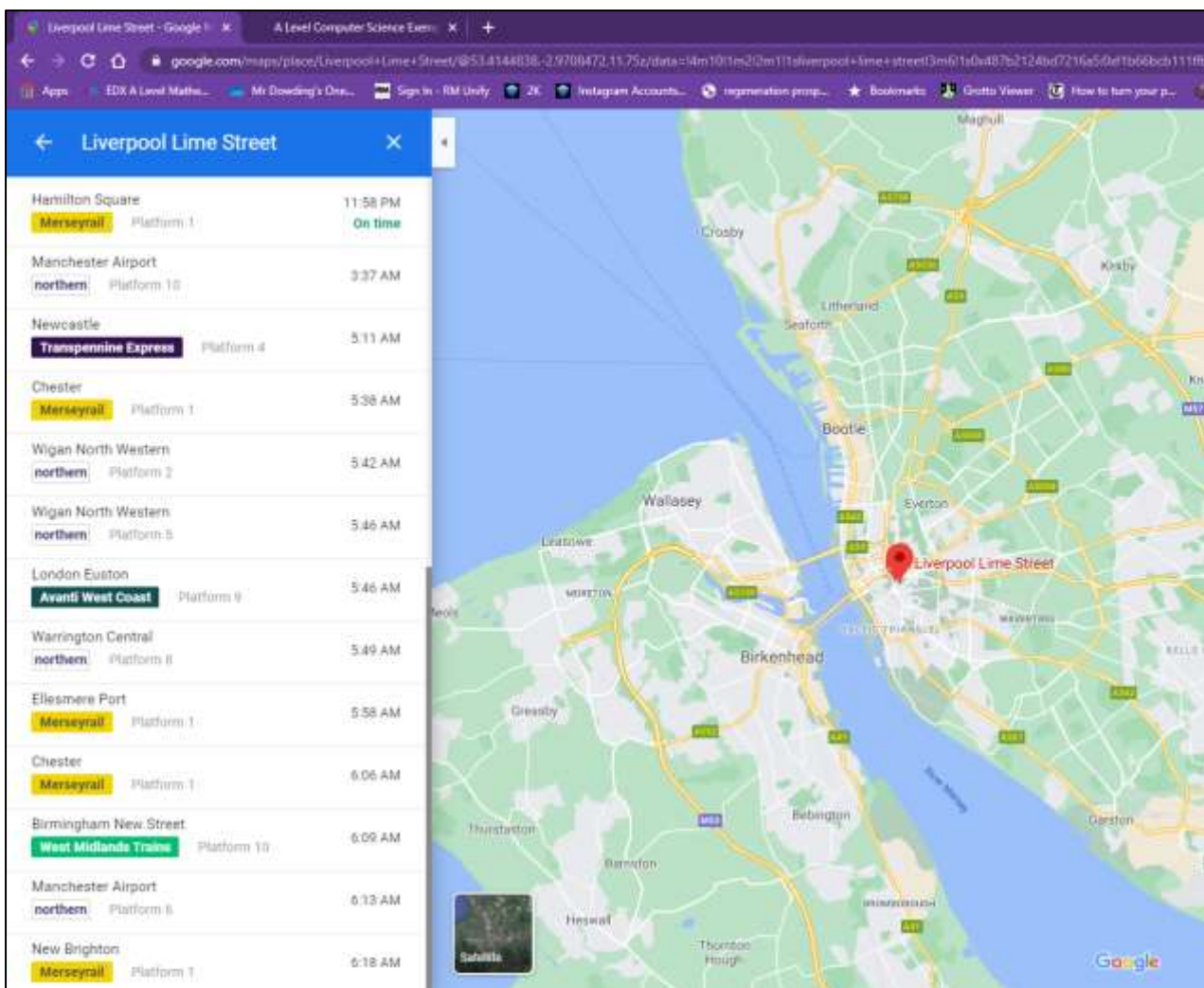
Answer: Not particularly.

Research

Existing Solution #1: Google Maps

Overview

Google Maps is widely used all over the world to display public transport information. It is located at <http://maps.google.com> and is heavily integrated into other Google services, such as search functionalities. Google Maps takes transport data and displays it in a human-readable format. The success of the program stems widely from its extended use of buttons, colours and visuals (such as differences in font size to distinguish different levels of data significance) making it the route planner of choice for many of the stakeholders, especially those in the category of regular travellers.



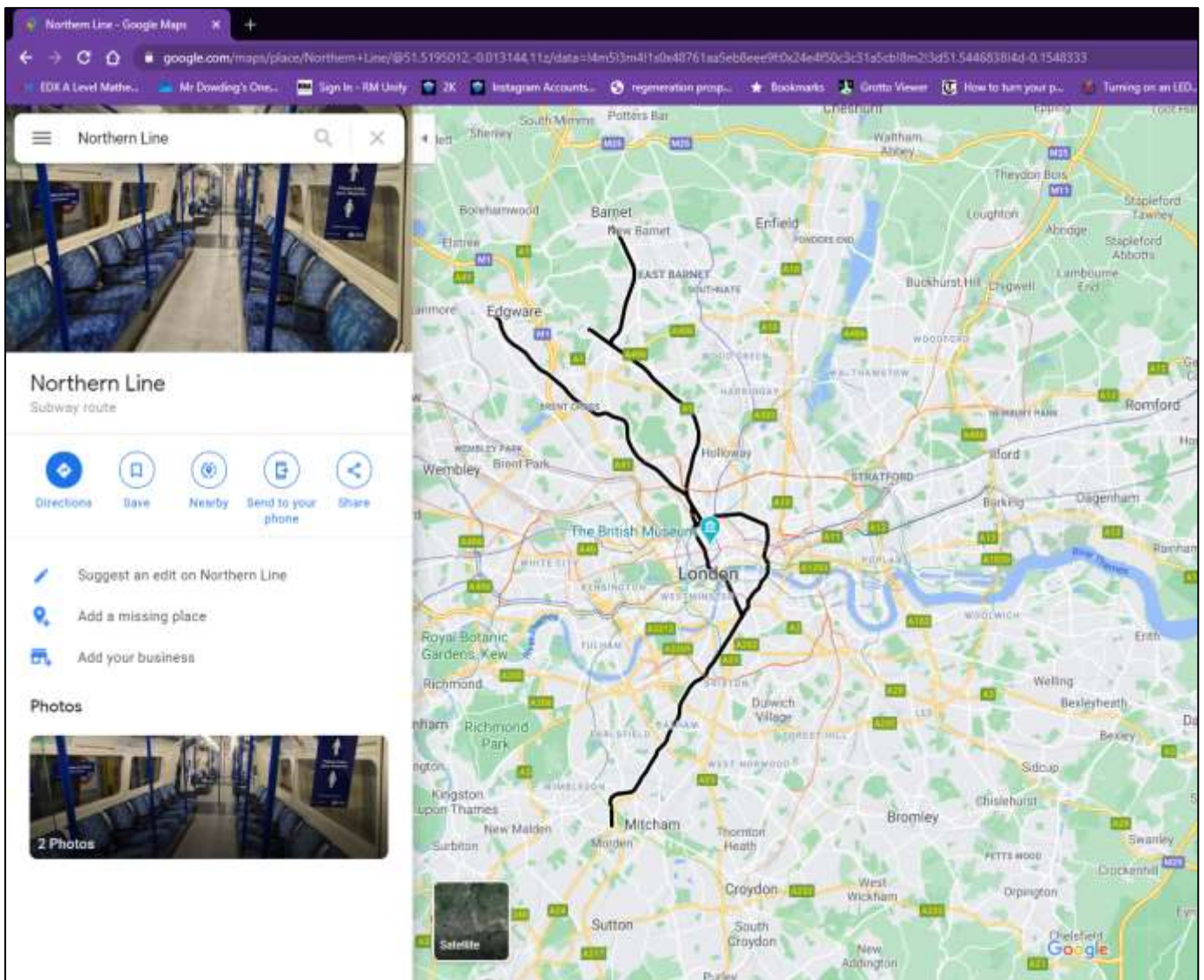
Application to Proposed Solution

Google Maps has several features from which inspiration can be drawn for development of the proposed solution. The first screenshot, shown above, shows train services being searched for at an example station. The application returns the location of the station on a graphical map and services departing from the station on the left-hand side. The use of different colours to represent different rail services is useful for users who may be in a rush and subconsciously know what they are looking for without reading the text; the human brain is wired to see visuals before words, making some level of abstraction of the text beneficial to the user. In this regard, the use of colours to show different rail services is applicable to the proposed solution and will be of benefit to the stakeholders and their priorities.

Limitations

The second screenshot shown below shows an entire route as opposed to a station. Once again the graphical map is shown with area names, roads and significant features accompanying the user's query. The queried rail route is shown on the left-hand side with photographs, options for user feedback and sharing. Additionally, a visual representation of rail routes is shown, making use of abstraction by showing the queried route in a bolder form than other connecting routes. The user can click on connecting routes, town names or station to be redirected to the search result for

these items.



Whilst these features suit Google Maps and its stakeholders/userbase well, they are not as applicable to the proposed solution. Implementation of a graphical map would be unnecessary to those in a rush as they will likely already have a degree of knowledge as to where they are going. The abstraction of routes shown is done already in the proposed solution by displaying relevant data only based on the user's query. The use of photographs of areas is also unnecessary for similar reasons to above; photographs will merely take up space, may require use of additional licencing and are already available publicly on the internet for users who wish to know more about station interiors or the appearance of trains.

The key difference between Google Maps and the proposed solution is that Google Maps is not rail-specific (hence it shows roads, towns and attractions) and has a

focus on providing all users with information to make their experience of the software feel 'friendlier' (such as adding photographs). The stakeholders of the proposed solution prefer speed and simplicity with respect to features included.

Existing Solution #2: RealTimeTrains

Overview

RealTimeTrains, located at <http://realtimetrains.co.uk> is a similar solution to the problem at hand. The website displays similar information to what is required and displays it in a readable format.

Simple / Detailed Modes

The site makes use of simple and detailed controls to display relevant data to users with different priorities. The 'simple' mode displays only the necessary data for an

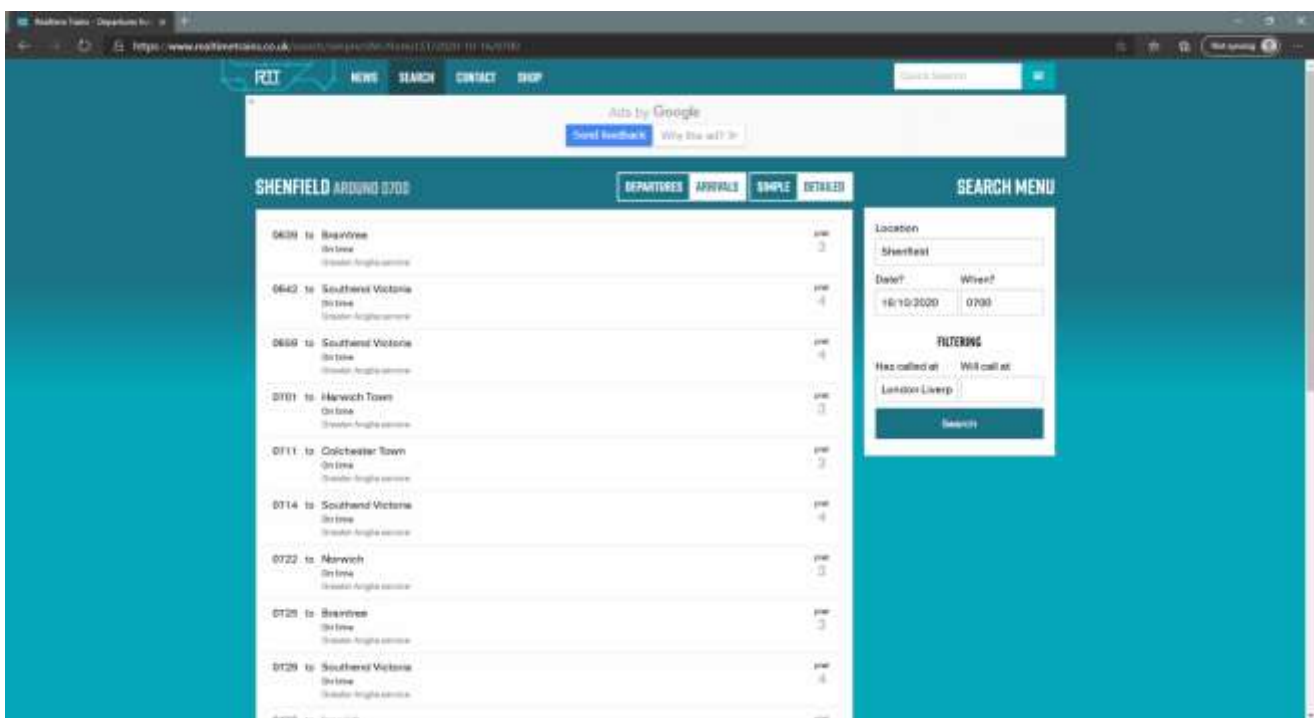


Figure 2: RealTimeTrains 'Simple' Mode

ordinary passenger, whereas the 'detailed' mode provides a greater level of flexibility. Example screenshots are shown below:

The simple mode screenshot contains a graphical list of train services, timings, platforms, and train operating company. This data is the most relevant for a user that simply needs to "get from A to B." Other search options such as specific destination / previous station is also available; the site has deemed these to be potentially useful but not essential in this scenario.

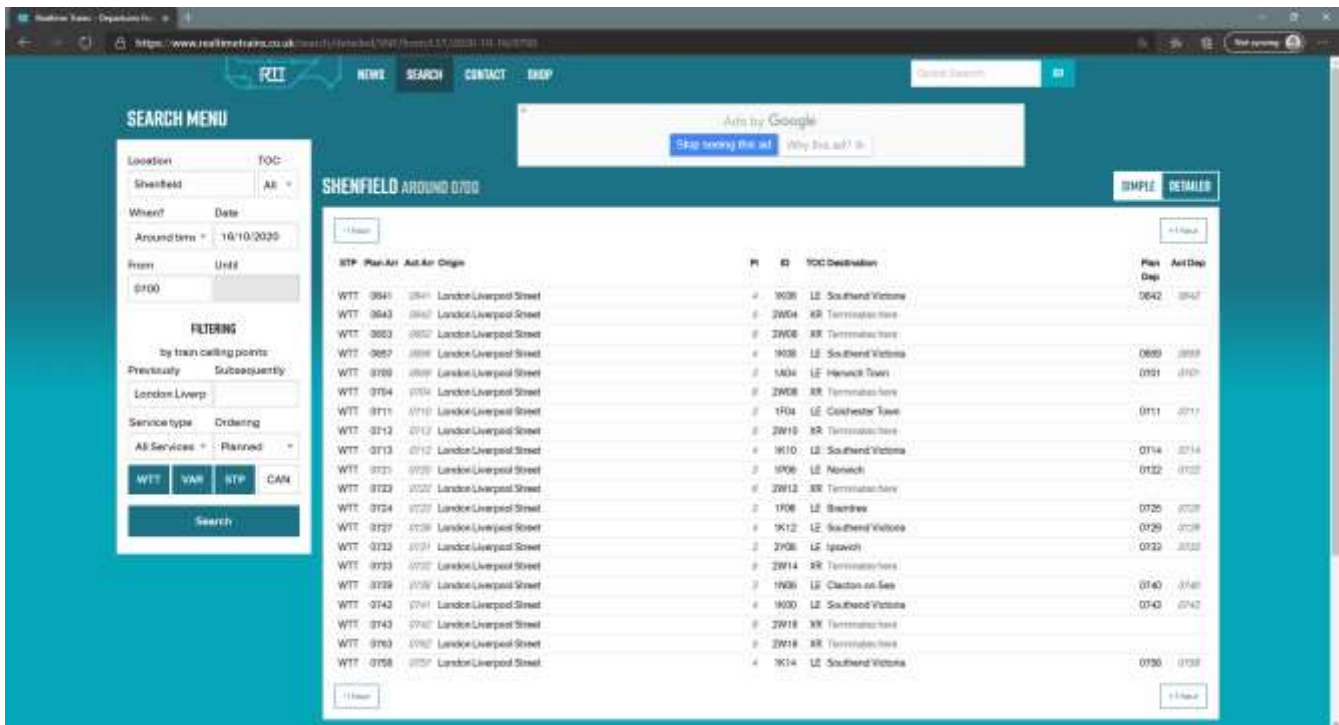


Figure 3: RealTimeTrains 'Detailed' Mode

The detailed mode screenshot provides more information, such as train ID and Service Type Pattern (STP). This data may be relevant for a rail enthusiast or railway employees who may need to access the data. This menu may also show freight/cargo services – an example of something irrelevant to ordinary passengers but of potential use to enthusiasts, photographers, and rail employees.

Application to Proposed Solution

The aspects of this site that can be applied to the proposed solution include the highly comprehensible format and the separation of complex and simple data. *In the design section, details are provided of the graphical interface being developed to allow readability and usability.*

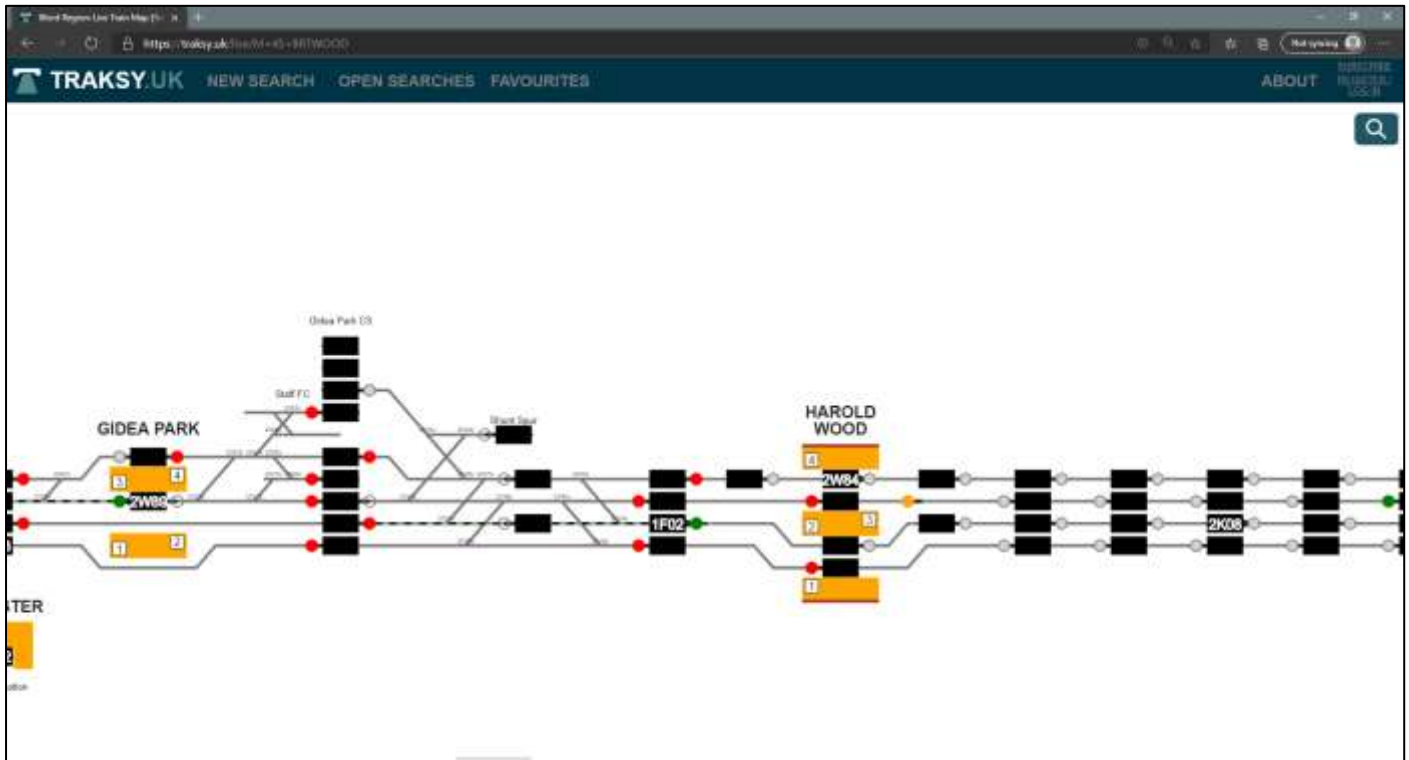
The use of features such as different font weights (bold, italic) and different variants of font colour allow distinguishment between different data and, to an extent, the relevance of the data. This can be applied to the proposed solution as it will minimise confusion and maintain the appeal of the interface as a whole.

Limitations

The proposed solution is solely focused on route planning and inherently will not include features such as a 'shop' or a 'news' tab. Whilst this example is commercialised, the proposed solution is not. However, a 'contact' feature, or something of a similar nature, could be implemented as way of receiving further feedback on the software from stakeholders including ways to improve the design or queries about the software itself.

It could also be argued that some aspects of the simple interface are too sophisticated for regular travellers, which is a good indicator that the proposed solution should be even simpler.

Existing Solution #3: Traksy



Overview

Traksy, located at <http://traksy.uk> provides live signal and track data for the national rail network. Signals are fundamental to the operation of the railway and can display either a green, orange, or red light depending on the occupation status of the track ahead.

This data will be of great use to enthusiasts and rail employees and the display shown is very similar to the information given to signal operators working for the rail network.

Application to Proposed Solution

The example shown here provides all of the relevant data in a visual format which is easily understandable. It could even be said that one of the objectives in mind for the developers of this example is to spark a level of enthusiasm for the subject of railway infrastructure without sacrificing functionality and the needs of those who use this software within a commercial setting.

Limitations

The proposed solution does not intend to spark enthusiasm in railway infrastructure due to its emphasis on convenience and presenting the required data. The proposed solution is very much a 'get the job done' solution and therefore the signal data could be displayed in a textual format rather than a visual format, something which is deemed time-constraining and unnecessary. For this reason, graphics can be abstracted here. Many users of this part of the proposed solution are likely to know the technicalities of the data being observed.

User & Server Requirements

Hardware – Host Server

A computer capable of running the required software. The majority of modern computers will have the necessary processing power to run the application. The standard peripherals including a mouse, keyboard and monitor will be required.

A stable internet connection. An internet connection is required to host the application and connect to the online APIs required to retrieve the data. The connection should have sufficient bandwidth to manage multiple users authenticating and connecting at the same time.

Software – Host Server

A webserver with capabilities to run PHP and Flask/Django. These components are required for the live data display and the pathfinder respectively, with the latter enabling the pathfinder to be displayed within the former.

Windows, Mac, or Linux operating system. These operating systems can run the required components.

Python Interpreter. This is needed to allow the pathfinder to function, since the pathfinder is written in python.

Python dependencies including **heapq**, the **math** module and the **time** module. These are required to run the pathfinder.

For the purposes of my A Level, the code will be hosted on my personal laptop within the Ubuntu Operating System, with all the required dependencies installed. I may provide sample data previously fetched from the API to negate the specific requirement for the application to be on the world wide web – however in a real scenario this application would require permanent and continuous internet access.

Hardware & Software – Stakeholders

Since the application is server-side, the requirements for the client are fairly standard. The requirement is simply a device capable of **internet access**. The application pulls from APIs and therefore cannot be run on a fully offline basis. Example devices could include a Windows PC, Android Phone, or iPad. The browser should include support for PHP, something the vast majority of browsers will have in place already.

Stakeholder Requirements

Graphical User Interface (GUI)		
Requirement #1	Lightweight and easy to follow; limited use of custom fonts or other graphics that slow the software down.	The stakeholders desire a design that works with time rather than against it – a visually appealing and function graphic design is necessary here as a result.
Requirement #2	Easy to follow timetable with large enough font for good readability	
Requirement #3	Interface that functions on a multitude of different devices.	The stakeholders desire that the application is useable by both portable and non-portable devices.

Functionality		
Requirement #4	Buttons/Options to filter trains by operator, train ID, time of day and platform number	The demands of advanced user stakeholders require that trains be filterable via the data returned by the API. The filterable options are crucial to the abstraction of the data fetched by the application.

Requirement #5	Simple, secure, user account system with a clear log in page, options to reset password and configurable settings.	The user account system is not mandatory for use of the application, but allows users to save routes and configure settings, making it desirable to the stakeholders.
Requirement #6	The facility to save routes for registered users - stored in SQL database.	The stakeholders desire this option; the database must be secure and protected against possible hijacks such as SQL Injection or data loss.
Requirement #7	Pathfinder integrated with the software and its own query displayed alongside the main query from the user.	The pathfinder, despite being coded in Python, must be integrated within the PHP interface. This can be done via Flask/Django or alternative methods.

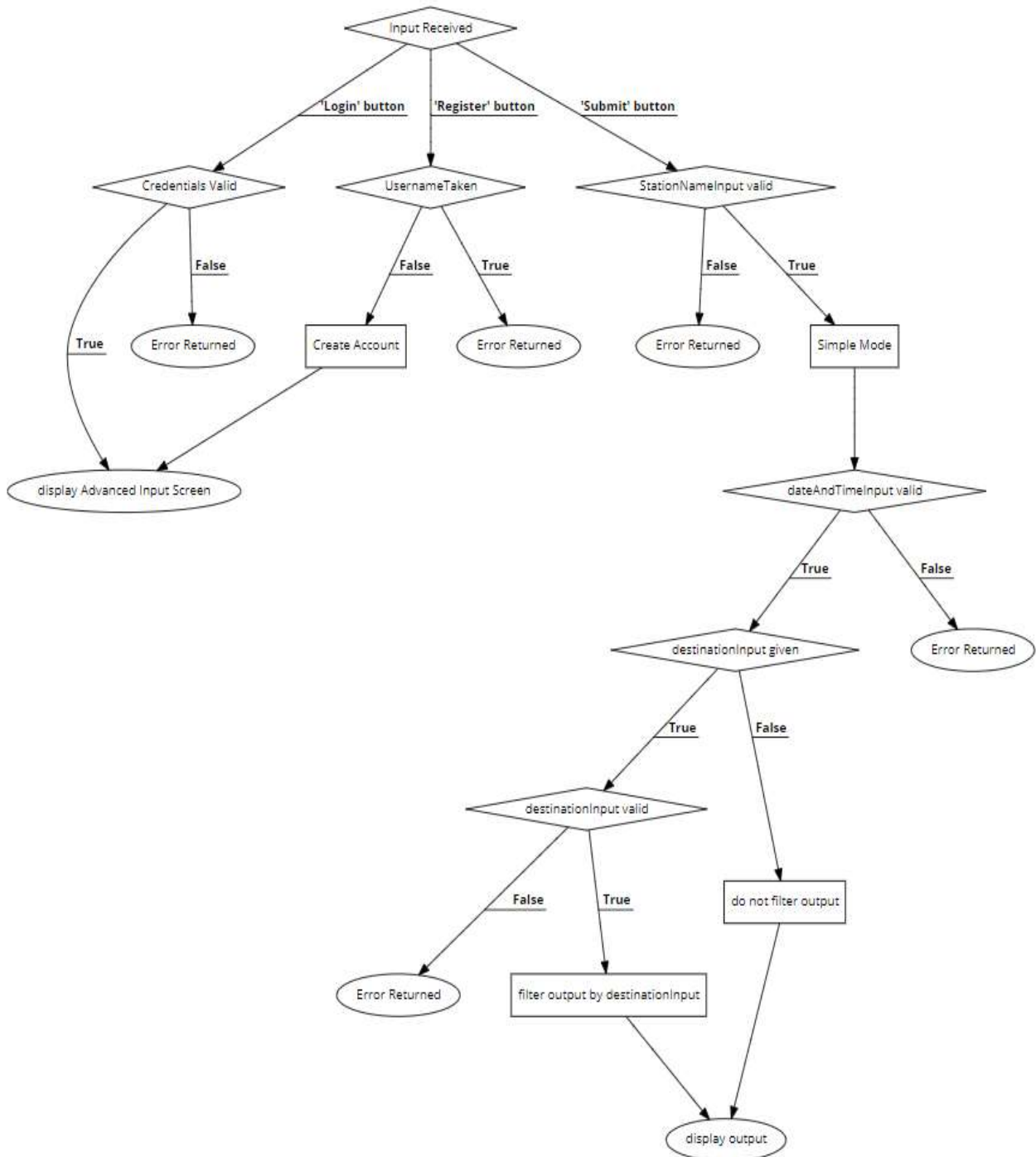
Success Criteria

Lightweight design with limited use of unnecessary graphics	Screenshots of the main pages and search queries
Mobile friendly, scalable interface	Two comparable screenshots of interface as shown on both mobile and PC
Buttons that function for both mobile and PC	
Easy to follow timetable with large enough font for good readability on all devices.	Screenshots of the timetable following a search query
Train filter options	Screenshots of the timetable following a search query
User Account System	Screenshot of the main page, settings pages, and the code behind the account system.
Facility to save routes in a database	The code behind the database
Pathfinder integrated with frontend	The code for all files behind the pathfinder and screenshots of the pathfinder results shown in both python and within the PHP frontend.

Design

Live Data Retrieval

Use of Computational Methods



Computational methods can be applied to the individual aspects of the user interface within the landing page and input query.

Divide and Conquer

Divide and Conquer as a problem-solving technique can be applied here; the different aspects of the program such as the error handling, output filtering, authentication, data type and data visualisation can be thought of as sub-problems which can then be tackled as individual components.

Abstraction

Abstraction is used to distinguish between the format of the input received and determine the next steps of the program, such as the software defaulting to 'Simple Mode' if an input is received via the 'Submit' button, whereas use of the 'Register' and 'Login' buttons will default the software to display the advanced input options. The use of abstraction via generalisation means that, for example, aspects regarding login and registration features can be tackled together whilst still being abstracted from the other sub-problems.

Decomposition

In addition to this, by decomposing the problem down into simpler steps and making use of visualisation as a problem-solving technique, it becomes easier to see where the divided sub-problems must be implemented in order to ensure a successful and well-built program. Taking error handling as an example, the flow diagram above shows that the software needs to return input errors when: the start station input is invalid; the login credentials are invalid; the date/time input is invalid; the destination station input is invalid (should the destination input not be blank); and when the username is taken upon registration.

Landing Page

Outline

Welcome to Railmapper!

Station Name Input

Destination Input (Optional)

Date & Time Input

Submit

Username

Register

Password

Login

This is a basic outline of the proposed 'welcome screen' for the program. This aspect of the program is fundamental to its success as it gives the user a first impression of the software.

User Input

The user can input the name of a station, an optional destination to filter by, and an input for the date and time. The user can also skip this entirely and instead choose to register or log in, providing access to advanced features.

Satisfying the success criteria

The stakeholders require the software to be rapid in order to satisfy the requirements of a user who may be in a rush. The lack of clutter within the proposed landing interface ensures this requirement is met. The search options displayed make up a focal point of the landing page and are easy to follow. The ability to

register and log in from the landing page is also a requirement of the stakeholders. This is also clearly shown and easy to follow.

Live Search of Fields

Outline

The GUI will contain several fields which will display autocomplete suggestions in order to maximise speed and convenience as stated within the success criteria. The fields which will contain autocompleting functionality are:

- Station input, via station codes and station names, based on the first letter input by the user.
- The date and time, which will default to the current date and time in autocompletion.

The live search is coded using PHP and AJAX within the HTML skeleton.

Wireframe diagram of the Railmapper GUI form:

Welcome to Railmapper!

Station Name Input: Example: London Euston, Epsom Downs, East Croydon, Ealing Broadway

Date & Time Input: Example: 2021.01.01 13:00

Submit

Username **Register**

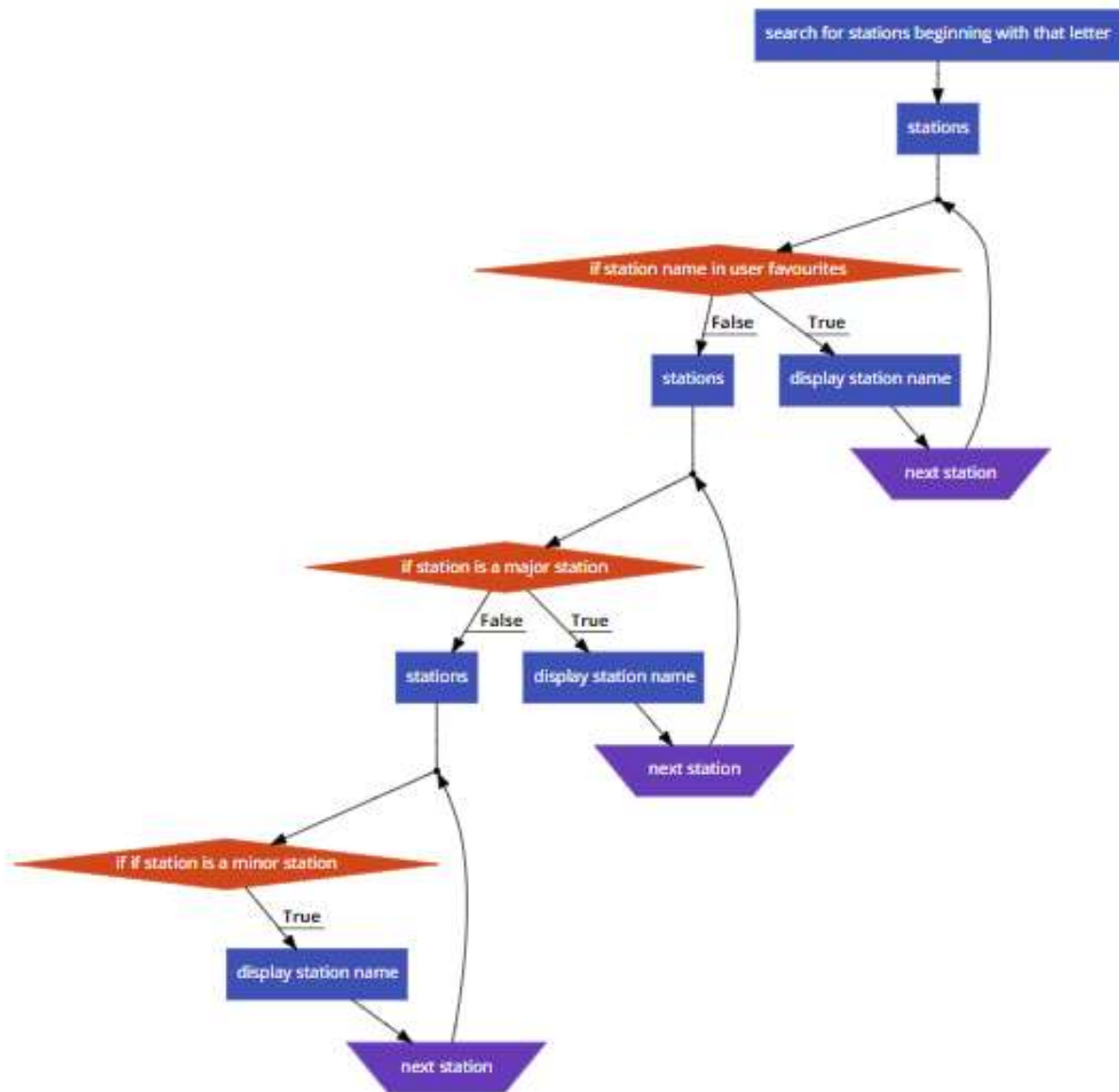
Password **Login**

Algorithm Decomposition

The flowchart shown describes the process used to determine the stations shown by the program as results are input by the user in real-time.

Autocompletion will maximise simplicity and speed within the application by prioritising specific categories of the returned data. By priority order, the code will determine:

- Is the station name contained within the user's favourite stations?
 - o If the user is not logged in, skip this step
- Is the station a major station, such as those found in major cities
E.g. Within results, Edinburgh Waverley will display higher than Enfield Town
- Display the rest of the stations starting with that letter up to a maximum of 8.



Mockup of iteration through the live search data.

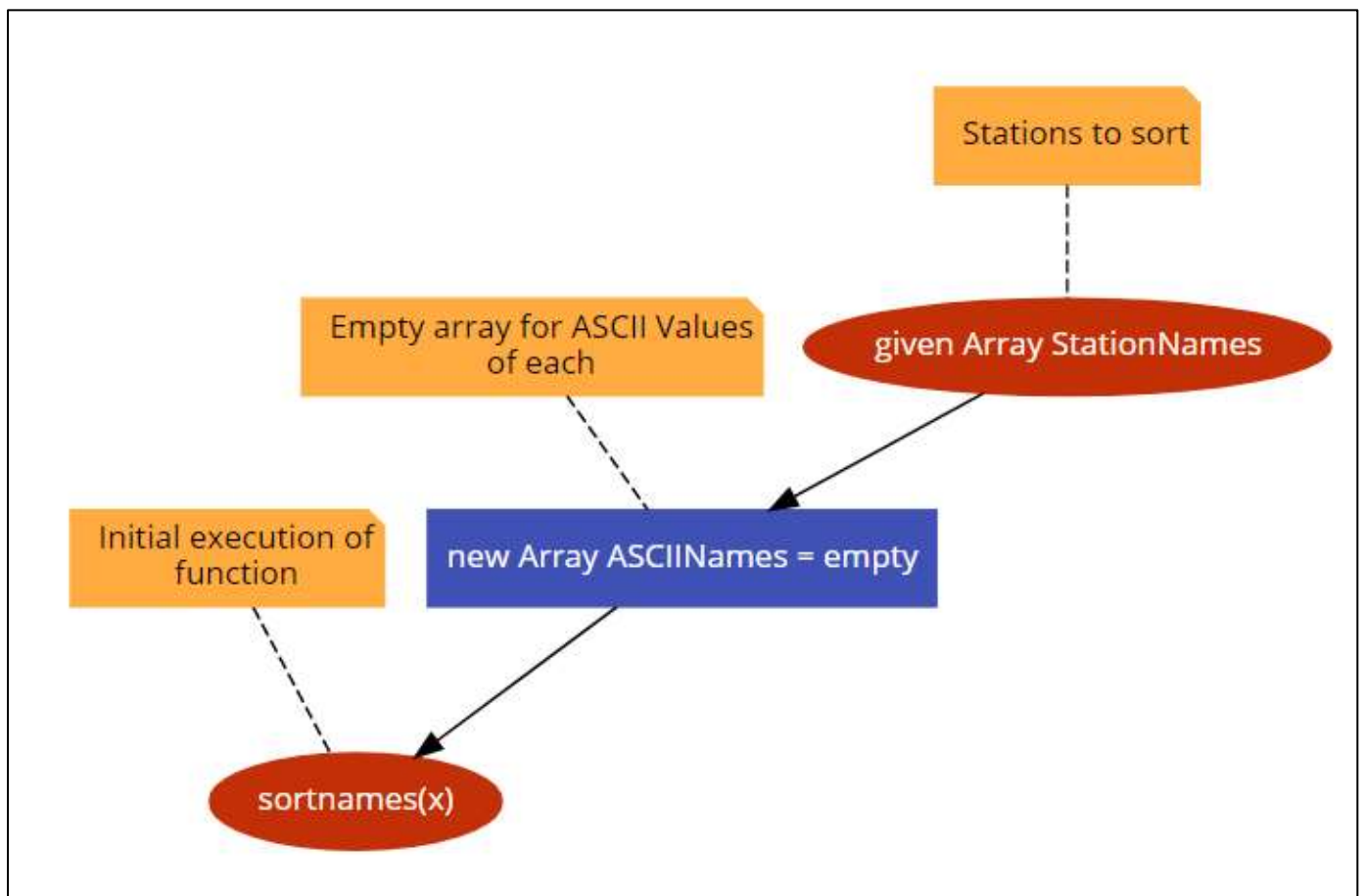
Data Types

The list of the user's favourite stations, the list of major stations and the remaining stations will be stored within a PHP Array. The PHP Array is an efficient data structure for storing information such as station names due to its limited use of memory since the array will not need to be updated after the refresh of the code. Upon retrieval of the code, the arrays will be concatenated together in the priority order.

Sorting, ASCII Representation & Iterative Methods

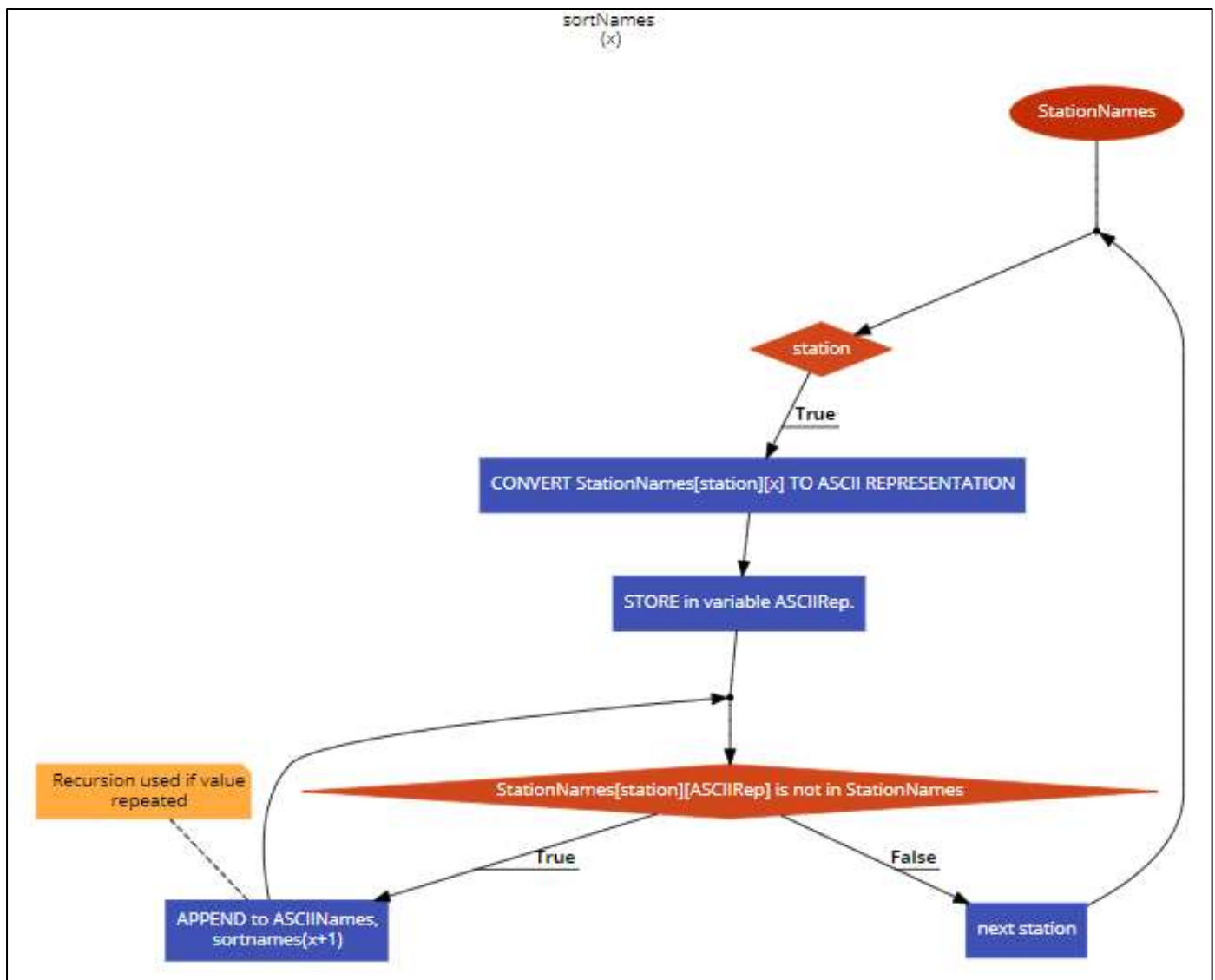
The stations returned will need to be displayed to the user alphabetically, for each category. Taking one category (for example, major stations) into account, a potential solution would be to create an array of the letters of the alphabet, and then compare the first letter of each station name to the corresponding position of the letter in within the array, with the lowest value being prioritised in a new array along with its corresponding name of station. Should the letters be the same, the second letter is taken instead and so on until the list is sorted.

Firstly, a new empty array is defined to store the ASCII values of the unsorted array.



Mockup of the method call for sortNames function and creation of ASCIINames array

From here, the function `sortNames(x)` is called and makes use of recursion to sort the ASCII Representation into the correct order.



A mockup of the sortNames function

The station name list is capitalised in order to ensure continuity among the ASCII value comparisons.

StationNames	Array Index	ASCIINames	New Array Index
CATFORD BRIDGE	0	[67]	3
ASHFORD INTERNATIONAL	1	[65, 83]	1
DEPTFORD	2	[68, 69]	5
LERWICK	3	[76]	7
EPSOM DOWNS	4	[70]	6
BARKING	5	[66]	2
DARTFORD	6	[68, 65]	4

APSLEY	7	[65, 80]	0
--------	---	----------	---

Example Data sorted from StationNames to ASCIINames

Results Display

Filtering of Results

In order to further enhance the user experience, the display of results will include options to filter by train operator. In addition to this, the option to filter by platform or calling points will also be added as a future consideration.

Time	Origin	Destination	Platform	Operator
12:00	ExampleStn	Leeds	1	Happy Trains
12:09	ExampleStn	Manchester	4	Northern
12:24	ExampleStn	Shenfield	6	TfL Rail
12:27	ExampleStn	Liverpool	2	Happy Trains
12:35	ExampleStn	Birmingham	2	Great Western
12:40	ExampleStn	Exeter	3	Great Western
12:58	ExampleStn	Glasgow	5	LNER

Happy Trains

Northern

TfL Rail

Great Western

LNER

Filtering will be done via buttons which can be selected and selected in order to display or hide results. An iterating loop will be used to update the visibility of table elements whenever one of the buttons is clicked.

In order to further enhance the user experience, the display of results will include options to filter by train operator. In addition to this, the option to filter by platform or calling points will also be added as a future consideration.

Filtering will be done via buttons which can be selected and selected in order to display or hide results. An iterating loop will be used to update the visibility of table elements whenever one of the buttons is clicked.

Time	Origin	Destination	Platform	Operator
12:35	ExampleStn	Birmingham	2	Great Western
12:40	ExampleStn	Exeter	3	Great Western

Happy Trains

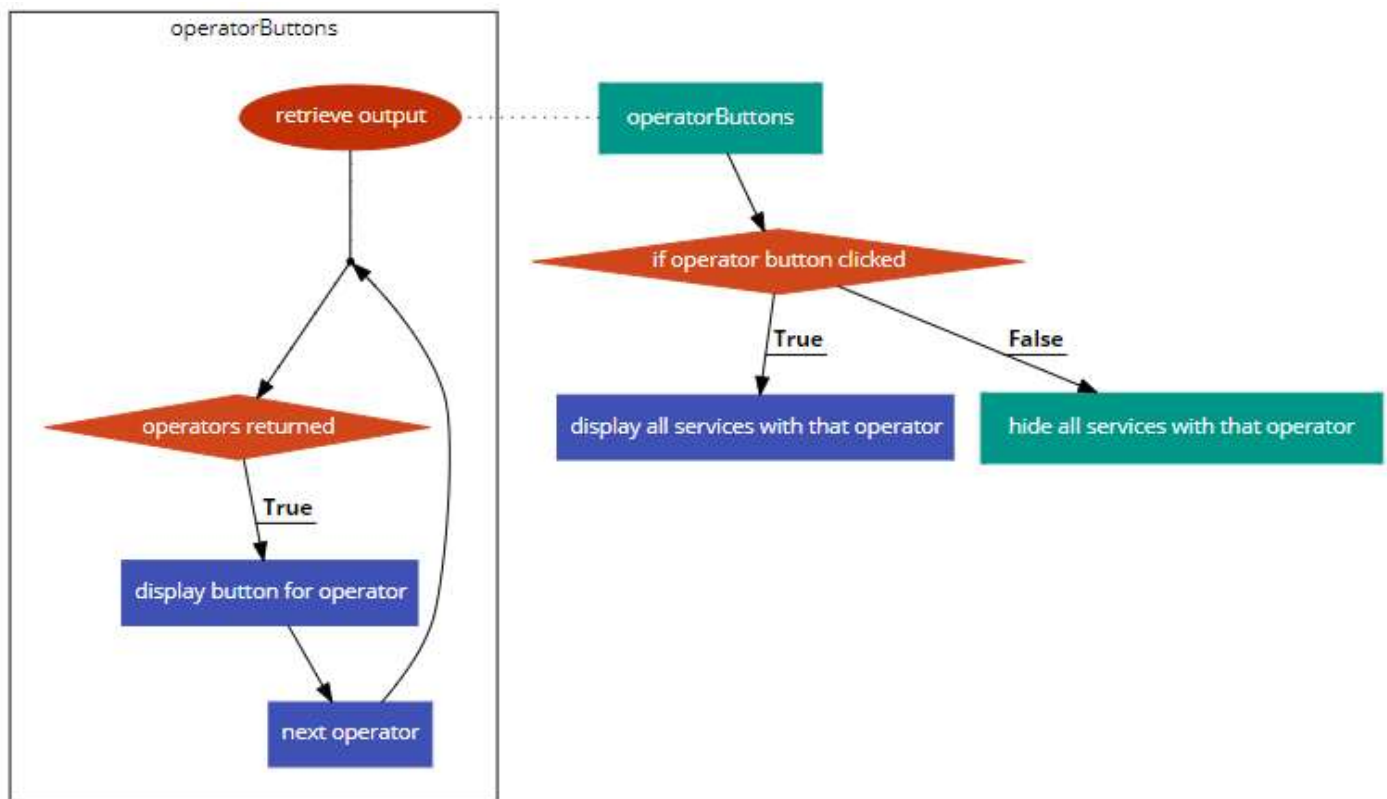
Northern

TfL Rail

Great Western

LNER

Algorithm for operator filter buttons



A PHP iterating loop will be used in the main code to retrieve the train operating companies returned and build respective buttons for them within the HTML skeleton.

Maximising Accessibility

Users with impaired vision may find it difficult to use the application. Use of different colours within the button design will improve the accessibility standard of the application.



This can be matched by the fact that many different rail operators within the UK make use of their own design idioms and branding schemes. Some of these are given in an example above.

Other accessibility solutions for this part of the results display could include:

- Larger text size for users with impaired vision
- Use of a sans-serif font which is easier to read
- Implementation into a screen-reader API for users of assistive technology

The results can also be tailored to suit those requiring physical assistance within stations. Rail companies often have an option to book travel assistance and query staffing levels at stations; this can be implemented into the results display, should accessibility options be set to show.

Time	Origin	Destination	Platform	Operator	Step-Free Access?	Book Travel Assistance
12:00	ExampleStn	Leeds	1	Happy Trains	Yes	[Link]
12:09	ExampleStn	Manchester	4	Northern	No	[Link]
12:24	ExampleStn	Shenfield	6	TfL Rail	Yes	[Link]
12:27	ExampleStn	Liverpool	2	Happy Trains	Yes	[Link]
12:35	ExampleStn	Birmingham	2	Great Western	No	[Link]
12:40	ExampleStn	Exeter	3	Great Western	Yes	[Link]
12:58	ExampleStn	Glasgow	5	LNER	No	[Link]

The operator colours can be determined two different ways:

- Random colours are assigned to each button

- A file or variable containing common operators and their respective 'colours' can be read from and used to display the buttons, with default colours chosen should no entry in this file be found.

API Integration

Data Retrieval



The API request is processed on the server side and then passed to the client whereby the DOM will construct the table of the output data. This will involve use of PHP and JSON including displaying JS variables in HTML.

The request is processed in different stages:

#	Description	Method / Type	Component
1	User input taken	HTML Form	Client Machine
2	User input sent to Application Host Server	HTTP GET Request	Client Machine
3	User input sent to API	PHP CURL Request	Server Machine
4	Data request processed	PHP JSON variable	RTT API
5	Data sent to Application Host Server	PHP JSON variable	RTT API
6	Data parsed	PHP JSON variable	Server Machine
7	Data passed to client	JS JSON object	Server Machine
8	Data implemented to DOM via browser	DOM	Client Machine
9	Output displayed	DOM	Client Machine

Problems & Solutions

Restrictions within API Usage

The problem at hand is to retrieve live rail data and display it in a human-readable format. This component of the application proved to be more challenging than initially thought due to a multitude of problems that would arise at various points throughout the development.

The initial plan was to use the TransportAPI located at <http://transportapi.com> to display the data. The data would be parsed via an XMLHttpRequest into a JavaScript object.

```
var obj; // Defines variable in a global manner
var request;

function updateData(code, date, time, dest) {

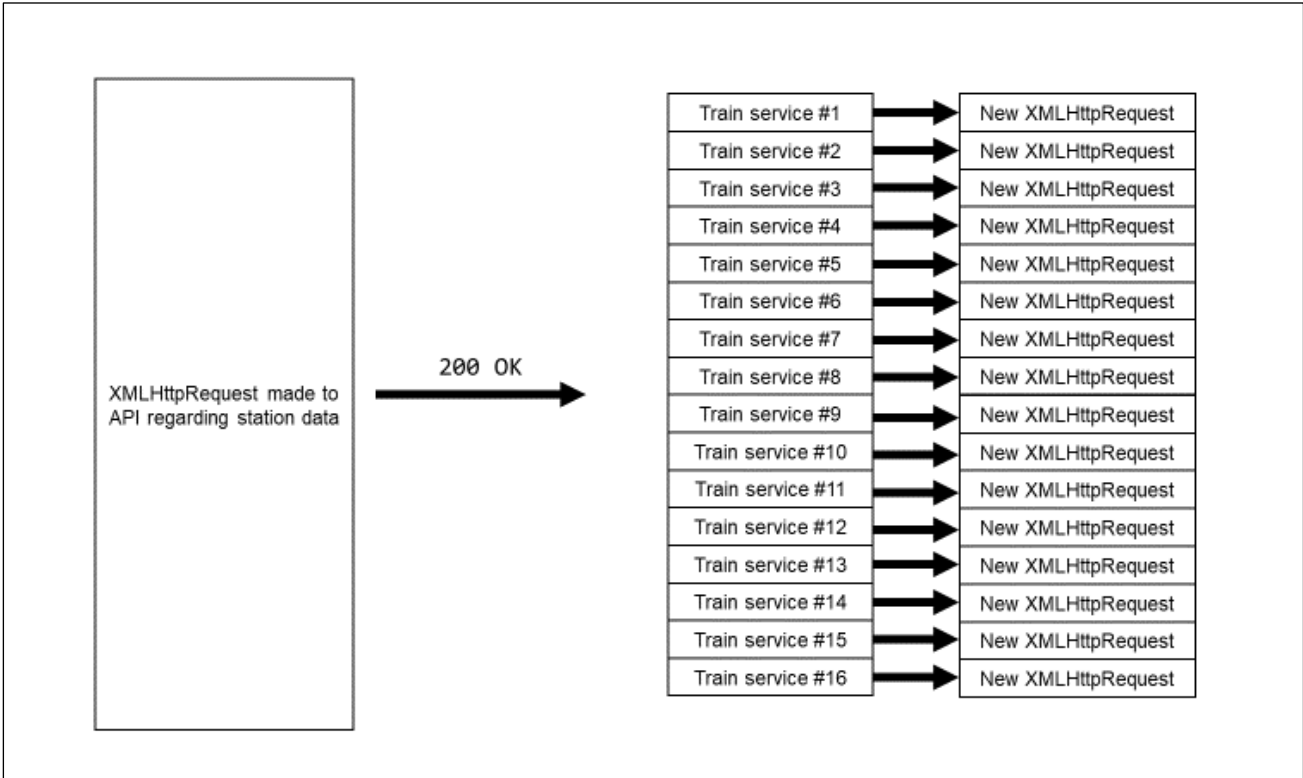
    if (typeof dest === 'undefined') {
        dest = 'none';
    }
    // 'Dest' is an optional parameter.
    // Should it not be included, its value will be set to none.

    var request = new XMLHttpRequest();

    request.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            obj = JSON.parse(this.responseText);
        }
    }
    request.open("GET", "https://transportapi.com/v1/uk/train/station" + code + "/" + date + "/" + time +
        "timetable.json?app_id=[appid]", true);
    request.send('code=' + code);
}
```

As shown above, the JavaScript function *updateData* would retrieve the data from the user input and store it in the *obj* variable. From here, a system of count-controlled loops would be used to iterate through the data and display it in an HTML table.

Limitation: Redundancy within requests



The TransportAPI had a fundamental limitation within retrieved requests; rail services returned by the main API call did not specify data regarding train stopping patterns. Instead the data retrieved would contain an *origin_station* string, a *destination_station* string and a *service_timetable* string which would contain a URL to another endpoint within the API.

For example, a request for services from West Ham Railway Station (with no destination input) will show only the start point and end point of said services (since it is a station request, not a train request).

	origin_name	destination_name	service_timetable
Random Example Services (requests here are for station, rather than train)	Fenchurch Street	Southend Central	[URL containing API request for this individual train]
One XHR needed	Fenchurch Street	Grays via Rainham	[URL containing API request for

			this individual train]
--	--	--	------------------------

This would not pose an issue unless the *dest* parameter is set, as without a *dest* parameter, the only data we need is what is shown above.

However, should we now make a request for services from West Ham to Basildon, for example, we would have to reference the *service_timetable* endpoint through multiple XMLHttpRequest, one for each train.

	origin_name	destination_name	service_timetable	New XHR	Includes <i>dest</i> ?	Display in table?
Random Example Services One XHR needed	Fenchurch Street	Southend Central	[URL]	+1 →	true	Yes
	Fenchurch Street	Grays	[URL]	+1 →	false	No
	(+ 14 more example services) *		[URL]	+ 14 → XHR needed		

(XHR refers to an XMLHttpRequest)

* the table displays 16 values by default.

Despite the impracticability of calling seventeen XMLHttpRequest per user input, the approach initially proved to be successful.

Evident issues soon arose regarding the speed of the application and then, problems with API request limits. Whilst the application initially used data from TransportAPI [<http://transportapi.com>], the limit of 1000 requests/day made it harder to develop the application.

Criteria (1) of the stakeholder requirements is the need for speed and simplicity. It was therefore decided that this approach was not feasible in the long term for the stakeholders, and the decision was soon made to change the API to RealTimeTrains, an example solution proposed earlier in this document.

API Migration

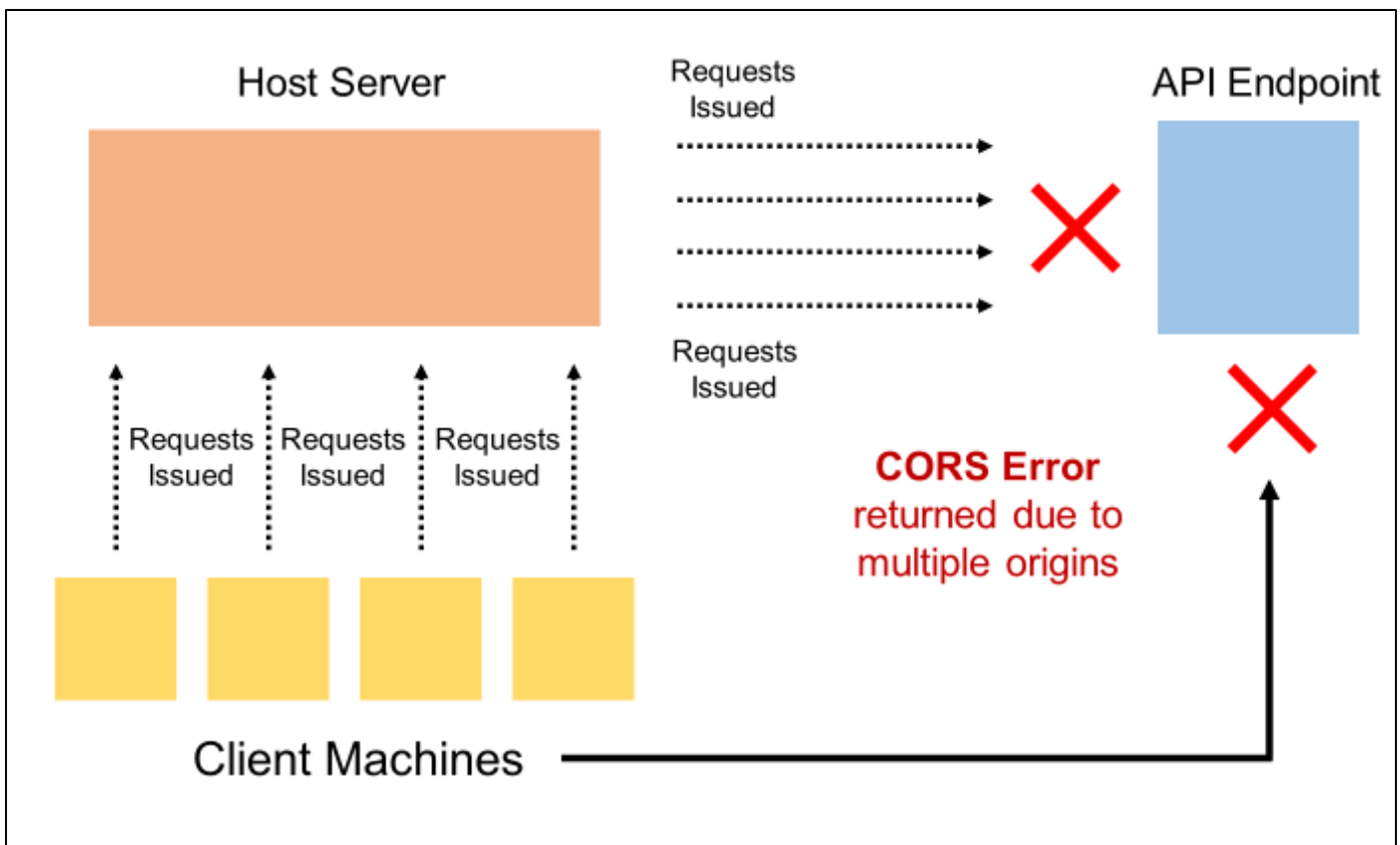
Switching the API proved challenging and numerous tasks had to be completed in order to maintain a working application.

References to object names and data values formatted to the previous API were inconsistent and therefore these references had to be updated to match the format of the new parsed JSON. This was not too difficult on its own, however it was amplified by further challenges.

One of the benefits of the new API was that a different endpoint was available for specifying a destination. The code was therefore reformatted to conditionally select which endpoint to send the request to depend on the status of the *dest* parameter. This greatly reduced the number of physical requests needing to be sent, maintaining stakeholder criteria (1) for speed and simplicity and ensuring limits are not exceeded.

Limitation: CORS and XMLHttpRequest

Cross-Origin Resource Sharing (CORS) is an internet security mechanism allowing access to restricted resources on a web server, where the original source of the request and the server sending the request are on different domains. Whilst the previous API's web server permitted CORS, the new one did not. This led to major problems with the fundamental structure of the code and its methods of retrieving the necessary data during testing, whereby the host machine and client machine are located on the same network.



CORS plays a significant role in maintaining user security in conventional web applications, particularly those in which requests are sent to an external server, such as for the APIs used in this application. In a situation where a client has accessed a host server exhibiting malicious intent and CORS is disabled, the JavaScript code executed may be able to tamper with the initial request, putting a user's privacy at risk. Hence, CORS ensures this cannot happen.

Separating Station Data and Train Data

The application will display data regarding services at a station. Each service returned, as displayed in the table, will link to another API URL containing individual train data, such as calling points, and the current location of the service.



The application will have a PHP file for the station service data, and another file for retrieving individual train data.

Ease of Navigation in the Application

To allow users to plan their routes easily, the application will make use of links within the pages, allowing users to link between stations and trains interchangeably, updating the time input.

This is best explained via an example. Let's say a user wants to plan a journey from Southend Victoria to Highbury & Islington, in London. This particular journey involves a change of train at Stratford. The user would be able to put 'Southend Victoria' as their input along with a time, to view all the trains available. Each result would be hyperlinked to the train data for that result. If a user clicks a train to view the train data, all the stopping points and their respective times are shown. The user would then click on 'Stratford' and be redirected back to the station page with an input of 'Stratford', and the time that particular service would arrive at Stratford. From here, the user can select the next train they need.

Accessibility Within the Site

The site will have certain options available within the settings pages for registered users. These will include the following:

- An option to increase the font size. This will benefit users with vision problems and make the application easier to use.
- A high contrast mode, whereby the site colours are inverted and only black and white are used. This, again, helps users with vision problems and also users with sensory issues who may be startled by bright screens.

Pathfinding

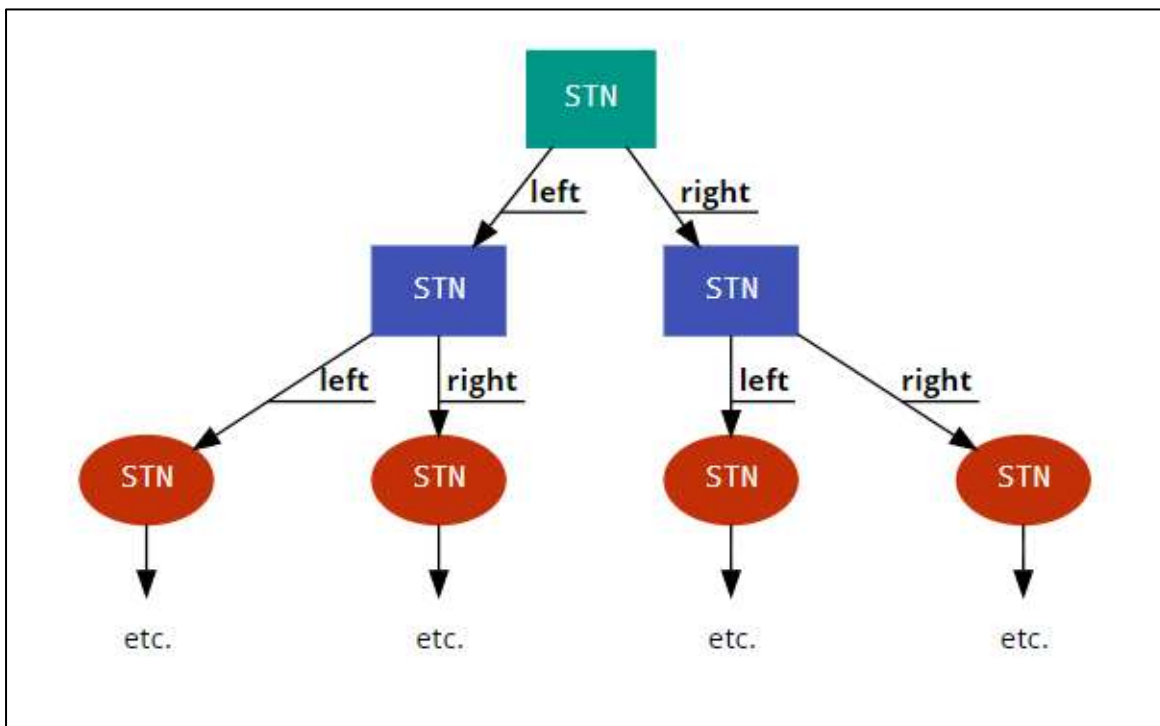
A*, Dijkstra and Heuristics

Outline

Ultimately, the pathfinder will make use of the A* algorithm, making use of a heuristic to maximise potential to find routes and efficiency. However, the pathfinder will be tested using both Dijkstra and A* to determine the level of efficiency achieved by using a heuristic over Dijkstra alone. This will be documented in the testing section.

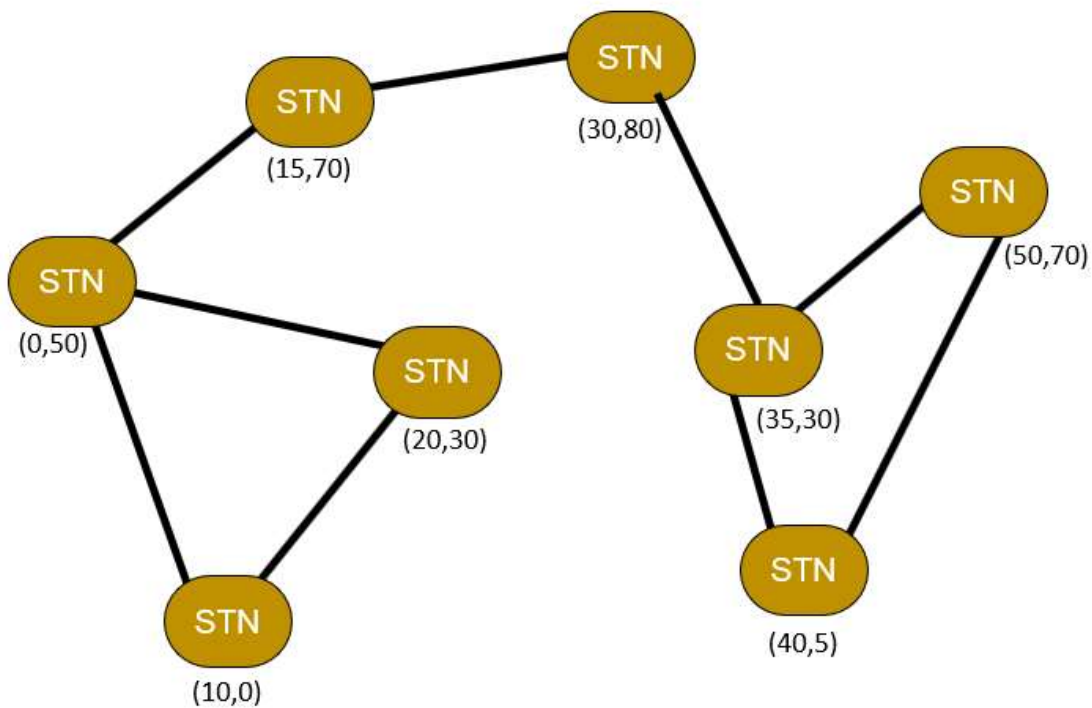
Start Point: Binary Tree Station Mapping

The pathfinder will make use of a binary tree initially in order to link stations together. A binary tree structure comprises a good start point for the development of the code as it is a simple way to begin to comprehend the task at hand and create a framework to build on later on. Limitations of the binary tree structure when compared to that of a node graph are described in the Development & Testing section.

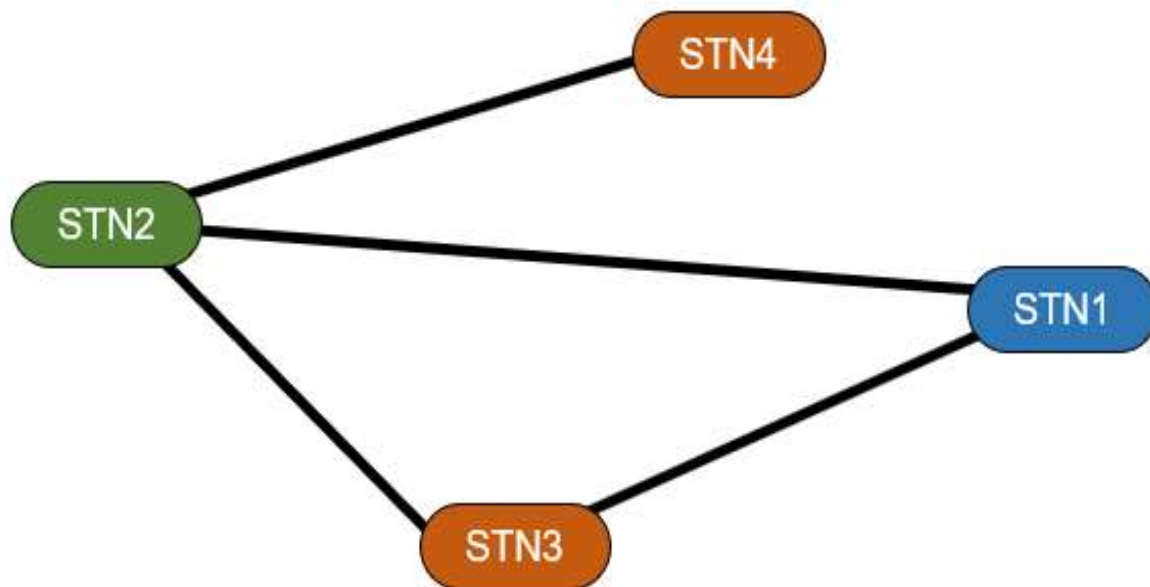


Next Stage: Node Graphing

The pathfinder will ultimately make use of a node graph data structure such as the one below. Nodes will be given an X and a Y coordinate. These will then be connected via linked list data structures to adjacent nodes with the distance between nodes given.



Breadth First Search (BFS)



Breadth First Search is a method of traversing a node graph so that all parent nodes are checked before child nodes are checked. This is as opposed to Depth First Search where a parent node's entire child structure will be checked before any sibling nodes are checked.

Dijkstra Algorithm

The Dijkstra algorithm can be used to find the distance between two points on the node graph via a breadth first search and a priority queue.

In the example above, a traversal from **STN1** to **STN4** using Dijkstra would follow the following algorithmic structure:

Get length STN1 to STN3

Get length STN1 to STN2

All parents of STN1 checked, move on to STN3

Get length STN3 to STN2

if $(\text{STN3 to STN2}) + (\text{STN1 to STN3}) < (\text{STN1 to STN2})$,
new length equals $(\text{STN1 to STN3}) + (\text{STN3 to STN2})$

else, new length equals (STN1 to STN2)

All parents of STN3 checked, move on to STN2

Get length STN2 to STN4

All parents of STN2 checked, move onto STN4

STN4 is our destination. newlength += (STN2 to STN4)

return newlength

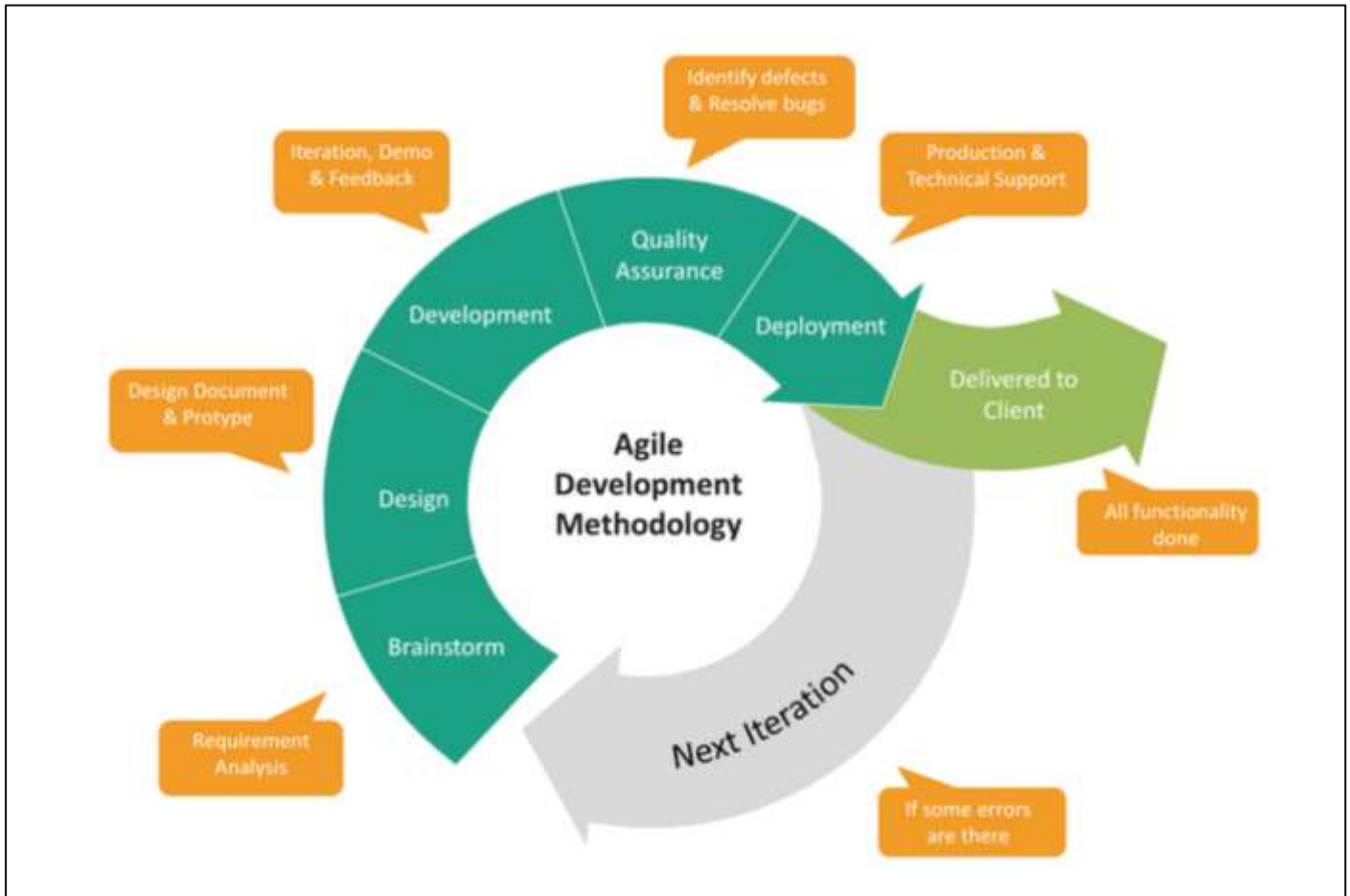
From Dijkstra to A*: Heuristics

From Wikipedia, the definition of a heuristic is as follows: A heuristic technique, or a heuristic, is any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect, or rational, but is nevertheless sufficient for reaching an immediate, short-term goal or approximation.

A heuristic can be thought of as an approximated shortcut to a problem using data that is already known. In a computational context, a heuristic can be thought of as a way of solving a problem more quickly by approximating the value of the solution via a heuristic function. Heuristics use discovery and analysis of trends in data to generate a 'rule of thumb' to apply to the data.

Development & Testing

Key Themes



- The application will follow the Agile Development Methodology as shown above until the stakeholder requirements are met and the stakeholders are satisfied with the application's usage.
- Development will take place in stages with feedback from the stakeholders at each stage. Phased development allows feedback to be submitted at each iteration which allows a more dynamic and tailored development for the application.

- The first stages of development will focus solely on functionality, with user-friendliness and ease of use implemented within the later stages.
- The proposal is to have as many stages of development as may be necessary.

Stage 1: Live Display Framework

Stage Objective

This is the first stage of development of the application. This section contains fundamental code, algorithms and logic diagrams added to the code in the early phases of development. At this stage, the data display and pathfinder are developed separately.

The first stage of development focuses in on developing a framework for a functioning data display without much emphasis on user friendliness or interface simplicity. Fine-tuning will be done at a later stage.

Development for Stage 1

Outline

The data display is integrated into the Graphical User Interface and makes use of PHP and SQL in order to store user authentication details, preferences and organise data within an HTML skeleton.

HTML Framework

At this stage of development, the data display is contained entirely within an HTML skeleton. During initial stages of development priority is placed on ensuring the code is functional and has a basic structure that the user can interpret without much focus on styling or comfort, which is focussed on in later stages of development.



The date and time input are currently done in a YYYY/MM/DD text format for the date, and an XXXX format for the time. Future development cycles may see this changed to autocomplete via the live search and the inputs may be combined to maximise user compatibility. The results are currently displayed in a text grid formatted as an HTML table, which is itself built using a JavaScript function.

localhost/result.php?stationcode=lb&destinationcode=&date=2020%2F12%2F07&time=1300

Welcome to Railmapper!

London Bridge

Time	Origin	Destination	Platform	Operator
1240	London Bridge	Couladon Town	12	Southern
1243	Ramsgate	London Charing Cross	9	Southeastern
1243	London Charing Cross	Sevenoaks	6	Southeastern
1243	London Cannon Street	London Cannon Street	2	Southeastern
1244	London Bridge	Caterham	11	Southern
1244	Orpington	London Cannon Street	3	Southeastern
1245	Gravesend	London Charing Cross	8	Southeastern
1245	Finsbury Park	Brighton	4	Thameslink
1246	Brighton	Cambridge	5	Thameslink
1247	London Bridge	Caterham	14	Southern
1247	Sevenoaks	London Charing Cross	9	Southeastern
1248	Luton	Rainham (Kent)	4	Thameslink
1249	London Charing Cross	Ramsgate	6	Southeastern
1250	Hayes (Kent)	London Cannon Street	3	Southeastern
1251	Gatwick Airport	Bedford	5	Thameslink
1251	London Charing Cross	Dartford	7	Southeastern
1251	Bedford	Gatwick Airport	4	Thameslink
1254	London Cannon Street	Orpington	2	Southeastern
1254	London Charing Cross	Hastings	6	Southeastern
1255	London Bridge	Epsom	13	Southern
1255	Hastings	London Charing Cross	9	Southeastern
1257	Dartford	London Charing Cross	8	Southeastern
1257	London Charing Cross	Gravesend	7	Southeastern

Data Retrieval

The data is retrieved in a JSON format, which is then iterated through a table constructor in order to display the data inside the HTML.

cURL is used to retrieve the data from the API.

```
<?php

// Get variables from user input

$stationcode = $_GET["stationcode"];
$date = $_GET["date"];
$time = $_GET["time"];
// Initialise cURL request
$curl_handle=curl_init();
curl_setopt($curl_handle, CURLOPT_USERPWD, "password");
```

```

curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, true);
curl_setopt($curl_handle, CURLOPT_URL,
https://apt.rtt.io/api/v1/json/search/.$stationcode."/".$date."/".$time);
$query = curl_exec($curl_handle);
curl_close($curl_handle);
?>

```

From here, the data is passed from PHP to JavaScript.

```

var obj = <?php echo $query;?>;
var i;

```

```

var obj = <?php echo $query;?>;
var i;

```

The JSON then becomes accessible to the DOM and can be manipulated on the client-side.

Example JSON Query

```

{
  "location": {
    "name": "London Liverpool Street",
    "crs": "LST",
    "tiploc": "LIVST"
  },
  "filter": null,
  "services": [
    {
      "locationDetail": {
        "realtimeActivated": true,
        "tiploc": "LIVST",
        "crs": "LST",
        "description": "London Liverpool Street",
        "gbttBookedDeparture": "0938",
        "origin": [
          {
            "tiploc": "LIVST",
            "description": "London Liverpool Street",
            "workingTime": "093800",
            "publicTime": "0938"
          }
        ],
        "destination": [
          {
            "tiploc": "CLCHRTN",
            "description": "Colchester Town",
            "workingTime": "104900",
            "publicTime": "1049"
          }
        ],
        "isCall": true,
        "isPublicCall": true,
        "realtimeDeparture": "0942",
        "realtimeDepartureActual": true,
        "platform": "10",
        "platformConfirmed": true,
        "platformChanged": false,
        "displayAs": "ORIGIN"
      },
      "serviceUid": "G71678",
      "runDate": "2021-01-07",
      "trainIdentity": "2F30",
      "runningIdentity": "2F30",
      "atocCode": "LE",
      "atocName": "Greater Anglia",
      "serviceType": "train",
      "isPassenger": true
    }
  ], {

```

(A single page call contains 24 of these).

An example of a section of the JavaScript **obj** variable when parsed is shown above. By default, 24 train services are retrieved per call. One is shown above.

The JSON is retrieved in a format which is human-readable and comprehensive, which can then be accessed via JS methods for retrieving object data. In the above example, the output of **obj.location.name** in the console would return 'London Liverpool Street'. This can then be passed into the HTML and displayed in the page.

```
document.getElementById("attribs").innerHTML = obj.location.name;
```

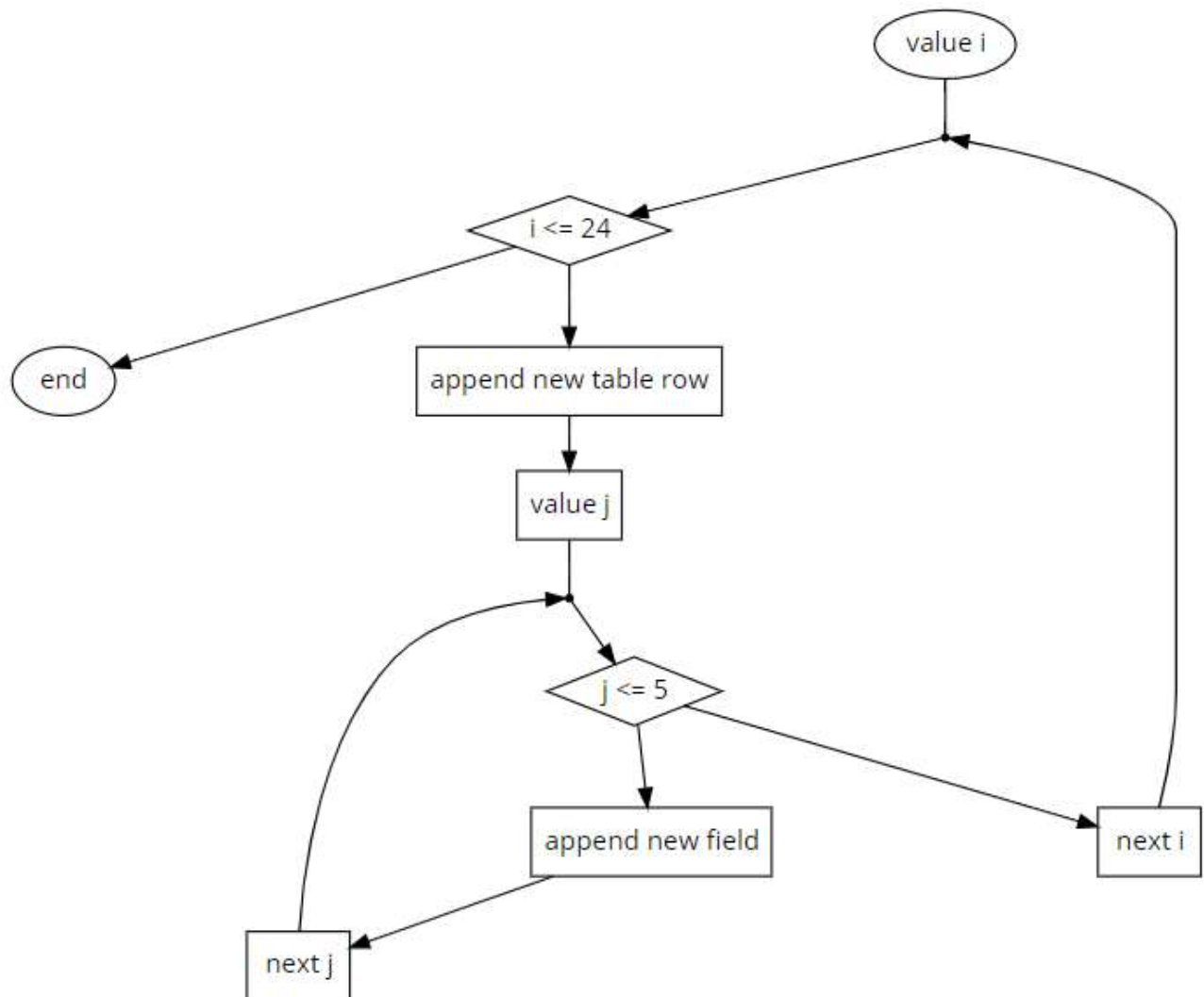
The data retrieved has a wide scope and use of the specifics can greatly improve the quality of the application in later development stages. The data distinguishes between whether platforms are confirmed for use before arrival (**Boolean platformConfirmed**), whether the train stops for staffing reasons or for passengers (**Boolean isPublicCall**) and also lists rail replacement bus services (**String serviceType**).

These options can be considered during development of the GUI in later stages; for example, displaying bus replacement services in italics or a different font colour or hiding staff trains entirely outside of the advanced mode.

Table Constructor

Once retrieved, the data is implemented into a visual HTML table via an iterating constructor function.

This involves a **for** loop nested inside another **for** loop.



```
var i;  
var j;  
var tr;  
var tdata;  
var currenttr;  
var datatable = document.getElementById("datatable");
```

```

    for (i=1;i<=24;i++){
        trow = document.createElement("TR");
        datatable.appendChild(trow);
        for (j=0;j<=5;j++){
            tdata = document.createElement("TD");
            trow.appendChild(tdata);
        };
    };

for (i=1;i<24;i++){
    currentrow = datatable.rows[i];
    currentrow.childNodes[0].innerHTML = obj.services[i-1].locationDetail.gbttBookedDeparture;
    currentrow.childNodes[1].innerHTML = obj.services[i-1].locationDetail.origin[0].description;
    currentrow.childNodes[2].innerHTML = obj.services[i-1].locationDetail.destination[0].description;
    currentrow.childNodes[3].innerHTML = obj.services[i-1].locationDetail.platform;
    currentrow.childNodes[4].innerHTML = obj.services[i-1].atocName;
};

```

JavaScript Table Constructor & Iterator

A number of variables are defined initially. **i** and **j** are iterative variables. Since the table can be visualised as a 2-dimensional plane, two iterator variables are needed to access every element: one for the vertical position of table rows, and another for the horizontal position of cells within these rows.

```

var i;
var j;
var trow;
var tdata;
b l

```

```
var currentrow;
```

Two variables are also needed for the data itself in a similar fashion Variable **trow** is defined as a blank table row element for each iteration of the loop, and **tdata** is defined as a blank table data element for each iteration of the second loop (explained below).

The API retrieves 24 results per query: with 5 values per result. Therefore, the iterator makes use of a **for** loop which will increment the value of variable **i** between values 0 and 24, and another nested **for** loop incrementing the value of variable **j** between values 0 and 5, for each value of **i**.

Once the table has been constructed, a loop can once again be used to add the retrieved JSON data to it. The built-in JavaScript **childNodes** object property is used to access the table.

```
for (i=1;i<24;i++){
    currentrow = datatable.rows[i];
    currentrow.childNodes[0].innerHTML = obj.services[i-1].locationDetail.gbttBookedDeparture;
    currentrow.childNodes[1].innerHTML = obj.services[i-1].locationDetail.origin[0].description;
    currentrow.childNodes[2].innerHTML = obj.services[i-1].locationDetail.destination[0].description;
    currentrow.childNodes[3].innerHTML = obj.services[i-1].locationDetail.platform;
    currentrow.childNodes[4].innerHTML = obj.services[i-1].atocName;
};
```

This gives a functional table displaying the results in a comprehensible format.

Testing for Stage 1

Outline

The data display has been tested with a range of inputs designed to mimic realistic inputs the program may receive. Testing identified a number of problems which are described in the **Stage 1 Review** below. These problems can then be fixed in the next stage of development.

Input Queries & Testing Results

Input Query #1: Control test using working values

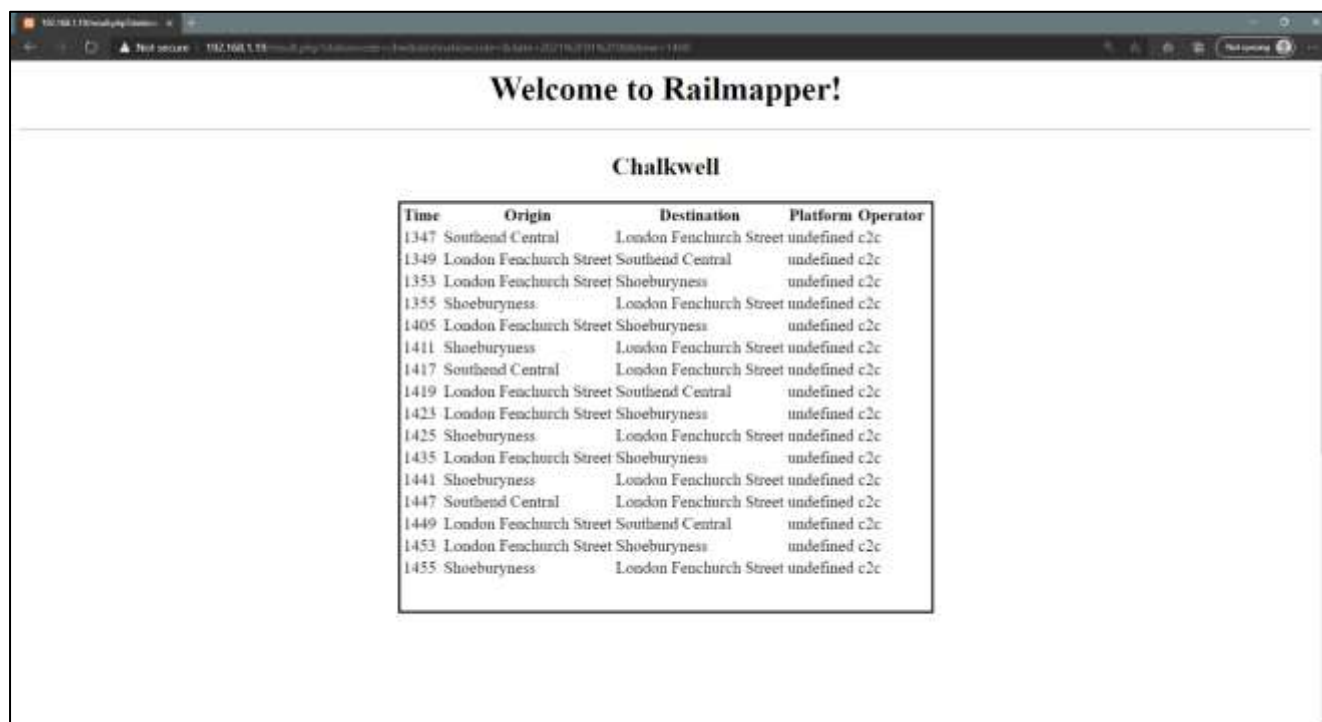
stationcode	"EUS"	date	"2020/01/08"	time	"1300"
-------------	-------	------	--------------	------	--------



Time	Origin	Destination	Platform	Operator
1243	London Euston	Edinburgh	4	Avanti West Coast
1245	London Euston	Watford Junction	9	London Overground
1246	London Euston	Crewe	12	West Midlands Trains
1251	London Euston	Milton Keynes Central	11	West Midlands Trains
1258	London Euston	Watford Junction	9	London Overground
1304	London Euston	Tring	10	West Midlands Trains
1307	London Euston	Liverpool Lime Street	14	Avanti West Coast
1310	London Euston	Glasgow Central	15	Avanti West Coast
1315	London Euston	Birmingham New Street	12	West Midlands Trains
1315	London Euston	Watford Junction	9	London Overground
1320	London Euston	Manchester Piccadilly	1	Avanti West Coast
1323	London Euston	Birmingham New Street	5	Avanti West Coast
1324	London Euston	Northampton	8	West Midlands Trains
1328	London Euston	Watford Junction	9	London Overground
1334	London Euston	Tring	10	West Midlands Trains
1340	London Euston	Manchester Piccadilly	3	Avanti West Coast
1343	London Euston	Blackpool North	6	Avanti West Coast
1345	London Euston	Watford Junction	9	London Overground
1346	London Euston	Crewe	12	West Midlands Trains
1354	London Euston	Milton Keynes Central	11	West Midlands Trains
1358	London Euston	Watford Junction	9	London Overground

Input Query #2: Test for station with no data for 'Platform'

stationcode	"CHW"	date	"2020/01/08"	time	"1400"
-------------	-------	------	--------------	------	--------



Chalkwell

Time	Origin	Destination	Platform	Operator
1347	Southend Central	London Fenchurch Street	undefined	c2c
1349	London Fenchurch Street	Southend Central	undefined	c2c
1353	London Fenchurch Street	Shoeburyness	undefined	c2c
1355	Shoeburyness	London Fenchurch Street	undefined	c2c
1405	London Fenchurch Street	Shoeburyness	undefined	c2c
1411	Shoeburyness	London Fenchurch Street	undefined	c2c
1417	Southend Central	London Fenchurch Street	undefined	c2c
1419	London Fenchurch Street	Southend Central	undefined	c2c
1423	London Fenchurch Street	Shoeburyness	undefined	c2c
1425	Shoeburyness	London Fenchurch Street	undefined	c2c
1435	London Fenchurch Street	Shoeburyness	undefined	c2c
1441	Shoeburyness	London Fenchurch Street	undefined	c2c
1447	Southend Central	London Fenchurch Street	undefined	c2c
1449	London Fenchurch Street	Southend Central	undefined	c2c
1453	London Fenchurch Street	Shoeburyness	undefined	c2c
1455	Shoeburyness	London Fenchurch Street	undefined	c2c

Input Query #3: Testing 'stationcode' variable for error handling

stationcode	"foobar"	date	"2020/01/08"	time	"1400"
-------------	----------	------	--------------	------	--------



Input Query #4: Testing 'date' for error handling: using hyphens instead of slashes

stationcode	"foobar"	date	"2020-01-08"	time	"1400"
-------------	----------	------	--------------	------	--------



Tests were all passed successfully.

Stage 1 Review

Achieved in This Stage

The data display is now functioning and displays the live data as required. At the moment there are no filter options, and the data is displayed in a mostly text-based format. The program can successfully take an input of a 3-digit station code, a time formatted as XXXX and a date formatted as YYYY/MM/DD and retrieve train operators, platforms, destinations, and origin points.

Improvements for Next Stages

The next task is, as requested by the stakeholders, to add an option to filter the input via train operators. This will provide great benefit to both regular travellers and advanced users, who have pointed out that the feature would speed up access for relevant data considerably.

Stage 2: Filterable Options

Stage Objective

This is the second cycle of development and builds off the skeleton structures shown in the previous section. The outlines shown previously are developed in this section to begin to create an application with levels of functionality. Graphical interfaces are still not prioritised in this section, although some aspects of graphical development may be considered as a gateway into full development of the user interface in this section.

Development for Stage 2

Filtering the Output (via Graphical Methods)

One of the key features requested by the stakeholders as a means of improving the delivery of the data is the ability to filter the output. In this stage, the option of filtering the results by the train operating company will be implemented.

The first stage in filtering the results by train operating company involves iterating through the returned API call and collecting the operator for each service. These results are then stored in a JavaScript array.

```
var operatorNames = new Array();
for(i=1;i<obj.services.length;i++){
    operatorNames.push(obj.services[i].atocName);
}
```

The next step is to remove duplicate values, so that we get an array of unique operator names. This can be done using a simple JavaScript **filter** method to create a new array which checks if values are already in the old array before adding to the new one. The new array directly replaces the old array.

```
operatorNames = operatorNames.filter((a,b) => operatorNames.indexOf(a) === b);
```

The next step is to iterate through the array and implement a clickable button into the DOM for each entry. This will create a series of buttons with the operator name given as the button text.

A variable, **filters**, is used to select the **<div>** with ID **filters** in the HTML DOM. From here, a button, variable **opButton**, is added to **filters** for each entry in the **operatorNames** array. **opButton** contains the operator name as the button label, and is given an HTML ID, allowing the button element to be accessed later with ease.

```
filters = document.getElementById("filters");
for(i=0; i < operatorNames.length; i++){
    opButton = document.createElement("BUTTON");
    opButton.innerHTML = operatorNames[i];
    filters.appendChild(opButton);
    opButton.style.margin = "1.5%";
    opButton.setAttribute("id","BTN-"+operatorNames[i]);
    opButton.setAttribute("onclick","filterTOC(\""+operatorNames[i]+"\")");
};
```

The button is set to call the filterTOC function, with the parameter of the button's operator name. The function is defined below.

```
function filterTOC(currentoperator) {
    button = document.getElementById("BTN-" + currentoperator);
    for (i = 1; i < 24; i++) {
        currentrow = datatable.rows[i];
        if (currentrow.childNodes[4].innerHTML == currentoperator) {
            if (currentrow.style.display === "none") {
                currentrow.style.display = "table-row";
                button.style.opacity = 1;
            } else {
                currentrow.style.display = "none";
                button.style.opacity = 0.6;
            }
        }
    }
}
```

```

};

};

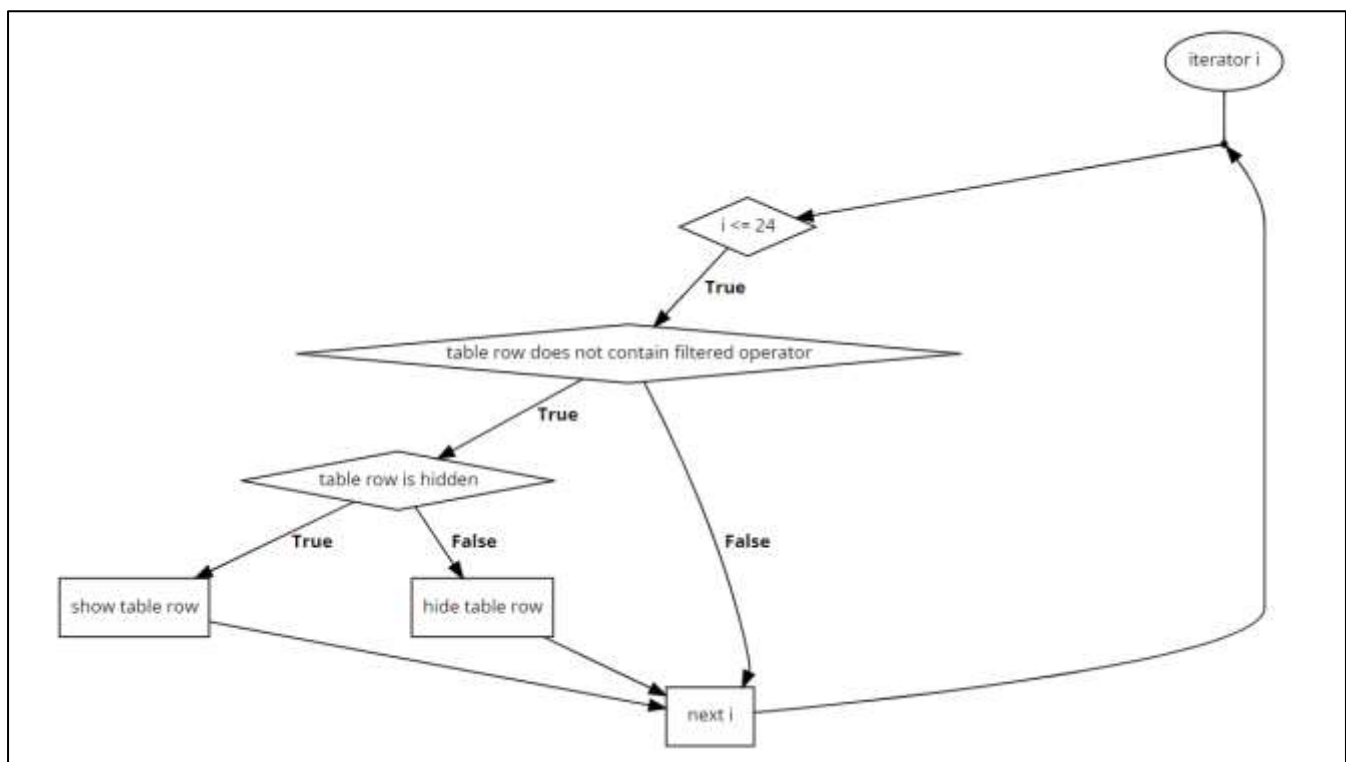
};

};

```

The function variable **button** is set to the button element that calls the function itself. This allows editing of the button opacity.

From here, the table rows are iterated through again, with a check being performed each time as to whether the current row contains the operator requested for filtration. If it does not, the table rows are hidden. If the button is pressed again, the table rows are replaced.



The **filters** <div> is itself displayed at the bottom of the page in a fixed footer. This allows the buttons to be toggled with ease, without the position of the buttons themselves moving when the table height is changed through the removal of rows.

Testing for Stage 2

Outline

In order to test the functionality added in Stage 2, a station with a high number of train operators was chosen in the input query in order to test that the filterable options work appropriately.

Input Queries & Results

Query given:

stationcode	"LIV"	date	"2021/01/29"	time	"1300"
-------------	-------	------	--------------	------	--------

With all filterable options activated (default):

The screenshot shows a web browser window with the URL `saenutheirpad/moad.php?stationcode=LiverpoolLimeStreet&date=2021%2F01%2F29&time=1300`. The page title is "Welcome to Railmapper!". Below the title, the station name "Liverpool Lime Street" is displayed. A table of train departures is shown, with columns for Time, Origin, Destination, Platform, and Operator. The table lists 20 train services. At the bottom of the page, there is a search bar and a row of buttons for different train operators: Arriva West Coast, Merseyrail, East Midlands Railway, Transpennine Express, Northern, West Midlands Trains, and Transport for Wales.

Time	Origin	Destination	Platform	Operator
1243	Chester	Chester	undefined	Merseyrail
1247	Liverpool Lime Street	London Euston	10	Avanti West Coast
1248	New Brighton	West Kirby	A	Merseyrail
1251	Liverpool Lime Street	Norwich	6	East Midlands Railway
1254	Liverpool Lime Street	Newcastle	3	Transpennine Express
1255	Liverpool Lime Street	Manchester Oxford Road	7	Northern
1258	Ellenborough Park	Ellenborough Park	A	Merseyrail
1308	Liverpool Lime Street	Birmingham New Street	10	West Midlands Trains
1313	Chester	Chester	undefined	Merseyrail
1315	Liverpool Lime Street	Wigan North Western	2	Northern
1318	West Kirby	New Brighton	A	Merseyrail
1320	Liverpool Lime Street	Manchester Oxford Road	6	Northern
1327	Liverpool Lime Street	Manchester Airport	4	Northern
1328	Ellenborough Park	Ellenborough Park	A	Merseyrail
1333	New Brighton	West Kirby	A	Merseyrail
1336	Liverpool Lime Street	Chester	10	Transport for Wales
1337	Liverpool Lime Street	Blackpool South	3	Northern
1342	Chester	Chester	undefined	Merseyrail
1347	Liverpool Lime Street	London Euston	9	Avanti West Coast
1354	Liverpool Lime Street	Newcastle	3	Transpennine Express
1357	Liverpool Lime Street	Manchester Oxford Road	7	Northern
1358	Ellenborough Park	Ellenborough Park	A	Merseyrail

With no filterable options activated:

Welcome to Railmapper!

Liverpool Lime Street

Time Origin Destination Platform Operator

Start Station: Platform XXX

End Station: Platform XXX

2021/01/08

Time: 09:00:00

Submit

Username

Password

Submit

Search Your Query Home East Midlands Railway Transpennine Express Northern West Midlands Railway Transport for Wales

With one filterable option activated:

Welcome to Railmapper!

Liverpool Lime Street

Time	Origin	Destination	Platform Operator
1255	Liverpool Lime Street	Manchester Oxford Road 7	Northern
1315	Liverpool Lime Street	Wigan North Western 2	Northern
1326	Liverpool Lime Street	Manchester Oxford Road 6	Northern
1327	Liverpool Lime Street	Manchester Airport 4	Northern
1337	Liverpool Lime Street	Blackpool North 3	Northern
1355	Liverpool Lime Street	Manchester Oxford Road 7	Northern

Start Station: Platform XXX

End Station: Platform XXX

2021/01/08

Time: 09:00:00

Submit

Username

Password

Submit

Search Your Query Home East Midlands Railway Transpennine Express Northern West Midlands Railway Transport for Wales

With two filterable options activated:

Not secure | saenutlrakpad@msub.php/ctvrmvzade+PdcJctvrmvzade+Route=20210110/1250/line=1202

Welcome to Railmapper!

Liverpool Lime Street

Time	Origin	Destination	Platform	Operator
1251	Liverpool Lime Street	Norwich	6	East Midlands Railway
1253	Liverpool Lime Street	Manchester Oxford Road	7	Northern
1313	Liverpool Lime Street	Wigan North Western	2	Northern
1320	Liverpool Lime Street	Manchester Oxford Road	6	Northern
1327	Liverpool Lime Street	Manchester Airport	4	Northern
1337	Liverpool Lime Street	Blackpool North	3	Northern
1353	Liverpool Lime Street	Manchester Oxford Road	7	Northern

Start Station: Liverpool LSC

End Station: Preston XGB

20210110

Time: From: 0000

Submit

Username:

Password:

Submit

Search Your Query

Home

East Midlands Railway

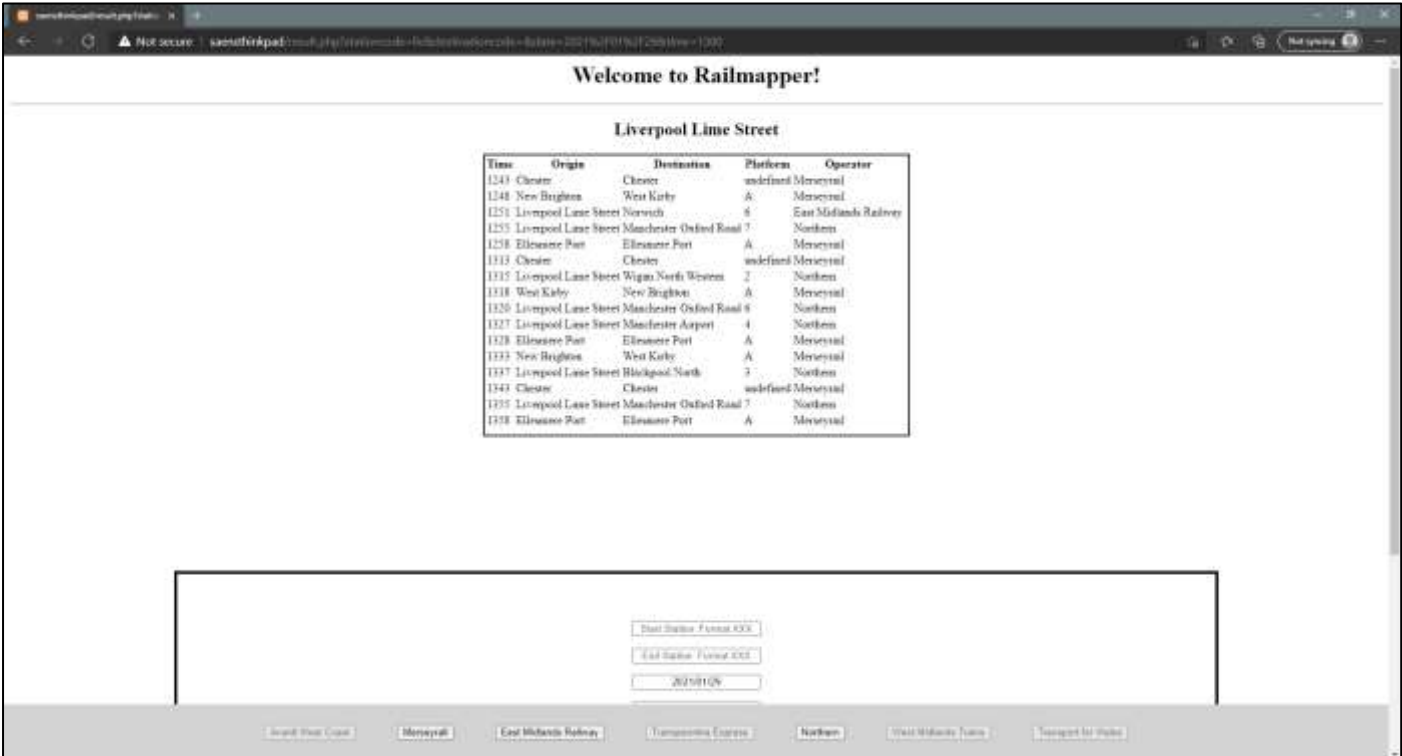
Transpennine Express

Northern

West Midlands Railway

Transport for Wales

With three filterable options activated:



The screenshot shows the Railmapper website interface. At the top, it says "Welcome to Railmapper!". Below this, the title "Liverpool Lime Street" is displayed. A table lists train services with columns for Time, Origin, Destination, Platform, and Operator. The table contains 18 rows of data. Below the table, there are three input fields: "Start Station: Format XXX", "End Station: Format XXX", and "20250109". At the bottom, there are several filterable options: "Search Your Query", "Merseyrail", "East Midlands Railway", "Transpennine Express", "Northern", "East Midlands Railway", and "Transport for London".

Time	Origin	Destination	Platform	Operator
1241	Chesam	Chesam	undefined	Merseyrail
1248	New Brighton	West Kirby	A	Merseyrail
1251	Liverpool Lime Street	Norwich	8	East Midlands Railway
1255	Liverpool Lime Street	Manchester Oxford Road	7	Northern
1258	Ellesmere Port	Ellesmere Port	A	Merseyrail
1312	Chesam	Chesam	undefined	Merseyrail
1315	Liverpool Lime Street	Wigan North Western	2	Northern
1318	West Kirby	New Brighton	A	Merseyrail
1329	Liverpool Lime Street	Manchester Oxford Road	8	Northern
1327	Liverpool Lime Street	Manchester Airport	4	Northern
1328	Ellesmere Port	Ellesmere Port	A	Merseyrail
1333	New Brighton	West Kirby	A	Merseyrail
1337	Liverpool Lime Street	Blackpool North	3	Northern
1341	Chesam	Chesam	undefined	Merseyrail
1355	Liverpool Lime Street	Manchester Oxford Road	7	Northern
1358	Ellesmere Port	Ellesmere Port	A	Merseyrail

These tests were all passed successfully.

Stage 2 Review

Achieved in This Stage

The train operator filter is now fully functioning and active and works as planned. The data is still displayed in a text-based format, and can now take an input, and then take further user input via the form of buttons in order to streamline the output.

Improvements for Next Stages

The filterable options have a number of minor inconsistencies which can be ironed out in later stages, particularly in development of graphics (Stage 5). These inconsistencies are as follows:

- When no filterable options are selected, the results table is simply shown to be empty. A better solution would be to implement a special case scenario for when no options are selected, allowing the user to know that no options are selected and clear all filters. This can be implemented in the next stage, which focuses on adding specifics.
- Resizing: As the options are added or removed to the input query, the table resizes to fit the new content. A fixed, albeit device-variable table would solve this problem. The challenge here is to ensure that text will continue to fit the table in all circumstances. This can be implemented in the development of graphics (Stage 5).
- When there are a lot of filterable options, it can become difficult to select/deselect all of them at once. An additional button to select all/deselect all can be added. This can be implemented in the next stage.

Stage 3: Implementing More Data

Stage Objective

Currently, the application achieves part of its task. Retrieval of the train services at a particular station allows the stakeholders to know when trains are, what platform the trains will be on and the final destination of the train. However, the application in its current stage has a number of pitfalls of which this stage of development will attempt to rectify.

Destination Indicators

The current layout of the application assumes the user will know whether or not their train stops at the station they need to get off at. The application shows start and end points and expects the user to know the route. However, trains (especially on high service routes) will often skip certain stations or include other stations that the user may not be aware of.

For example, trains on the c2c rail route are often timetabled in the afternoon to separate the student cohort that travels from Southend to London each day. In order to avoid overcrowding, some stations will receive a service from all trains, some trains will run into London, skipping stops through southwest Essex before stopping at stations in east London, while other trains will carry south Essex passengers and then skip east London stations afterwards. For a traveller, this can be confusing and may result in them ending up on a train that does not stop at their station, since only the endpoint is shown.

In order to solve this problem, two aspects of development need to be implemented. Firstly, use of the destination box on the input will ensure that only trains that stop at the given station will be displayed to the user. Secondly, each train in the table needs to have a link to a page that will:

- retrieve another API call for the given train service
- display all the station points for the given train service.

Development for Stage 3

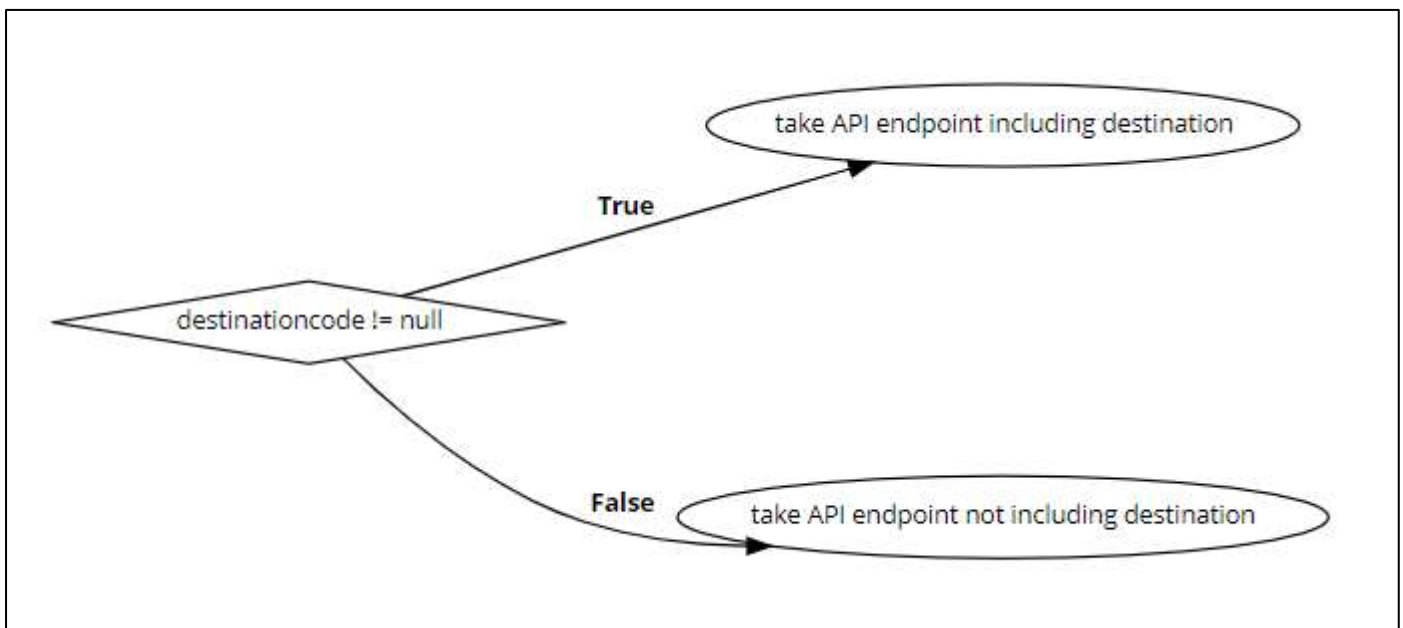
Determining which API query to use

If a destination station is specified, a different API call needs to be used.

```
$stationcode = $_GET["stationcode"];  
$destinationcode = $_GET["destinationcode"]; // Stage 3 Added  
$date = $_GET["date"];  
$time = $_GET["time"];
```

Initially, a PHP variable is added to retrieve the **destinationcode** from the HTML input.

From here a conditional statement is used to determine which CURL request pathing to take.



This will depend on whether the user has left the **destinationcode** box blank or has filled it in. Should the box be left blank, all services at a station will be shown regardless of endpoint or direction.

```
$curl_handle=curl_init();  
  
curl_setopt($curl_handle, CURLOPT_USERPWD, "password");
```

```

curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, true);

// This IF was added in Stage 3

if ($destinationcode == ""){
    curl_setopt($curl_handle,
CURLOPT_URL,"https://api.rtt.io/api/v1/json/search/" . $stationcode . "/" . $date . "/"
.$time);
} else {
// Added in Stage 3
curl_setopt($curl_handle,
CURLOPT_URL,"https://api.rtt.io/api/v1/json/search/" . $stationcode . "/to/" . $desti
nationcode . "/" . $date . "/" . $time);
}
$query = curl_exec($curl_handle);
curl_close($curl_handle);

```

Increasing Service Detail

The next task is to ensure that individual train services can be accessed.

For each API request to a station that is returned, a service UID is returned along with each train service. This links to the individual train data.

Firstly, for each element in the table, an HTML link needs to be created to link to another page, which will then return the service detail.

```

for (i = 1; i < 24; i++) {

    trainLink = document.createElement("a");
    trainLink.setAttribute("href", "./trainservice.php?serviceuid=" +
        obj.services[i - 1].serviceUid);
    currentrow = datatable.rows[i];
    currentrow.childNodes[0].innerHTML =
        "<a href='./trainservice.php?serviceuid=" + obj.services[i -
//

```

```

        1].serviceUid + "&date=<?php echo $date;?>'>" + obj
        .services[i - 1].locationDetail.gbttBookedDeparture;
currentrow.childNodes[1].innerHTML =
    "<a href='./trainservice.php?serviceuid=" + obj.services[i -
        1].serviceUid + "&date=<?php echo $date;?>'>" + obj
        .services[i - 1].locationDetail.origin[0].description;
currentrow.childNodes[2].innerHTML =
    "<a href='./trainservice.php?serviceuid=" + obj.services[i -
        1].serviceUid + "&date=<?php echo $date;?>'>" + obj
        .services[i - 1].locationDetail.destination[0].description;
currentrow.childNodes[3].innerHTML =
    "<a href='./trainservice.php?serviceuid=" + obj.services[i -
        1].serviceUid + "&date=<?php echo $date;?>'>" + obj
        .services[i - 1].locationDetail.platform;
currentrow.childNodes[4].innerHTML =
    "<a href='./trainservice.php?serviceuid=" + obj.services[i -
        1].serviceUid + "&date=<?php echo $date;?>'>" + obj
        .services[i - 1].atocName;
};

```

The table constructor from Stage 1 has been edited to include an HTML **anchor** for each element, creating a link. For each row in the returned query, a new element is created.

The link goes to another file, **trainservice.php**, appending the serviceUID to the end of the file via a PHP GET variable in the URL.

```

"<a href='./trainservice.php?serviceuid="+obj.services[i-
1].serviceUid+"&date=<?php echo $date;?>'>"

```

Inside this file, another cURL request is called to access the service data.

```

// File: trainservice.php

```

```

$date = $_GET['date'];
$serviceuid = $_GET['serviceuid'];

$curl_handle=curl_init();
curl_setopt($curl_handle, CURLOPT_USERPWD,
"rttapi_saenkazak"." ":"2b6ad3b266f6daeb92fa47f6c6ceca776db10f06");
curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, true);
curl_setopt($curl_handle,
CURLOPT_URL,"https://api.rtt.io/api/v1/json/service/" . $serviceuid . "/" . $date);

$query = curl_exec($curl_handle);
curl_close($curl_handle);

?>

```

The **date** and **serviceuid** variables are taken from the GET attribute of the URL

The service data will then be retrieved and displayed in a table constructor similar to the last file, albeit with different data.

Table Construction

```

document.getElementById('operator').innerHTML = obj.atocName;
document.getElementById('rundate').innerHTML = obj.runDate;
var i;
var j;
var datatable = document.getElementById("datatable");
for (i = 0; i < obj.locations.length - 1; i++) {
    currentrow = document.createElement("TR");
    datatable.appendChild(currentrow);
    for (j = 0; j < 3; j++) {

```



```

        var tablecell = document.createElement("TD")
        currentrow.appendChild(tablecell);
    }
    for (j = 0; j < 3; j++) {
        currentrow.childNodes[0].innerHTML =
obj.locations[i].gbttBookedDeparture[0] +
obj.locations[i].gbttBookedDeparture[1] + ":" +
obj.locations[i].gbttBookedDeparture[2] +
obj.locations[i].gbttBookedDeparture[3];

        currentrow.childNodes[1].innerHTML = "<a
href='./result.php?stationcode=" + obj.locations[i].crs + "&date=" + date +
"&destinationcode=&time=" + obj.locations[i].realtimeDeparture + "'>" +
obj.locations[i].description;

        currentrow.childNodes[2].innerHTML = obj.locations[i].platform;
    }
}
lastRow = document.createElement("TR");
datatable.appendChild(lastRow);
for (j = 0; j < 3; j++) {
    lastRow.appendChild(document.createElement("TD"))
};
lastObj = obj.locations[obj.locations.length - 1];

```

Firstly, the operator name and running date are applied to the HTML skeleton. From here, a table is created in the skeleton and a table row is created for each returned train service to be appended to the table via the iterating loop shown above. Additionally, and similarly to the previous table, an internal iterating loop is used to add three table cells to each row. From here, another loop will add the corresponding departure time, the station name (within a link, described below), and a platform number.

Linking Back to the Original File

One of the key elements in enabling the program to be easily navigated is allowing the user to link from stations to train services and then back to stations, maintaining the correct time input and updating it as required. This would allow a user that may need to check a journey involving multiple trains to be able to do so. (This is explained in the design section). In order to ensure this is possible, the **innerHTML** for the second child node of each table row will have a link embedded.

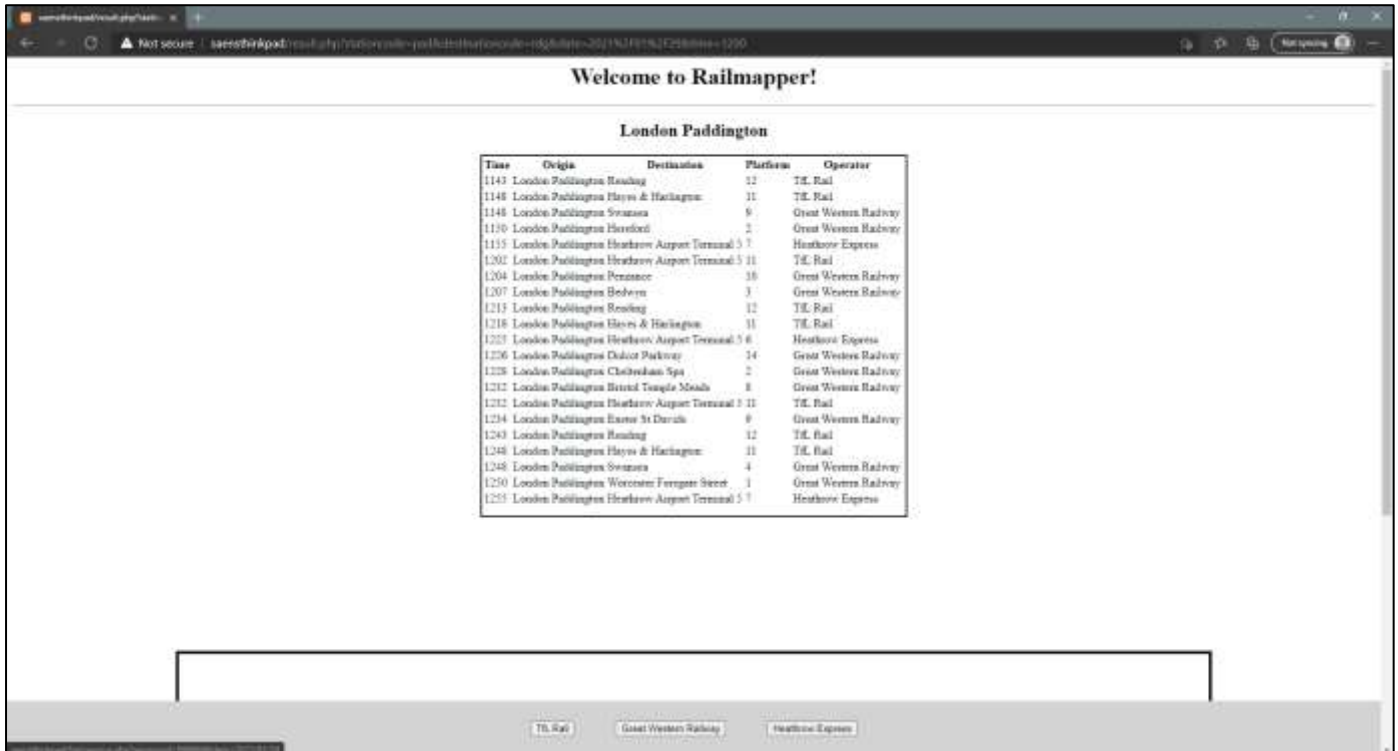
```
currentrow.childNodes[1].innerHTML =  
"<a  
href='./result.php?stationcode="+obj.locations[i].crs+"&date="+date+"&destinati  
oncode=&time="+obj.locations[i].realtimeDeparture+"'>" +obj.locations[i].descrip  
tion;
```

The **&time** variable in the URL input, is taken from the time the train arrives at the station, as opposed to the earlier time input from the original query.

Testing for Stage 3

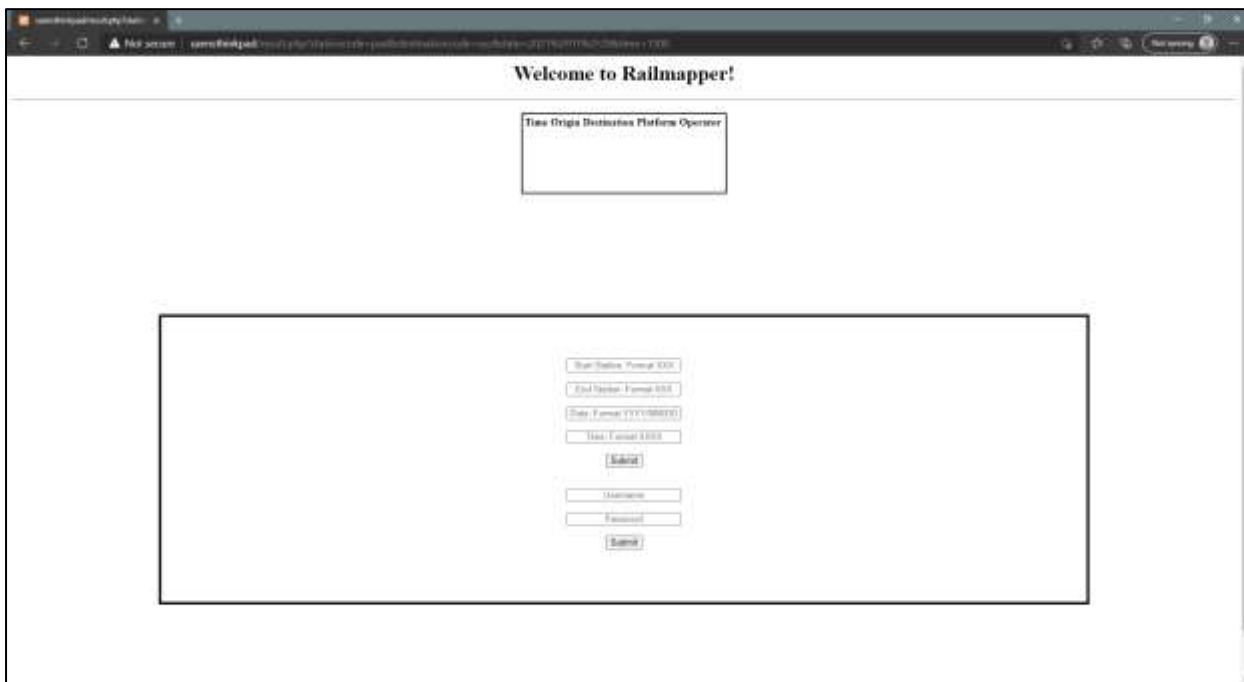
Control test with a destination set:

stationcode	"PAD"	destinationcode	"RDG"	date	"2021/01/29"	time	"1200"
-------------	-------	-----------------	-------	------	--------------	------	--------



Test with an invalid destination set:

stationcode	"PAD"	destinationcode	XYZ	date	"2021/01/29"	time	"1200"
-------------	-------	-----------------	-----	------	--------------	------	--------



This test passed successfully. However, it will need to be tested again once error handling is implemented in the next stage.

Testing hyperlinks:

stationcode	"IVR"	destinationcode	"RDG"	date	"2021/02/02"	time	"1200"
-------------	-------	-----------------	-------	------	--------------	------	--------

Welcome to Railmapper!

Iver

Time	Origin	Destination	Platform	Operator
11:29	London Paddington	Reading	3	TfL Rail
12:00	London Paddington	Reading	3	TfL Rail
12:30	London Paddington	Reading	3	TfL Rail

Start Station: Format XXX

End Station: Format XXX

2021/02/02

Time: Format XXXX

Submit

Username

Password

Submit

TfL Rail

Third option selected:

Operated by TfL Rail

Service on 2021-02-02 activated 12:13 at London Paddington

Time	Station	Platform
12:13	London Paddington	12
12:21	Leamington	5
12:28	Stratford	5
12:31	Heathrow	5
12:35	Windsor	5
12:38	Leamington	5
12:40	Leamington	5
12:44	Stratford	4
12:48	Stratford	1
12:51	Leamington	5
12:54	Leamington	5
13:01	Leamington	5
13:13	Leamington	14

Last option selected, expected for time value to change from 12:00 to 13:13

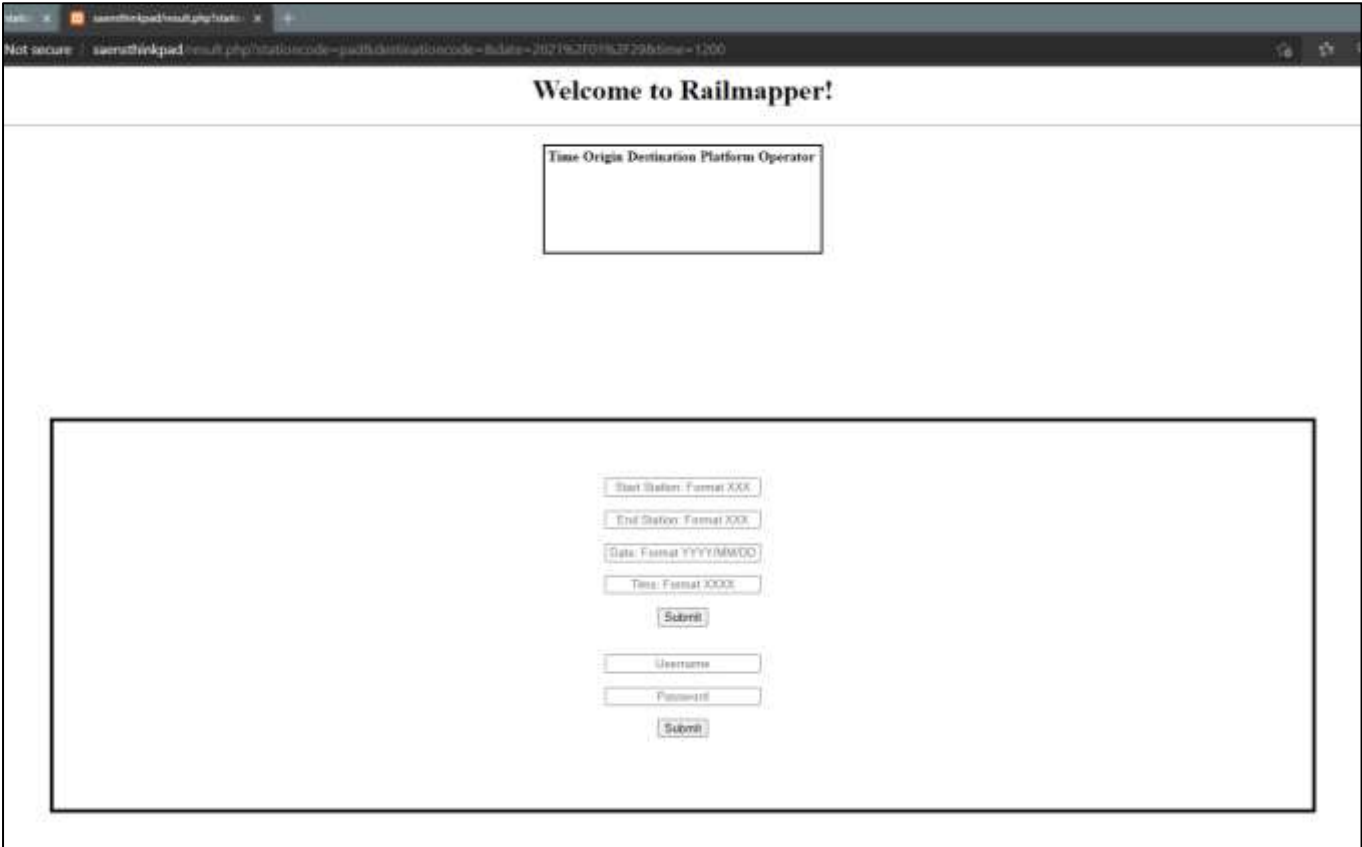
Errors During Development

This section posed a lot of issues in developing, involving invalid character recognitions, API limitations and the need for some error handling to be implemented. Errors are described alongside development in this section.

Errors in Destination Input Handling

Upon initial testing, the code failed to achieve its purpose. An input of 'pad' for London Paddington, whilst leaving the 'end station' destination indicator blank, returned a blank result. This suggests the new added code broke the initially used code and must be modified.

stationcode	"PAD"	destinationcode		date	"2021/01/29"	time	"1200"
-------------	-------	-----------------	--	------	--------------	------	--------



Removing the &destinationcode= from the search result yielded a functioning display, albeit with an error message regarding the **destinationcode** variable at the top.

Notice: Undefined index: destinationcode in C:\xampp\htdocs\railmap.php on line 57

Welcome to Railmapper!

London Paddington

Time	Origin	Destination	Platform	Operator
1143	London Paddington Reading		12	TfL Rail
1148	London Paddington Hayes & Harlington		11	TfL Rail
1148	London Paddington Swansea		9	Great Western Railway
1158	London Paddington Hereford		2	Great Western Railway
1155	London Paddington Heathrow Airport Terminal 5		7	Heathrow Express
1202	London Paddington Heathrow Airport Terminal 5		11	TfL Rail
1204	London Paddington Penzance		10	Great Western Railway
1207	London Paddington Bodva		3	Great Western Railway
1213	London Paddington Reading		12	TfL Rail
1218	London Paddington Hayes & Harlington		11	TfL Rail
1223	London Paddington Heathrow Airport Terminal 5		4	Heathrow Express
1226	London Paddington Dulou Parkway		14	Great Western Railway
1228	London Paddington Chelmsford Spa		2	Great Western Railway
1232	London Paddington Heathrow Airport Terminal 5		11	TfL Rail
1232	London Paddington Bristol Temple Meads		8	Great Western Railway
1234	London Paddington Exeter St Davids		9	Great Western Railway
1243	London Paddington Reading		12	TfL Rail
1248	London Paddington Hayes & Harlington		11	TfL Rail
1248	London Paddington Swansea		4	Great Western Railway
1250	London Paddington Worcester Foregate Street		1	Great Western Railway
1253	London Paddington Heathrow Airport Terminal 5		7	Heathrow Express

Upon closer inspection, the error is shown to be caused by the PHP **is_null** function. Despite being unset, the **destinationcode** variable is not recognised as **null** by the program.

This can be fixed by modifying the following line of code from:

```
if(is_null($destinationcode))
```

to:

```
if($destinationcode=="")
```

The initial query was repeated following this change to give a successful test result.

Not secure | saeuthinkpad/mob.../londonpaddington?date=2021%2F01%2F2000line=1200

Welcome to Railmapper!

London Paddington

Time	Origin	Destination	Platform	Operator
1143	London Paddington	Reading	12	TfL Rail
1148	London Paddington	Hayes & Harlington	11	TfL Rail
1148	London Paddington	Swans	9	Great Western Railway
1150	London Paddington	Hitcham	2	Great Western Railway
1153	London Paddington	Heathrow Airport Terminal 5	7	Heathrow Express
1202	London Paddington	Heathrow Airport Terminal 5	11	TfL Rail
1204	London Paddington	Perth	10	Great Western Railway
1207	London Paddington	Swans	3	Great Western Railway
1213	London Paddington	Reading	12	TfL Rail
1218	London Paddington	Hayes & Harlington	11	TfL Rail
1223	London Paddington	Heathrow Airport Terminal 5	8	Heathrow Express
1226	London Paddington	Dulou Parkway	14	Great Western Railway
1228	London Paddington	Chesham Spa	2	Great Western Railway
1232	London Paddington	Heathrow Airport Terminal 5	11	TfL Rail
1232	London Paddington	Bristol Temple Meads	8	Great Western Railway
1234	London Paddington	Swans	9	Great Western Railway
1243	London Paddington	Reading	12	TfL Rail
1248	London Paddington	Hayes & Harlington	11	TfL Rail
1248	London Paddington	Swans	4	Great Western Railway
1250	London Paddington	Worcester Foregate Street	1	Great Western Railway
1253	London Paddington	Heathrow Airport Terminal 5	7	Heathrow Express

TfL Rail | Great Western Railway | Heathrow Express

Errors in Date Input Handling

```
var obj = <?php echo $query;?>;
var date = <?php echo htmlspecialchars($date);?>;
date = obj.runDate.toString().replace("-", "/").replace("-", "/");
```

Two minor problem arose when dealing with the 'date' input query:

- Initially, the date as taken from the API would be returned in the format YYYY-MM-DD, although a YYYY/MM/DD format (including slashes instead of hyphens) was needed for the string to be added to a URL successfully. This problem was solved by applying the JS **replace** function. This function swaps the '-' character for a '/' character. (The function had to be called twice as each instance only replaces the first character found).
- Following on from this error, retrieval of the date made the program interpret the '/' characters in the date as a mathematical expression, of which it would then divide to retrieve a completely invalid value for the date. A value of '2021/01/30', for example, would return a date of '67.3666667.' This problem was solved by applying the JS **toString** function to the **obj.runDate** object,

allowing the value to be interpreted as a string rather than a mathematical expression.

API Limitation Errors

An important limitation of the API being used was discovered in this stage of development.

Upon checking the RealTimeTrains website, an error message is presented in a certain circumstance: the API cannot retrieve data about a train's stopping points if the train has run more than 24 hours in the past. This is not something that is able to be bypassed.



SEARCH RESTRICTION

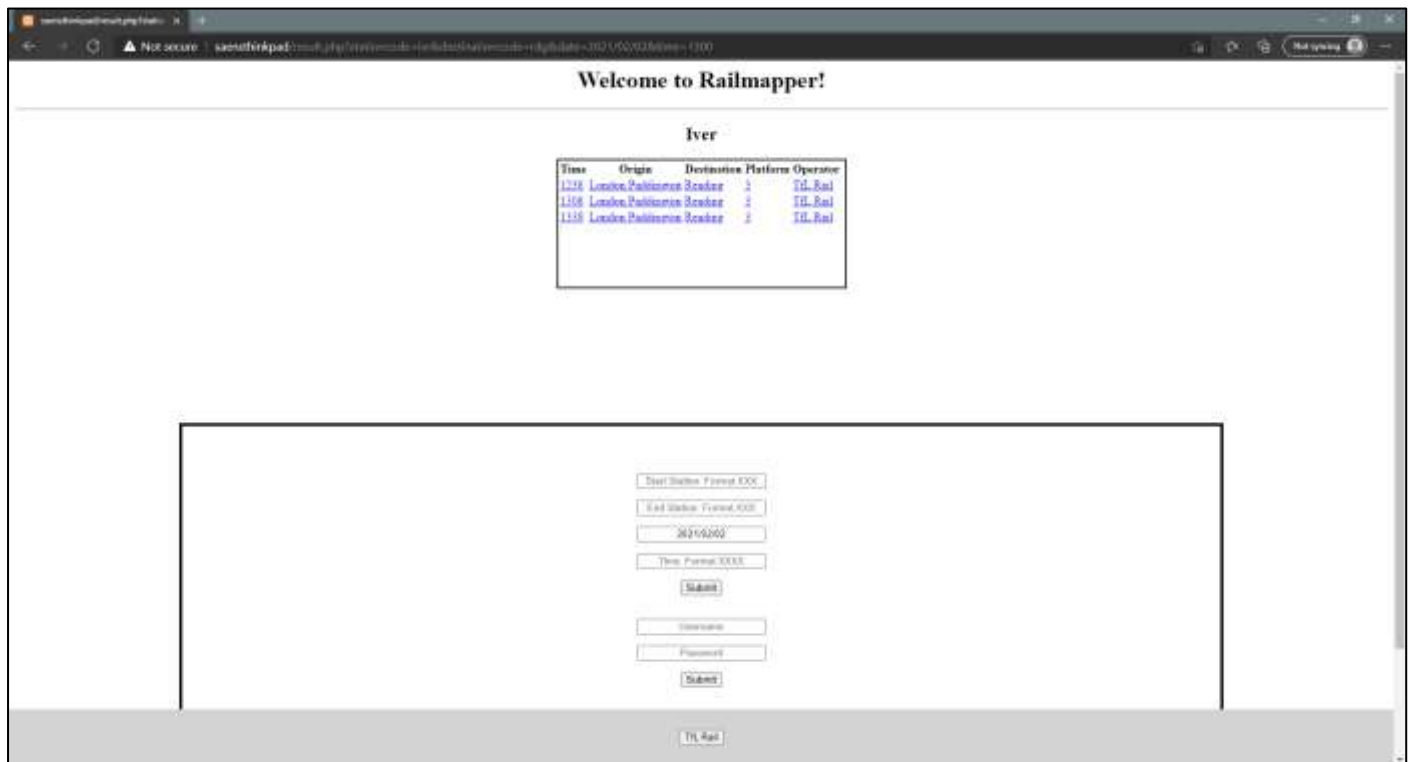
Due to a system restriction, we cannot filter by prior or future calling points if a train more than 24 hours in the past.
This constraint has been removed for your search results.

Therefore, this needs to be considered when feedback for error handling is implemented in a future stage of development.

Errors in Testing Hyperlinks

By now, the links between train pages and station pages have been added in. Tests were run to determine their functionality with different input queries

stationcode	"IVR"	destinationcode	"RDG"	date	"2021/01/31"	time	"1200"
-------------	-------	-----------------	-------	------	--------------	------	--------



Upon selecting the first train service in the list, the station points failed to show.

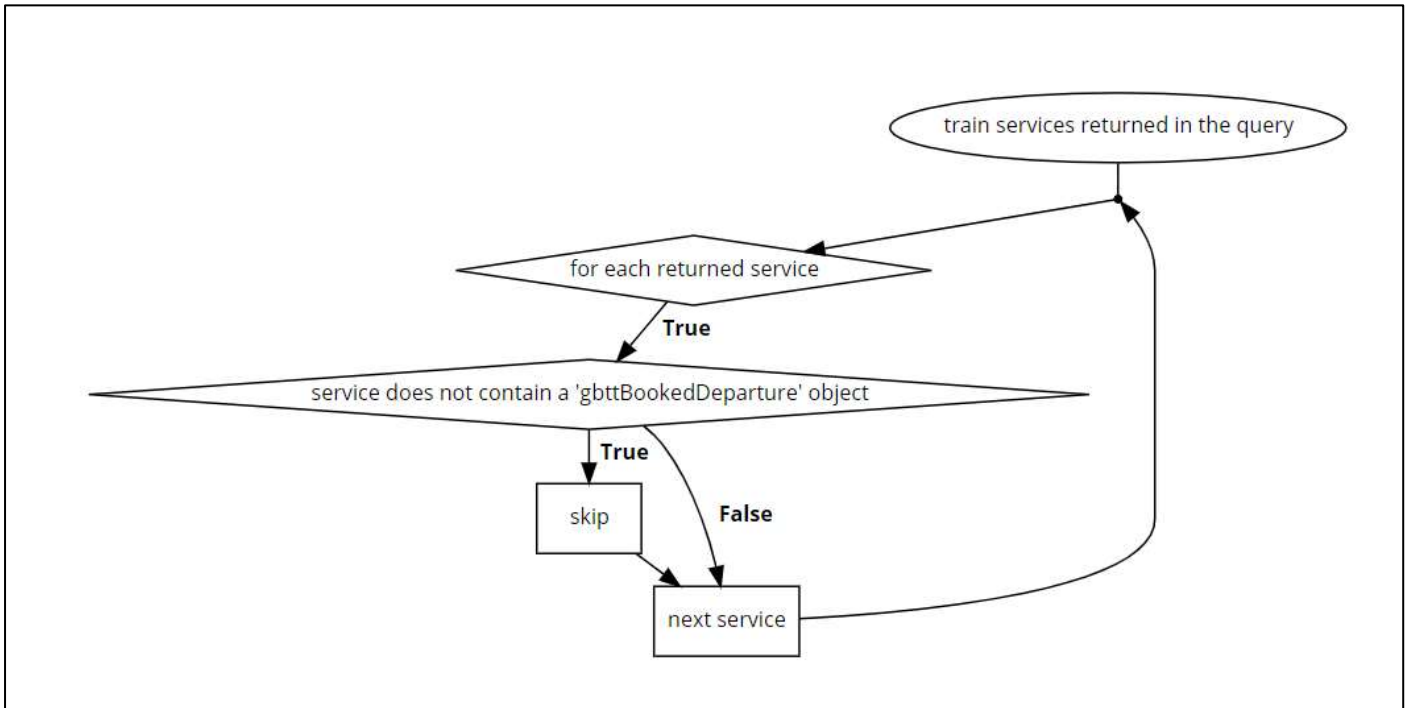


Inspecting the JSON query returned revealed the following:

- For train services more than 24 hours in the future, additional points for important junctions (and other locations that are not stations) are listed in the train query. This occurs because the train has not been 'activated' in railway systems.

- Train points that are not stations do not have a **gbttBookedDeparture** object within their query (as they do not 'depart' or 'stop'). This causes the iterator to break when retrieving this object.

In order to fix this issue, a check for each result to verify whether it is a station point is needed. Should this return false, the point should be skipped.



```
// Verify if the service has a 'gbttBookedDeparture', else skip
if(obj.locations[i].hasOwnProperty("gbttBookedDeparture") == false){continue;}
```

Following the implementation of this small verification, the test completed successfully.

Operated by TfL Rail

Service on 2021-02-01 activated 12:13 at London Paddington

Time	Station	Platform
12:13	London Paddington	12
12:21	Reading	5
12:28	Reading	5
12:31	Reading	5
12:35	Reading	5
12:38	Reading	5
12:40	Reading	5
12:44	Reading	4
12:48	Reading	1
12:51	Reading	3
12:54	Reading	3
13:01	Reading	3
13:13	Reading	14

Stage 3 Review

Achieved in This Stage

This stage has been monumentally significant in the development of the application as a useable software as opposed to a draft concept. This stage has seen the implementation of:

- Detailed service information regarding specific trains, developed in a separate file
- Use of a separate part of the API to streamline data down to specifics as required
- Dynamic navigation throughout the site, allowing the user to navigate between train descriptions and service descriptions easily
- Fixing of numerous errors as well as discovery of API limitations

Improvements for Next Stage

The work completed in this stage is a solid framework for extension in the next stage. The next tasks are to:

- Improve error handling as displayed to the user, make the user aware of errors, including issues with the input query or API restrictions
- Break queries down into data desired by advanced users and data desired by regular travellers.

Stage 4: User Functionality

Stage Objective

At this point in the development of the application, the data display functions as intended. The next stage is to increase the useability of the application and tailor the services towards the different stakeholders.

User Account System for Advanced Use

For advanced users, a user account system will be in place. It will function in the following ways:

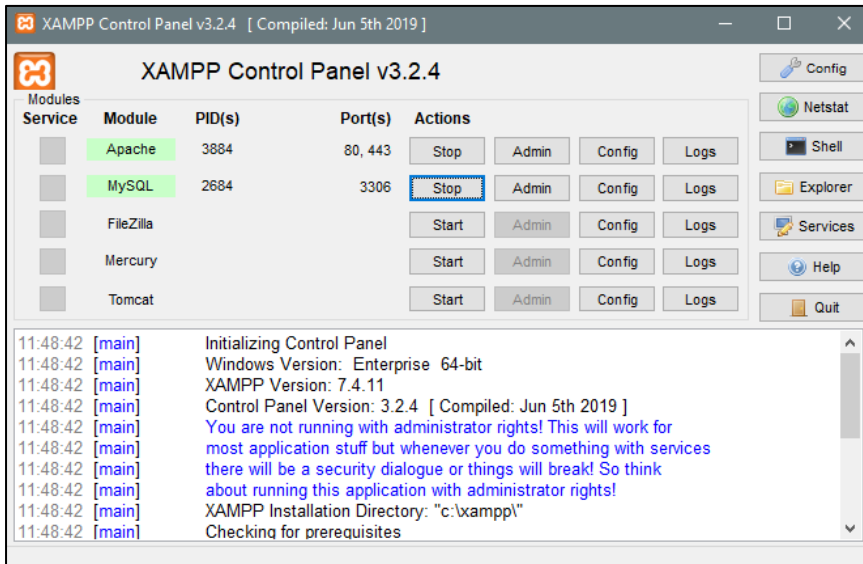
- All users can access the 'regular traveller' information without the need for an account
- Users with a valid account can access the 'advanced user' information, and the 'regular traveller' information if they desire.
- Users with an account can save their favourite stations, destinations, and times of day for quick access.

The database will operate using MySQL within the PHP environment.

Development for Stage 4

Enabling MySQL

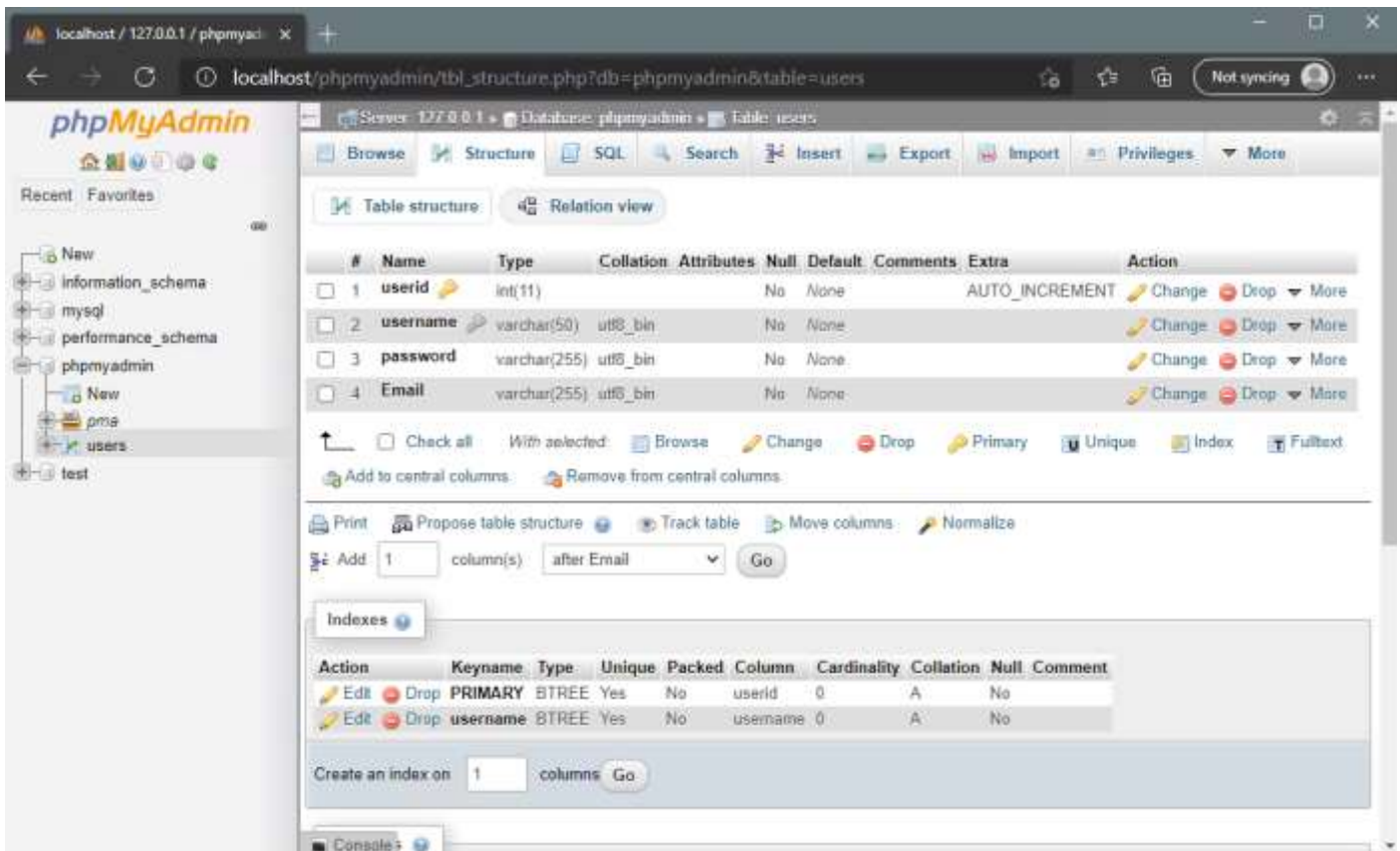
The very first step is to activate MySQL in the XAMPP control panel.



From here, MySQL can be accessed via the phpMyAdmin SQL query box.

Creating a User Account Table

```
CREATE table users (  
    userid INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    username VARCHAR(50) NOT NULL UNIQUE,  
    password VARCHAR(255) NOT NULL,  
    email VARCHAR(255) NOT NULL  
);
```



This code creates a new table within the default 'phpmyadmin' database with four columns: **userid**, which stores a unique integer for each username and will increment for each entry added (hence creating a primary key), **username**, which is a string and is also unique to each user, and **password** and **email** which are also strings and do not have to be unique.

PHP Configuration File

The next step is to connect to the database from PHP. A new file, **config.php** is created and the database information is included.

```
// file: config.php
<?php
define('DATABASE_SERVER','localhost');
define('DATABASE_USERNAME','localhost');
define('DATABASE_PASSWORD','');
define('DATABASE_NAME','phpmyadmin');
/* Connecting to the database */
```



```

$connect =
mysqlis_connect(DATABASE_SERVER,DATABASE_USERNAME,DATABASE_PASSWORD,DATABASE_NAME)

if ($connect === false){
    die("Error: Could not connect to the authentication database. Please try again later.");
}

?>

```

Creating the Registration Form

Once this has been done, the form allowing the user to register an account needs to be created. Another new file, **register.php**, is created and the file above is included within it. The code is described with annotations as to how it works below:

```

<?php
// file: register.php
// Require config.php to run

require_once "../config.php";

// Variable Definition

$username = "";
$password = "";
$password_confirm = "";
$email = "";

$username_err = "";
$password_err = "";
$password_confirm_err = "";
$email_err = "";

```

A number of variables are defined first and set to be blank. These variables are the given username, password, confirmation of password and email, along with a variable for each data type that will store an error message for display should an error arise in creation.

```

if($_SERVER["REQUEST_METHOD"] == "POST"){

```

If the page is loaded via the POST method, we need to verify that the four inputs are all valid. We start with the username:

```
// Ensure username entered
if(empty(trim($_POST["username"]))) {
    $username_err = "Please enter a username.";
} else {
    $query = "SELECT userid FROM users WHERE username = ?";

    if($stmt = mysqli_prepare($connect, $query)){
        // Bind parameters to the sql query
        mysqli_stmt_bind_param($stmt, "s", $param_username);
        $param_username = trim($_POST["username"]);

        if(mysqli_stmt_execute($stmt)){
            mysqli_stmt_store_result($stmt);

            if(mysqli_stmt_num_rows($stmt)==1){
                $username_err = "Username taken. Please try again.";
            } else {
                $username = trim($_POST["username"]);
            }
        } else {
            echo "Something went wrong, please try again later.";
        }
        mysqli_stmt_close($stmt);
    }
};
```

Firstly, we check if the username POSTed is not simply a blank variable. If it is not, we create a variable **query** which contains an SQL query to select the specified username.

From here we utilise the **connect** variable to prepare a statement for SQL execution, a new variable **param_username** contains the result of the POSTed username. The **param_username** value is then checked against the table rows to ensure it does not already exist. If it does, **username_err** is set to the 'username taken' error message. Otherwise, the initial **username** variable is set to the POSTed username. In case the database fails to connect, another error message is shown.

Now the username has been verified, we need to verify the password:

```
// Ensure password entered

if(empty(trim($_POST["password"]))){
    $password_err = "Please enter a password.";
} elseif(strlen(trim($_POST["password"])) < 6){
    $password_err = "Your password is too short. Please enter at least 6
characters.";
} else{
    $password = trim($_POST["password"]);
}

// Ensure password confirm validated
if(empty(trim($_POST["password_confirm"]))){
    $password_confirm_err = "Please confirm your password.";
} else{
    $password_confirm = trim($_POST["password_confirm"]);
    if(empty($password_err) && ($password != $password_confirm)){
        $password_confirm_err = "Passwords did not match, please try
again";
    }
}
}
```

Next, we check that a password has been entered. If blank, the **password_err** variable is set to a message that tells the user this.

Following this we check to ensure the password is longer than 6 characters. Again, if it is not, the **password_err** is set to a message telling the user to use a longer password.

This is then repeated with the password confirmation, with an error message thrown if the **password_confirm** variable does not match the **password** variable.

```
// Ensure email entered
if(empty(trim($_POST['email']))){
    $email_err = "Please enter your email address.";
} elseif(filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $email_err = "Invalid email format. Please try again.";
} else {
    if($stmt = mysqli_prepare($connect, $query)){
        // Bind parameters to the sql query
```

```

        mysqli_stmt_bind_param($stmt, "s", $param_email);
        $param_email = trim($_POST["email"]);

        if(mysqli_stmt_execute($stmt)){
            mysqli_stmt_store_result($stmt);

            if(mysqli_stmt_num_rows($stmt)==1){
                $email_err = "Email already in use. Maybe you signed up
already?";
            } else {
                $email = trim($_POST["email"]);
            }
        } else {
            echo "Something went wrong, please try again later.";
        }
    }
}

```

A similar process is repeated for the confirmation of email, along with an additional verification that the input matches an email format, and a verification for ensuring the email does not already exist in the table.

```

// Check input errors before inserting in database

    if(empty($username_err) && empty($password_err) &&
empty($password_confirm_err) && empty($email_confirm_err)){

        // Statement to insert into database prepared
        $sql = "INSERT INTO users (username, password, email) VALUES (?, ?,
?)" ;

        if($stmt = mysqli_prepare($connect, $sql)){
            // Bind variables to the prepared statement as parameters
            mysqli_stmt_bind_param($stmt, "sss", $param_username,
$param_password, $param_email);

            // Set parameters
            $param_username = $username;
            $param_password = password_hash($password, PASSWORD_DEFAULT);

            // Password hash used to increase security
            $param_email = $email;

            // Attempt to execute the prepared statement

```

```

        if(!mysqli_stmt_execute($stmt)) {
            echo "Something went wrong. Please try again later.";
        }

        // Close statement
        mysqli_stmt_close($stmt);
    }
}

// Close connection
mysqli_close($connect);
}
};
?>

```

Following the completion of the PHP section, the next step is to create an HTML register form, which will post the query to the PHP in the same page.

```

<div class="wrapper">
<h2>Sign Up</h2>
<p>Please fill this form to create an account.</p>
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
method="post">
<div class="form-group <?php echo (!empty($username_err)) ? 'has-error' : '';
?>">
    <label>Username</label>
    <input type="text" name="username" class="form-inputs" value="<?php echo
$username; ?>">
    <span class="help-block"><?php echo $username_err; ?></span>
</div>
<div class="form-group <?php echo (!empty($password_err)) ? 'has-error' : '';
?>">
    <label>Password</label>
    <input type="password" name="password" class="form-inputs" value="<?php echo
$password; ?>">
    <span class="help-block"><?php echo $password_err; ?></span>
</div>
<div class="form-group <?php echo (!empty($password_confirm_err)) ? 'has-error'
: ''; ?>">
    <label>Confirm Password</label>
    <input type="password" name="password_confirm" class="form-inputs"
value="<?php echo $password_confirm; ?>">
    <span class="help-block"><?php echo $password_confirm_err; ?></span>
</div>

```

```

<div class="form-group <?php echo (!empty($email_err)) ? 'has-error' : ''; ?>">
  <label>Email</label>
  <input type="text" name="email" class="form-inputs" value="<?php echo $email;
?>">
  <span class="help-block"><?php echo $email_err; ?></span>
</div>
<div class="form-group">
  <input type="submit" class="btn btn-primary" value="Submit">
  <input type="reset" class="btn btn-default" value="Reset">
</div>

```

This code does the following:

- Creates an input field for each column in the database and a submit button
- Creates a label to display an error message, should one arise, for each input field.

Creating the Login Form

To accompany the registration form, a login form needs to be created. The login form will attempt to match the username and password to a value that already exists in the database, returning an error if the value does not exist. Another new file, **login.php**, is created for this purpose.

```

<?php

// Initialise PHP session
session_start();

// Check if the user is already logged in, if yes then redirect to the
index.php file
if(isset($_SESSION["loggedin"]) && $_SESSION["loggedin"] === true){
    header("location: index.php");
    exit;
}

require_once('config.php');

```

Logins will not require a email address. Once again, the username and password are set to blank initially, along with their error messages.

```
$username = "";
$password = "";
$username_err = "";
$password_err = "";

if($_SERVER["REQUEST_METHOD"] == "POST"){

    // Check if username is empty
    if(empty(trim($_POST["username"]))){
        $username_err = "Please enter username.";

    } else{
        $username = trim($_POST["username"]);
    }

    // Check if password is empty
    if(empty(trim($_POST["password"]))){
        $password_err = "Please enter your password.";

    } else{
        $password = trim($_POST["password"]);
    }

    // Validate credentials
    if(empty($username_err) && empty($password_err)){
        // Prepare a select statement
        $sql = "SELECT id, username, password FROM users WHERE username = ?";
```

```

if($stmt = mysqli_prepare($connect, $sql)){
    // Bind variables to the prepared statement as parameters
    mysqli_stmt_bind_param($stmt, "s", $param_username);

    // Set parameters
    $param_username = $username;

    // Attempt to execute the prepared statement
    if(mysqli_stmt_execute($stmt)){
        // Store result
        mysqli_stmt_store_result($stmt);

        // Check if username exists, if yes then verify password
        if(mysqli_stmt_num_rows($stmt) == 1){
            // Bind result variables
            mysqli_stmt_bind_result($stmt, $id, $username,
$hashed_password);
            if(mysqli_stmt_fetch($stmt)){
                if(password_verify($password, $hashed_password)){
                    // Password is correct, so start a new session
                    session_start();

                    // Store data in session variables
                    $_SESSION["loggedin"] = true;
                    $_SESSION["id"] = $id;
                    $_SESSION["username"] = $username;

                    // Redirect user to welcome page

```



```

        header("location: index.php");
    } else{
        // Display an error message if password is not valid
        $password_err = "The password you entered was not
valid.";
    }
}
} else{
    // Display an error message if username doesn't exist
    $username_err = "No account found with that username.";
}
} else{
    echo "Something went wrong, Please try again later.";
}

// Close statement
mysqli_stmt_close($stmt);
}
}

// Close connection
mysqli_close($link);
}
?>

```

Following the PHP, an HTML form needs to be created for login.

```

<html>
<body>
    <div class="wrapper">
        <h2>Login</h2>
    </div>

```

```

    <p>Login Page</p>
    <form action="" method="post">
        <div class="form-group <?php echo (!empty($username_err)) ? 'has-error' : ''; ?>">
            <label>Username</label>
            <input type="text" name="username" class="form-control"
value="<?php echo $username; ?>">
            <span class="help-block"><?php echo $username_err; ?></span>
        </div>
        <div class="form-group <?php echo (!empty($password_err)) ? 'has-error' : ''; ?>">
            <label>Password</label>
            <input type="password" name="password" class="form-control">
            <span class="help-block"><?php echo $password_err; ?></span>
        </div>
        <div class="form-group">
            <input type="submit" class="btn btn-primary" value="Login">
        </div>
        <p>Or alternatively,<a href="register.php"> sign up.</a></p>
    </form>
</div>
</body>
</html>

```

The design for the login page is very similar to that of the registration page.

Modifying the Welcome Page

Now that users can login, the welcome page, **index.php**, needs to be modified to include the following options:

- An option to logout
- A link to a settings page

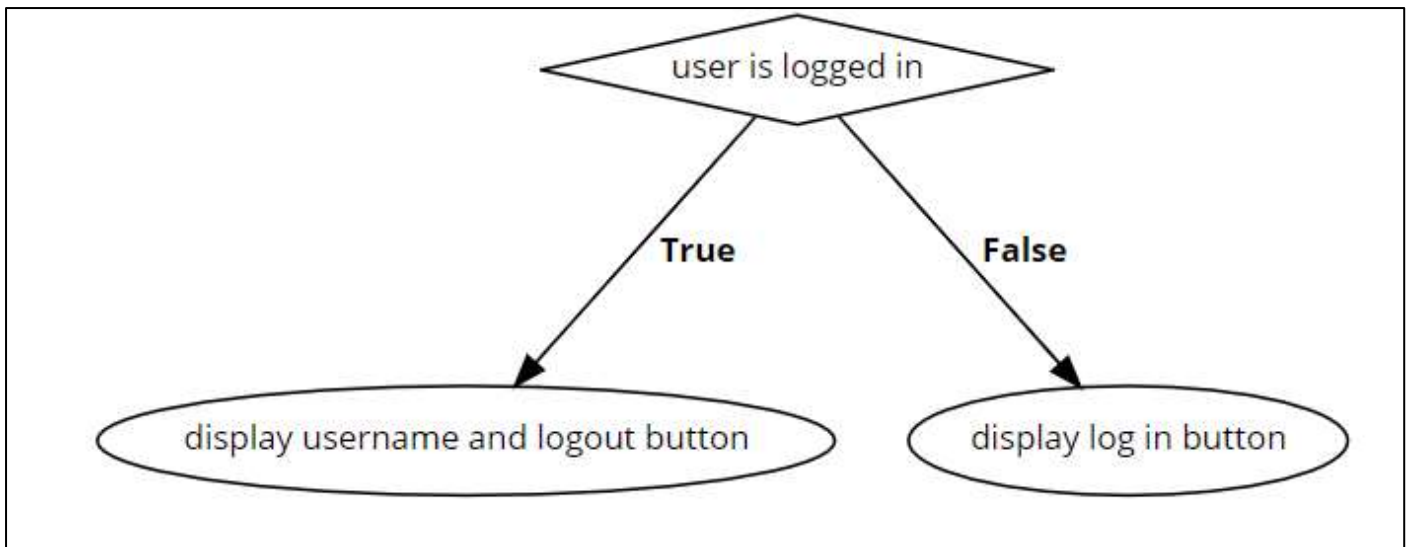
- This will contain an option to reset password
- When the user is not logged in, a link to the login page

```
<h1>Welcome to Railmapper!</h1>
<?php if(isset($_SESSION['loggedin'])){
    echo "<p> Logged in as ".$_SESSION['username']. "<br/>Not you? <a
href='logout.php'>Logout.</a>";
};
?>
```

This is the code for **logout.php**. It simply sets all the session variables to blank, destroys the session, and then redirects back to **index.php**.

```
// file: logout.php
<?php
session_start();
$_SESSION = array();
session_destroy();
header("location: index.php");
exit;
?>
```

Additionally, the welcome page needs to be modified to include a **log in** button which will simply link to **login.php**. A future consideration could be the use of an **iframe** to link the different pages together.



Creating the Settings Page

The settings page will allow the user to save certain queries for future use (saving queries will be developed in the next stage).

For this, a link to a settings page is created within the main page.

Testing for Stage 4

Testing register.php

Input Query #1: Control Test, all values valid

Username	Password	Confirm Password	Email
bob	bob123	bob123	bob@bob.com

Sign Up

Please fill this form to create an account.

Username

bob

Password

Confirm Password

Email

bob@bob.com

Submit

Reset

Or [login here](#).

Showing rows 0 - 1 (2 total). Query took 0.0015 seconds.

SELECT * FROM "users" WHERE 1

Show all

Number of rows: 25

Filter rows: Search this table

Sort by key: None

Options

T +

userid

username

password

Email

☐

Edit

☐

Copy

☐

Delete

3

bob

\$2y\$10\$jp6Fk5Perlx.CyIma3pPKRow4EV5Ptd87WfMwB4d km...

bob@bob.com

☐

Check all

With selected

Edit

Copy

Delete

Export

Show all

Number of rows: 25

Filter rows: Search this table

Sort by key: None

Input Query #2: Leaving username blank:

Username	Password	Confirm Password	Email
	bob123	bob123	bob@bob.com

Notice: Undefined variable: query in C:\xampp\htdocs\register.php on line 71

Sign Up

Please fill this form to create an account.

Username

Please enter a username.

Password

Confirm Password

Email

Submit

Reset

Or [login here](#).

This test passed with issue. (**query** variable error returned).

Input Query #3: Username taken:

Username	Password	Confirm Password	Email
bob	bob123	bob123	bob3@bob.com

Sign Up
Please fill this form to create an account.
Username Username taken. Please try again.
Password *****
Confirm Password *****
Email bob3@bob.com

Or [login here](#).

Input Query #4: Password left blank:

Username	Password	Confirm Password	Email
bob2			bob4@bob.com

Sign Up
Please fill this form to create an account.
Username bob2
Password Please enter a password.
Confirm Password Please confirm your password.
Email bob4@bob.com

Or [login here](#).

Input Query #5: Password too short:

Username	Password	Confirm Password	Email
bob2	bob12	bob12	bob5@bob.com

Sign Up

Please fill this form to create an account.

Username

Password Your password is too short. Please enter at least 6 characters.

Confirm Password

Email

Or [login here](#).

Input Query #6: Confirm Password left blank:

Username	Password	Confirm Password	Email
bob3	bob123		bob6@bob.com

Sign Up

Please fill this form to create an account.

Username

Password

Confirm Password Please confirm your password.

Email

Or [login here](#).

Input Query #7: Confirm Password doesn't match Password:

Username	Password	Confirm Password	Email
Bob4	bob123	bob124	bob7@bob.com

Sign Up

Please fill this form to create an account.

Username

Password

Confirm Password Passwords did not match, please try again.

Email

Or [login here](#).

Input Query #8: Email left blank:

Username	Password	Confirm Password	Email
bob5	bob123	bob123	

Sign Up

Please fill this form to create an account.

Username

Password

Confirm Password

Email Please enter your email address.

Or [login here](#).

Input Query #9: Email invalid:

Username	Password	Confirm Password	Email
bob6	bob123	bob123	bob8noturl

Login

Login Page

Username

Password

Or alternatively, [sign up](#).

Showing rows 0 - 1 (2 total. Query took 0.0015 seconds.)

SELECT * FROM "users" WHERE 1

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Options

		userid	username	password	Email
<input type="checkbox"/>	Edit Copy Delete	3	bob	\$2y\$10\$jp0Fk5Pwix.Cslma3pPKRuw4EVSPb8B7W/FMw84d.km	bob@bob.com
<input type="checkbox"/>	Edit Copy Delete	4	bob6	\$2y\$10\$XLzK.itWpGVJhyQ45KuiHmeEuPzKkmiFdmfzr0CV5u	bob@noturl

☐ Check all
 With selected:

This test failed. (Signup went through successfully without detecting error).

Input Query #10: Email already in use:

Login

Login Page

Username

Password

Login

Or alternatively, [sign up](#).

This test failed. (Signup went through successfully without detecting error).

Errors During Development

The program gave multiple errors during this stage, all of which were subsequently fixed by changing a small number of inconsistencies in the code.

Verification of Valid Email

Issues regarding email validation were caused by an error in the code. One line needed to be changed from this:

```
elseif(filter_var($email, FILTER_VALIDATE_EMAIL))
```

to this:

```
elseif(filter_var(trim($_POST['email']), FILTER_VALIDATE_EMAIL)==false)
```

Following this, the input of an invalid email address returned an error message, meaning the error handling was successful.

Username	Password	Confirm Password	Email
Bob21	Bob123	Bob123	notanemail

Sign Up

Please fill this form to create an account.

Username

Password

Confirm Password

Email Invalid email format. Please try again.

Or [login here](#).

Checking if Email Already in Use

The program also failed to throw an error when an email already existing in the table is used at sign-up, instead validating the address and leaving a situation whereby it was possible to have multiple accounts signed up with the same username.

This was due to a variable naming error in the code when verifying the error variables were all unset, which was changed from this:

```
if(empty($username_err) && empty($password_err) && empty($password_confirm_err)
&& empty($email_confirm_err))
```

to this:

```
if(empty($username_err) && empty($password_err) && empty($password_confirm_err)
&& empty($email_err))
```

i.e., the variable `$email_err` was not correctly referenced.

Testing to see if an email already exists in the database returned an error message after this change, meaning the error handling was successful.

This was done on the premises that the email address **bob@bob.com** already existed in the table when the test was done, and the username did not:

Username	Password	Confirm Password	Email
bob3	bob123	bob123	bob@bob.com

Sign Up

Please fill this form to create an account.

Username

Password

Confirm Password

Email Email already in use. Maybe you signed up already?

Or [login here](#).

Stage 4 Review

The progress made in this stage has been significant. The application now contains a functioning account system with facilities to register and login and an SQL database containing user information.

Achieved in This Stage

- Accounts can now be created and stored within a database
- User information can also be stored within the database

Improvements for Next Stages

- Now that accounts exist, one of the next tasks is to allow advanced users (who will have an account) to access a wider range of data.
- The ability to save stations into the database and retrieve when necessary in a 'bookmark' fashion should also be implemented.
- Additionally, when searching for stations, a list of stations should be implemented to search through, and then the stations should be converted to their three-digit CRS code for the URL. This will make it far easier to search for stations. This can be done using PHP.

Stage 5: Implementing Advanced Data

Stage Objective

Development for Stage 5

A minor adjustment is first made to make the site easier to code. The top banner, displaying the text 'Welcome to Railmapper!' and the login information has been moved into its own file, **banner.php**.

This file can then be included via a PHP statement. This has been done as it ensures the banner is kept constant across all the pages in the site.

Additionally, as a constant block of code, site-wide configurations can be applied within this file.

Creating Site Accessibility Options

The first step in the development of site-wide accessibility options is the need to link an external CSS stylesheet to all the pages in the site. The stylesheet can then be applied via PHP conditionally, should the user select the accessibility options.

Accessibility options include use of a high contrast mode and use of a larger font size. For the high contrast mode, a column is added to the user table to store a Boolean value of either 0 or 1 to determine whether this value is set or not.

```
ALTER TABLE users ADD ContrastSet Boolean
```

The SQL query on **register.php** is modified to include the ContrastSet value, which will default to 0.

```
$sql = "INSERT INTO users (username, password, email, ContrastSet) VALUES (?, ?, ?, 0)";
```

```
if($stmt = mysqli_prepare($connect, $sql)){  
    // Bind variables to the prepared statement as parameters
```

```

        mysqli_stmt_bind_param($stmt, "ssss", $param_username,
$param_password, $param_email, $param_contrastset);

        // Set parameters
        $param_username = $username;

        $param_password = password_hash($password, PASSWORD_DEFAULT); //
Creates a password hash

        $param_email = $email;

        $param_contrastset = $contrastset;

        // Attempt to execute the prepared statement
        if(mysqli_stmt_execute($stmt)){
            // Redirect to login page
            header("location: login.php");
        } else{
            echo "Something went wrong. Please try again later.";
        }

        // Close statement
        mysqli_stmt_close($stmt);
    }

```

The **ContrastSet** value is then retrieved along with the other values in the table.

Since the **ContrastSet** value in the table is a boolean value, we can use an SQL statement within PHP to toggle it.

The setting of accessibility options will follow this method:

Upon registration, the **ContrastSet** value is set to its default value of **FALSE** along with the other components of user registration.

Additionally, the value itself is cached in the PHP **session** upon each page login to minimise rework and prevent slowing down of the application upon each page load.

```
// Store data in session variables. File: login.php
$_SESSION["loggedin"] = true;
$_SESSION["id"] = $id;
$_SESSION["username"] = $username;
$_SESSION["ContrastSet"] = $contrastset;
```

The value can be changed via a toggle button in **settings.php**. Clicking the button will redirect the user to **contrast-change.php** whereby an SQL query is executed to toggle the value in the database, and the **session** value is also changed simultaneously, removing the need to synchronise the database again during the current session.

```
<?php
require_once('config.php');
session_start();
$user = $_SESSION['username'];
$sql = "UPDATE users SET ContrastSet = !ContrastSet WHERE
username='".$user."'";
mysqli_query($connect,$sql);
mysqli_close($connect);

if($_SESSION["ContrastSet"]==1){
    $_SESSION["ContrastSet"]=0;
} else {
    $_SESSION["ContrastSet"]=1;
}
```

```
header('Location: settings.php');
```

```
?>
```

A verification check is applied to **banner.php** for every page load (since it is included in every page in the site). The verification check confirms whether the **session ContrastSet** variable is set to 0 or 1, and should it be set to 1, the CSS link is included in the file, putting the pages into high contrast mode.

```
if(isset($_SESSION["ContrastSet"])){
```

```
if($_SESSION["ContrastSet"]==1){echo'<link type="text/css" rel="stylesheet"
href="high-contrast.css"/>';}
```

```
}
```