Testing

Overview

Throughout development I tested features to make sure they worked on their own before implementing them into my project. In this section I will test each function of my finished system to see if I observe the results I expect.

Where data can be entered, I will test using valid, invalid and boundary data. Where data that is input will be used in an SQL query to my database, I will also make sure my project does not break when SQL symbols are entered to check that my parametrised statements are protecting my project against SQL injections.

Details of individual tests

The evidence for each test is included in these videos:
Video 1: https://youtu.be/SKqqxd1AO50
Video 2: https://youtu.be/MvopYK3y8Sk
Video 3: https://youtu.be/s_4g6u5X7KU

Test group and Number	Purpose	Description	Test Data	Expected Result	Video Timings	Actual Result
1 Validation / Navigation	Valid test	I will enter two different valid stations into the ‘From’ and ‘To’ text boxes. Then I will press the ‘Find Route’ button.	Start station = 'Vauxhall' To station = 'London Bridge’	The ‘Journey Results’ panel is displayed when the ‘Find Route’ button is pressed.	Video 1 01:05 to 01:34	As predicted
2 Validation	Invalid test	I will enter nothing into both the ‘From’ and ‘To’ text boxes. Then I will press the ‘Find Route’ button.	Start station = null To station = null	An error message will appear on the GUI	Video 1 00:30 to 00:38	As predicted
3 Validation	Invalid test	I will enter two invalid station names into both the ‘From’ and ‘To’ text boxes.	Start station = ‘test station’	An error message will appear on the GUI	Video 1 00:39 to 00:52	As predicted

		Then I will press the 'Find Route' button.	To station = 'made up station'			
4 Validation	Invalid test	I will enter the same valid station into both the 'From' and 'To' text boxes. Then I will press the 'Find Route' button.	Start station = 'London Bridge' To station = 'London Bridge'	An error message will appear on the GUI	Video 1 00:52 to 01:04	As predicted
5 Validation / Navigation / Algorithm output	Invalid test	I will find a route between two stations. I will then close a line that is mandatory for this journey. I will then press the 'Find Route' button again.	Start station = 'Vauxhall' To station = 'London Bridge' Victoria Line status = closed Other lines are open	When the line is closed an error message will appear on the GUI and the panel being shown will not change.	Video 1 01:05 to 02:22	As predicted
6 Validation / Navigation / Algorithm output	Valid test	I will find a route between two stations. I will then close a line that is used on this journey but is not mandatory to the journey. I will then press the 'Find Route' button again.	Start station = 'Vauxhall' To station = 'London Bridge' Northern Line status = closed Other lines are open	A route is found with the path taken being redirected onto other train lines.	Video 1 01:05 to 03:05	As predicted
7 Algorithm output	Valid test	Start entering a station name. Test to see if there is autocomplete for the station name being entered	'lon' entered into the station name text box	When typing in a station name the user should have the option to use autocomplete to automatically fill out a station name suggested by the system.	Video 1 00:52 to 01:00	As predicted

				In this case when typing in 'lon' the station name 'London Bridge' should be suggested.		
8 Navigation	Valid test	I will press the 'Admin Access' button	Pressing the 'Admin Access' button	The 'Admin Login' panel is displayed when the 'Admin Access' button is pressed.	Video 1 01:49 to 01:52	As predicted
9 Algorithm output	Valid test	The 'Minimum Changes' checkbox is not selected, and a route is found between two stations where having more changes would make the journey quicker than have fewer changes.	The 'Minimum Changes' checkbox is not selected. Start station = 'Victoria' To station = 'Liverpool Street'	The route is found with more changes than the result in test 10 but the journey time is less.	Video 1 03:35 to 04:49	As predicted
10 Algorithm output	Valid test	The 'Minimum Changes' checkbox is selected, and a route is found between the same two stations in test 9	The 'Minimum Changes' checkbox is selected. Start station = 'Victoria' To station = 'Liverpool Street'	The route is found with less changes than the result in test 9 but the journey time is more.	Video 1 03:35 to 04:49	As predicted
11 Algorithm output	Valid test	The 'Live Train Changes' checkbox is not selected and then a route is found.	The 'Live Train Changes' checkbox is not selected Start station = 'Dollis Hill'	In the journey results there is no platform information or arrival times.	Video 1 04:55 to 05:43	As predicted

			To station = 'Holborn'			
12 Algorithm output	Valid test	The 'Live Train Changes' checkbox is selected and then a route is found.	The 'Live Train Changes' checkbox is selected Start station = 'Dollis Hill' To station = 'Holborn'	The API is called and in the journey results there is platform information and arrival times for each line change returned. If live information for a station is not available an error will be shown instead.	Video 1 05:56 to 06:03	As predicted
13 Algorithm output	Valid test	The 'Live Train Changes' checkbox is selected and then a route is found. Then arrival times and platform names are compared to that of TfL's website.	The 'Live Train Changes' checkbox is selected Start station = 'Dollis Hill' To station = 'Holborn'	The arrival times and platform names output by my system are the same as those displayed by TfL on their website. Arrival times may be off by ± 1 min due the exact time the API was called, and the accuracy of the API compared to the website.	Video 1 05:56 to 06:48	As predicted
14 Algorithm output	Valid test	The 'Live Train Changes' checkbox is selected and then a route is found while not connected to the internet. This will mean the API cannot be accessed.	The 'Live Train Changes' checkbox is selected	All route details will be returned apart from platform and arrival time information. An error message will	Video 1 07:10 to 07:30	As predicted

				appear on the GUI instead.		
15 Algorithm output	Valid test	A route is found between two stations that are reasonably close together. (about 10 stops)	Start station = ‘Poplar’ To station = 'Regent’s Park’	A route between the stations will be calculated and list of station names will appear in the console showing the exact route taken by my path finding algorithm. This route should appear to be the shortest path.	Video 1 07:37 to 08:57	As predicted (where the same station is printed out twice it shows a line change)
16 Algorithm output	Valid test	A route is found between two stations that are far apart, across the whole tube map	Start station = 'Epping’ To station = ‘Amersham’	A route between the stations will be calculated and list of station names will appear in the console showing the exact route taken by my path finding algorithm. This route should appear to be the shortest path.	Video 1 09:02 to 10:02	As predicted (where the same station is printed out twice it shows a line change)
17 Algorithm output	Valid test	Test to see that the cost of the journey is displayed and that the cost is the correct value from the database.	Start station = 'Epping’ To station = ‘Amersham’	The price is displaced on the GUI and is the correct value given the range of zones travelled through. This range of zones is printed in the console.	Video 1 10:08 to 10:45	As predicted

18 Algorithm output	Valid test	Test to see if the system returns the total estimated travel time for the journey.	Start station = 'Epping' To station = 'Amersham'	The total time for the journey is displayed.	Video 1 10:50 to 10:58	As predicted
19 Algorithm output	Valid test	Test to see if the system returns the estimated travel time for each section between train changes in the journey.	Start station = 'Epping' To station = 'Amersham'	The estimated travel time for each section between train changes is displayed.	Video 1 10:59 to 11:09	As predicted
20 Algorithm output	Valid test	Test to see if the system returns the names of stations where each line change occurs.	Start station = 'Epping' To station = 'Amersham'	The names of stations where line changes occur should be displayed.	Video 1 11:09 to 11:19	As predicted
21 Algorithm output	Valid test	Test to check that the system shows the names and colours of the lines being travelled on in the journey.	Start station = 'Epping' To station = 'Amersham'	The system shows the names and colours of the lines being travelled on in the journey.	Video 1 11:22 to 11:36	As predicted
22 Algorithm output	Valid test	Are the route details returned in less than 200ms, when the 'Live Changes' checkbox is not selected and therefore the API is not accessed for live train times?	Start station = 'Richmond' To station = 'Seven Sisters' The 'Live Train Changes' checkbox is not selected	The route details are returned in less than 200ms. This time is printed in the console.	Video 1 11:40 to 12:02	As predicted The route was found in 16ms which was less time than I expected.
23	Valid test	Perform a path find. The priority queue is printed whenever a station is	Perform a path find	The queue is in order of each station's time to be reached from the	Video 3 08:10 to 11:48	As predicted

Algorithm output		popped from the front of the queue.	Start station = 'Devons Road' To station = 'Loughton'	start station. When an index is added, it is inserted based on its time to be reached.		NB: the same station name sometimes appears multiple times in the queue due to some stations being represented by multiple nodes. This is explained below in the 'Graph problem' section.
24 Algorithm output	Valid test	Print out the adjacency list when the program begins. Print out the station names for each corresponding list of edge indexes.	Run the program	For each station there is a list of edge indexes. These edges correspond to connected station names which are printed out. These names should show that connections between stations are	Video 3 02:08 to 05:13	As predicted NB: some stations are represented by multiple nodes and are therefore listed

				correctly representing the tube map.		multiple times. This is explained below in the 'Graph problem' section.
25 Algorithm output	Valid test	Test the binary search by printing out the array of station names still being considered every time half of the array is discarded.	Perform a path find First find two valid stations: Start station = 'Devons Road' To station = 'Loughton' Then search for an invalid station: Start station = 'Devons'	The binary search should find the station name that was entered if it exists in the array.	Video 3 05:20 to 08:07	As predicted
26 Navigation	Valid test	Testing the 'New Route' button on the 'Journey Results' panel	Pressing the 'New Route' button	The 'Tube Planner' main menu panel is displayed when the 'New Route' button is pressed.	Video 1 12:02 to 12:08	As predicted
27 Data storage / Validation / Navigation	Valid test	Can an admin login to an admin account by entering a username and password that exist in the tblAdminDetails table?	Any valid admin account. In this case: Username = 'Toby' Password = 'password'	The GUI displays the admin menu.	Video 1 12:17 to 12:42	As predicted

28 Validation	Invalid test	An invalid username and password are entered and then the 'Login' button is pressed.	Username = 'not valid' Password = 'not a password'	An error message will appear on the GUI.	Video 1 12:55 to 13:10	As predicted
29 Data security / Validation	Invalid test	Enter an SQL symbol into the username field on the 'Admin Login' panel to check that the parametrised statement in preventing against SQL injections.	Username = '''' Password = ''''	An error appears on the GUI saying account does not exist.	Video 1 13:13 to 13:22	As predicted
30 Navigation	Valid test	Testing the 'BACK' button on the 'Admin Login' panel	Pressing the 'BACK' button	The 'Tube Planner' main menu panel is displayed when the 'BACK' button is pressed.	Video 1 2:10 to 2:14	As predicted
31 Navigation	Valid test	Testing the 'Log out' button on the 'Admin Menu'	Pressing the 'Log out' button	The 'Admin Login' panel is displayed when the 'Log out' button is pressed.	Video 1 12:41 to 12:44	As predicted
32 Navigation	Valid test	Testing the 'Edit Zone pricing' button on the 'Admin Menu'	Pressing the 'Edit Zone pricing' button	The 'Edit Zone pricing' panel is displayed when the 'Edit Zone pricing' button is pressed.	Video 1 13:34 to 13:47	As predicted
33 Navigation	Valid test	Testing the 'Edit Lines' button on the 'Admin Menu'	Pressing the 'Edit Lines' button	The 'Edit Lines' panel is displayed when the 'Edit Lines' button is pressed.	Video 1 13:49 to 13:52	As predicted
34 Navigation	Valid test	Testing the 'New Admin account' button on the 'Admin Menu'	Pressing the 'New Admin account' button	The 'New Admin Account' panel is displayed when the	Video 1 13:53 to 13:56	As predicted

				'New Admin account' button is pressed.		
35 Data storage	Valid test	Enter the details for a new admin account on the 'New Admin Account' panel. Then press 'Create Account'	Username = 'Felix' Password = 'securePass7' Confirm Password = 'securePass7' Press the 'Create Account' button	A message appears on the GUI saying an account was created. This new account now appears in the tblAdminDetails table in the database.	Video 1 14:08 to 14:45	As predicted
36 Data storage / Validation	Invalid test	When creating a new admin account, enter a username that already exists. Then press 'Create Account'	Username = 'Toby' Password = 'password123' Confirm Password = 'password123' Press the 'Create Account' button	An error message appears on the GUI saying that this username has been taken.	Video 1 15:57 to 16:21	As predicted
37 Validation	Boundary test – invalid test	When creating a new admin account, enter a username with 21 characters. Then press 'Create Account'	Username = '123456789012345678901' Password = 'password123' Confirm Password = 'password123' Press the 'Create Account' button	An error message appears on the GUI saying that the username needs to be <= 20 characters.	Video 1 16:25 to 16:42	As predicted
38 Validation	Boundary test – valid test	When creating a new admin account, enter a username with 20	Username = '12345678901234567890'	A message appears on the GUI saying an account was created.	Video 1 17:00 to 17:22	As predicted

		characters. Then press ‘Create Account’	Password = ‘password123’ Confirm Password = ‘password123’ Press the ‘Create Account’ button	This new account now appears in the tblAdminDetails table in the database.		
39 Validation	Invalid test	When creating a new admin account, leave the username field blank. Then press ‘Create Account’	Username = null Password = ‘password123’ Confirm Password = ‘password123’ Press the ‘Create Account’ button	An error message appears on the GUI saying that a username must be entered.	Video 1 17:25 to 17:48	As predicted
40 Data security / Validation	Valid test	Creating a new admin account entering SQL characters into the username and password fields. Then press ‘Create Account’	Username = '’’ Password = ‘’testing(’ Confirm Password = ‘’testing(’ Press the ‘Create Account’ button	A message appears on the GUI saying an account was created. This new account now appears in the tblAdminDetails table in the database.	Video 1 17:52 to 18:23	As predicted
41 Validation	Boundary test – invalid test	When creating a new admin account, enter a password and confirmation password of 7 characters. Then press ‘Create Account’	Username = ‘Jacob’ Password = ‘1234567’ Confirm Password = ‘1234567’ Press the ‘Create Account’ button	An error message appears on the GUI saying that the password needs to be >= 8 characters.	Video 1 18:28 to 18:57	As predicted

42 Validation	Boundary test – valid test	When creating a new admin account, enter a password and confirmation password of 8 characters. Then press ‘Create Account’	Username = ‘Jacob’ Password = ‘12345678’ Confirm Password = ‘12345678’ Press the ‘Create Account’ button	A message appears on the GUI saying an account was created. This new account now appears in the tblAdminDetails table in the database.	Video 1 18:58 to 19:21	As predicted
43 Validation	Boundary test – invalid test	When creating a new admin account, enter a password and confirmation password of 21 characters. Then press ‘Create Account’	Username = ‘Dan’ Password = ‘123456789012345678901’ Confirm Password = ‘123456789012345678901’ Press the ‘Create Account’ button	An error message appears on the GUI saying that the password needs to be <= 20 characters.	Video 1 19:25 to 19:53	As predicted
44 Validation	Boundary test – valid test	When creating a new admin account, enter a username of 20 characters. Then press ‘Create Account’	Username = ‘Dan’ Password = ‘12345678901234567890’ Confirm Password = ‘12345678901234567890’ Press the ‘Create Account’ button	A message appears on the GUI saying an account was created. This new account now appears in the tblAdminDetails table in the database.	Video 1 19:54 to 20:23	As predicted
45 Validation	Invalid test	When creating a new admin account, enter different values for the password and confirmation	Username = ‘Dave’ Password = ‘davesPassword’ Confirm Password = ‘davesPassword72’	An error message appears on the GUI saying that the password and	Video 1 20:27 to 20:54	As predicted

		password. Then press 'Create Account'	Press the 'Create Account' button	confirmation password must be the same.		
46 Algorithm output / Data security	Valid test	Create a new admin account. Check this account's password when it is hashed in the database. Compare this hash with a hash generated with a SHA-256 generator online.	Username = 'Felix' Password = 'securePass7' Confirm Password = 'securePass7' Enter 'securePass7' into hash generator online.	The hash generated online from the string 'securePass7' is the same as the hash that appears in my database for the account with username 'Felix'	Video 1 14:08 to 15:48	As predicted
47 Algorithm output / Data storage	Valid test	Create a new admin account. Then check this account's adminID in the database	Create a new account with any valid details, then view this account in the database.	The adminID should be automatically set to the maximum existing adminID +1.	Video 1 14:32 to 14:59	As predicted
48 Navigation	Valid test	Testing the 'BACK' button on the 'New Admin Account' panel	Pressing the 'BACK' button	The 'Admin menu' panel is displayed when the 'BACK' button is pressed.	Video 1 20:57 to 21:00	As predicted
49 Data storage	Valid test	Test to see if all zone pricing data from the database if shown correctly in table on the 'Edit Zone pricing' panel.	Navigate to the 'Edit Zone pricing' panel	The prices displayed on the GUI should show the same data from the database.	Video 1 21:01 to 22:05	As predicted
50 Data storage	Valid test	Enter new prices into the zone pricing table. Then press Apply Changes. Check the database table 'tblZonePricing' before and after these changes.	Zone1, Zone1 = '1' Zone1, Zone2 = '£5' Zone1, Zone3 = '20.3' Zone1, Zone4 = '0.1' Zone1, Zone5 = '3.02'	The database is updated with the new prices when the 'Apply Changes' button is pressed.	Video 1 21:14 to 23:05	As predicted

51 Algorithm output / Validation	Valid test	Enter valid prices into the fields in the table on the 'Edit Zone pricing' panel.	Zone1, Zone1 = '1' Zone1, Zone2 = '£5' Zone1, Zone3 = '20.3' Zone1, Zone4 = '0.1' Zone1, Zone5 = '3.02'	A message appears on the GUI saying the database was updated.	Video 1 22:11 to 22:47	As predicted
52 Algorithm output / Validation	Invalid test	Test the price regular expression by entering invalid data into a price field on the 'Edit Zone pricing' panel.	New price = 'text'	An error message appears on the GUI saying an invalid price has been entered.	Video 1 23:15 to 23:25	As predicted
53 Algorithm output / Validation	Invalid test	Test the price regular expression by entering invalid data into a price field on the 'Edit Zone pricing' panel.	New price = '0.701'	An error message appears on the GUI saying an invalid price has been entered.	Video 1 23:30 to 23:40	As predicted
54 Algorithm output / Validation	Invalid test	Test the price regular expression by entering invalid data into a price field on the 'Edit Zone pricing' panel.	New price = '1.70.81'	An error message appears on the GUI saying an invalid price has been entered.	Video 1 23:41 to 23:54	As predicted
55 Navigation	Valid test	Testing the 'BACK' button on the 'Edit Zone pricing' panel	Pressing the 'BACK' button	The 'Admin menu' panel is displayed when the 'BACK' button is pressed.	Video 1 23:58 to 24:02	As predicted
56 Data storage	Valid test	Change line statuses using the drop-down lists displayed on the 'Edit Lines' panel. Then press 'Apply Changes'. Check the	Bakerloo Line status = closed Central Line status = closed	Each station should have a drop-down list showing options for 'open' or 'closed'. When the 'Apply	Video 1 24:06 to 25:43	As predicted

		database before and after these changes.	District Line status = closed DRL Line status = closed Second commit: Bakerloo Line status = open Central Line status = open District Line status = open DRL Line status = open	Changes' button is pressed changes to these statuses should be updated in the database.		
57 Navigation	Valid test	Testing the 'BACK' button on the 'Edit Lines' panel	Pressing the 'BACK' button	The 'Admin menu' panel is displayed when the 'BACK' button is pressed.	Video 1 25:43 to 25:46	As predicted
58 Algorithm output	Valid test	Test to see that a URL request can be sent to the TfL API and parsed to JSON.	The 'Live Train Changes' checkbox is selected. Find a route. Then view the console.	The URL is printed in the console along with the JSON response returned from the API.	Video 1 26:00 to 27:38	As predicted
59 Algorithm output	Valid test	When the program begins stations should be sorted using a merge sort into alphabetical order based on their UNICODE values. Printing out the array of stations in the console	Run the program	The array of stations should be printed in the console before and after sorting. After sorting, the array should be sorted.	Video 3 00:00 to 01:55	As predicted NB: most stations are already sorted as the list

		should be used to check the sort has worked.				proved is sorted but not using UNICODE values. I talk more about this in the 'Sorted Stations' section below.
60 Algorithm output / Data storage	Valid test	Test to see if station and connection information from my CSV files can be transferred into my database.	Run the program	When the program is run the CSV files are read, and the data is transferred into the database tables. After running, the database contains records for both connections and stations.	Video 2	The data is copied from the CSV files into the database as predicted. However, the station's naptanIDs, which are fetched from the API for each station, could not be found for 6 of the stations. These will

						need to be entered manually. Also, it must be noted that the transfer of data took a long time as I am limited by the number of requests I can make to the API.
61 Algorithm output / Data storage	Valid test	The icon image of the application is set when the program is run.	Run the program Image file is called 'UndergroundIcon.png'	The icon image is set when the program is run, using an image in the files of the project.	Video 1 00:00 to 00:29	As predicted

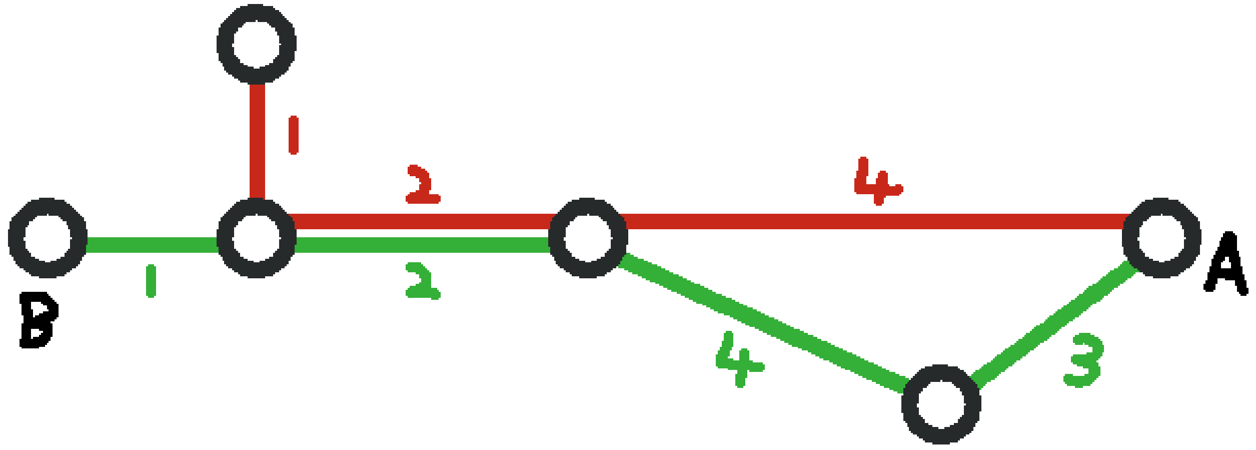
Solving problems with my program

Although everything went as expected in my testing section, this was only because I had tested throughout development and fixed any problems I came across. Below I have written about some of my issues and how I fixed them during development.

<ins>Graph problem</ins>

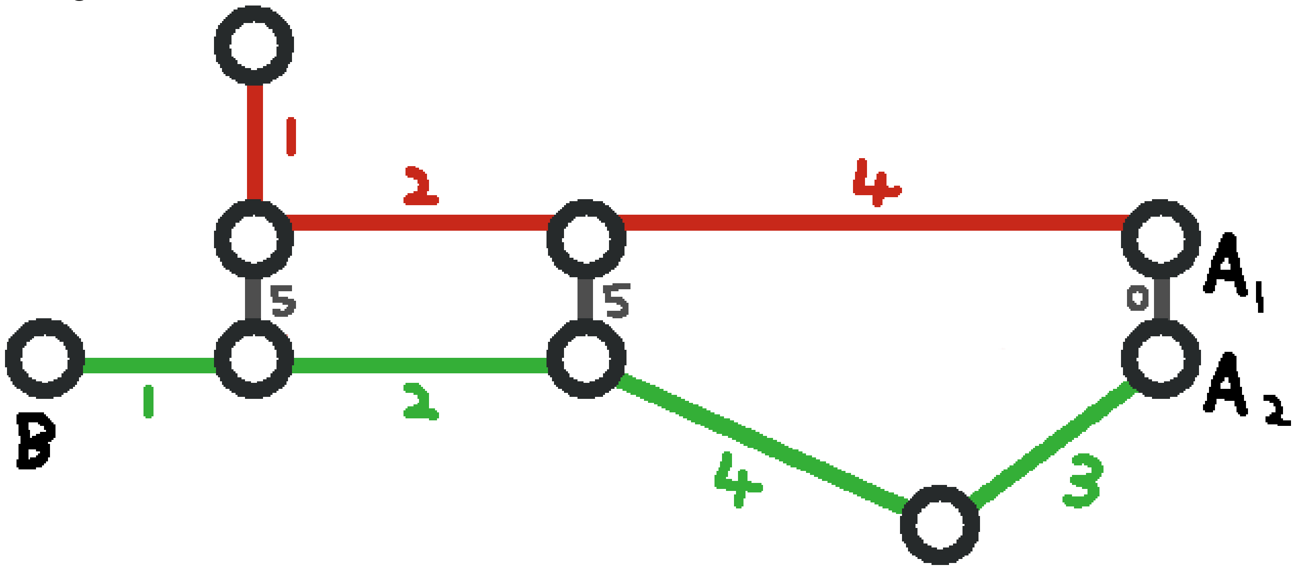
By far the biggest issue I came across when creating my project was with the graph and my path finding solution. Although I got Dijkstra's algorithm working perfectly with a simple graph and then with the London Underground graph, I soon realised that there was a major problem involving line changes.

Using stations as nodes in my graph, and connections between stations as the edges of my graph did not work as the algorithm had no way of pre-emptively knowing which line to travel on when there were parallel lines and therefore this graph would sometimes lead to unnecessary line changes. A route between A and B below has this problem as Dijkstra's algorithm would first travel on the red line and then change to the green, as it has no way of knowing at A that it must travel on the green line to finish the journey and is not aware that changing lines takes time.



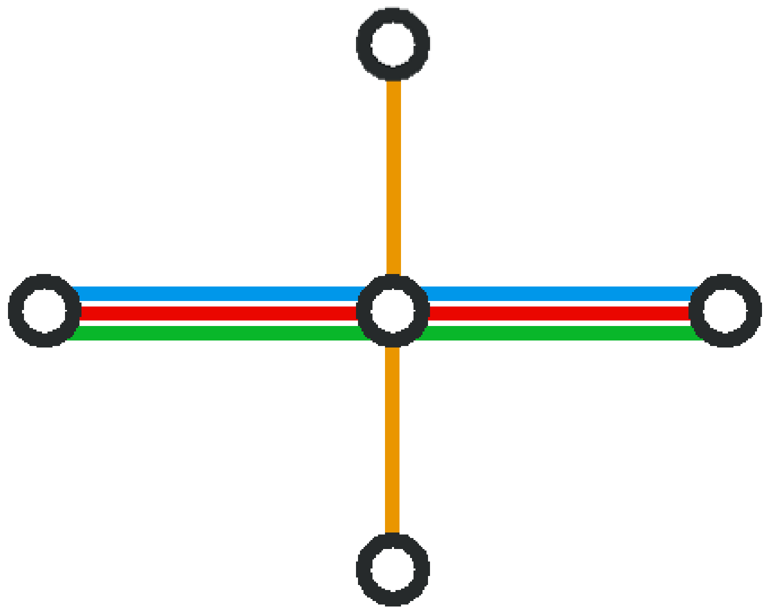
The solution I came to was to create a new and improved graph where line changes were added as edges to the graph, while I replaced each station node with multiple nodes to represent each connection to a different line at that station. I would now create this improved graph of 'GraphNode' and 'GraphEdge' objects every time the program is run using the original station and connection information in the database. I create these graph objects when the program is run so that I am still storing all the station and connection information in my database, which is a lot easier to view and understand. This means that it would then be a lot easier to add new stations and connections in the future if I wanted to implement this feature.

The solution is shown below when it is applied to the graph above. As you can see each station node is replaced by a group of nodes, connected by grey edges representing line changes. The weight of these changes can be edited where the weight is the time it takes to change trains.

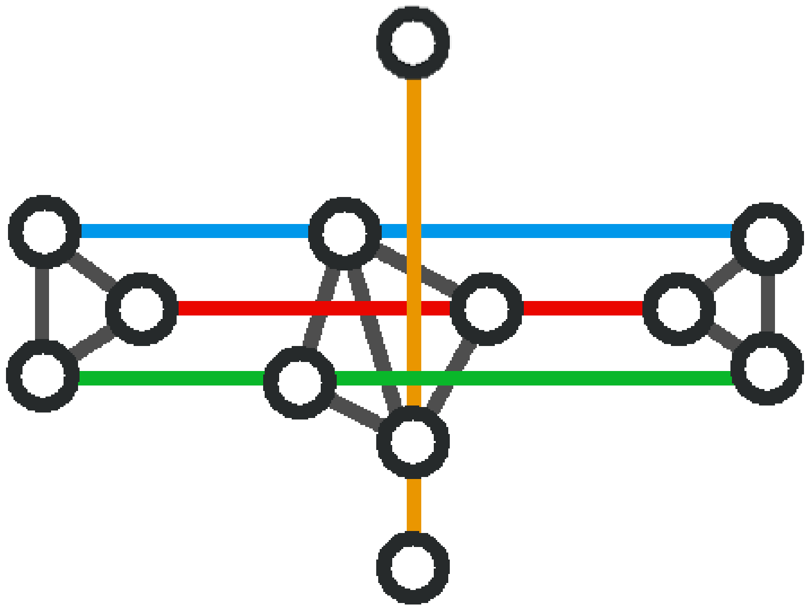


This makes my graph look a lot more complex even though the path finding algorithm works in the exact same way. An example of one station with six lines going through it, connected to four different stations is draw below. As you can see each group of nodes in the improved graph represents one station, with the grey edges representing line changes.

Before:



After:



Sorted Stations
I was using binary search on the stations from the dataset, however, sometimes there would be issues with stations not being found when they should be even through the station names appeared to be sorted in alphabetical order. The problem was that the I was comparing strings in my binary search using their Unicode value, but the station were not sorted according to Unicode but instead something else.
To solve this, I applied a merge sort on my stations based on the Unicode values of their names. This now meant that the binary search always worked as intended.

Adjacency List
I realised that creating my adjacency list every time the path finding ran was adding unnecessary time to the path finding, as I had imbedded for loops looping though all the nodes and edges of the graph. This was a simple fix where I now create the adjacency list when the program is run and then store it a 2D array while the program is running so that it isn't created every time a route needs to be found. This and a few other changes took my path-finding time to < 30ms from around 50-100ms.

Evaluation

Comparison of Project Performance against objectives

Objective		How I met this objective	How well I met this objective
1	The user can input a starting station and destination station.	I met this objective as on the 'Tube Planner' main menu panel there are two text boxes for the 'From' and 'To' station names to be entered, for a route to be found between them. Evidence of this in test 1.	I think that I met this objective well as not only can users enter a start station and destination station, these text boxes also have autocomplete to make it even easier for users to input the information. The only alternative of doing text boxes would be using the console as the interface and having the user type in their station names there. I think this would have been less user friendly, less visually pleasing, and I would have likely not been able to implement autocomplete.
2	There is autocomplete when the user types in the station names.	I met this objective as when a user starts typing in a station name into either the 'From' or 'To' text boxes, there is a suggestion filled out by the system. The user can then press Tab to complete this name or continue typing in a different name. Evidence of this in test 7.	I think this objective is met well as the autocomplete works, is intuitive to use, and speeds up the time to type a station name which was the main reason for this feature. However, this autocomplete could be improved if I were to make an updated verson of this feature for future programs. Firstly, I could have a drop-down list that suggests multiple station names rather than just suggesting the first one that matches what the user types in. Secondly, I could also suggest station names that have a similar spelling to what the user has typed in so that the user will still get the suggestion they need if they make a spelling error or typo.
3	A binary search is used to search an array of station	I met this objective as I use this algorithm when I need to find the station names entered by the user.	I think that I met this objective well as my implementation works as a binary search should, with a

	names for a station name entered by the user.	The code for this binary search is in the 'binarySearchForNode' method in the 'RouteFinder' class. Evidence of this in test 25.	Big-O notation of O(log n), returning true or false depending on whether the station name is found. The binary search algorithm is also the best search algorithm I could have implemented as it is much faster than linear search for searching long arrays.
4	The user can choose whether they would like the quickest route or the route with minimum train changes, via a checkbox.	I met this objective as there is a checkbox labelled 'Minimum Changes' which allows users to choose whether they would like the quickest route or the route with minimum train changes. Evidence of this in tests 9 and 10.	I think that I met this objective well as there is a checkbox letting the user choose whether they would like the quickest route or the route with minimum train changes. Selecting this checkbox changes how my path finding algorithm works so the correct route can be found given the state of this checkbox. This was a very simple objective so after implementing the checkbox there is nothing I could improve or do differently.
5	The user can choose whether they would like to receive live train times, via a checkbox.	I met this objective as there is a checkbox labelled 'Live Train Times' which allows users to choose whether they would like to view live train arrival times. Evidence of this in tests 11 and 12.	I think that I met this objective well as there is a checkbox doing the function specified. This was a very simple objective so after implementing the checkbox there is nothing I could improve or do differently.
6	An error is displayed if an invalid station is entered.	I met this objective by displaying an error message if a station name entered by the user cannot be found in the array of station names. Evidence of this in tests 2 and 3.	I think that this objective is met well as a clear error message is displayed on the GUI specifying that 'Stations do not exist'. Although this error message meets the requirements I have set in my objective, I could have been more specific in saying which of the two station name fields

			is invalid. I could have also had a different error message for when the fields are left completely blank.
7	An error is displayed if the journey is not possible as a line is closed.	I met this objective by displaying an error message if the journey is not possible as a line is closed. Evidence of this in test 5.	I think that this objective is met well as a clear error message is displayed on the GUI specifying that a 'Route cannot be found due to closed lines'. Although this error message meets the requirements I have set in my objective, I could have been more specific in saying which line is closed and preventing the journey. To improve on just an error message I could also suggest alternative modes of transport and information about when the line may reopen, however this would be a large feature and would not be quick and easy to implement.
8	An error is displayed if the start and end station entered are the same station.	I met this objective by displaying an error message if the start and end station entered by the user are the same. Evidence of this in test 4.	I think that this objective is met well as the error message is displaced on the GUI saying, 'Stations entered are the same', which highlights the exact problem causing the user's input to be invalid. I have considered two alternative ways of displaying the error message however I think that neither of these are as good as displaying the error on the GUI next to where the information is typed in. The first is printing the message in the console, however this is less clear and cannot be viewed if my program is compiled into an external application. The second is having the error pop-up on a small GUI that requires the user to press 'OK'. I don't like this method as it unnecessarily

			requires the user to press an extra button, and the pop-up can seem annoying.
9	Dijkstra's algorithm is used to find the shortest travel time between stations and the route of that journey.	I have met this objective by programming it in my class called 'DijkstrasAlgorithm'. Testing the program shows that the journey with the shortest travel time between stations is always found. Evidence of this in tests 15, 16 and 18.	I think that I have met this objective well as my implementation of Dijkstra's algorithm works as intended and I have used some effective programming techniques such as a priority queue and recursion. A faster alternative to Dijkstra's algorithm I could have used instead is A* pathfinding, however I have explained in my design section why I was unable to use this algorithm. If I was able to get round some of these problems, such as getting the real-world distance information between stations, this algorithm may be an improvement to my path finding and decrease the time for a journey to be found.
10	Dijkstra's algorithm uses a priority queue to hold indexes of nodes Objects, ordered by their time to be reached.	I have met this objective by including an ArrayList called 'queueOfGraphNodeIndexes' which acts as my priority queue of nodes sorted by their time to be reached from the start node. Evidence of this in test 23.	I think that I have met this objective well as I believe I have programmed the queue efficiently. I have used a binary search to find the position in the queue where a new node should be inserted, instead of using a linear search. I do not check to see if an index is already in the queue when it is added, instead I just add the new index so the queue may have multiple of the same index. These duplicates are removed when they get to the front of the queue eliminating the need for a linear search. This implementation is fast, making adding an item to the queue as quick as O(log n), while popping an item from the queue is O(1). There are other possible methods to using an ArrayList that I could have used to implement my priority queue such as a Binary Heap or linked list. However, both

			methods are used to overcome using a fixed size array which was not a problem for me as I was using an ArrayList. A linked list is very inefficient, and a Binary Heap has a complexity of O(log n) for popping and O(log n) for adding new values, so the method I have used is the most efficient.
11	My implementation of Dijkstra's algorithm should use recursion.	I have met this objective as my 'recursiveAlgorithum' method in Dijkstra's algorithm calls itself. Evidence of this in screenshot 1.	I think that I have implemented recursion well, making my code easier to read than if I had used iteration. I think that recursion was a better solution than iteration for Dijkstra's algorithm as I need to run the algorithm until the end node has been found, or no route can be found, rather than setting a condition before looping through the algorithm. However, it would be possible to program Dijkstra's algorithm using iteration, so in hindsight I should have programmed both methods to see if iteration could make a worthwhile impact on the execution speed of the route finder. This is something I could try in the future to improve my program.
12	An adjacency list is used to store indexes of all edge objects connected to each node object.	I have met this objective by creating a 2D array as my adjacency list, with the index of each row representing the index of a node in the node ArrayList, and the list of indexes in each row representing indexes of edges in the edge ArrayList that are connected to the corresponding node object.	I think that my adjacency list is programmed well, as I have initialised the list when the program begins, rather than at the start of every path find, to improve path finding speeds as mentioned above in the section called 'Adjacency List'. The traditional way to store a graph would be to use an adjacency list storing all nodes connected to each node or to use an adjacency matrix. Instead, I used my own adaptation of an adjacency list where I store all edges

		Evidence of this in test 24.	connected to each node. This allows me to store more information about each edge using the edge object, such as the train line information object as well as the travel time for that edge. I did not use an adjacency matrix as there are very few edges compared to nodes so most cells in the matrix would be empty, wasting memory space. Storing my data in an adjacency matrix would mean a 2D array with 170,569 fields. My adjacency list has 2,891 fields. I only need to use 1,136 of these fields. As you can see here an adjacency list is a much better solution than an adjacency matrix, and the 1,755 unused fields in my adjacency list is acceptable when memory is not a huge problem as it comes with the convenience of an easy to use and understand adjacency list structure. However, if I wanted to make my program even more optimised and memory efficient, I should definitely look at how I could improve my current adjacency list.
13	My pathfinding algorithm redirects routes if a line's status is set to 'Closed'.	I have met this objective by altering my path finding algorithm so that it does not used edges that are sections of closed lines. This means my program finds an alternative route, redirecting the user onto other lines. Evidence of this in test 6.	I have met this objective well as my program redirects routes when a line is closed. I also think that this objective is met well as I display a useful error message to the user if a journey becomes impossible due to the closed lines. I have commented on possible improvements to this error message when analysing objective 7, however I don't think there are any possible improvements I could make for this objective.
14	If the 'Minimum Changes' checkbox has been selected, my pathfinding algorithm redirects routes if	I have completed this objective by weighting the algorithm more in favour of staying on the same line if the minimum changes checkbox is	I have executed this objective well as my path-find solution successfully adapts to the user's choice of selecting the 'Minimum Changes' checkbox to get a route with less train changes where possible.

	there is an alternative route with less train changes, even if the journey time is made longer.	selected. This means that a route will still be found, but if this is possible with less line changes that is the route the algorithm will take, even if it means the total journey time is longer. Evidence of this in tests 9 and 10.	I don't think there are any improvements I can make for this objective.
15	Objects are used to store the nodes and edges of the graph representing the tube map. With nodes representing stations and edges representing train lines.	I have completed this objective by creating 'Station' objects to represent stations and 'StationConnection' objects to represent train lines between stations. 'GraphNode' objects represent the nodes of the graph and 'GraphEdges' represent the edges of the graph. GraphNodes associate with StationNodes and are needed as one station is represented by multiple nodes in the graph. The GraphEdges are the connections between these GraphNodes. This is explained in 'Graph Problem' section above. Evidence of this in screenshot 2.	Although I have used objects for the nodes and edges of my graph, they do not directly represent stations and train lines as I expected they would in my objective. However, I think that I have met this objective in a slightly different and better way. I talk in detail about how I represented the graph, and why, in the 'Graph Problem' section above. One thing that I could have done differently is to use inheritance instead of association between the 'GraphNode' and 'StationNode' objects. However, I did not use this as the 'GraphNode' objects use the data in the 'StationNode' objects but do not clearly share a child-parent relationship with them. I decided to use association as I think it makes my objects' relationships clearer and easier to understand in this context, and because I am using association everywhere else in my program.
16	The system returns the cost of the journey.	I have reached this objective by storing the different prices in my 'tblZonePricing' table in my database, which can be updated by admins via the admin panels. The appropriate	I have met this objective well as the cost of the journey is returned as required, based on the zones of that route, and selected from the database of prices entered by the admins.

		cost is then displayed with the other journey results depending on the zones that are travelled through on the journey. Evidence of this in test 17.	To improve on this feature, I could add more tables in my database for prices, allowing me to store alternative costs for children, families, and other special case tickets.
17	The system returns the total estimated travel time for the journey.	I have met this objective by displaying the total time calculated by my pathfinding solution, from adding all the times of the edges on the shortest path. Evidence of this in test 18.	I have completed this objective well as the total estimated travel time for each route is displayed on the journey results panel. I don't think there are any improvements I can make for this objective as this was a simple objective that was either met, or not.
18	The system returns the estimated travel time for each section between train changes in the journey.	I have achieved this objective by displaying the time for each section between train changes. Evidence of this in test 19.	I have executed this objective well by displaying the estimated travel time for each section, as required by my objective. It is useful for the user to know how long each section of the route between line changes will be, so that they are ready to get off at their stop. There are not any improvements I can make for this objective.
19	The system returns the names of stations where each line change occurs.	I have reached this objective by displaying the names of stations where each line change occurs on the GUI. Evidence of this in test 20.	I have met this objective well by showing the names of the stations as required. This is useful information to the user as it tells them what stations to change at. There are not any improvements to be made for this objective.
20	The system shows the names and colours of the lines being travelled on in the journey.	I have achieved this objective by storing the hex colour values of each line record, along with the line name, in the database. I then show the	I have met this objective well as having the colour displayed as well as the line name makes it clearer for the user to identify and see the different lines when they are displayed with the journey results.

		colour of each line next to the name of the lines that appear on the route. Evidence of this in test 21.	There are not any improvements I can make for this objective.
21	Using the TfL API, the system returns the live arrival times of the next three trains for each stage of the user's journey.	I have met this objective by calling the TfL API and parsing the response to JSON. I then display the arrival times on the GUI panel. Evidence of this in tests 12, 13 and 58.	I have met this objective quite well with three arrival times being display most of the time, but there are a few circumstances when getting three arrival times is not possible: • At night the lines are closed so there are no train arrivals • Due to Covid-19 some lines were often closed meaning no live times • Some sections of lines are just not busy enough for three trains to be arriving in the near future, and therefore less than three times are returned by the API • I was unable to get live train times for the DLR and East London lines as these two lines have no vehicleIDs. This means that I can get the arrival times for trains at the stations but cannot check if they are going in the right direction. Even though this is a problem with TfL's API, I think that I may be able to find a solution as they say that all information that they display on their website can be obtained from the API. Therefore, I will add this to my improvements section and try to find a solution. From this I have realised that always getting three arrival times was an unrealistic objective as there will always be times where trains are not running.

22	Using the TfL API, the system returns the platforms that the user's trains leave from.	I have met this objective by calling the TfL API and parsing the response to JSON. I then display the platform names on the GUI panel. Evidence of this in tests 12 and 58.	I have met this objective well, displaying platform names most of the time. However, for the same situation as mentioned above for objective 21, there is no train information returned and therefore no station name can be displayed. I also think this objective is met well because I display an error message on the GUI to let the user know when platform information cannot be found, instead of just leaving the panel blank when the user expects to see live information.
23	An error message is displayed to the user if the API can't be accessed.	I have met this objective by displaying a message on the GUI panel whenever no station information is available. This may be for when there are no trains currently running or when the user has no internet connection, as shown in test 14. Evidence of this in test 14.	I have met this objective well as the error message is displayed, preventing the program from failing, and letting the user know when no live information can be displayed, rather than just leaving the panel blank.
24	If the 'Live Train Changes' Checkbox was not selected, the API is not called, and the live train times and platform information are not displayed.	I have completed this objective by not calling the API when the checkbox is not selected. This means the user doesn't need to view live information if they don't want to, speeding up the time for a route to be returned. Evidence of this in test 11.	I have executed this objective well because whenever the 'Live Train Changes' checkbox is deselected the API is not called, and no live information is displayed. There are not any improvements I can make for this objective.

25	The route details are returned in less than 200ms, when the API is not accessed for live train times.	I have met this objective by finding a route using an efficient path finding algorithm, priority queue, and binary search. I have also reduced calculation time by creating the adjacency list when the program is run rather than every time there is a path find. Evidence of this in test 22.	I have met this objective very well with routes being returned in 15-30ms. To improve on this, I could try to decrease the time to access the API for live train information. However, this may not be possible as it is linked to the user's internet speed.
26	An Admin can login to the admin menu if they enter a valid username and password.	I have achieved this objective by checking if the username and password entered on the login panel exists in the database. If the account exists, the admin is sent to the admin menu. Evidence of this in test 27.	I have met this objective well as the account validation works as planned, letting users in if they have an account and denying incorrect information with a message on the GUI.
27	An error is displayed if the username and/or password entered is invalid.	I have met this objective by displaying a message: 'Account does not exist', when an invalid username or password is entered on the admin login panel. Evidence of this in test 28.	I have met this objective well because an error is displayed as required by my objective – when the username and/or password entered is invalid. There are not any improvements I can make for this objective.
28	The username and password entered are checked against the records in the database, using a parameterised prepared SQL statement.	I have met this objective by creating a prepared statement and inserting the username and hashed password as the parameters.	I have met this objective well as I have used a parameterised prepared statement as required. I used this method, instead of using a stored procedure, to protect my database against SQL injection attacks.

		Evidence of this in test 29 and screenshot 3.	
29	Admins are displayed a menu from which they have three different options: edit zone pricing, edit lines, and create new admin accounts.	I have met this objective by adding these functions to the admin menu, with the buttons leading to their respective panels. Evidence of this in tests 32, 33 and 34.	I have completed this objective well as the options appear as buttons on the admin panel, allowing admins to edit zone pricing, edit lines, and create new admin accounts. The only improvement I can make to this objective is adding new functions for the admins. This may include adding new stations and lines, and the ability to close stations.
30	Admins can create new admin accounts for fellow admins.	I have met this objective. The admins can go to the 'New Admin Account' panel and create a new admin account. If a valid username and password are entered the password is then hashed, and the username and hashed password are stored as a new record in the 'tblAdminDetails' table. Evidence of this in test 35.	This objective is met well as admin accounts can be created and added to the database. When doing this the password is hashed using a secure SHA-256 hash, and an adminID is assigned to the new account using a SQL 'MAX' operation. I cannot think of any improvements I can make to this. I have set it up so only admins can create accounts for other admins, to prevent any user from creating an account.
31	A username entered for a new admin account must be unique.	I have met this objective by checking if the entered username already exists in the database, before allowing an account to be created using the username. Evidence of this in test 36.	I have met this objective well as my program validates that a username is unique and returns an error if it already exists for an account in the database. There are not any improvements I can make for this objective.

32	The username must be <= 20 characters.	I have reached this objective by making sure the username is <= 20 characters before allowing an account to be created. Evidence of this in tests 37 and 38.	I have met this objective well as my program validates that the username is <= 20 characters for the account to be created, otherwise it returns an error message. There are not any improvements I can make for this objective.
33	The username must not be blank.	I have completed this objective by making sure the username field is not empty before allowing an account to be created. Evidence of this in test 39.	I have met this objective well as my program validates that the username is not blank and returns an error if it is. There are not any improvements I can make for this objective.
34	The password must be between 8 and 20 characters (inclusive).	I have reached this objective by making sure the password is <= 20 and >= 8 characters before allowing an account to be created. Evidence of this in tests 41, 42, 43 and 44.	I have met this objective well as my program validates that a password is between 8 and 20 characters and returns an error if the password is too long or too short. I have set the restraints of the password to this length so that it is long enough to be secure, while not being unnecessarily long. I cannot make any improvements for this objective.
35	Password and Confirming Password must be the same when creating a new admin account.	I met this objective by comparing the password to the confirming password to validate they are the same before an account can be created using this password. Evidence of this in test 45.	I have met this objective well as my program validates that the password and confirming password are the same and returns an error if they are not. There are not any improvements I can make for this objective.
36	If any of the conditions for the username and password	I have completed this objective by displaying a range of different error messages depending on the	I think that I have met this objective very well as I display different error messages depending on the condition that has been broken, to let the admin know

	are not met, an error message is displayed.	condition violated that makes the account entered by the admin invalid. Evidence of this in tests 36, 37, 39, 41, 43 and 45.	specifically what they need to change for their new account to be valid. I don't think that there are any improvements I can make for this objective.
37	The new admin account is inserted into the database using a parameterised prepared SQL statement.	I have met this objective by using a parameterised prepared SQL statement when adding new admin accounts to the database. Evidence of this in test 40 and screenshot 4.	I think that I have met this objective well as I have used a parameterised prepared SQL statement. I think that using a parameterised statement is a lot better than using a standard stored procedure. This is because a parametrised statement will protect against SQL injection attacks, while also making my code easier to read and understand.
38	Admins passwords are hashed using a SHA-256 hash.	I have successfully completed this objective by applying a SHA-256 hash to passwords before they are stored in the 'tblAdminDetails' table. Evidence of this is test 46 and screenshot 5.	I have met this objective well, using a SHA-256 hash as required and then storing the hashed passwords in my database to protect my user's data. I think that this objective has been met well as SHA-256 is a secure hash that cannot be reverse engineered and is much better than using an outdated hashing algorithm such as MD5 which is more susceptible to brute force attacks due to it being fast and memory-conserving. A newer hash such as SHA-512 could be used instead but not provide any huge gain and would just require longer hashes to be stored.

			The best way to improve security would be to use a salt, along with SHA-256, to protect my stored passwords against rainbow table attacks.
39	Use the aggregate SQL ‘MAX’ function when assigning a unique ID to a new admin account.	I have completed this objective by using the SQL ‘MAX’ function to get the current greatest admin account ID. I then add one for the new adminID. Evidence of this is test 47 and screenshot 6.	I think that I have met this objective well by using the ‘MAX’ function in my SQL statement as required. I have decided to use this function as it means I do not need to loop through all the admin records. The admin names also don’t need to be sorted by their adminID, which the alternative ‘TOP’ function requires.
40	Admins can update the pricing for the different Underground zones by editing fields in an interactable table.	I have achieved this objective by having an interactable table on the ‘Edit Zone pricing’ panel, allowing admins to update the costs of journeys through different zones. Evidence of this in test 50.	I have met this objective well as the table lets admins update the pricing for the different Underground zones, as required by my objective. An extra feature I could add would be the ability for admins to update all prices at once using a multiplier. This would be useful as they could increase all prices by the same percentage without having to change each field individually.
41	A cross-table SQL statement is used to fetch all zone pricing information from the database.	I have completed this objective by using a cross-table SLQ statement to get zone pricing information from the ‘tblZonePricing’ and ‘tblZone’ tables in the database. Evidence of this in test 49 and screenshot 7.	I have met this objective well as my SQL statement for getting zone price information uses a cross-table query, as required by my objective. I think that the objective is met well as using a cross-table query means that I don’t need to connect to the database twice and send two different requests, instead all the information I need is returned at once. This makes my program more efficient and makes my code easier to understand.

42	Admins are only able to enter prices with a valid price format, checked by a regular expression.	I have met this objective by adding a validation check to the prices entered by the admins, using a regular expression. Evidence of this in tests 51, 52, 53 and 54 and screenshot 8.	I have completed this objective as my regular expression successfully validates strings to check they are in the correct format. I think I have met this objective well as a regular expression is a better method than coding validation checks for the price input. I have also included a useful error message to let admins know when an invalid price has been entered. To improve my program, I could try to shorten my regular expression. I have already done this but shortening it further may be possible. I could also make the error message more specific by saying exactly which of the prices, entered by the admin into the table, is in the wrong format.
43	Admins have an 'Apply Changes' button to commit their changes to the local database.	I have completed this objective by adding an 'Apply Changes' button for admins to commit their new prices to the local database. Evidence of this in test 50.	I have met this objective well as having this button means that the database only needs to be accessed once when the admin has finished editing all the price fields, rather than each time the admin makes a single change. There are no improvements to be made for this objective.
44	Admins can set each line's status to open or closed using drop down lists.	I have met this objective by having a drop-down list for each line, allowing the line statuses to be changed. Evidence of this in test 56.	I have met this objective well, because having drop down lists is an easy way for admins to select from a few select options of statuses. There are no improvements to be made for this objective.
45	The changes to line statuses are updated in the local database when the admin	I have completed this objective by adding an 'Apply Changes' button for	I have met this objective well as having this button means that the database only needs to be accessed

	presses the 'Apply Changes' button.	admins to commit their status changes to the local database. Evidence of this in test 56.	once when the admin is finished editing the statuses, rather than each time an admin makes a single change. There are no improvements to be made for this objective.
46	Send a URL request to the TfL API, and parse to JSON.	I have met this objective by calling the TfL API with a URL request, and then parsing the result to a JSON array. Evidence of this in test 58 and screenshot 9.	I have met this objective well as my subroutine returns a JSON array from a URL parameter. I have done this by sending the URL as a request to the TfL open REST API, and then parsing the response to JSON format. For this objective I have used JSON rather than XML because JSON is a lot easier to read and debug and has arrays which makes it easier to get specific data from each API response. As for the API, there is no other option for getting train information than from the TfL API, because it is their train network.
47	Use a normalised database for all the details of the system.	I have met this objective by using primary and foreign keys to link multiple tables to build up a database in third normal form. Evidence of this in screenshot 10.	I think that I have met this objective very well as I have used third normal form to eliminate data redundancy. I have used third normal form so that my tables have no non-key dependencies, making my database clearer to use and understand. Preventing duplicate data is also very important as it makes updating the database easier and means that memory is not wasted.
48	Merge sort stations in alphabetical order based on their UNICODE values, when the program begins.	I have met this objective by coding a merge sort in my 'MergeSortStations' class. Evidence of this in test 59.	I have met this objective well as I have coded an efficient merge sort using recursion, sorting the station names into alphabetical order based on their UNICODE values. I am using this sort over insertion sort or quicksort as merge sort has a worst-case time complexity of O(n*log n) whereas insertion sort and

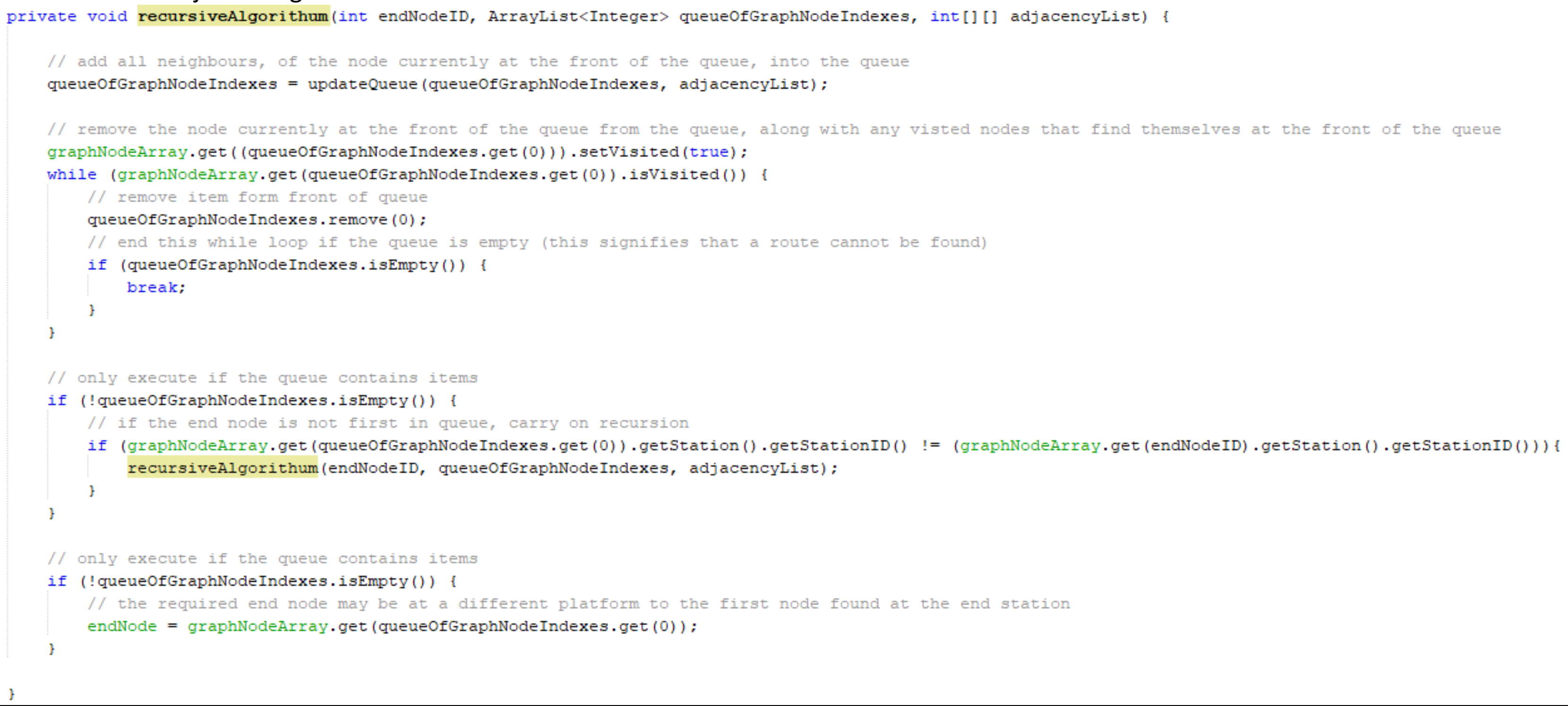
			quicksort both have a complexity of $O(n^2)$. Insertion sort and quicksort only run faster for short arrays, and my array of stations is not short. I do not think I could have coded my merge sort to be any more efficient. I think recursion is the best way to code a merge sort as the algorithm is naturally recursive.
49	My merge sort should use recursion.	I have met this objective by using recursion in my merge sort in the 'MergeSortStations' class. Evidence of this in screenshot 11.	I have met this objective well by using recursion when the 'mergeSort' method calls itself in the 'MergeSortStations' class. I think that I have met the objective well because recursion is the best way to code a merge sort as the algorithm is naturally recursive. Recursion also makes my algorithm easier to read and understand for anyone who needs to read and understand my code.
50	I will read from CSV files in order to enter all station and connection information into my database from these CSV datasets.	I have successfully met this objective by reading the datasets from the CSV files and parsing the result to a CSV format. I have then stored this station and connection information in my database tables. Evidence of this in test 60.	I think that this objective has been met well as the data from the CSV files is successfully transferred to my database. Transferring the data into the database takes a while, as shown in test 60, but this is because the TfL API needs to be accessed multiple times and is not to do with the speed of accessing and processing the CSV files. Nevertheless, it should be noted that there is not a great need for this process to be fast and optimised as it will only be run once when the database is first created, so the efficiency of the algorithm will never affect the user.

			I have used CSV files as this is the format provided with the datasets on GitHub.
51	I will read an image file to set it as the icon image of my application.	I have successfully met this objective by getting the 'UndergroundIcon' PNG stored in the files of my program and setting this as the icon of my application whenever it is run. Evidence of this in test 61.	I think that I have met this objective well as the icon of my application is set making my program look cleaner, with more polish, an unequivocal upgrade to the default Java logo. I have used a PNG image format as JPEG images cannot have a transparent background. A transparent background is very important for the icon so it can display on the taskbar without a background behind it.

Screenshot evidence for objectives

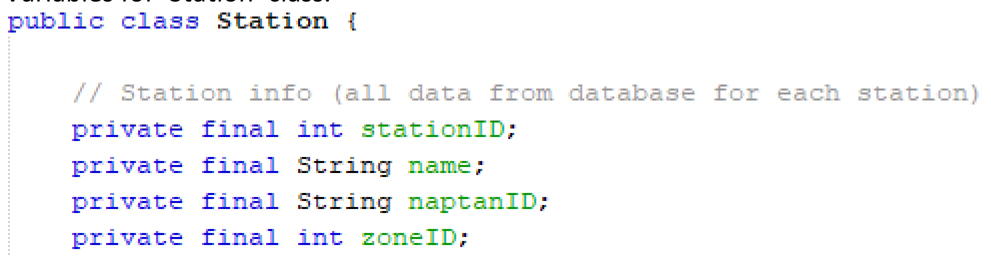
Screenshot

1 Method from ‘DijkstrasAlgorithm’ class.



```
private void recursiveAlgorithum(int endNodeID, ArrayList<Integer> queueOfGraphNodeIndexes, int[][] adjacencyList) {

    // add all neighbours, of the node currently at the front of the queue, into the queue
    queueOfGraphNodeIndexes = updateQueue(queueOfGraphNodeIndexes, adjacencyList);

    // remove the node currently at the front of the queue from the queue, along with any visted nodes that find themselves at the front of the queue
    graphNodeArray.get((queueOfGraphNodeIndexes.get(0))).setVisited(true);
    while (graphNodeArray.get(queueOfGraphNodeIndexes.get(0)).isVisited()) {
        // remove item form front of queue
        queueOfGraphNodeIndexes.remove(0);
        // end this while loop if the queue is empty (this signifies that a route cannot be found)
        if (queueOfGraphNodeIndexes.isEmpty()) {
            break;
        }
    }

    // only execute if the queue contains items
    if (!queueOfGraphNodeIndexes.isEmpty()) {
        // if the end node is not first in queue, carry on recursion
        if (graphNodeArray.get(queueOfGraphNodeIndexes.get(0)).getStation().getStationID() != (graphNodeArray.get(endNodeID).getStation().getStationID())){
            recursiveAlgorithum(endNodeID, queueOfGraphNodeIndexes, adjacencyList);
        }
    }

    // only execute if the queue contains items
    if (!queueOfGraphNodeIndexes.isEmpty()) {
        // the required end node may be at a different platform to the first node found at the end station
        endNode = graphNodeArray.get(queueOfGraphNodeIndexes.get(0));
    }

}
```

2 Variables for ‘Station’ class.



```
public class Station {

    // Station info (all data from database for each station)
    private final int stationID;
    private final String name;
    private final String naptanID;
    private final int zoneID;
```

Variables for ‘StationConnection’ class.

```
public class StationConnection {

    // Connection info (all data from database for each connection)
    private final int stationID1;
    private final int stationID2;
    private final int travelTime; // the time to travel between these two stations
    private final Line line;     // Line object holds all line info (id,name,colour,status)
```

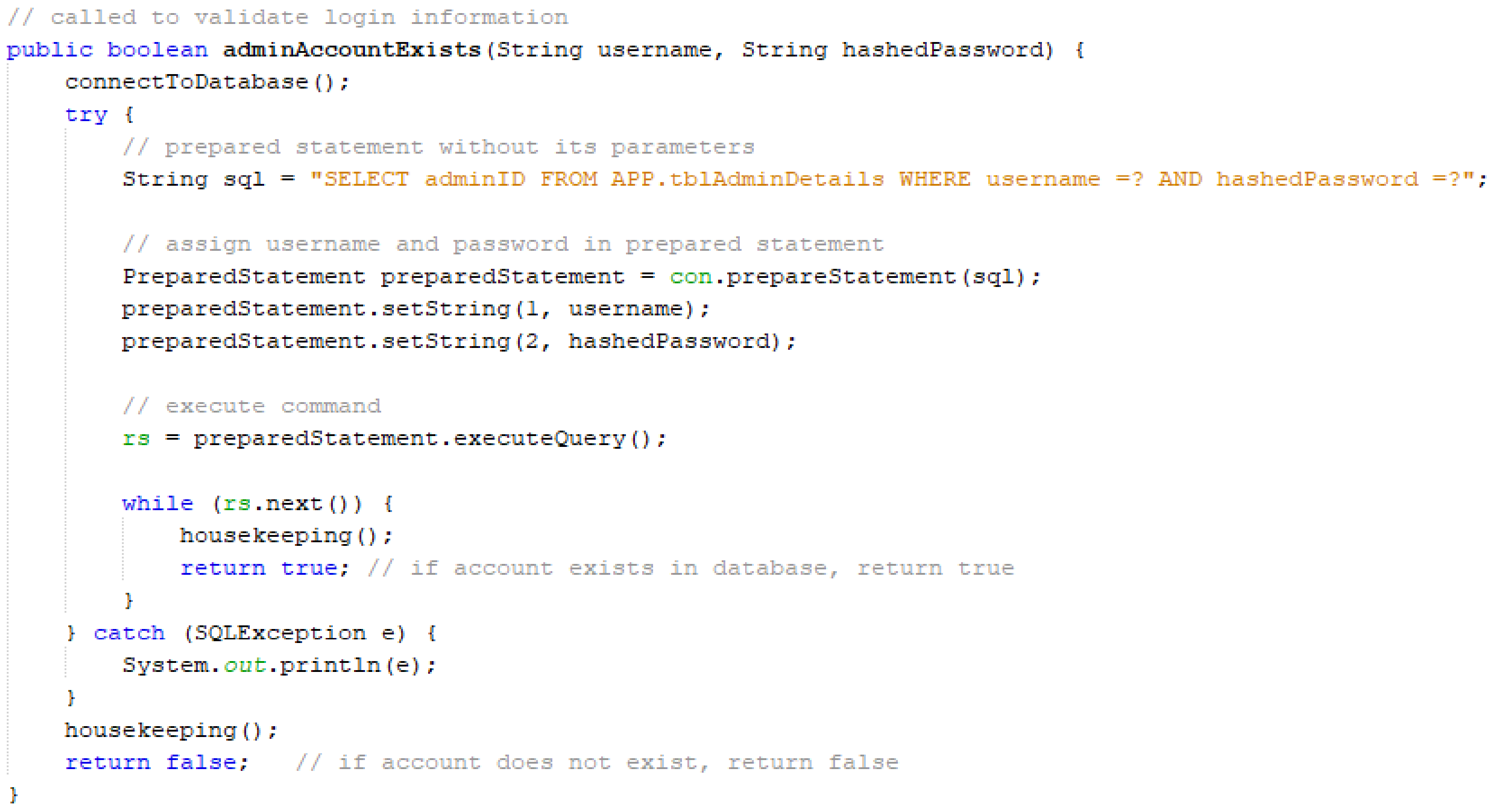
Variables for ‘GraphNode’ class.

```
public class GraphNode {

    // variables used for pathfinding
    private final int indexInGraphNodeArray;
    private int timeFromStartStation = 999;
    private boolean visited = false;
    private GraphNode previousNode;
    private Line previousLine = null;

    // Station info
    private final Station stationThisRepresents;
```

Variables for ‘GraphEdge’ class.

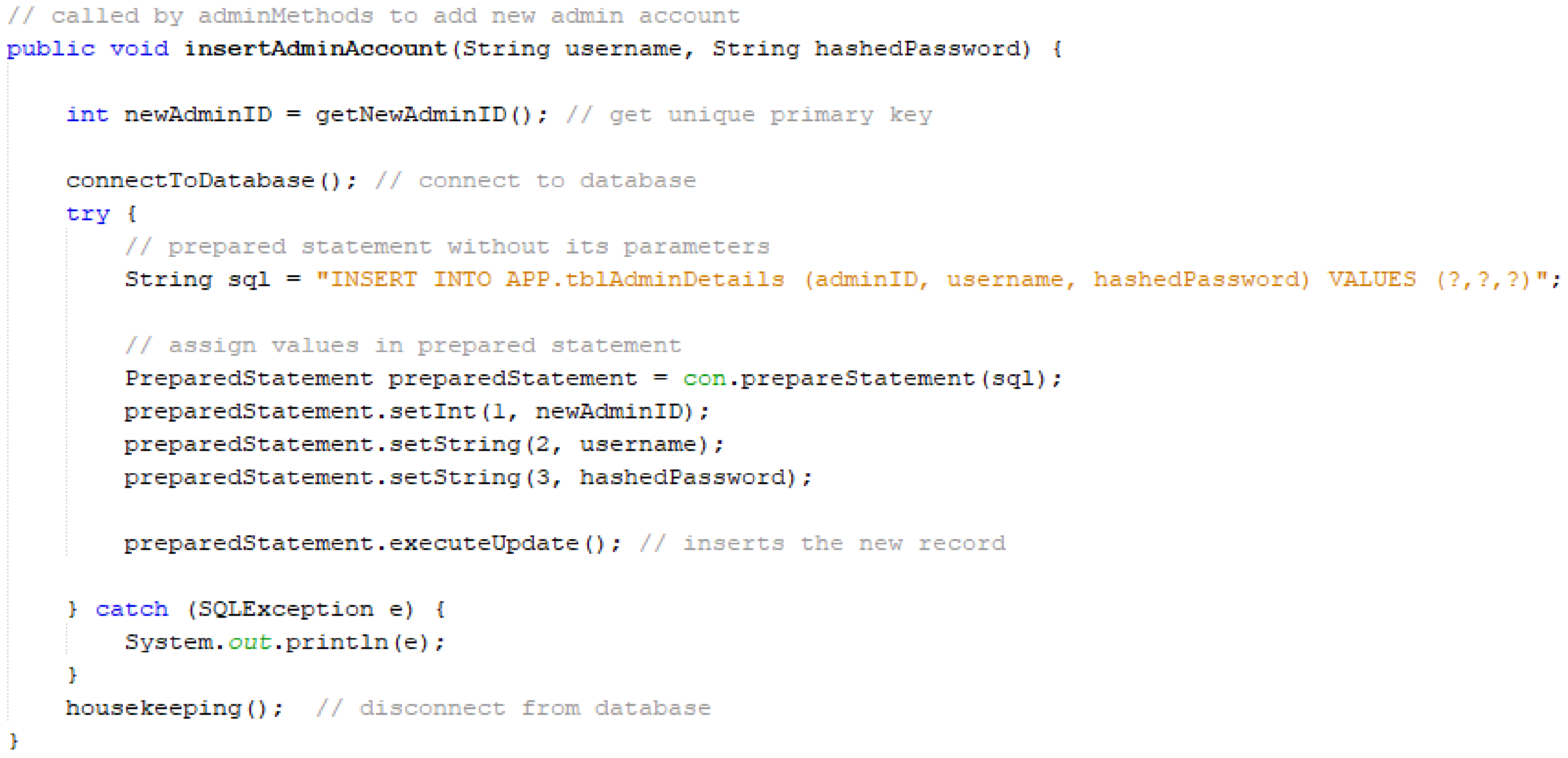
```
public class GraphEdge {

    private final GraphNode node1;
    private final GraphNode node2;
    private int travelTime;  // the time to travel between these two nodes
    private final Line line;   // Line object holds all line info (id,name,colour,status)
```

3 Method from ‘AccessDatabase’ class. Called from ‘doesAdminAccountExist’ method in ‘AdminMethods’ class.



```java
// called to validate login information
public boolean adminAccountExists(String username, String hashedPassword) {
    connectToDatabase();
    try {
        // prepared statement without its parameters
        String sql = "SELECT adminID FROM APP.tblAdminDetails WHERE username =? AND hashedPassword =?";

        // assign username and password in prepared statement
        PreparedStatement preparedStatement = con.prepareStatement(sql);
        preparedStatement.setString(1, username);
        preparedStatement.setString(2, hashedPassword);

        // execute command
        rs = preparedStatement.executeQuery();

        while (rs.next()) {
            housekeeping();
            return true; // if account exists in database, return true
        }
    } catch (SQLException e) {
        System.out.println(e);
    }
    housekeeping();
    return false;   // if account does not exist, return false
}
```

4 Method from ‘AccessDatabase’ class.



```
// called by adminMethods to add new admin account
public void insertAdminAccount(String username, String hashedPassword) {

    int newAdminID = getNewAdminID(); // get unique primary key

    connectToDatabase(); // connect to database
    try {
        // prepared statement without its parameters
        String sql = "INSERT INTO APP.tblAdminDetails (adminID, username, hashedPassword) VALUES (?,?,?)";

        // assign values in prepared statement
        PreparedStatement preparedStatement = con.prepareStatement(sql);
        preparedStatement.setInt(1, newAdminID);
        preparedStatement.setString(2, username);
        preparedStatement.setString(3, hashedPassword);

        preparedStatement.executeUpdate(); // inserts the new record

    } catch (SQLException e) {
        System.out.println(e);
    }
    housekeeping();  // disconnect from database
}
```

5 Method from ‘AdminMethods’ class.

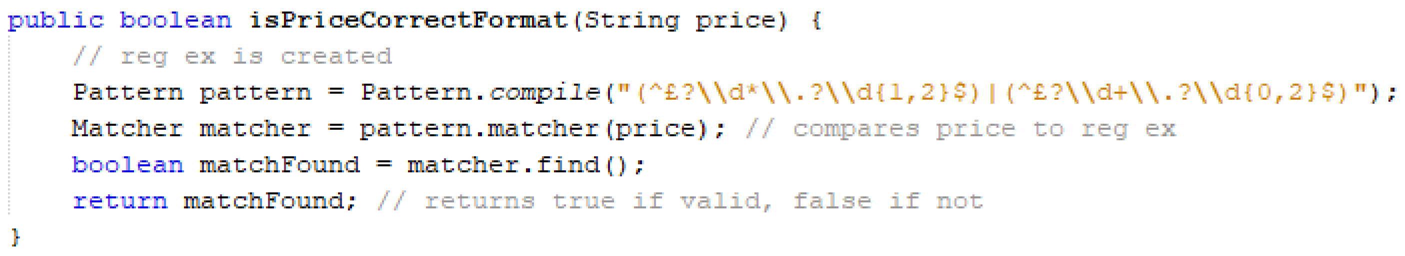
```
// SHA-256 hash on password
private String getSHA256hash(String password) {
    String hashedPassword = null;
    try {
        MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");  // select SHA-256 hash
        byte[] msgDigest = messageDigest.digest(password.getBytes());        // apply hash to password

        BigInteger bigInt = new BigInteger(1, msgDigest);   // convert from byte array
        hashedPassword = bigInt.toString(16);               // to a hashed string
    } catch (NoSuchAlgorithmException e) {
        System.out.println(e);
    }
    return hashedPassword;
}
```

6 Method from ‘AccessDatabase’ class.

```
// generates a unique primary key for a new admin account
// by returning the number one greater than the current highest adminID
private int getNewAdminID() {
    connectToDatabase();
    int newAdminID = 1;
    try {
        String SQL = "SELECT MAX(adminID) as maxID FROM APP.tblAdminDetails";  // the number returned is to be named maxID

        rs = stmt.executeQuery(SQL);
        rs.next();

        newAdminID = rs.getInt("maxID") + 1;  // the new primary key must be one greater than the current maxID

    } catch (SQLException e) {
        System.out.println(e);
    }
    housekeeping();
    return newAdminID;
}
```

7 Method from ‘AccessDatabase’ class.

```java
// called by 'AdminMethods' to fetch pricing information
public String[][] getZonePrices() {
    connectToDatabase();
    String[][] prices2Darray = new String[10][10];

    try {
        // create cross-table SQL statement
        String sql = "SELECT * FROM APP.tblZonePricing\n"
                + "JOIN APP.tblZone ON APP.tblZonePricing.zoneID1 = APP.tblZone.zoneID\n"
                + "UNION\n"
                + "SELECT * FROM APP.tblZonePricing\n"
                + "JOIN APP.tblZone ON APP.tblZonePricing.zoneID2 = APP.tblZone.zoneID";
        // execute SQL query
        rs = stmt.executeQuery(sql);

        while (rs.next()) {
            // get zone price information
            int rowZoneID = rs.getInt("zoneID1");
            int columnZoneID = rs.getInt("zoneID2");
            String rowName = rs.getString("zoneName");
            String columnName = rs.getString("zoneName");
            String price = rs.getString("price");
            if (columnZoneID != rowZoneID) {
                rs.next();
                rowName = rs.getString("zoneName");
            }
            // add zone price information to 2D array
            prices2Darray[columnZoneID][0] = columnName;
            prices2Darray[0][rowZoneID] = rowName;
            prices2Darray[columnZoneID][rowZoneID] = price;
        }
    } catch (SQLException e) {
        System.out.println(e);
    }
    housekeeping();
    return prices2Darray; // return zone price information
}
```

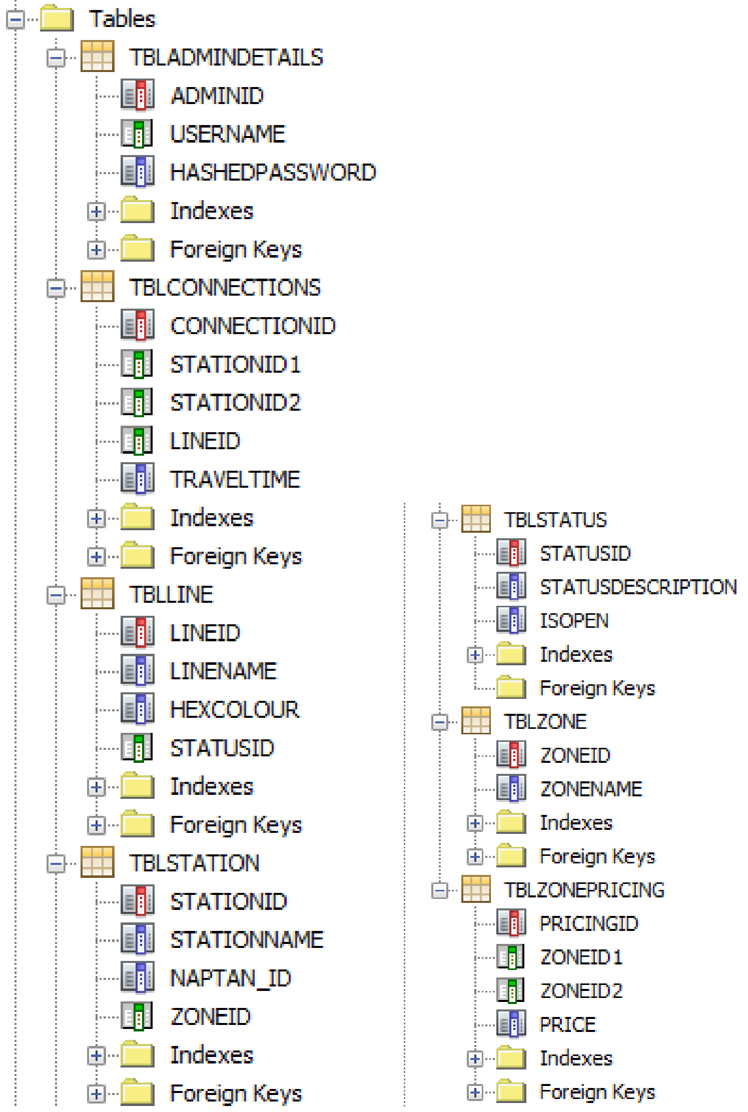
8	Method from ‘AdminMethods’ class. 

```
public boolean isPriceCorrectFormat(String price) {
    // reg ex is created
    Pattern pattern = Pattern.compile("(^£?\\d*\\.?\\d{1,2}$)|(^£?\\d+\\.?\\d{0,2}$)");
    Matcher matcher = pattern.matcher(price); // compares price to reg ex
    boolean matchFound = matcher.find();
    return matchFound; // returns true if valid, false if not
}
```

9

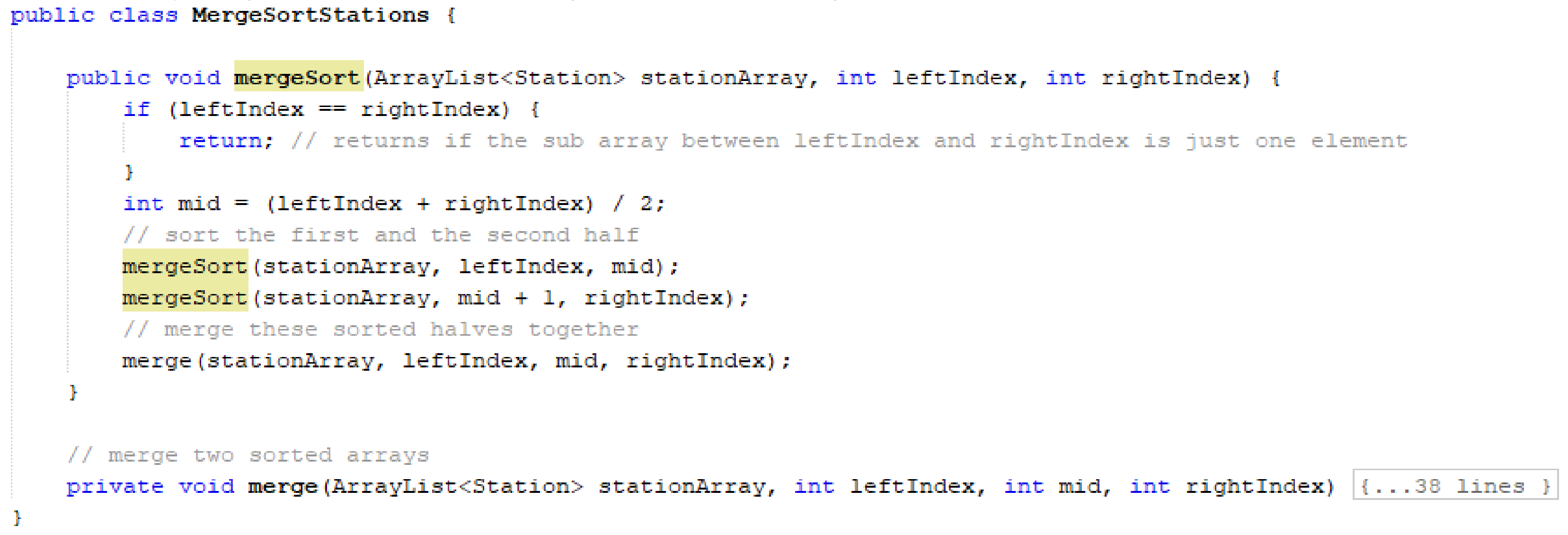
Method from 'AccessAPI’ class.

```java
private JSONArray sendUrlRequest(String url) {
    try {
        URL URLobj = new URL(url);

        HttpURLConnection con = (HttpURLConnection) URLobj.openConnection();   // defult connection is GET

        // print out for debugging:
        //
        //int responseCode = con.getResponseCode();
        //System.out.println("Sending 'GET' request to URL : " + url);
        //System.out.println("Response Code : " + responseCode); // returns response code so I know what the error is (e.g. 404, 490..)

        // read the response
        StringBuilder response;
        try (BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()))) {
            String inputLine;
            response = new StringBuilder();               // StringBuilder is used to store every line received in the URL request
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);               // each line in the response is added to the response variable
            }
        }

        // response parsed to JSON. JSONArray is used to store the different
        // JSONObjects from the response, there are different objects for each train
        JSONArray JSONtrainObjectArray = new JSONArray(response.toString());

        //System.out.println(JSONtrainObjectArray.toString() + "\n");  // prints the whole JSON array

        return JSONtrainObjectArray;  // return the JSONObject Array

    } catch (IOException | JSONException e) {
        System.out.println("ERROR:" + e);
    }
    return new JSONArray();
}
```

10	Below is the structure of my database showing all column names for each table. Red indicates a primary key; green indicates a foreign key, and blue indicates a standard column for data. 

```
Tables
  TBLADMINDETAILS
    ADMINID
    USERNAME
    HASHEDPASSWORD
    Indexes
    Foreign Keys
  TBLCONNECTIONS
    CONNECTIONID
    STATIONID1
    STATIONID2
    LINEID
    TRAVELTIME
    Indexes
    Foreign Keys
  TBLLINE
    LINEID
    LINENAME
    HEXCOLOUR
    STATUSID
    Indexes
    Foreign Keys
  TBLSTATION
    STATIONID
    STATIONNAME
    NAPTAN_ID
    ZONEID
    Indexes
    Foreign Keys
  TBLSTATUS
    STATUSID
    STATUSDESCRIPTION
    ISOPEN
    Indexes
    Foreign Keys
  TBLZONE
    ZONEID
    ZONENAME
    Indexes
    Foreign Keys
  TBLZONEPRICING
    PRICINGID
    ZONEID1
    ZONEID2
    PRICE
    Indexes
    Foreign Keys
```

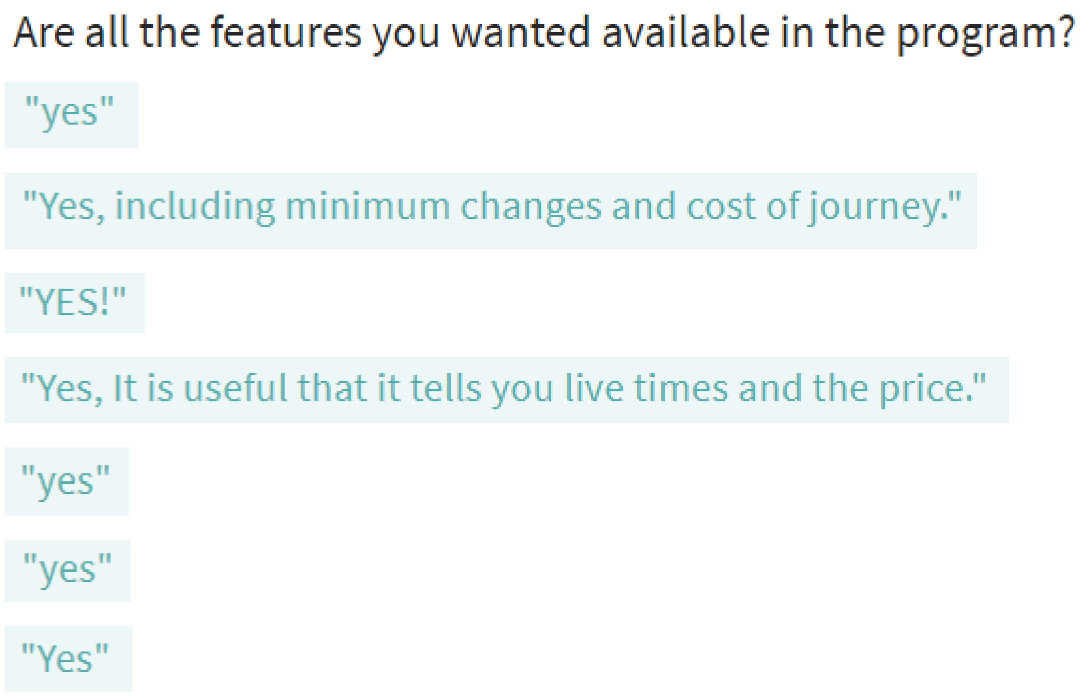
11 Contents on my ‘MergeSortStations’ class, showing recursion where ‘mergeSort’ calls itself.



```
public class MergeSortStations {

    public void mergeSort(ArrayList<Station> stationArray, int leftIndex, int rightIndex) {
        if (leftIndex == rightIndex) {
            return; // returns if the sub array between leftIndex and rightIndex is just one element
        }
        int mid = (leftIndex + rightIndex) / 2;
        // sort the first and the second half
        mergeSort(stationArray, leftIndex, mid);
        mergeSort(stationArray, mid + 1, rightIndex);
        // merge these sorted halves together
        merge(stationArray, leftIndex, mid, rightIndex);
    }

    // merge two sorted arrays
    private void merge(ArrayList<Station> stationArray, int leftIndex, int mid, int rightIndex) {...38 lines }
}
```

Analysis of User Feedback

I have asked users of my system some questions about what they thought of my program in a user feedback survey. Below are the questions and some of the answers given in this survey. I have then commented on their feedback and the action I will now take.



Are all the features you wanted available in the program?

"yes"

"Yes, including minimum changes and cost of journey."

"YES!"

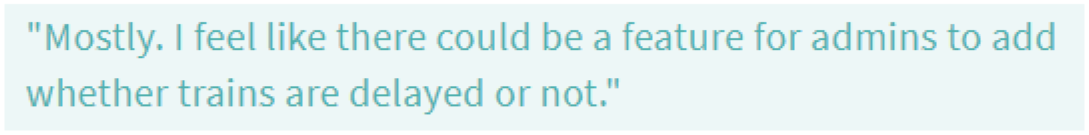
"Yes, It is useful that it tells you live times and the price."

"yes"

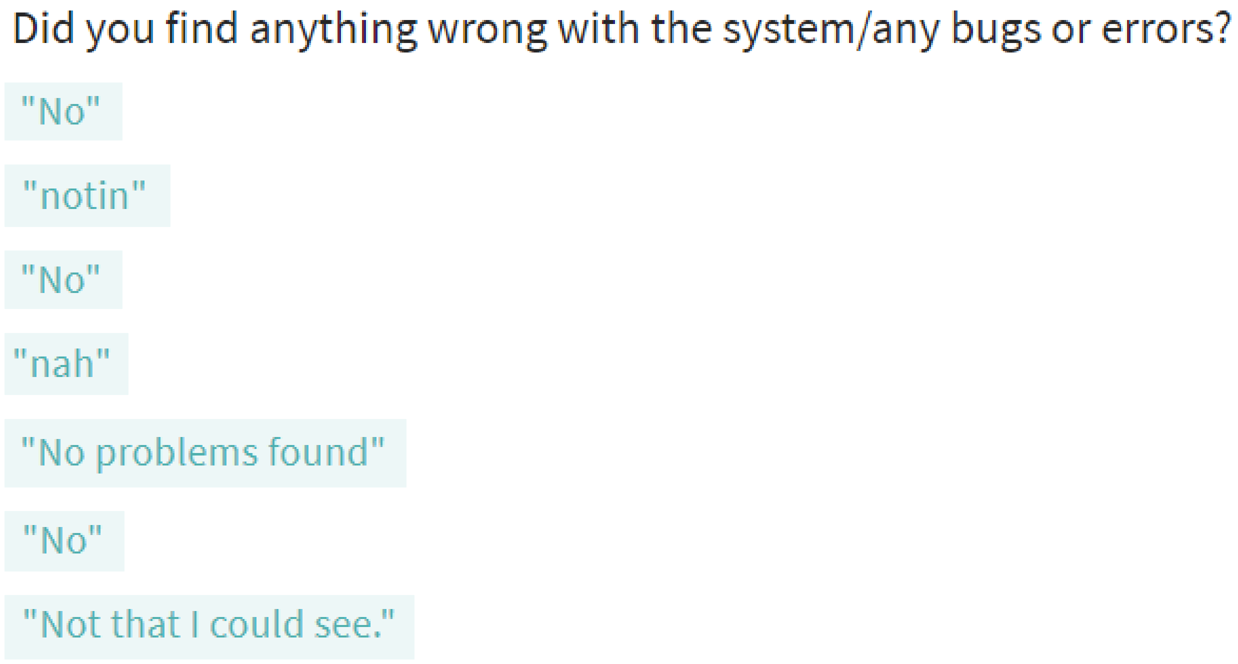
"yes"

"Yes"

All the above responses are from people who have seen all the feature they wanted added to the program, with two people mentioning how they liked the minimum changes, live train times, and price of journey.


"Mostly. I feel like there could be a feature for admins to add whether trains are delayed or not."

Despite having live expected arrival times, which should mean that delays are accounted for, I agree with this feedback. It would be useful for users to know if there are possible extra delays to their arriving trains, as well as possible delays on their route that may increase their journey time. This could be implemented by giving admins the ability to enter what lines and trains have delays as suggested by this user, with delays being shown to users with their journey results, or on a separate panel where users could go to view all delays.


Did you find anything wrong with the system/any bugs or errors?

"No"

"notin"

"No"

"nah"

"No problems found"

"No"

"Not that I could see."

From the results above we can see that most users faced no bugs when running my program. This shows that my program has been tested well and there are no obvious errors.

"Could not enter a large value for price into the table"

This is not a problem under any normal circumstances.

What did you like about the system?

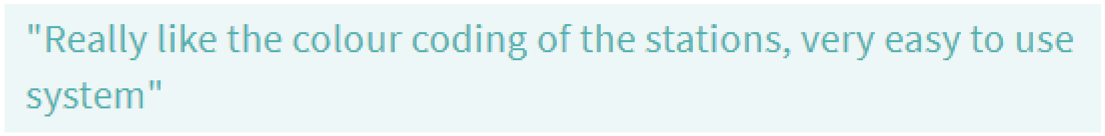
"How well it worked to show the journey/route. Without requesting train arrival times, the route returned very quickly."

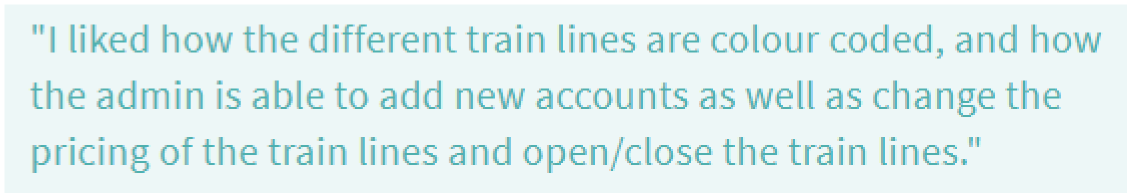
"It is very quick to find results"

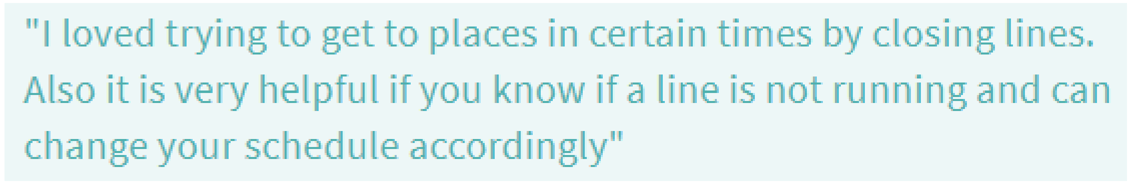
This is great news as this was one of my objectives that I met very well. This user feedback shows that having a fast pathfinding algorithm was noticed by users and it was worth the effort I put in to tuning it to be faster.

"Autocomplete on text fields"

This user liked the autocomplete function. This is useful to know as the main purpose of adding this feature was to improve the user's experience by making it easier for them to type in station names. To further improve the user experience I could improve my autocomplete as well as adding other possible features such as speech-to-text.


"Really like the colour coding of the stations, very easy to use system"


"I liked how the different train lines are colour coded, and how the admin is able to add new accounts as well as change the pricing of the train lines and open/close the train lines."


"I loved trying to get to places in certain times by closing lines. Also it is very helpful if you know if a line is not running and can change your schedule accordingly"

The above feedback shows users that liked; the colour coded lines, how routes were redirected when there were line closures, and the abilities of admins to change prices and open new accounts.

The use of colours was useful to these users but for further development I would have to think about how the colour blind and visually impaired would view the system. I could start by asking people who are colour blind and with poor eyesight to test my program.

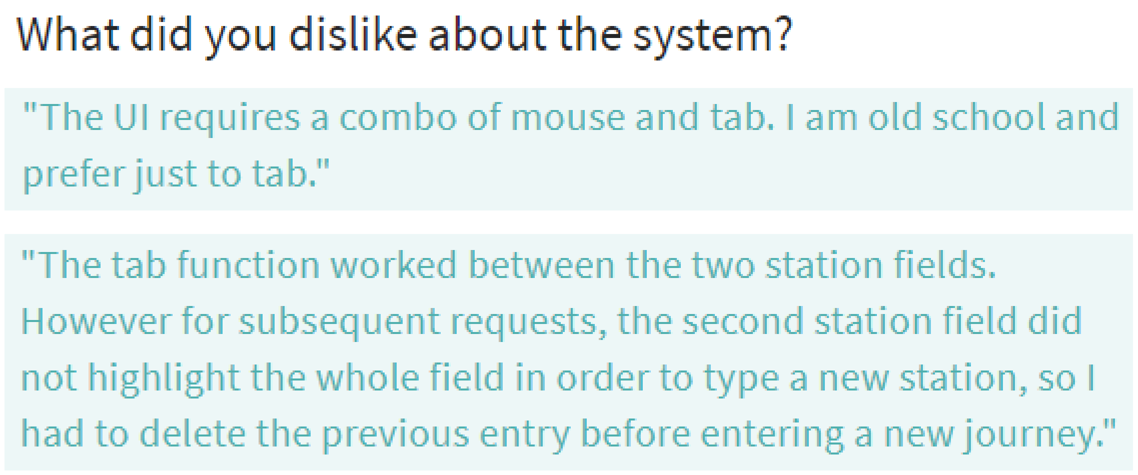
The last of the three responses above points out how redirecting routes when there are line closures allows them to change their schedule if there are delays caused by closed lines.

> "the pricing and time features were cool"

This user liked how the cost and time of journeys were displayed which were some of my objectives and main functions of my system.

> "I like how precise everything is with all the data. It must have taken a long time to input it all. It's also very easy to understand and I also like how you've hashed the passwords for extra security."

This user liked to see the hash function being used to secure the passwords in my database. They also thought that entering all the data into my database would have taken a long time, but due to me importing the datasets from CSV files the process was quick and easy, and much better than entering it manually.

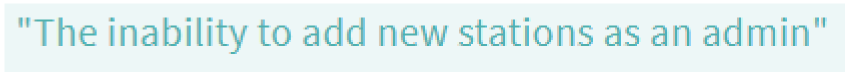


What did you dislike about the system?

"The UI requires a combo of mouse and tab. I am old school and prefer just to tab."

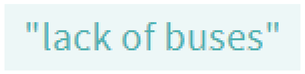
"The tab function worked between the two station fields. However for subsequent requests, the second station field did not highlight the whole field in order to type a new station, so I had to delete the previous entry before entering a new journey."

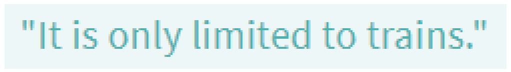
The two responses above show these users did not like how my system could not tab between all fields. This is something that NetBeans has done automatically, allowing users to tab between text fields but not onto buttons or checkboxes. However, this is a change I could make myself and I think it would be useful as it would improve the quality of my system, making it more natural to use as many other systems allow tabbing between all fields.



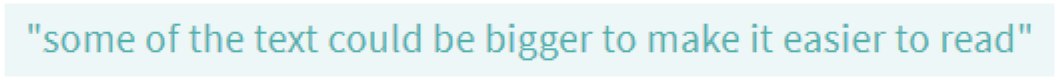
"The inability to add new stations as an admin"

I agree with this feedback and think that the ability of adding new stations to the system would be useful as new stations are added to lines on the Underground in real life. I would also like to combine this with being able to add new lines to the system as new lines such as Crossrail are currently under construction.

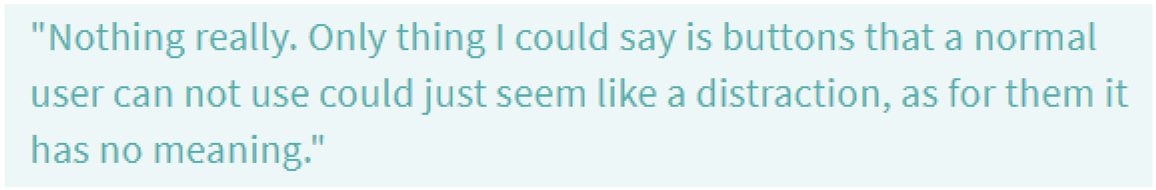

"lack of buses"


"It is only limited to trains."

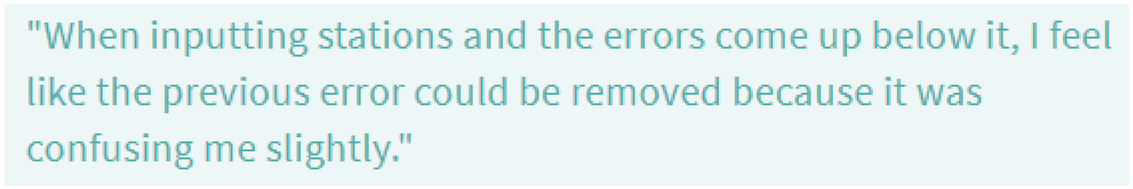
I like the idea of expanding to other modes of transport in London, allowing users to travel the fastest possible route even if this means not on trains. In particular, I would like to add routes for buses as well as walking between stations as sometimes walking between two stations is a lot faster than taking the Underground, and buses can access areas of London where the train lines do not extend.


"some of the text could be bigger to make it easier to read"

This user found the text on my GUI difficult to read. This is something I did not consider but I should make considerations for in the future, especially for people with poor eyesight or using smaller displays. These considerations may include the ability to change the size of the text and automatically adjusting the size of the font based on the user's screen size.

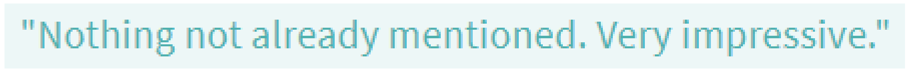

"Nothing really. Only thing I could say is buttons that a normal user can not use could just seem like a distraction, as for them it has no meaning."

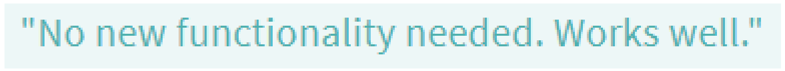
One user has pointed out how the 'Admin Access' button can be seen by users, even though it cannot be used by them. This could be avoided by redesigning my system to work with two programs running off the same database, one for users with no admin access, and one for admins with full admin access to update the shared database. This would require a large redesign and would benefit from having an online database that can be connected to by all users and admins, with the users and admins having access to different software.


"When inputting stations and the errors come up below it, I feel like the previous error could be removed because it was confusing me slightly."

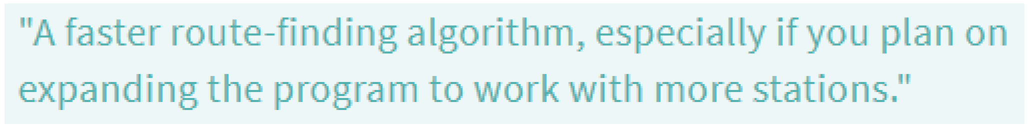
When implementing error messages, I thought it would look good to have new error messages appear above previous messages, with these previous messages being moved down on the GUI. However, this was clearly not the right decision by me as something as simple as error messages should not be confusing to view. Therefore, in the future I will change this so a new error message removes previous messages because the functionality and ease of use of the system should be the priority.

Are there any improvements to be made?


"Nothing not already mentioned. Very impressive."


"No new functionality needed. Works well."

These responses show that some users saw nothing that could be improved.


"A faster route-finding algorithm, especially if you plan on expanding the program to work with more stations."

This user is thinking ahead at whether I would need a faster route-finding algorithm for a larger graph, if I were to expand my network. However, as I have talked about before, other implementations such as A* are not possible as I don’t have the datasets required. I also do not think that a larger graph would be much of a problem for my current path finding solution, not that I plan to expand my system anywhere outside of London.

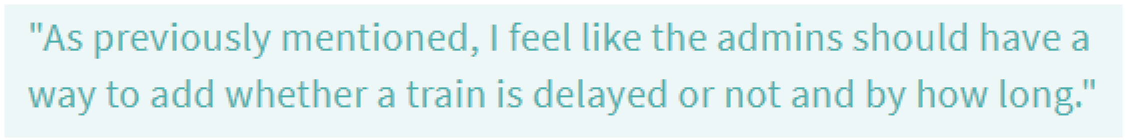
"Include areas outside London"

"Expand it to include more of the rail network."

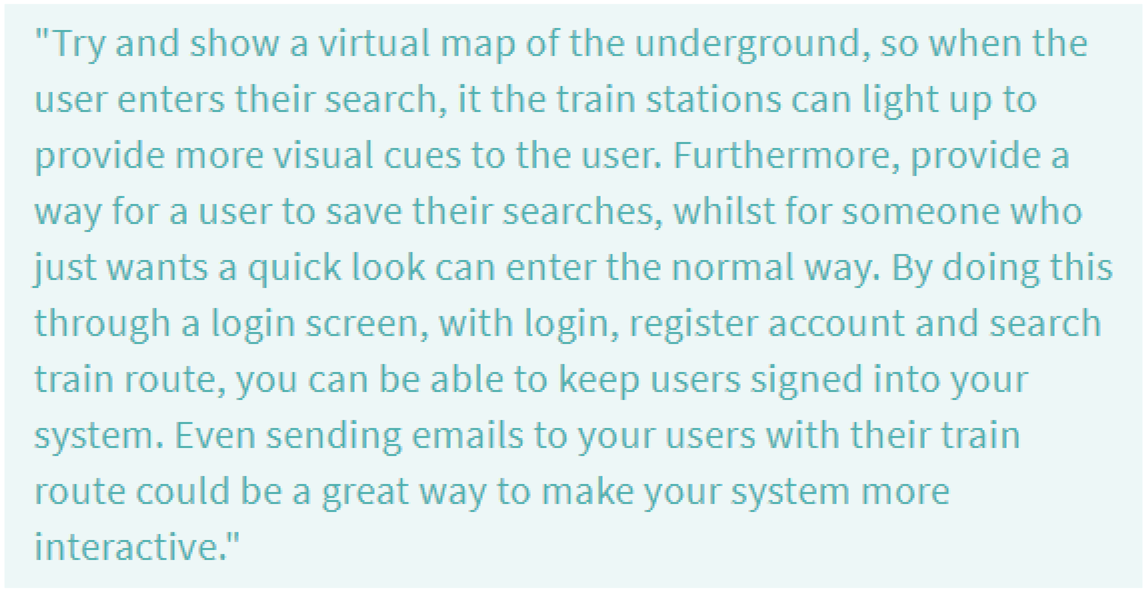
For my system I do not want to expand outside London as I want my system's purpose to be for people needing to travel around London, and not across the whole country. Expanding my project on that scale would also be a real challenge to code and to get all the data needed.

"make the error messages dissapear if they are no longer needed"

A second user has mentioned not liking how I am displaying error messages. As said before, I would like to change this in order to make error messages disappear when a new one appears.


"As previously mentioned, I feel like the admins should have a way to add whether a train is delayed or not and by how long."

I have analysed this user's idea when they mentioned it for the first question. I have agreed that I would like to add a feature like this so admins can enter whether there are delays for trains.


"Try and show a virtual map of the underground, so when the user enters their search, it the train stations can light up to provide more visual cues to the user. Furthermore, provide a way for a user to save their searches, whilst for someone who just wants a quick look can enter the normal way. By doing this through a login screen, with login, register account and search train route, you can be able to keep users signed into your system. Even sending emails to your users with their train route could be a great way to make your system more interactive."

This user has suggested a couple of improvements for my system. I would like to include something similar to their first suggestion, such as having an option where the user can choose to open a new window displaying the tube map. I agree that this may be helpful to some users even though I said I didn’t want this feature in my analysis section.

However, I think the second suggestion by this user of having user accounts would be a pointless addition to the system as I don’t think that saving routes is needed as they are very quick to find by searching so nothing is gained. This would add unnecessary complexity to the system, make it more complicated for the users, and would require me to store personal information when it is avoidable.

Changes and features I want to include based on this user feedback:	Changes suggested by users that I will not include:
• Admins can enter whether there are delays for trains. • Tab between all text fields and buttons. (currently only works between text fields) • The option of adding new stations and lines. • Include other modes of transport. • Adjusting text size to make it easier to read. • Separate systems for users and admins that connect to the same database. • Deleting previous error messages when a new one appears, rather than printing the new one above. • An option to open and view the tube map.	• Faster pathfinding algorithm. • Routes outside of London. • User accounts.

Improvements/Extensions

This section mentions some improvements and extensions in detail that would make my system better. These are based on user feedback and areas that I know could be improved.

1) Use a salt to improve password security.

Although my SHA-256 hashing algorithm is secure and cannot be reversed it is still susceptible to rainbow table attacks, where the attacker uses a large dataset of hashed data to crack unsecure passwords such as commonly used passwords and plain English words. To protect against these attacks, and make my admin passwords more secure, I can use a salt alongside my hashing algorithm.

Adding a salt would not be difficult. It requires an extra column in the admin table of my database for the salt to be stored and simple random generation for the salt to be added. This makes this extension very appealing as it will offer increased security while it should take little time to implement.

2) Train times for DLR and East London Line.

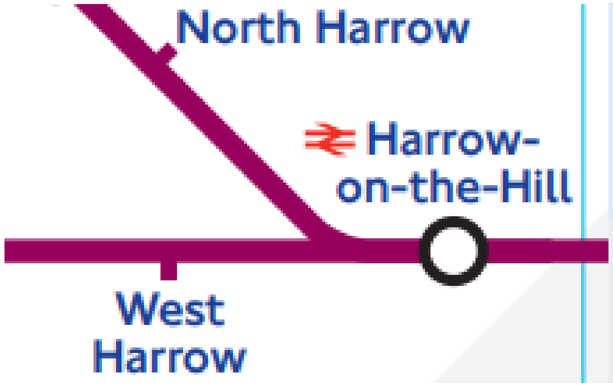
I was unable to get live train times for the DLR and East London lines as these two lines have no vehicleIDs. This means that I can get the arrival times for trains at the stations but cannot check if they are going in the right direction. Even though these vehicleIDs are not displayed by TfL for these two lines, I think there is a solution as they say that all information displayed on their website can be obtained from the API.

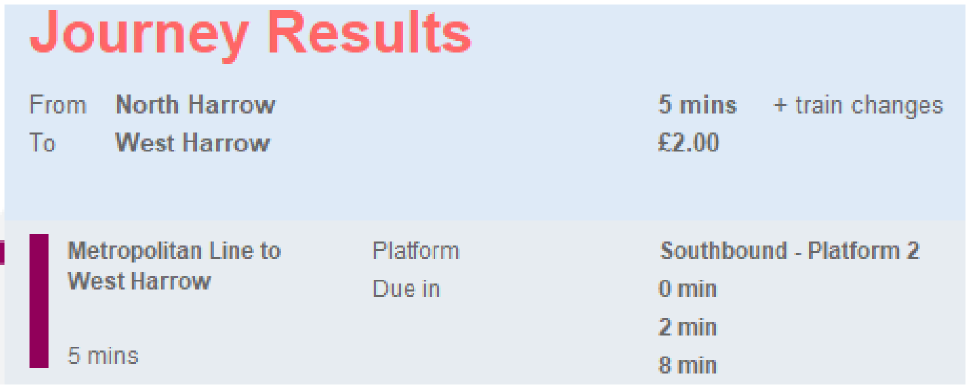
Therefore, to improve my system I want to find a way to obtain these DLR and East London line arrival times. I may need to contact people at TfL about this as I have already looked but I have not found a reliable way to get live information for these lines.

This improvement is very important as some users may travel on the DLR and East London lines and I can't currently show them any live train information.

3) Mention Train changes when the same line forks.

When there is a split in a train line my program does not say that there is a train change involved, even though train changes are sometimes required even if they are not line changes. For example: when going from North Harrow to West Harrow it says to take the metropolitan from North Harrow to West Harrow and does not mention that you must change at Harrow-on-the-Hill.





Journey Results

From North Harrow
To West Harrow
5 mins + train changes
£2.00

Metropolitan Line to West Harrow
5 mins

Platform: Southbound - Platform 2
Due in: 0 min
2 min
8 min

To improve my system, I must change my pathfinding to work with new information about what sections of the same line are connected in what directions. This may require me to rethink how I set out my graph to represent the Tube map, but it is achievable.

4) The option of adding new stations and lines. (based on user feedback)

A useful new feature for admins would be the ability for new stations and lines to be added. This extension would make my system better as it could adapt to changes to the network such as the new line 'Crossrail' being constructed and possible new line extensions. New stations are also added along new lines and to existing lines. It would also be useful to be able to remove stations as sometimes they are closed when running them becomes unprofitable for TfL.

I would add these new features by adding more options to the admin menu that allow admins to make changes to the local database. The station, line and connection tables will all be updated by admins.

I have coded my project in such a way that my route finding is dynamic with any size graph and so would require minimal changes to implement this feature. This feature would be very useful while not taking too long to implement.

5) Ability to close stations and certain sections of track rather than whole lines.

Closing individual stations and sections of track would be more useful than whole lines as it is more likely that sections of track fail (e.g. due to low temperatures or signal failure) than the whole line closing (e.g. for a tube strike). Stations can also be closed when there are bags left unattended (posing a bomb threat) or there can be suicides on the platforms causing both platform closures and line closures. Video evidence linked below shows documentaries recording moments where sections of lines, platforms, and whole stations need to be closed.

I plan to implement this in a similar way to how I currently close whole lines. I will use statuses and then allow admins to change the statuses of stations and line sections via drop-down lists. I will need to have some way of storing which section each station is in on a line. This may link in to how I store lines in sections in order to meet improvement No.3 above.

Evidence:
https://youtu.be/1eO93NFMsgU?t=124
https://youtu.be/em-Rd6-axms?t=131
https://youtu.be/em-Rd6-axms?t=529

6) Include other modes of transport. (based on user feedback)

Some of my user feedback mentioned expanding the area covered by my system to include more rail networks on a larger scale and to add new modes of transport such as buses. For my system I do not want to expand outside London as I want my system to be for people needing to travel around London, and not across the whole country. Expanding my project on that scale would be a real challenge, would take a lot of time to get all the data needed, and defeats the purpose of a system which is to help people commute around London.

However, the idea of expanding to other modes of transport is more reasonable and would be a good addition to my route finder as sometimes buses can be more direct and quicker than trains, and cover areas of London where there aren't stations. I also like the idea of having walking as another mode of transport as some stations are physically very close but require a long journey on the London Underground. Having this walking option would allow users to save money by not taking the train, as well as saving them precious time.

To implement this feature I would have to start by finding adequate datasets and then adding more tables to my database for modes of transport, as well as adding new records to many of my existing tables. This would be one of my more difficult and time-consuming extensions, but it would be a useful addition to make a perfect route finder for London rather than just the London Underground.

7) Adjusting text size to make it easier to read. (based on user feedback)

After receiving user feedback saying that the text was hard to read, I would like to change my system to have dynamic font sizes based on the size of the user's screen, as well as having options to increase the font size manually. This should allow people with poor eyesight or smaller displays to view my system with no problems.

To adjust the size of the text based on the user's screen size, I will have to get the screen size and automatically change the font settings when the program runs.

To allow the user to further manually edit the text size I will include an options menu where they can select their desired font size which will change the size of text across the system. I may include a few options such as 'small', 'medium', 'large', and 'very large', like other systems do as this will allow me to make sure everything is formatted correctly for each level of font sizes.