

Finding the fastest route to travel between London Underground stations

Analysis

Identification of problem

Overview:

I am going to make a system that allows users to find the route between two London Underground stations. It will allow passengers on London's Tube to enter start and destination stations and it will then display the route with the shortest travel time between them.

Other information will also be displayed such as the time of the next train leaving, the number of changes, and the price of the journey.

My system will also allow admins, for example TfL staff, to login and change the current status of a train line if it is under maintenance. My system will then redirect passengers, creating the best new route available.

Solving a problem:

This system solves the problem faced by many non-Londoners and tourists who are unfamiliar with the layout of the Tube and don't know how to get from A to B in the quickest possible time.

A knock-on effect of this problem is that people may have less time to experience London and view sites and are instead stuck on the not too pleasant Tube. Others who have tight schedules may be late for meetings and important engagements.

With my system users will be able to find the fastest route for their journey, allowing them to save precious time. The user will also be given important information such as the time of the next train and the platform they should go to.

Area of interest:

I have chosen this problem as I love the idea of solving mathematical problems (like traversing a graph) in real life. I have also used the Tube many times and have been left wondering if I could have gone on a more efficient route, in what appears to be a spaghetti-like layout of Tube lines.

Target user:

My target users will be people who are unfamiliar with the Tube as they may be tourists, foreign businessman, or just from outside London. These people may find it daunting to look at the confusing Tube map, yet would also like to know the quickest route to their destination. This system I am creating will abstract the complicated process of finding the quickest route across London for the user.

My application will also be useful when some lines are not available, as passengers can be shown new routes that can get them to their destination.

As the problem with the normal Tube map is its complexity, my system must be simplistic and easy to use so it can be a much-needed solution to people's traveling problems.

System constraints

My system must be simple and easy to use. This is because my users may have no ICT knowledge so they would be unable to navigate a complex system. Instead the UI must be simple, and it must be clear what each UI element does.

My users may also be in a rush if they need to catch a train as quickly as possible. This makes it important that my system is simple, so that a new user can find it easy to navigate quickly. The route must also be calculated quickly, and live information used to give the arrival times of Tube trains.

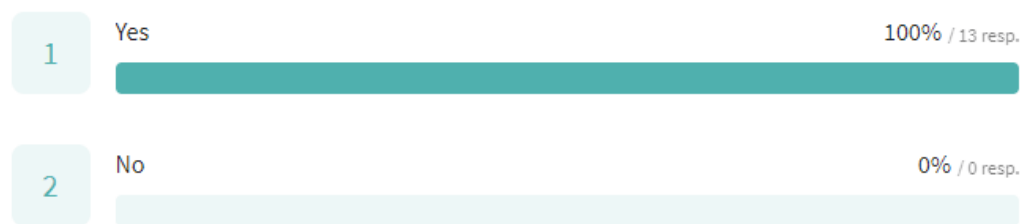
Research Techniques

Survey

Below I have asked questions to my potential users about the current system and what they would like to see in a new system:

Would you be interested in this system if you are planning to use the Tube?

13 out of 13 answered

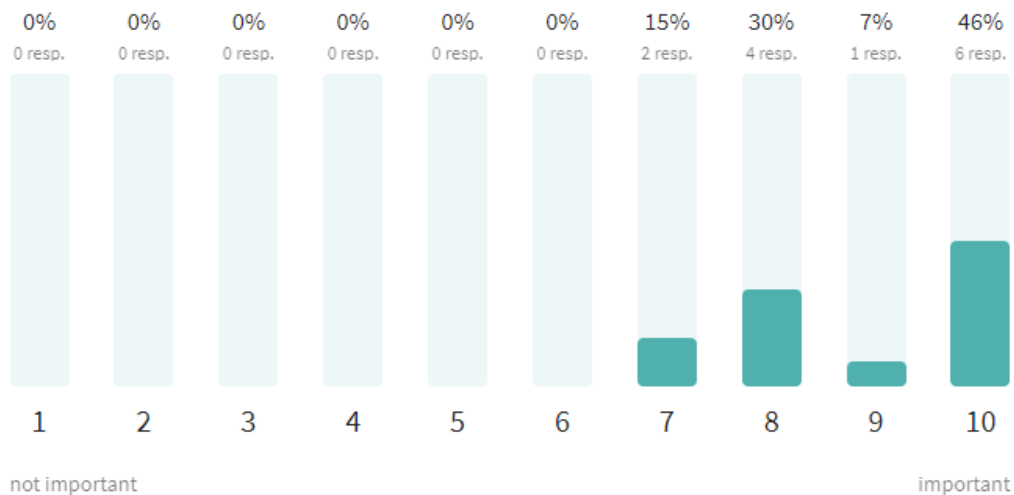


I have asked this question to check there would be a userbase for my system. From the response, I can clearly see that people are interested in this system to help them navigate the Tube.

How important is it for you to know the quickest route when using the Tube?

13 out of 13 answered

8.8 Average rating



Possible feature – finds the quickest route between stations

This question asks the importance of the main feature of my system.

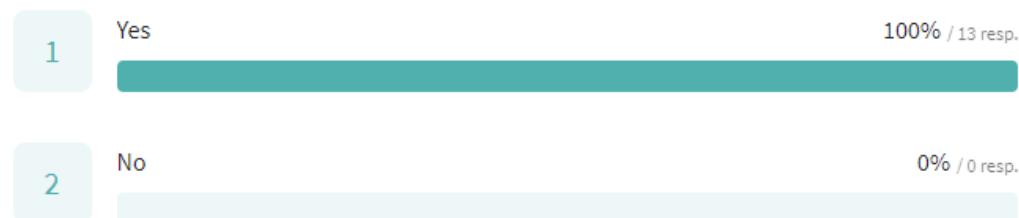
With an average of 8.8 I can see that it is very important that people know the quickest route on the Tube. It shows that people may be in a rush when using the Tube, need to reach a destination quickly or would rather not be on the busy Underground longer than needed. This also shows that there is a userbase for my system.

Implementation

To implement this, I will use a pathfinding algorithm as well as a database storing all stations and station connections.

Would you like to have the option of selecting a route with minimum train changes?

13 out of 13 answered

**Possible feature – route with minimum changes**

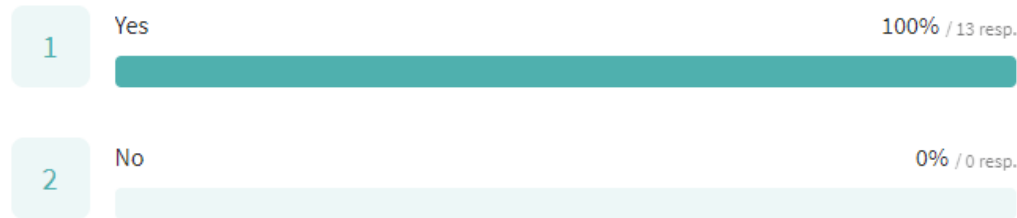
This result shows that everyone would like the option of selecting a route with minimum changes, due to this making journeys easier to navigate for the user, with less walking and train changes required.

Implementation

I will be able to implement this as a checkbox that the user can choose to select, if they want to know the route with minimum train changes. I will also have to alter my path finding algorithm to prioritise routes that remain on the same lines and have few changes.

Would you like to know when the next train is due?

13 out of 13 answered



Possible feature – train arrival times

All users answered saying they would like to see when the next train is due.

I also think this would be a useful feature as it lets the user know if they should rush to catch an earlier train, or just take their time knowing that their train is not arriving right away. This allows people to spend less time waiting on platforms, as well as preventing people from running for trains they can't catch.

Implementation

I will be able to add this feature by calling the TfL API which can return details of train arrivals for specific stations. I will do this by sending a URL request to TfL's REST API containing the station's unique ID. From this request a JSON response will be returned, including details of arriving train times for that station.

I will display the next few train arrival times, rather than just one, so the user has a choice of trains to catch if they miss the first one.

I will also add a checkbox which allows the user to choose if they want to view live train times, as they may not need to travel right away, and calling the API will take relatively longer than just finding the path using the local database.

Would you like to see any other features?

11 out of 13 answered

"Amount of changes"

Possible feature – showing the number of train changes

This is a feature I plan to include, as I want to display to the user the most important information about their route. I think this is important information as the user must be ready to change trains so that they don't miss their stop.

Implementation

To show a train change, I will tell the user the change of line, as well as telling the user the station at which the change occurs.

"the cost of travel/compare cost of travel"

Possible feature – showing the cost of the journey

Here someone has indicated the importance of a feature that displays the cost of a journey, as the user will likely want to know how much they are paying.

Implementation

To do this I will need to create a database for the stations in each zone, and the prices to travel between different zones. I will then find the range of zones on the route, using the stations on the route, and my database. Using another table, I will find the price information for the different zones travelled through.

Possible feature – comparing cost of travel

Although I think it would be useful to show the price of the journey, I don't think there is any need to compare journey costs or find the route with the lowest price. This is because most journeys are a fixed price whatever route is taken, as the user is only traveling on TfL's Underground and has no other options that could be cheaper while still taking a route on the Underground.

"graphical way of seeing the route you are taking"

Possible feature – map of route

Some people may like the idea of a route displayed graphically if they are interested in the exact route the train is taking around London. However, this is not a mandatory feature for helping the average user get from A to B, as the user is not driving the train and instead only needs to know when to get on and off. Therefore, I think a map would just add unnecessary information to the system, when my main goal is to make the system as simple and easy to use as possible.

Is there something you would not like to see in this system?

10 out of 13 answered

"i don't want it to be complicated - just a simple gui"

"overcomplications, keep it simplistic"

"Clutters of information, keep the UI clean"

"overcomplication and making it confusing to see your route you are taking"

Possible feature – simple UI

When asked what they would not like in the system, many said they don't want the UI to be overcomplicated, and instead want a simplistic system that is easy to use.

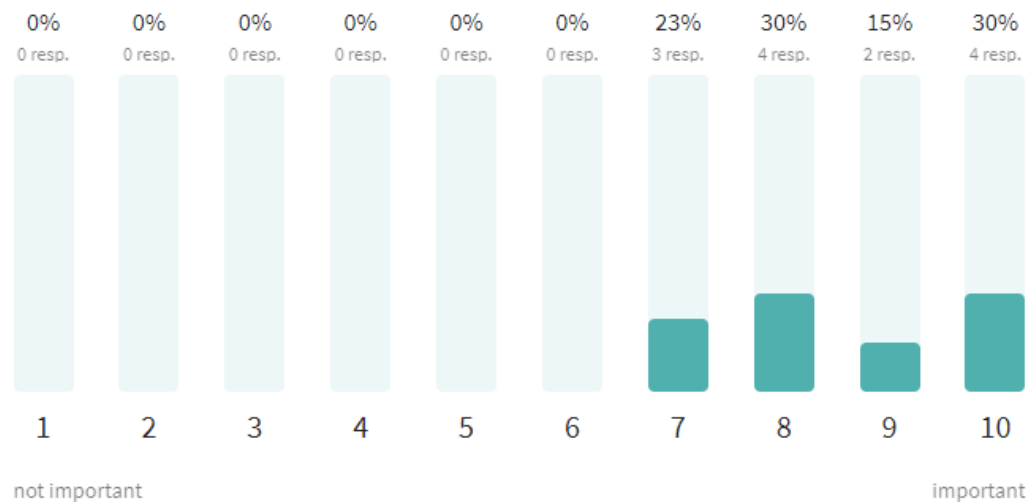
Implementation

To do this I will keep my UI simple, with only a few GUI elements so that the important fields are obvious, and so the user can receive their desired route as quickly and easily as possible. To ensure I do this I will not have a login system for users, as this is unnecessary and adds nothing to the main purpose of the system. Instead, the first thing the user should see when opening the application is an option to enter two stations, for a route between them to be returned.

How important is ease of use and simplicity of the system?

13 out of 13 answered

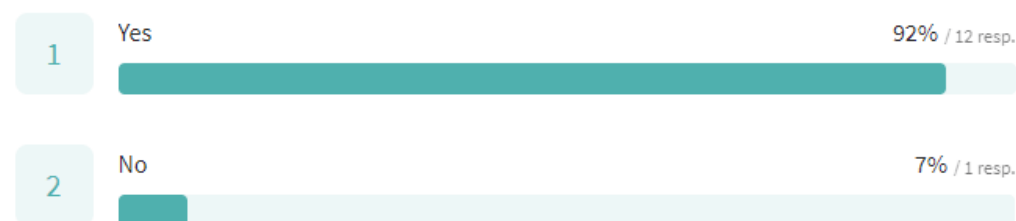
8.5 Average rating



The result of this question shows again the importance of a simplistic system that is easy to use. This means my UI should not be cluttered with information, and it should be clear to the user what they need to do to find their route.

Would you like to be re-directed if lines are under maintenance?

13 out of 13 answered



Possible feature – route redirection

Most people thought it was a good idea for routes to be redirected if lines are under maintenance. I think this would be a useful feature of the system because users will always be directed on the best route despite any interruptions on the Underground.

Implementation

To do this I plan to have admin accounts, allowing staff to login and update line statuses in order to redirect traffic onto other lines. My path finding algorithm will then avoid any closed lines when finding the fastest route.

Do you know of any similar systems?

If so, what do they do well, or need to improve on?

13 out of 13 answered

"thetrainline.com"

"Trainline"

"trainline. It doesn't always show the best route"

"thetrainline.com is ok but not really for the underground.
<https://raildar.co.uk/radar.html> to see where trains are on a map."

A few different people mentioned Trainline as a similar system. I will look more closely at this system, and at its pros and cons, as it is similar to my planned system, but for the overground network instead of the London Underground.

"Google Maps, tell the user when they have to leave, when they will arrive, which platform they have to be on. Through doing something that replicates this easy to use system, this interface will be able to tell the user everything he needs to know about his journey. The only thing Google Maps does not do is tell the user the price of the journey. Therefore, through your system telling the user which route is the cheapest, and fastest, it will be a very good system."

Possible feature – showing the cost of the journey

Here a user has pointed out that Google Maps does not tell you the price of a journey and thinks this would be a useful feature. I would therefore like to add this feature as it would be very helpful to know the price of your journey.

I have talked about the implementation of this above when someone mentioned it as a feature they would like to see added.

Survey Features - Conclusion

Features to include <ul style="list-style-type: none">• Route with minimum changes• Display live train arrival times• Display number of train changes• Showing the price of the journey• Admins can close lines for the route to be redirected• Checkbox for the user to choose if they would like to view live train times for their route	Things not to include <ul style="list-style-type: none">• Comparing journey prices• A cluttered and overcomplicated UI• Map of route
---	---

Similar systems currently available

TfL journey planner <https://tfl.gov.uk/plan-a-journey/>

Plan a journey

From
To

Leaving Arriving

Today 16:15

Hide preferences <

Plan my Journey

My Journeys Recents

London Bridge to Liverpool Street >

Liverpool Street to London Bridge >

Liverpool Street to New Cross >

Victoria to Cockfosters Underground Station >

Add favourites Turn off / clear

Public transport Cycling Walking

Travel by [select all](#) | [deselect all](#)

<input checked="" type="checkbox"/>	Bus	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	National Rail	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	London Overground	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	River Bus	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Emirates Air Line	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Tube	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	DLR	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	TfL Rail	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Tram	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Coach	<input checked="" type="checkbox"/>

Show me

The fastest routes

Routes with fewest changes

Routes with least walking

Access options

No accessibility requirement

Use escalators, not stairs

Use stairs, not escalators

Step-free to platform only

Full step-free access

Preferences

I only want to walk for a maximum of

40 mins

My walking speed is

Average

Travel via

Enter a location to travel via

Outside London

Search outside London

Optimise for walking

I'd rather walk if it makes my Journey quicker

Save these preferences for future visits

The TfL journey planner takes the input of your start and end Underground station, and then finds the quickest route between them. This requires a path finding algorithm, which would most likely be some form of Dijkstra's algorithm. To perform this algorithm the information of each node (i.e. station) and edge (i.e. travel time for each connection) must be stored. This data is likely stored in a normalized database.

The TfL planner returns a route, along with each change and the time for each section.

The journey planner also returns the time until when the next train is due. This information can be found using the TfL open API which can return the arrival times of future trains, along with platform information and other data.



In my project I will include Dijkstra's algorithm as my path finding solution, similar to how TfL may do their route finding. I will also include references to the TfL API so I can also display when the next train is due, as I think it is a very important feature that people will find helpful.

One thing I would like to avoid after viewing TfL's journey planner is displaying too much information at once, so that the important information is obscured. I would like my software to be simplistic so that a user can easily find a journey without any prior knowledge of the system, even if they have little ICT skills.

Journey results





From: **London Bridge**
To: **Liverpool Street**
Leaving: **Sunday 17th May, 16:15**

Travel preferences & accessibility:
Showing the fastest routes Using Tube Max walk time 40 mins

[Edit journey](#) [Add favourites](#) [Edit preferences](#)

Fastest by public transport

16:11 - 16:20 9 mins
£2.40 anytime

-  Northern line to Moorgate
-  Special Service +
- 3 min [Hide stops](#)
- Bank Underground Station
-  Metropolitan line or Hammersmith & City line to Liverpool Street
-  Special Service +
- 1 min

[View details](#) [Map view](#)

Access, lifts and escalators +

<p>Good</p> <ul style="list-style-type: none"> Tells you when the next train is due Displays information on train changes 	<p>Bad</p> <ul style="list-style-type: none"> Displays a lot of information at once
<p>What I will include in my solution:</p> <ul style="list-style-type: none"> Simple UI as this make it easier to use, it is also what people wanted in my survey Getting the time of the next train using the TfL API I will use Dijkstra's algorithm to find a route between the stations, I will then display details of the journey such as the number of stops and changes. 	

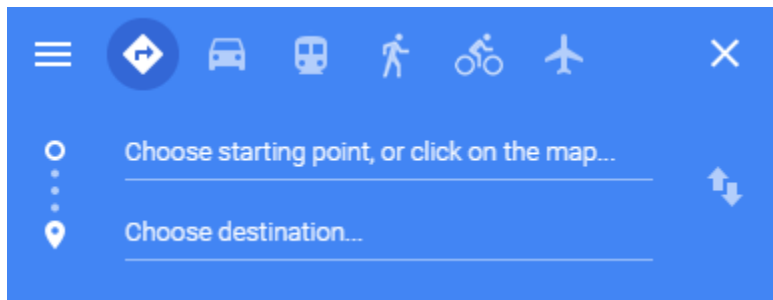
Trainline journey planner <http://thetrainline.com/>

Trainline's journey planner is used to plan routes for train journeys across Europe, taking inputs of starting and ending station, and then displaying a ticket price.

When typing in a location, Trainline auto fills the word making it quick and easy to search for destinations with long names. I like this feature as it makes place names that are longer or more difficult to spell quicker to find, saving users time. For Trainline to have this system a database must be stored with all names of stations. In my project I will need to store all names of London Underground Stations, which can be done in the same normalized database that my information for Dijkstra's algorithm is using.

<p>Good</p> <ul style="list-style-type: none"> Autocomplete makes searching for a station quick and easy 	<p>Bad</p> <ul style="list-style-type: none"> If there is not a direct route the system sometimes displays an error rather than finding a route with train changes
<p>What I will include in my solution:</p> <ul style="list-style-type: none"> I will include autocomplete for the fields where the user must type station names 	

Google Maps <https://www.google.co.uk/maps/dir/>



DELAYS

Light traffic in this area

- Road closure on Cranfield Rd

- Road closure on Mill Rd

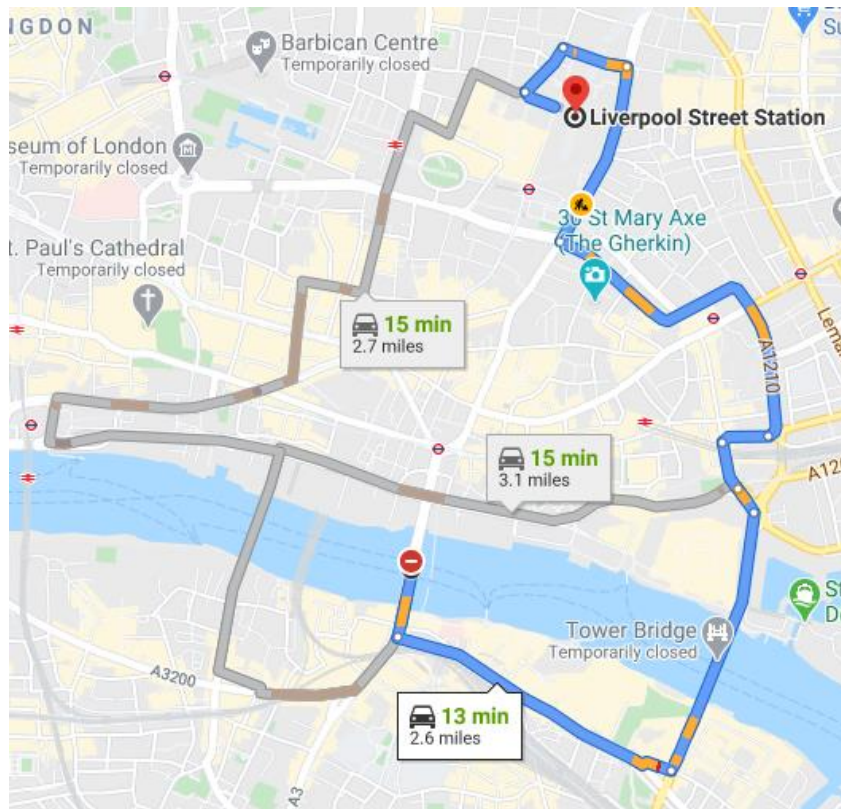
- Road closure on Mill Rd

- Road closure on B5120

Google Maps includes a path finding solution along with information about delays and closures to help keep traffic flowing well and giving people the quickest route in all circumstances.

I would like to include a system like this, which redirects passengers onto another route on the Underground if there are closures to a line. I will do this by allowing admins to sign in and update line statuses.

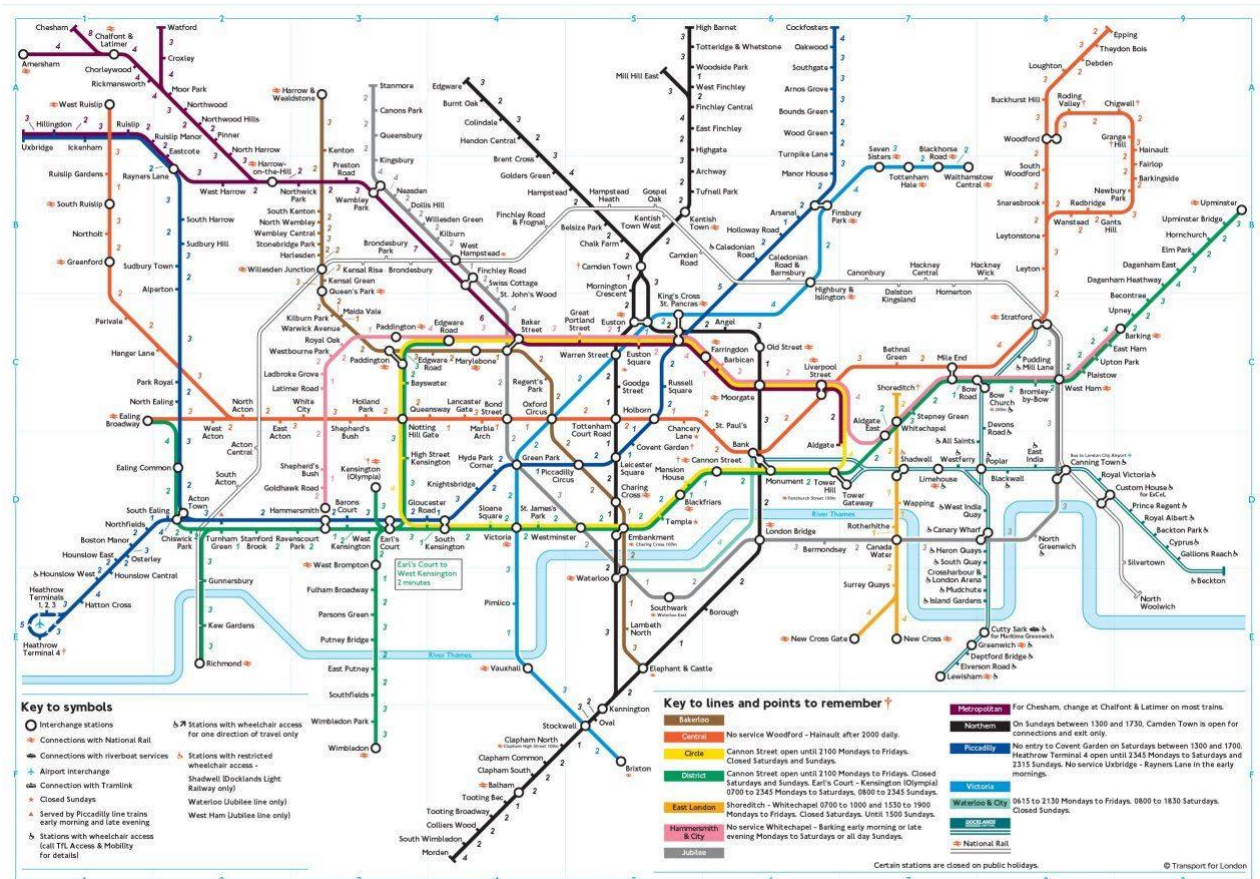
Google Maps also displays a visual route, making it easy for users to navigate their journey. However, I will not include a map like this for my system for the Tube, as the only important thing is that the user can catch the correct train, so the exact path can be abstracted from the user.



<p>Good</p> <ul style="list-style-type: none"> • Include information on road closures, so routes can be redirected • Simple UI 	<p>Bad</p> <ul style="list-style-type: none"> • Does not display journey prices
<p>What I will include in my solution:</p> <ul style="list-style-type: none"> • I will include the prices of journeys which will be displayed to the user once their route has been found • I will redirect routes if lines are closed. I will do this by allowing admins to set each line's status, for example, to open, or closed due to maintenance. I will need to make this a secure feature so that only admins can say that lines are closed. 	

Other relevant details

Tube map



Above is the Tube map that I will be basing my project on. I will store the station and line information in a database.

As the Tube map is very complex, I will develop my route-finding application in stages, starting with just a few train lines, in order to build up the complexity and test as I go.

Research on Dijkstra's Algorithm

Dijkstra's Algorithm finds the shortest path between two nodes. I will use a priority queue in my implementation.

Dijkstra's Algorithm

1. Mark all nodes as unvisited
2. Assign every node a time to be reached of infinity, apart from the starting node which should be assigned a time of 0.

3. Add the starting node to the priority queue.
4. Consider all unvisited neighbouring nodes of the first node in the queue. Update each neighbour's time to be reached, if the time to reach the station at the front on the queue + the connection time to this neighbour is less than the current time to reach that neighbour.
5. Insert all updated neighbours into the priority queue, based on their time to be reached.
6. Remove the node from the start of the queue marking it as visited.
7. Repeat steps 4 to 6 until the destination node reaches the front of the priority queue.
8. To find the shortest path trace backwards from the final node.

Overview of evidence gathered

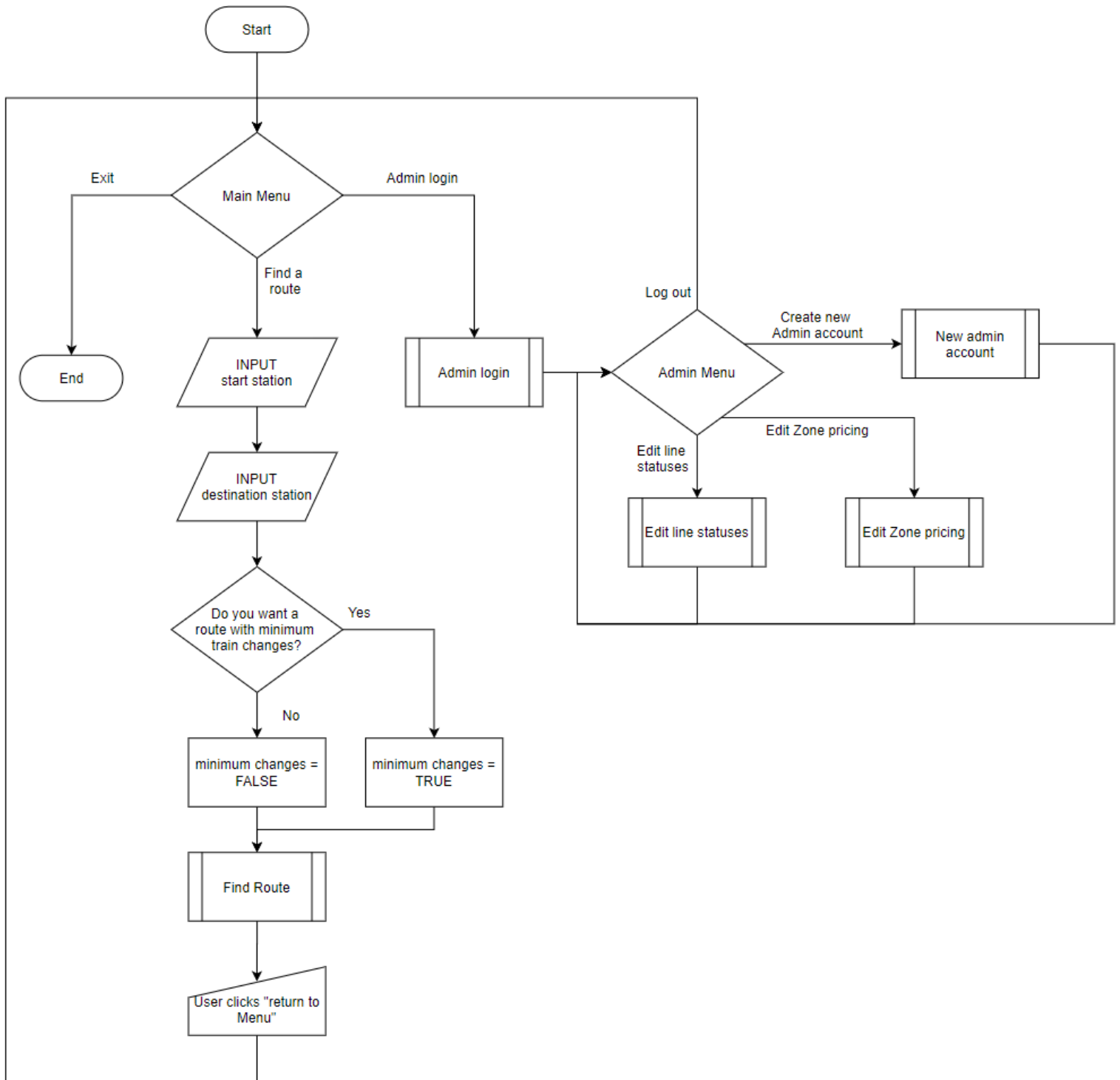
Firstly, I have discovered that many of my potential users would like a simple UI that is easy to use. I have also learnt from my survey that users would like to see the price of their journey, and how many train changes are needed. People would also like to have the option to select the route with minimum train changes.

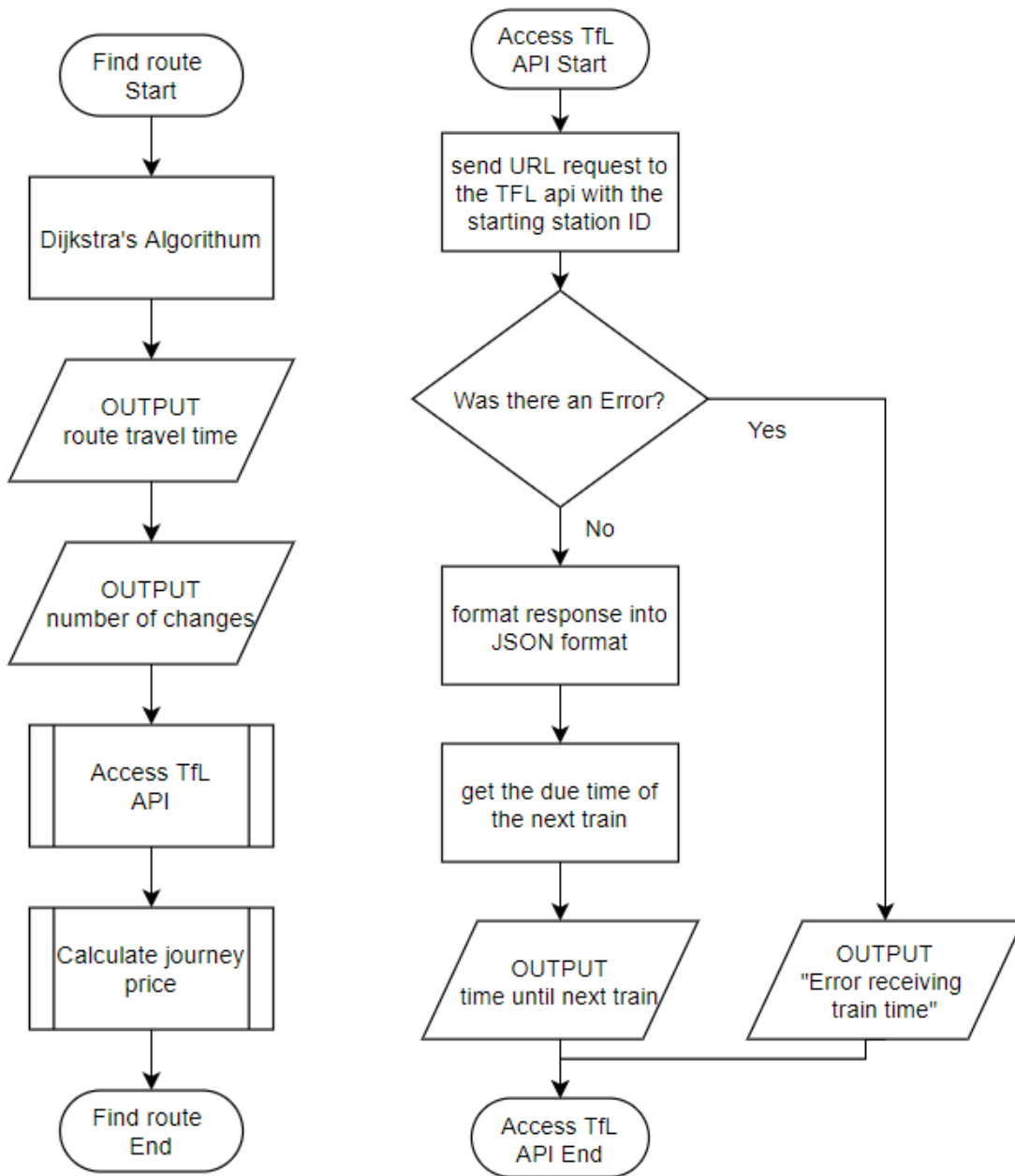
After looking at similar systems and seeing what features they include, I would like to access the TfL API to display the train times for the user's journey. This is also a feature that 100% of people voted for in my survey.

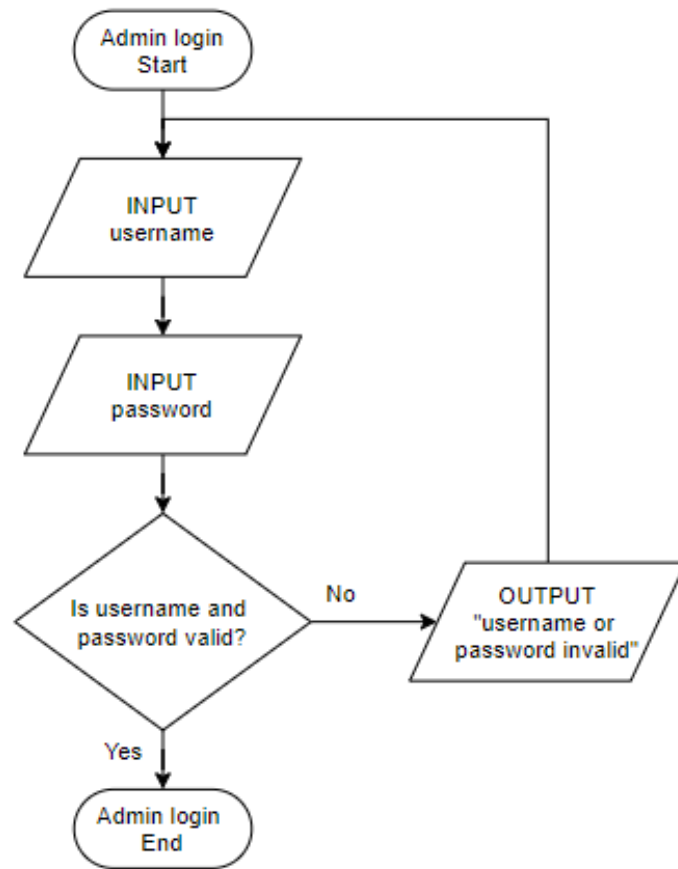
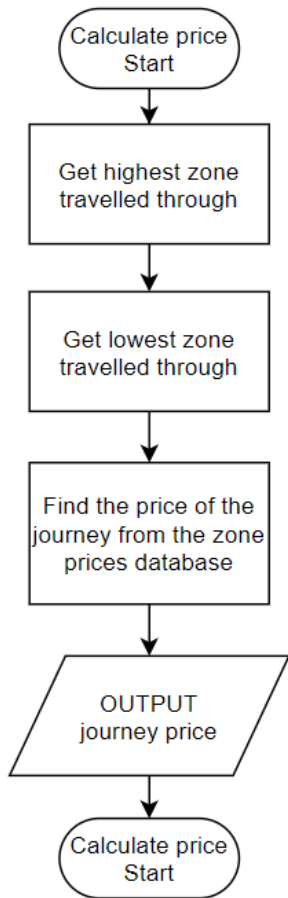
Another feature that was voted for, and is used in systems such as Google Maps, is redirecting journeys when lines are under maintenance or closed. To do this I will have admin accounts, allowing admins to login to my system and change the status of a line (e.g. from open to closed). This will also be a useful feature for my testing of the graph traversal algorithm.

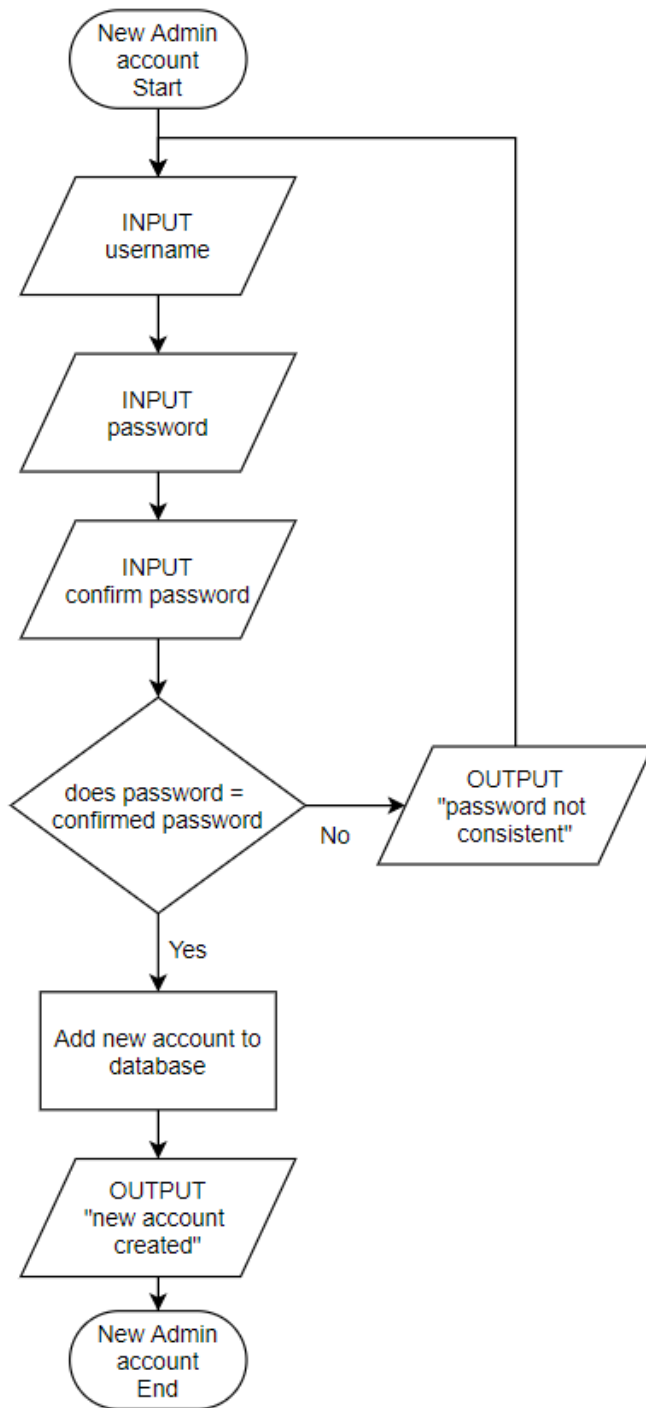
To get some of these features to work (such as route redirects, and finding a route with minimum train changes), I will need to alter Dijkstra's Algorithm to suit my needs.

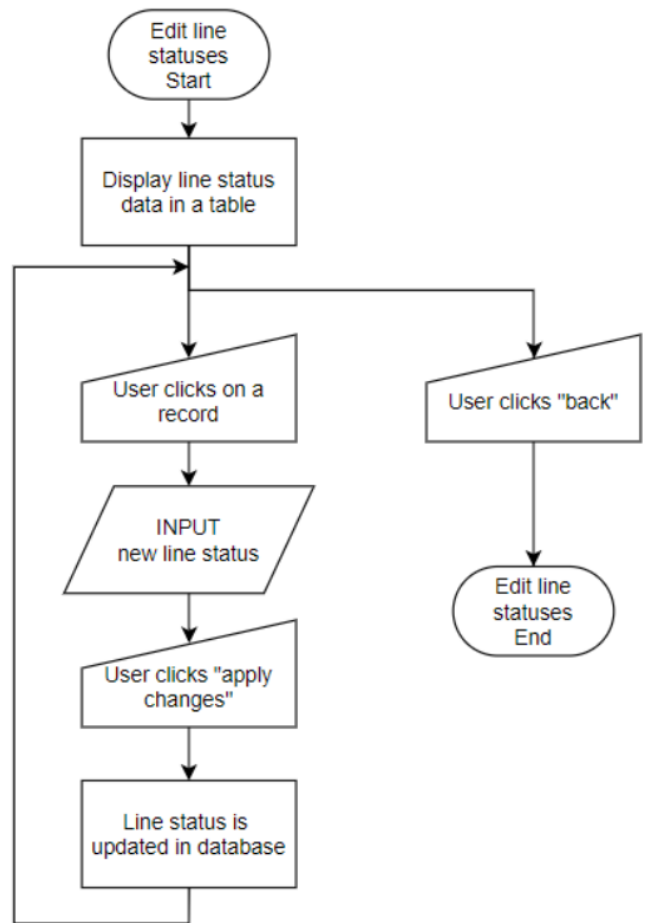
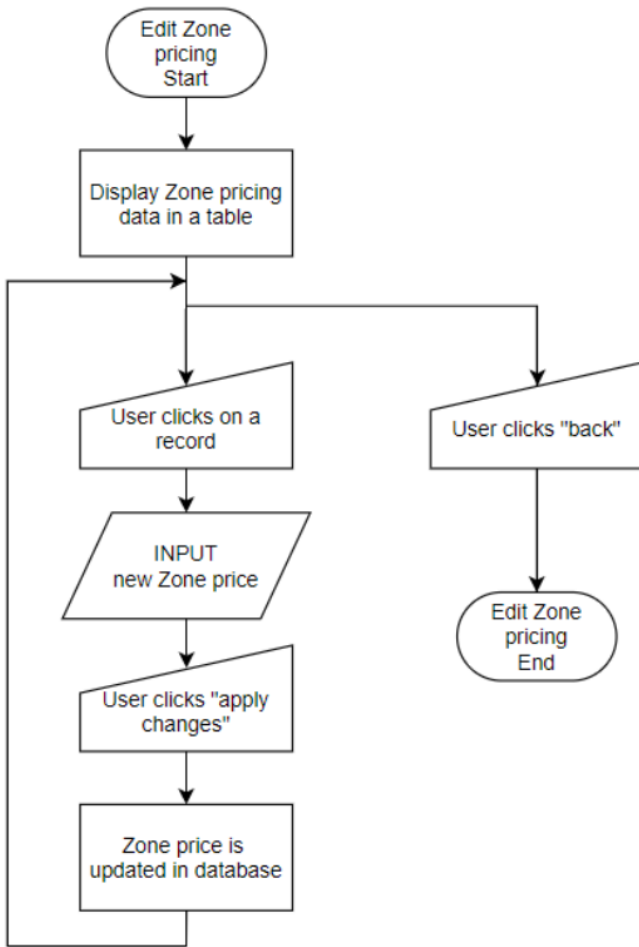
Flowchart



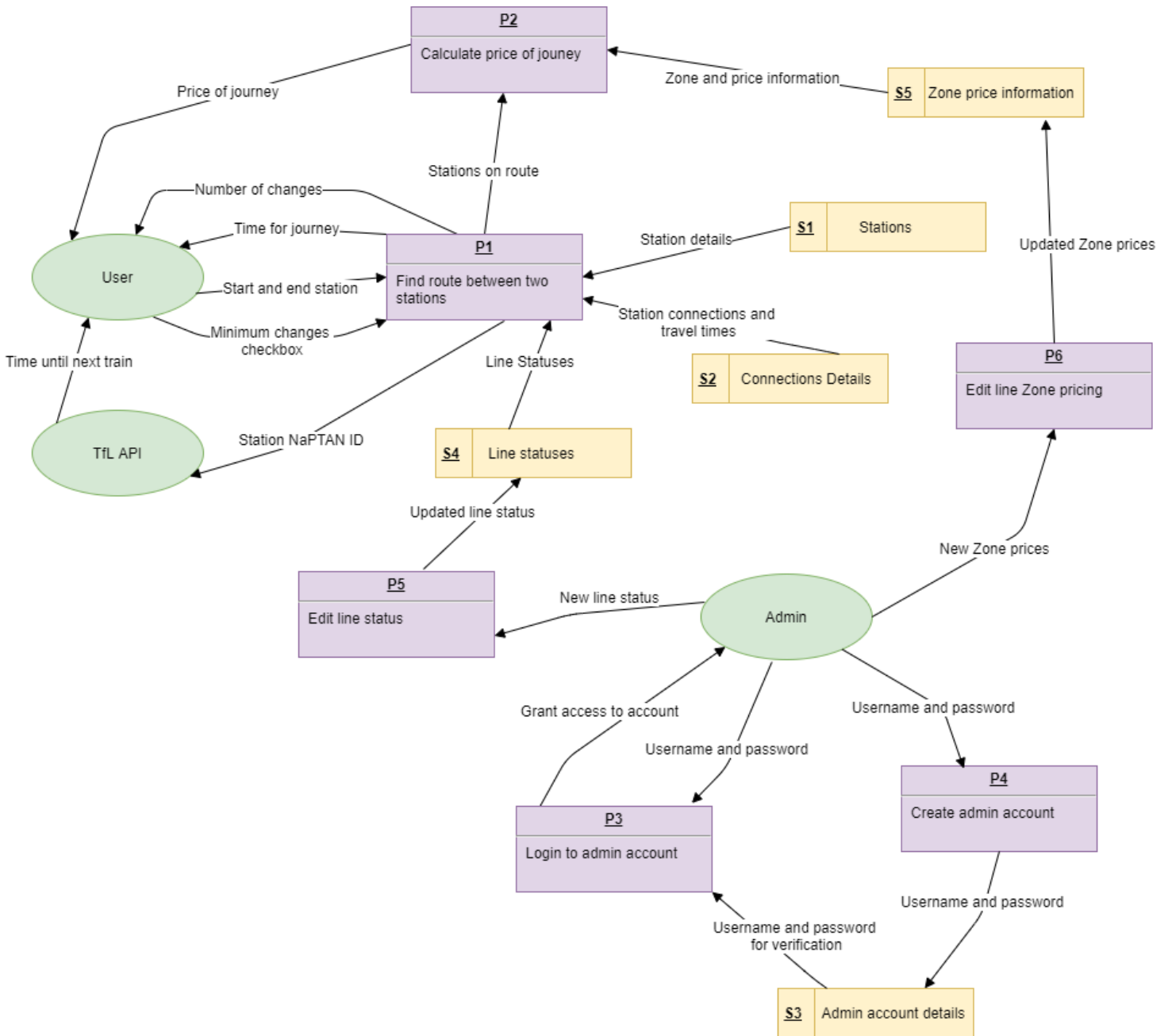








Data flow Diagram



Storage		Processes	
S1	Stations table – includes all information about each station. <u>Fields:</u> <ul style="list-style-type: none"> • Station ID • Station name • NaPTAN ID (ID used to access the TfL API) • Station Zone (1-9) 	P1	Finds a route between the two stations stated by the user, using Dijkstra's Algorithm.
S2	Connections table – includes details on all connections between stations. <u>Fields:</u> <ul style="list-style-type: none"> • Station IDs in the connection • Travel time of the connection • The train line the connection is on 	P2	Calculates the price of the journey using the price information for the zones travelled through.
S3	Admin account details – includes details of each admin account <u>Fields:</u> <ul style="list-style-type: none"> • Hashed password • Username 	P3	Allows admins to login to their admin account by entering their username and password. The username is compared with the username in the database, and the password is hashed and then compared with the hashed password in the database.
S4	Line status table – includes details on each train line, including the status of the line <u>Fields:</u> <ul style="list-style-type: none"> • Line ID • Line name • Line status ID 	P4	Creates a new admin account with the username and password the admin has entered and adds it to the account database.
S5	Zone pricing information table – includes each price for every possible zone combination (e.g. 1-3, 2-5, 7-7) <u>Fields:</u> <ul style="list-style-type: none"> • Zone ID 1 (any zone 1 to 9) • Zone ID 2 (any zone 1 to 9) • Price (for this zone combination) 	P5	Updates the line status with the new status entered by the admin.
		P6	Updates the zone price with the new price entered by the admin.

Data Sources and Destinations

Input	Process	Storage	Output
- Start and end Station - Minimum train changes checkbox	Find route between stations		- Time for journey - Number of changes
Stations on route that has been calculated	Calculate price of journey		Price of journey
Starting station	Access TfL API via a URL request		Time until next train leaves
Username and password	Validate username and password, then login to admin account		
Username and password	- Check password is a sensible length, then hash it - Create new admin account	Store username and hashed password in database	
New Zone prices	Edit Zone pricing	Update Zone pricing information in database	
New line status (e.g. non-operational)	Edit line status	Update line status database	

Data Volumes

The system should be able to have 100 users with the ability for this to increase over time. This number could increase to over 2,000 users in six months. This should not be a problem for data storage as users will not have accounts.

There should be the ability for at least 10 admin accounts. This number should be able to double after 2 years.

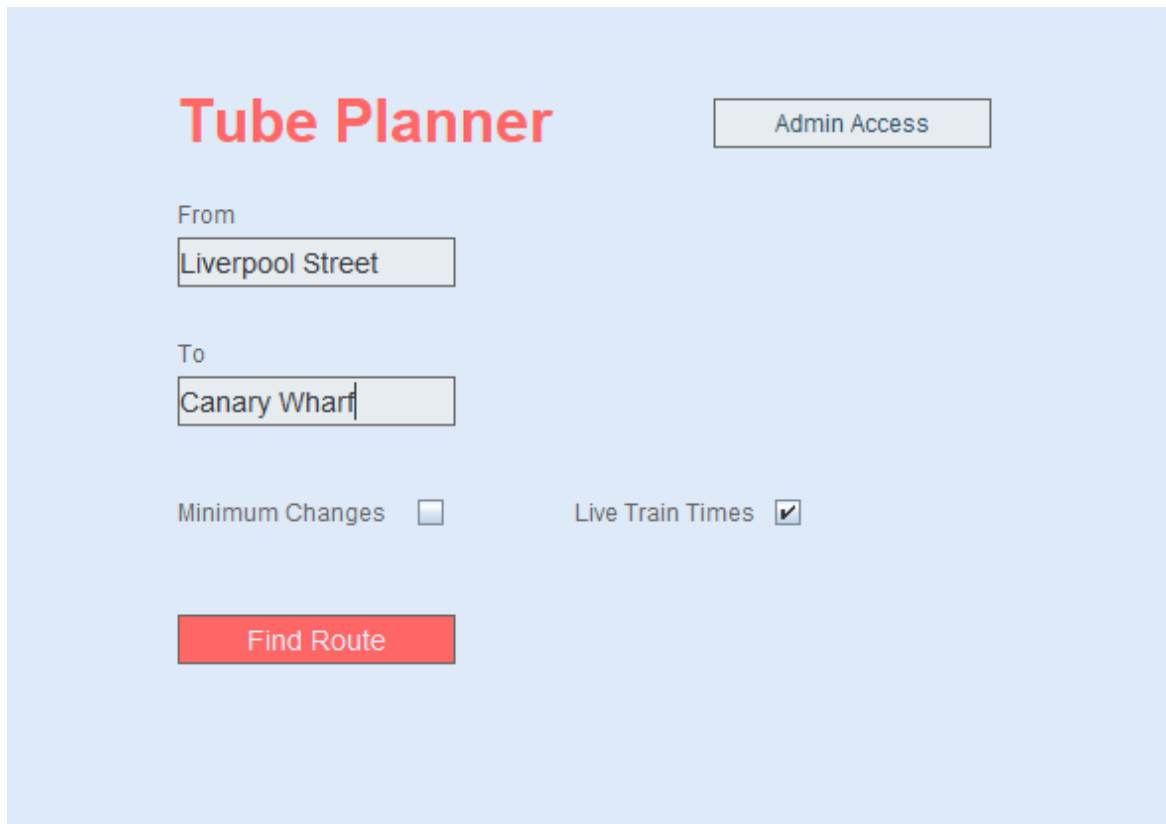
GUI ideas

Overview of GUI

For my GUI I have decided to use only a few colours to keep a simple look, with red against a light blue background to make important parts stand out. I have also used negative space to make sure my GUI doesn't look cluttered. I have chosen to do this because many people in my survey wanted to see a simplistic UI.

Main Menu Panel

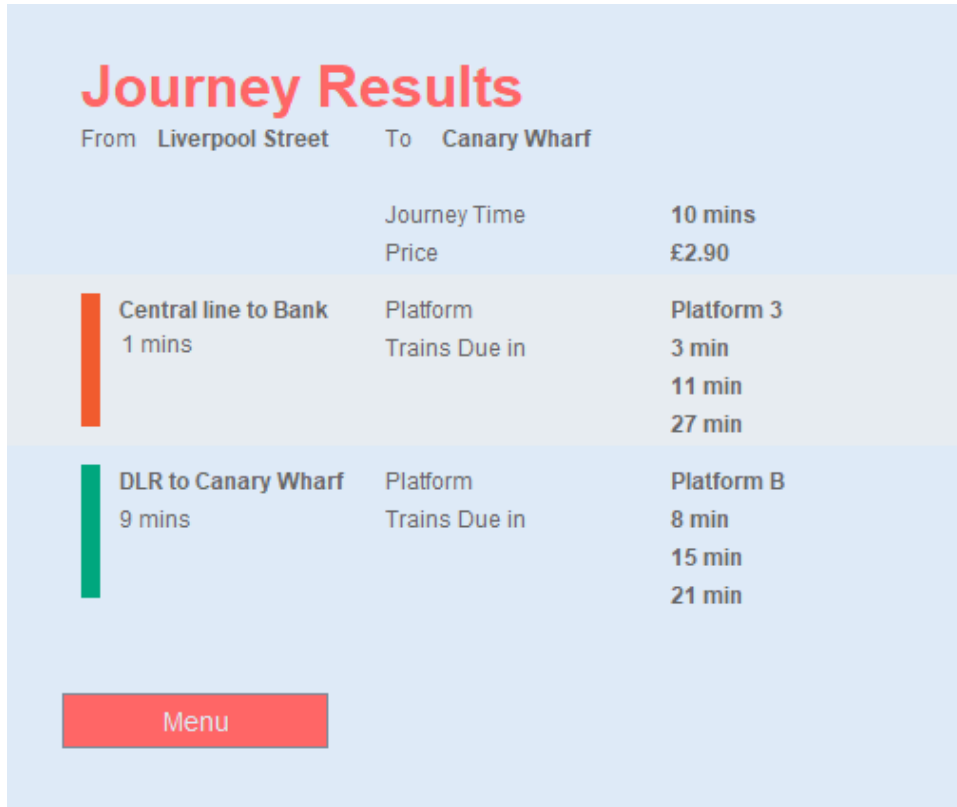
This is the first panel the user will see and where they can type in the stations that they want to find a route between. There is also a checkbox so the user can choose if they want a route with minimum train changes. I have also included a button for 'Admin Access', to allow admins to login and then carry out admin tasks such as changing zone prices.



The screenshot displays the 'Tube Planner' interface on a light blue background. At the top left, the title 'Tube Planner' is written in a large, bold, red font. To its right is a rectangular button with a thin border and the text 'Admin Access' in a light blue color. Below the title, there are two input fields: 'From' containing 'Liverpool Street' and 'To' containing 'Canary Wharf'. Underneath these fields are two checkboxes: 'Minimum Changes' which is unchecked, and 'Live Train Times' which is checked. At the bottom of the form is a prominent red button with the text 'Find Route' in white.

Journey Results Information

Here all information about the user's journey will be displayed, such as their journey time and when the next train leaves.



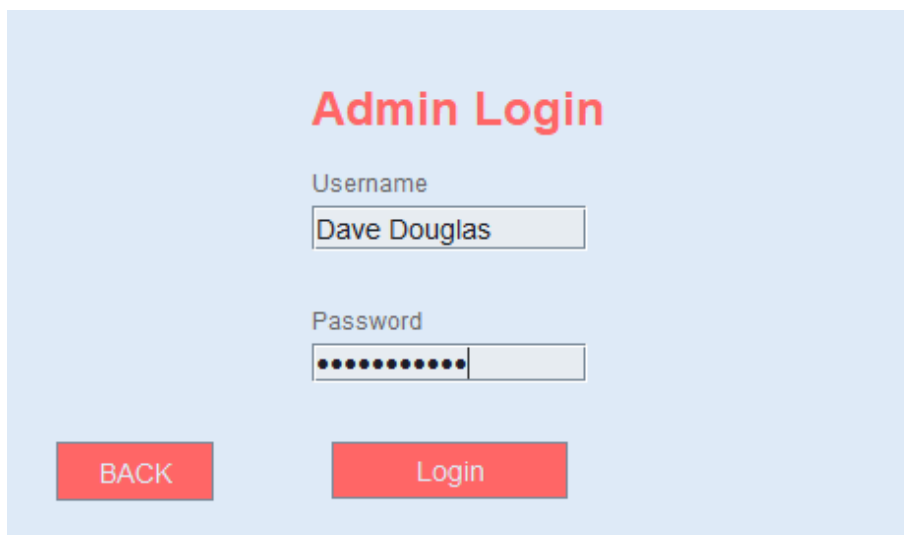
The screenshot shows a 'Journey Results' page with a light blue background. At the top, the title 'Journey Results' is in red. Below it, the route is 'From Liverpool Street To Canary Wharf'. The main content is a table with three rows. The first row shows 'Journey Time' as '10 mins' and 'Price' as '£2.90'. The second row, highlighted with an orange bar on the left, shows 'Central line to Bank' with a '1 mins' duration, 'Platform 3', and 'Trains Due in' times of '3 min', '11 min', and '27 min'. The third row, highlighted with a green bar on the left, shows 'DLR to Canary Wharf' with a '9 mins' duration, 'Platform B', and 'Trains Due in' times of '8 min', '15 min', and '21 min'. At the bottom left, there is a red button labeled 'Menu'.

Journey Results		
From Liverpool Street To Canary Wharf		
	Journey Time	10 mins
	Price	£2.90
Central line to Bank 1 mins	Platform	Platform 3
	Trains Due in	3 min 11 min 27 min
DLR to Canary Wharf 9 mins	Platform	Platform B
	Trains Due in	8 min 15 min 21 min

Menu

Admin – Login

Here admins can login to their account. There is no option to create an account to prevent just anyone making an account. Instead, other admins can create new admin accounts from their existing account.



The screenshot shows an 'Admin Login' page with a light blue background. The title 'Admin Login' is in red. Below the title, there are two input fields: 'Username' with the text 'Dave Douglas' and 'Password' with a masked password of 12 dots. At the bottom, there are two red buttons: 'BACK' and 'Login'.

Admin Login

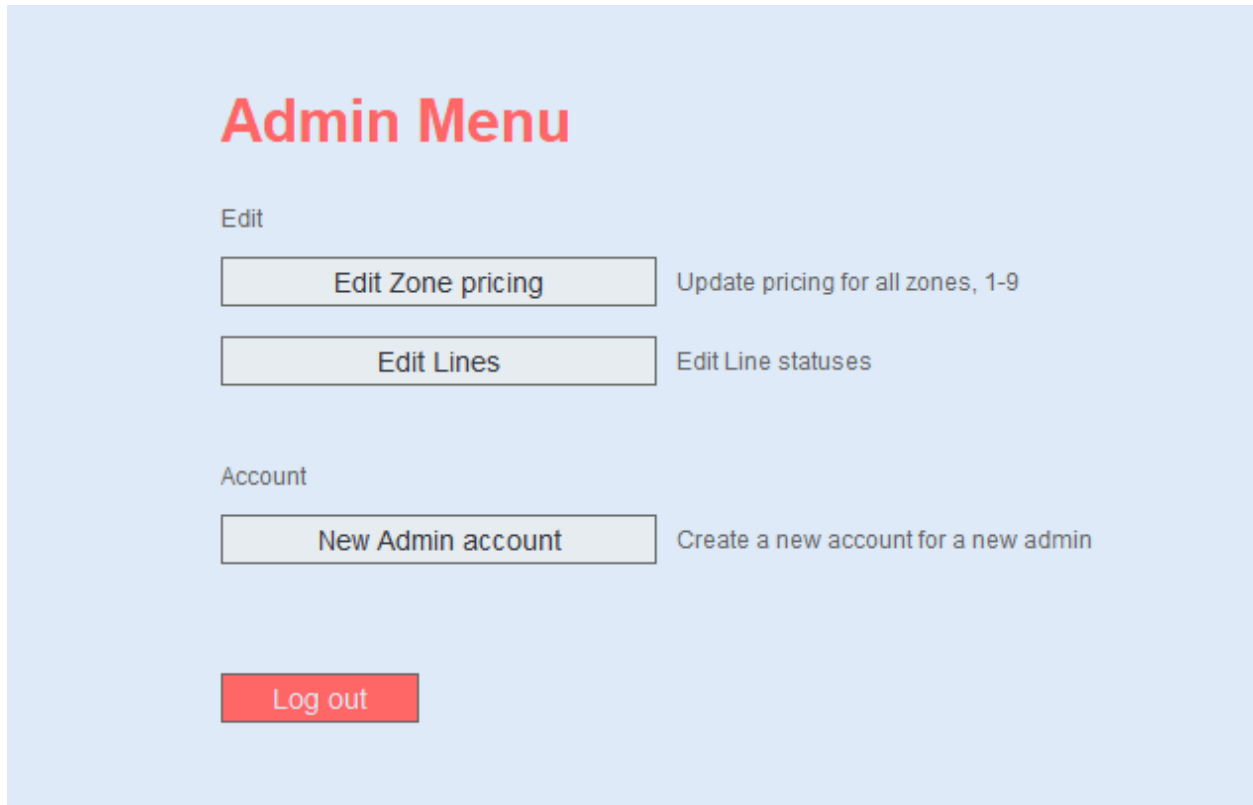
Username
Dave Douglas

Password
●●●●●●●●●●●●

BACK Login

Admin - Menu

This menu is displayed to admins when they login. It can be used to access different admin tasks such as editing zone pricing. It can also be used to add new admin accounts so that new admins can login to the admin menu.



Admin – Edit Zone pricing

Below is an example of a table admins can use to edit details in the system. This table allows admins to edit specific zone prices, while another table displaying line details will allow admins to edit line statuses. The apply button must be pressed to commit the changes to the local database.

Edit Zone pricing

	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5	Zone 6	Zone 7	Zone 8	Zone9
Zone 1	£2.40								
Zone 2	£2.90	£1.70							
Zone 3	£3.30	£2.90	£1.70						
Zone 4	£3.90	£3.30	£2.90	£1.70					
Zone 5	£4.70	£3.90	£3.30	£2.90	£1.70				
Zone 6	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70			
Zone 7	£5.60	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70		
Zone 8	£6.90	£5.60	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70	
Zone 9	£7.00	£6.90	£5.60	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70

[BACK](#)[Apply Changes](#)

Complex Algorithms

- Graph/Tree Traversal – I will use Dijkstra's algorithm as my path finding solution in order to find routes between stations, with the shortest travel time
- Queue operations – I will use a priority queue in my implementation of Dijkstra's algorithm to hold the next nodes to be checked, in order of their time to be reached
- Calling parameterised Web service APIs and parsing JSON/XML to service a complex client-server mode – I will access the TfL API so that I can display train arrival times
- Hashing – I will use a SHA256 hash to securely store admin passwords in my database
- Complex data model in database (e.g. several interlinked tables) – I will include several interlinked tables that store data about stations, station connections, pricing and lines. These tables will be used by my path finding algorithm, as well as to display the journey price and other information.
- Cross-table SQL – I will use cross-table SQL to get information from my tables about zone pricing
- Parameterised SQL – I will use prepared parameterised SQL statements when inserting admin accounts into my database. I will also use them when checking whether an admin account exists for a username and password entered.
- Aggregate SQL functions – I will use the MAX function when assigning a unique ID to a new admin account
- Object-orientated programming model – I will create node and edge objects, containing information about the Tube map, which will be used by my path finding solution
- Merge Sort – Used to sort stations into alphabetical order when the program starts, so binary search can be used on them
- Recursion - Used in my merge sort and in my implementation of Dijkstra's algorithm
- Regular expression - I will use a regular expression to check the format of zone prices entered by admins.
- Binary Search - I will use a binary search to search for a station name entered by the user, in an array of stations.
- Reading from files – I will read from CSV files to import data into my database. I will also be reading an image file to be used as the application's icon

Any other relevant details

Connecting to TfL API

URL request example: <https://api.tfl.gov.uk/StopPoint/940GZZLUCWR/arrivals>

This returns a JSON response with information about all trains arriving soon for the station specified. The station is specified by its NaPTAN ID in the URL, for example, Canada Water = 940GZZLUCWR.

Objectives

User Input

1. The user can input a starting station and destination station.
2. There is autocomplete when the user types in the station names.
3. A binary search is used to search an array of station names for a station name entered by the user.
4. The user can choose whether they would like the quickest route or the route with minimum train changes, via a checkbox.
5. The user can choose whether they would like to receive live train times, via a checkbox.
6. An error is displayed if an invalid station is entered.
7. An error is displayed if the journey is not possible as a line is closed.
8. An error is displayed if the start and end station entered are the same station.

Pathfinding

9. Dijkstra's algorithm is used to find the shortest travel time between stations and the route of that journey.
10. Dijkstra's algorithm uses a priority queue to hold indexes of nodes Objects, ordered by their time to be reached.
11. My implementation of Dijkstra's algorithm should use recursion.
12. An adjacency list is used to store indexes of all edge objects connected to each node object
13. My pathfinding algorithm redirects routes if a line's status is set to 'Closed'.
14. If the 'Minimum Changes' checkbox has been selected, my pathfinding algorithm redirects routes if there is an alternative route with less train changes, even if the journey time is made longer.
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.

Output to user

16. The system returns the cost of the journey.
17. The system returns the total estimated travel time for the journey.
18. The system returns the estimated travel time for each section between train changes in the journey.
19. The system returns the names of stations where each line change occurs.
20. The system shows the names and colours of the lines being travelled on in the journey.
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.
22. Using the TfL API, the system returns the platforms that the user's trains leave from.
23. An error message is displayed to the user if the API can't be accessed.
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.
25. The route details are returned in less than 200ms, when the API is not accessed for live train times.

Admin Login

26. An Admin can login to the admin menu if they enter a valid username and password.
27. An error is displayed if the username and/or password entered is invalid.
28. The username and password entered are checked against the records in the database, using a parameterised prepared SQL statement.

Admin Menu

29. Admins are displayed a menu from which they have three different options: edit zone pricing, edit lines, and create new admin accounts.

New admin account

30. Admins can create new admin accounts for fellow admins.
31. A username entered for a new admin account must be unique.
32. The username must be ≤ 20 characters.
33. The username must not be blank
34. The password must be between 8 and 20 characters (inclusive)
35. Password and Confirming Password must be the same when creating a new admin account
36. If any of the conditions for the username and password are not met, an error message is displayed.
37. The new admin account is inserted into the database using a parameterised prepared SQL statement.
38. Admins passwords are hashed using a SHA256 hash.
39. Use the aggregate SQL 'MAX' function when assigning a unique ID to a new admin account.

Edit Zone pricing

40. Admins can update the pricing for the different Underground zones by editing fields in an interactable table.
41. A cross-table SQL statement is used to fetch all zone pricing information from the database.
42. Admins are only able to enter prices with a valid price format, checked by a regular expression.
43. Admins have an 'Apply Changes' button to commit their changes to the local database.

Edit Lines

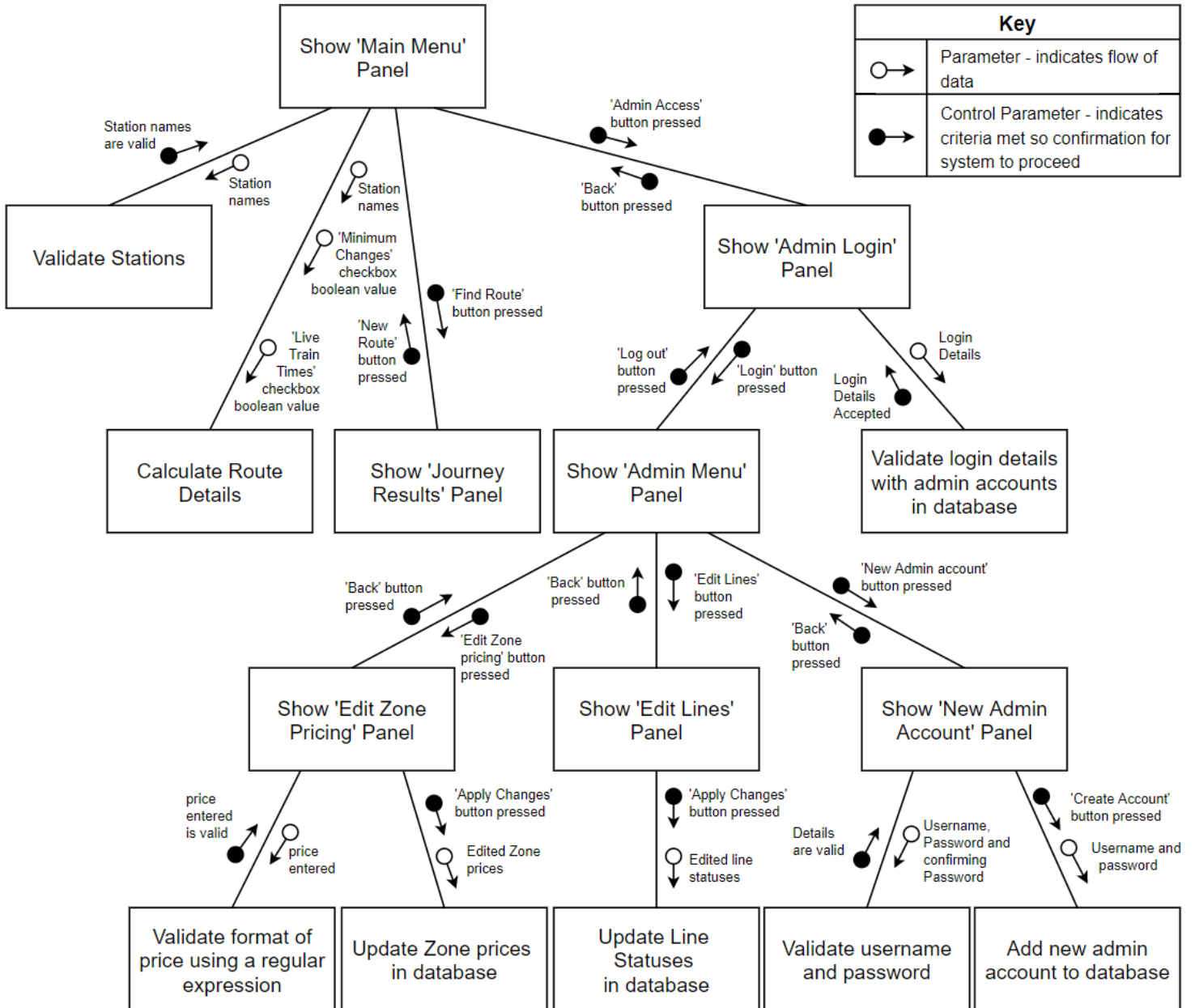
44. Admins can set each line's status to open or closed using drop down lists.
45. The changes to line statuses are updated in the local database when the admin presses the 'Apply Changes' button.

Other

46. Send a URL request to the TfL API, and parse to JSON.
47. Use a normalised database for all the details of the system.
48. Merge sort stations in alphabetical order based on their UNICODE values, when the program begins.
49. My merge sort should use recursion.
50. I will read from CSV files in order to enter all station and connection information into my database from these CSV datasets.
51. I will read an image file to set it as the icon image of my application.

Design

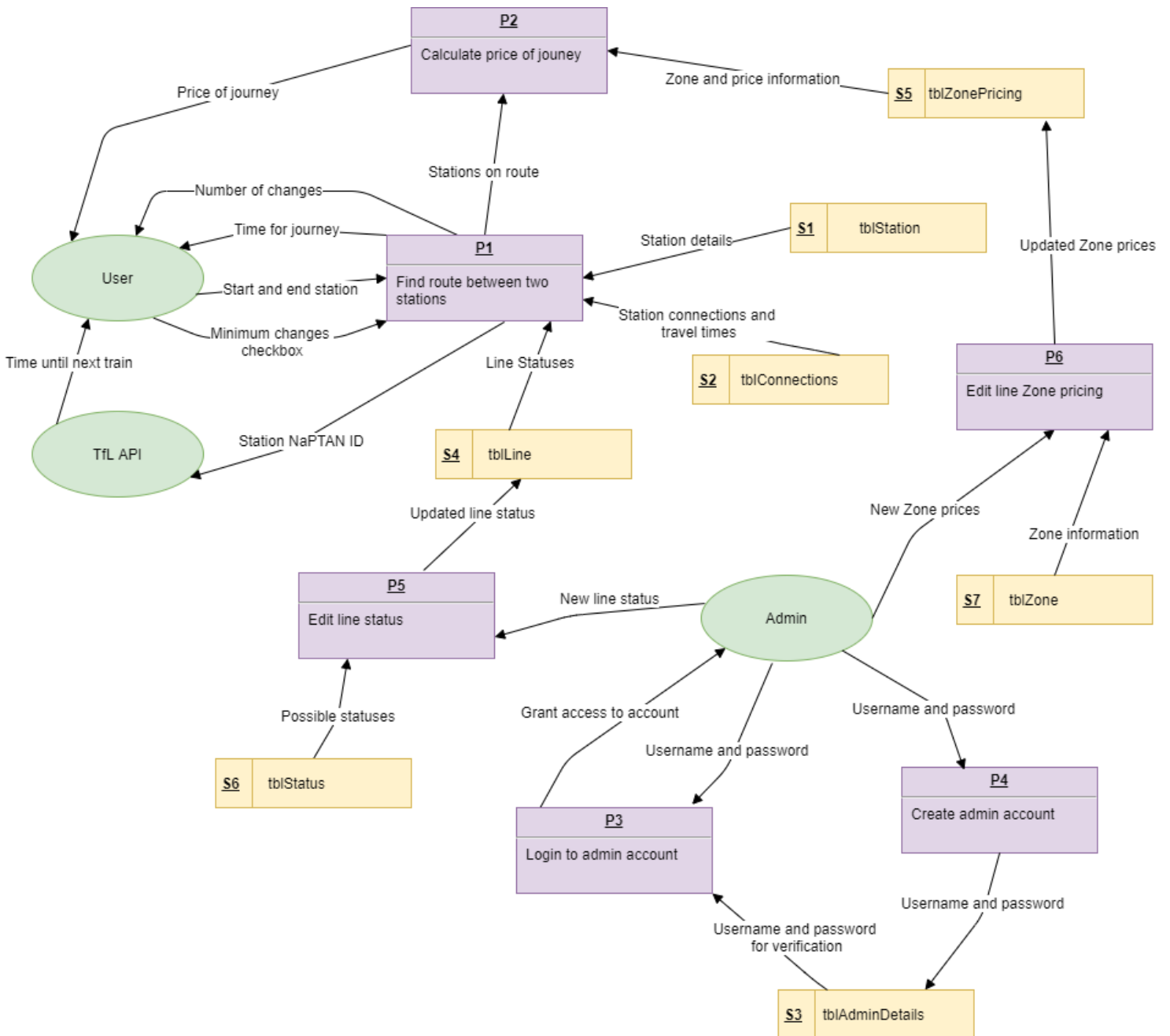
Structure Chart



DFD of proposed system

My DFD here is very similar to my DFD in my analysis section. I have made a few alterations and have included the new diagram below. The changes I have made are:

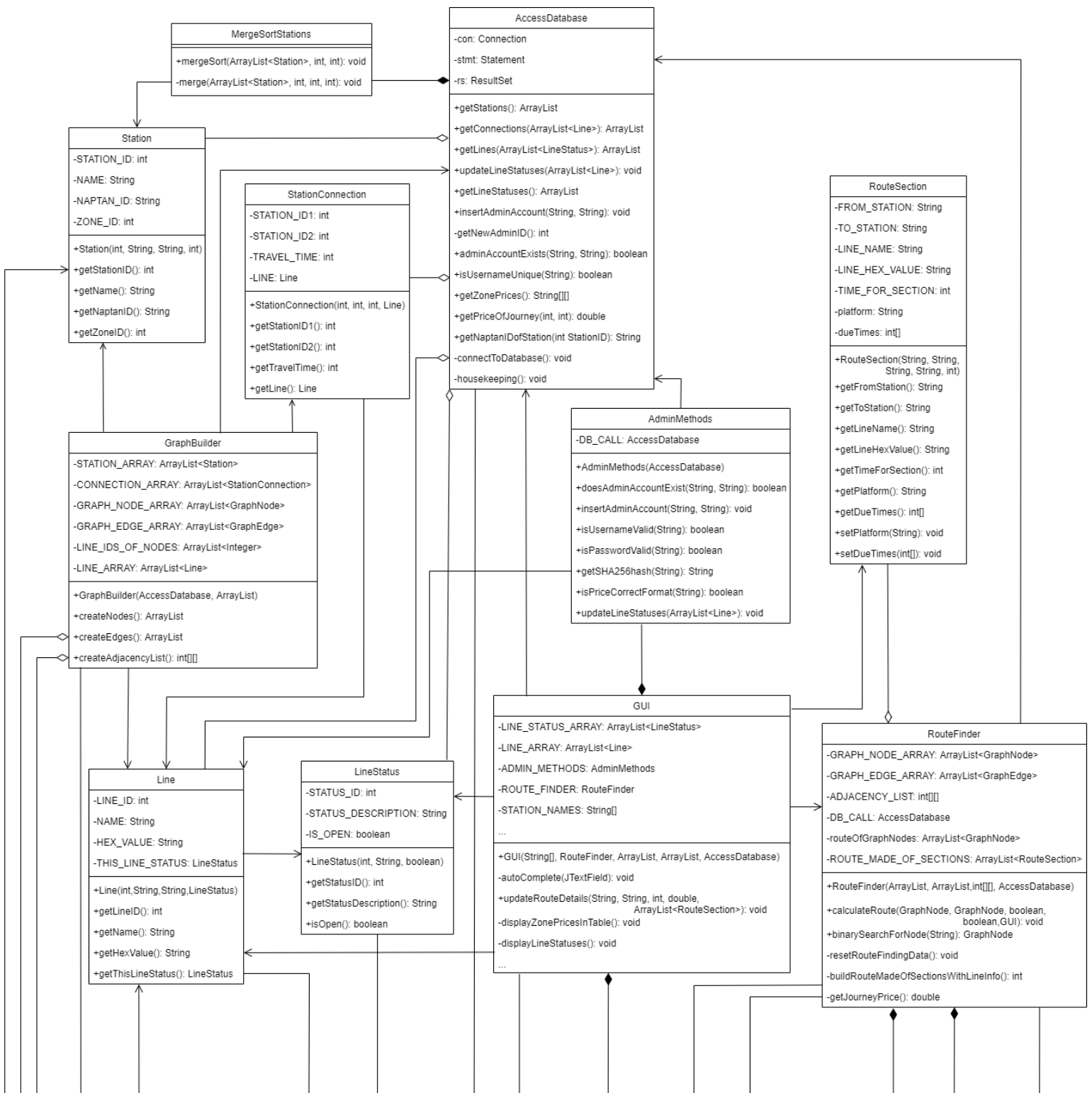
- Updated the names of all storages to show the table names in my database.
- Added the tblStatus storage, S6.
- Added the tblZone storage, S7.

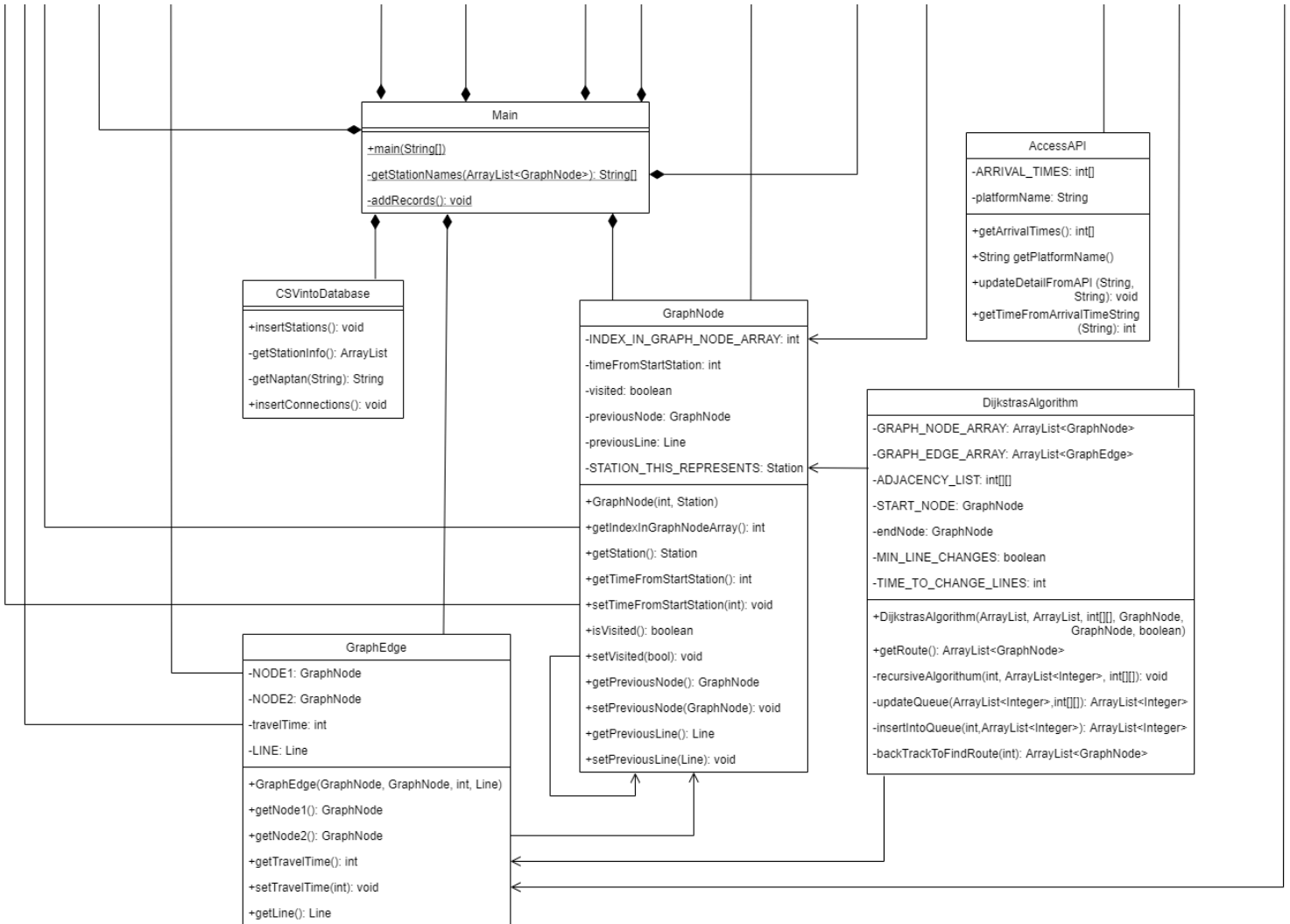


This table includes information on the two new storages. The other storages and processes are explained under my DFD in the design section.

Storage	
S6	<p>tblStatus – includes records to represent different statuses that lines can have. Status ID is referenced as a foreign key in tblLine.</p> <p><u>Fields:</u></p> <ul style="list-style-type: none">• Status ID• Status Description• isOpen (boolean value representing whether a status allows a line to be travelled on)
S7	<p>tblZone – stores a zone name for each zone ID. Zone ID is referenced as a foreign key in tblZonePricing.</p> <p><u>Fields:</u></p> <ul style="list-style-type: none">• Zone ID• Zone Name

Class diagram



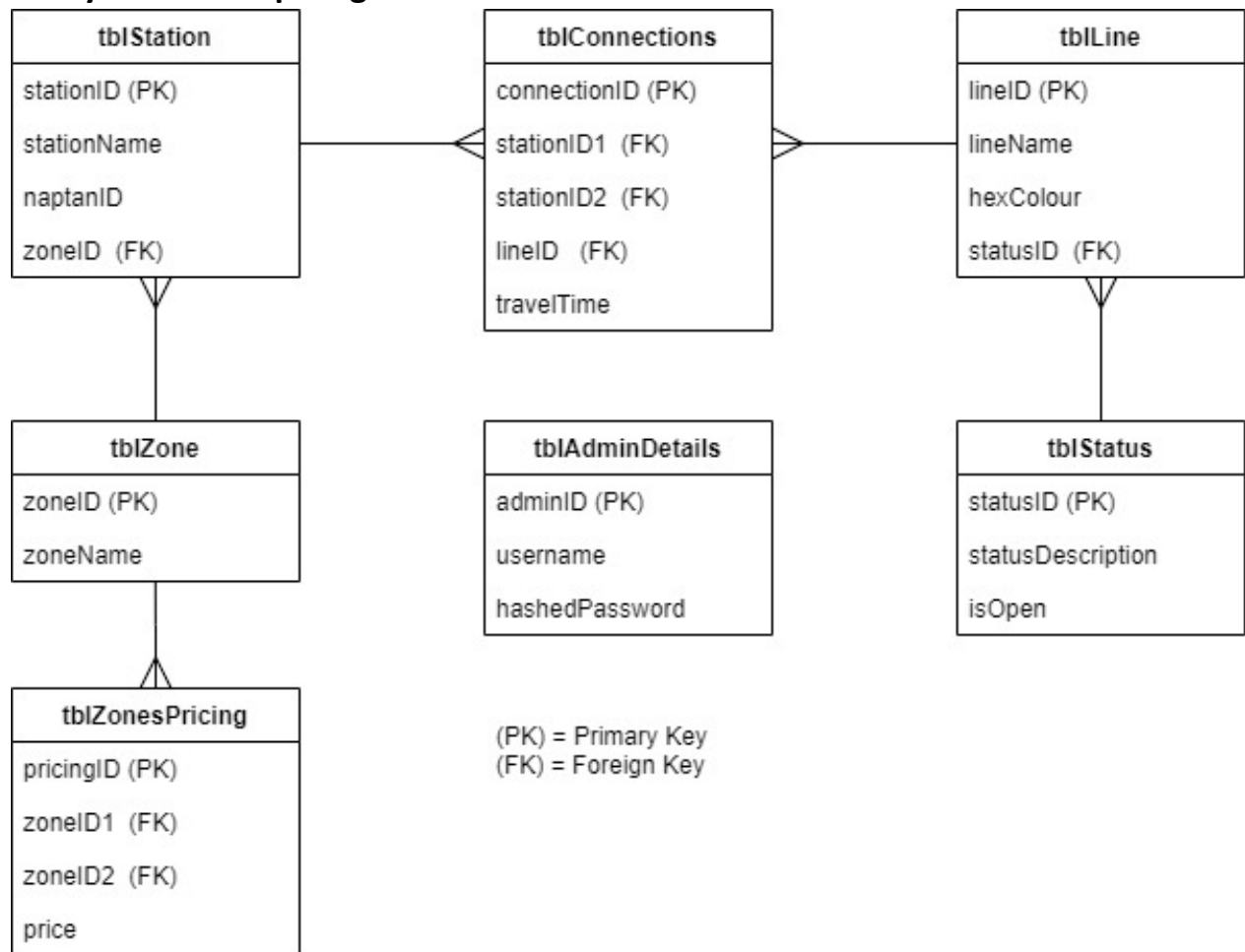


Description of Record Structure

- I'm going to make a local database in NetBeans.
- I'm going to use CSV files to import data into my databases.
- I will also be storing a PNG image file for the application icon.

I will store my database and other files in separate folders within my project folder.

Entity-relationship diagram



Data Dictionary

tblStation

Item Name	Example	Data Type	Size	Validation?
stationID	157	Integer	10 NB: 10 is the size assigned to an Integer data type	N/A
stationName	London Bridge	Varchar	30 NB: the longest station name is 'Crossharbour & London Arena'	N/A
naptanID	940GZZLULNB	Varchar	11 NB: NaPTAN IDs are 11 characters or less	N/A
zoneID	1	Integer	10	N/A

tblLine

Item Name	Example	Data Type	Size	Validation?
lineID	3	Integer	10	N/A
lineName	Circle Line	Varchar	25 NB: the longest line name is 'Hammersmith & City Line'	N/A
hexColour	FFE02B	Varchar	6	N/A
statusID	1	Integer	10	N/A

tblConnections

Item Name	Example	Data Type	Size	Validation?
connectionID	26	Integer	10	N/A
stationID1	82	Integer	10	N/A
stationID2	139	Integer	10	N/A
lineID	7	Integer	10	N/A
travelTime	2	Integer	10	N/A

tblStatus

Item Name	Example	Data Type	Size	Validation?
statusID	1	Integer	10	N/A
statusDescription	Open	Varchar	50	N/A
isOpen	TRUE	Boolean	1	N/A

tblZone

Item Name	Example	Data Type	Size	Validation?
zoneID	1	Integer	10	N/A
zoneName	Zone 1	Varchar	7	N/A

tblZonePricing

Item Name	Example	Data Type	Size	Validation?
pricingID	1	Integer	10	N/A
zoneID1	1	Integer	10	N/A
zoneID2	1	Integer	10	N/A
price	2.4	Double	52 NB: 52 is the size assigned by default to a Double data type	The format of the price is checked using a regular expression. Error message: "Price must be in correct format"

tblAdminDetails

Item Name	Example	Data Type	Size	Validation?
adminID	1	Integer	10	N/A
username	User1	Varchar	20	Presence check Error message: "Username must be entered" Length must be <= 20. Error message: "Maximum Username length is 20 characters" Duplicate check Error message: "Username already exists"
hashedPassword	83878c91171338902e0fe0fb97a8c47a	Varchar	64 NB: a SHA256 hash is 64 Characers long	N/A

File Organisation and Processing

tblStation(stationID(10), stationName(30), naptanID(11), zoneID(10))

- There will be 302 station records in this table = $(10+30+11+10)*302 = 18422$ bytes
- I have not planned to implement a system to add new stations

tblLine(lineID(10), lineName(25), hexColour(6), statusID(10))

- There will be 14 records in this table = $(10+25+6+10)*14 = 714$ bytes
- I have not planned to implement a system to add new lines

tblConnections(connectionID(10), stationID1(10), stationID2(10), lineID(10), travelTime(10))

- There will be 406 records in this table = $(10+10+10+10+10)*406 = 20300$ bytes
- New connections will not be added as I will not be adding new lines or stations.

tblStatus(statusID(10), statusDescription(50), isOpen(1))

- There will be 2 possible status records in this table, open and closed = $(10+50+1)*2 = 122$ bytes
- There will not be a feature to add new statuses. However, this may change in the future and the number of records may increase to 6 over 2 years = $(10+50+1)*6 = 366$ bytes

tblZone(zoneID(10), zoneName(7))

- There will be 9 zone records in this table = $(10+7)*9 = 153$ bytes
- This number of zones will not need to increase

tblZonePricing(pricingID(10), zoneID1(10), zoneID2(10), price(52))

- There will be 45 records in this table = $(10+10+10+52)*45 = 3690$ bytes
- This number of records will not need to increase as the number of zones will not change

tblAdminDetails(adminID(10), username(20), hashedPassword(256))

- There will be 10 admin accounts in this table = $(10+20+64)*10 = 940$ bytes
- Over 2 years the number of admins may increase by 100% = $940*2 = 1880$ bytes

Total Data size

$18422 + 714 + 20300 + 366 + 153 + 3690 + 1880 = 45525$ bytes $\approx 45.5\text{Kb}$

Adding Records to my Database

In order to represent the Tube map I will need to enter all 302 station records and 406 connection records into my database. Whereas with other tables it will be quickest to enter the data manually when I first create the system, with these two tables I plan to automate the process.

To do this I will read the CSV files that I am taking my data from, using each line to create the records in my tables. I found these CSV files in a GitHub repository containing almost all the information I need to map out the Tube graph.

However, the 'naptanID' field needed for each station record is not included in the CSV files. To get these IDs I will need to call the Transport for London API with the name of each station. Each JSON response will include the station's NaPTAN ID.

GitHub datasets: <https://github.com/nicola/tubemaps/tree/master/datasets>

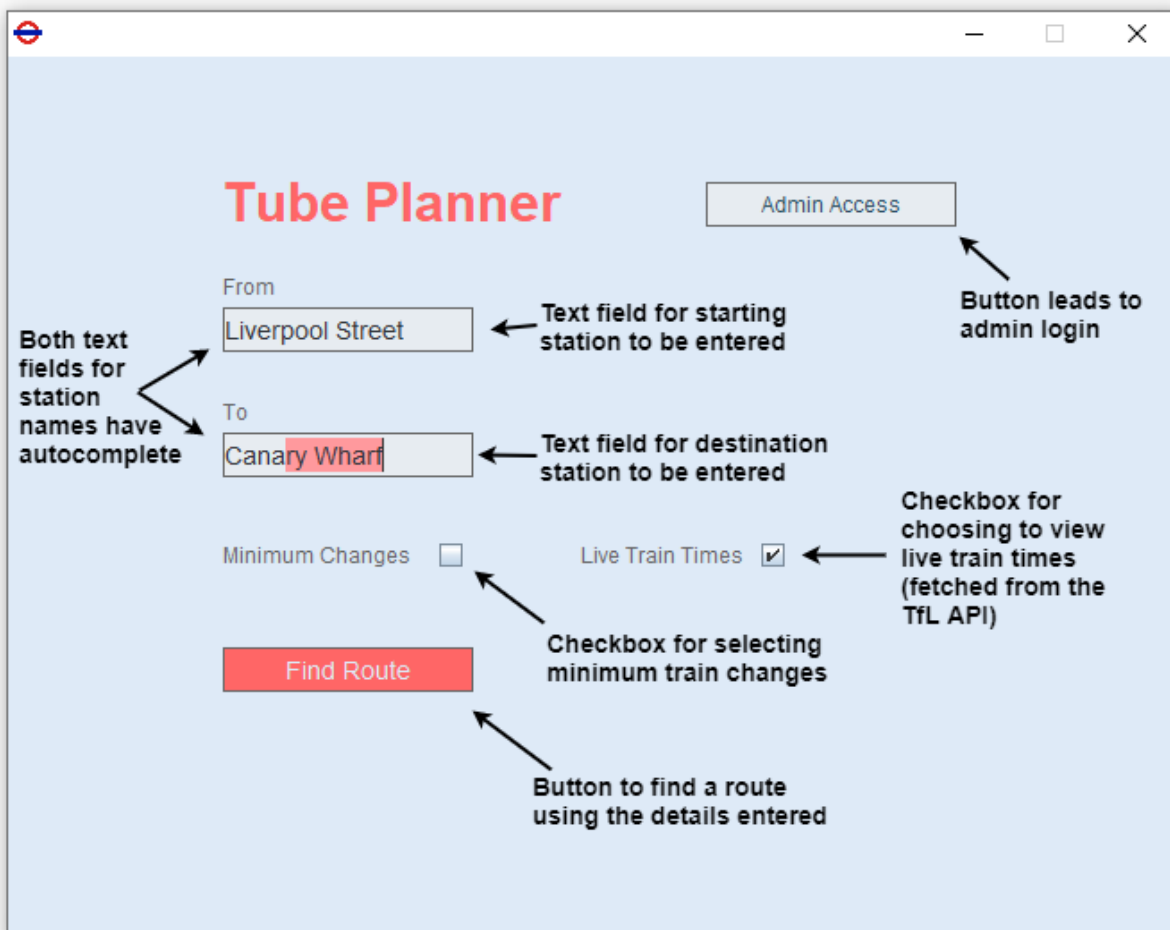
User Interface Design

The aesthetic of my GUI design has stayed the same as in my Analysis section. Now I have included screenshots of all my panels, with labels describing the functions of each element.

Main Menu Panel

When the 'Find Route' button is pressed the two stations entered are validated. If they are not valid an error message will be displayed, otherwise the route between these stations is calculated. An error message is displayed if:

- One or both stations have been left blank
- One or both stations do not exist
- The stations entered are the same
- A route cannot be found due to a closed train line.



Journey Results Information

Here all information about a route is displayed. If the API cannot be accessed at the current time, an error message will display instead of the live train times, with the rest of the information being shown as normal. Live train times will also not be shown if the 'Live Train Times' checkbox was not selected on the previous panel.

Journey Results

From Liverpool Street
To Canary Wharf

12 mins + train changes
£2.90

Key Route information

Line	Duration	Platform	Due in	Platform 1	Train times
Central Line to Bank	2 mins	Platform	Due in	Platform 1	6 min, 12 min, 21 min
DLR to Canary Wharf	10 mins	Platform	Due in	Platform 1	6 min, 12 min, 21 min

Line information

Train times and platforms from API

New Route

Button returns the user to the menu where they can find a new route

Scroll bar for when there are more lines travelled on, and therefore more information displayed on this panel

Admin – Login

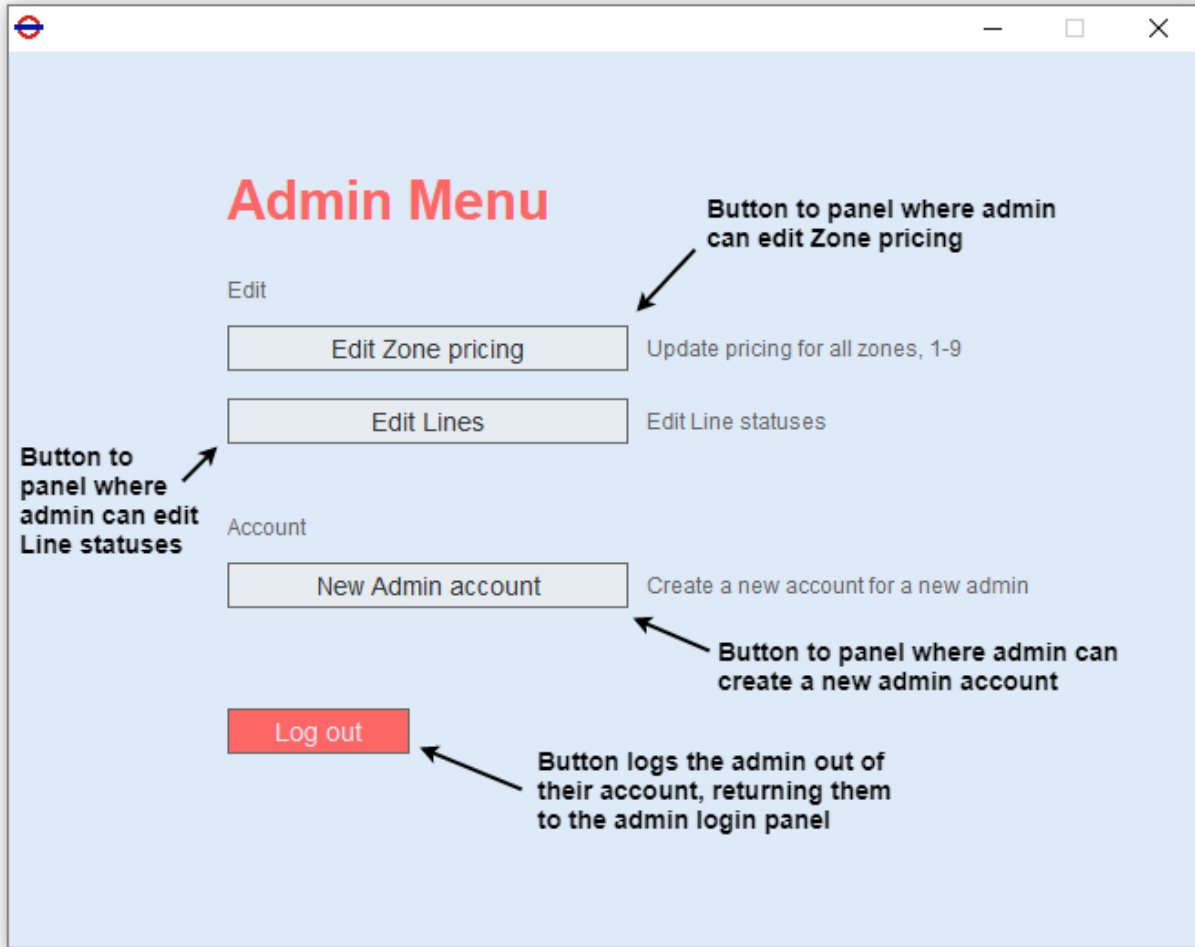
Admins can use this panel to login to their admin account. If the username and password entered match a record in the admin details table, they will enter the admin menu. Otherwise, an error message will be displayed.

The image shows a screenshot of a web application window titled "Admin Login". The window has a light blue background and a white title bar with standard window controls (minimize, maximize, close). The main content area contains the following elements:

- Username:** A text input field containing "Dave Douglas". An annotation points to it: "Text field for admin account username to be entered".
- Password:** A password input field with masked characters (dots). An annotation points to it: "Text field for admin account password to be entered".
- Login:** A red button with the text "Login". An annotation points to it: "Button to Login to an admin account - If the username and password are valid".
- BACK:** A red button with the text "BACK". An annotation points to it: "Button returns the user to the main menu where a route can be found".

Admin – Menu

The admin menu allows admins to access all three admin tasks: edit zone pricing, edit lines statuses, and creating a new admin account.



Admin – Edit Zone pricing

This panel allows admins to make changes to zone prices using an interactive table. Changes are not updated in the database unless the admin commits their changes using the 'Apply Changes' button.

Changes to these zone prices will change the prices displayed to users when they view route information.

The price fields in the table will use a regular expression to check each value entered is in the correct price format. A pound sign, £, will be accepted, but will not be compulsory.

Edit Zone pricing

Table displaying prices for each zone combination

	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5	Zone 6	Zone 7	Zone 8	Zone 9
Zone 1	£2.40								
Zone 2	£2.90	£1.70							
Zone 3	£3.30	£2.90	£1.70						
Zone 4	£3.90	£3.30	£2.90	£1.70					
Zone 5	£4.70	£3.90	£3.30	£2.90	£1.70				
Zone 6	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70			
Zone 7	£5.60	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70		
Zone 8	£6.90	£5.60	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70	
Zone 9	£7.00	£6.90	£5.60	£5.10	£4.70	£3.90	£3.30	£2.90	£1.70

A price can be selected and edited

Button returns the admin to the admin menu

BACK

Button commits changes in the table above to the local database

Apply Changes

Admin – Edit Lines

The Edit Lines panel is where admins can update the statuses of lines. All lines are displayed, and an admin can change each one's status using the drop-down lists. Changes will not be saved to the database unless the 'Apply Changes' button is pressed.

Closing lines means they can no longer be travelled on. My path finding algorithm will avoid these lines, redirecting user's routes. If a journey can no longer be completed due to a line closure, an error will be displayed to the user.

The screenshot shows a web interface titled "Edit Lines" with a list of lines and their current status. The interface includes a "BACK" button and an "Apply Changes" button. Annotations with arrows point to specific elements: "Each line's status is displayed" points to the status column; "Drop-down list allows admin to change the status of a line" points to the District Line's status dropdown, which is open showing "Open" and "Closed" options; "Button returns the admin to the admin menu" points to the "BACK" button; and "Commits changes to line statuses to the database" points to the "Apply Changes" button.

Line	Status
Bakerloo Line _____	Open
Central Line _____	Open
Circle Line _____	Open
District Line _____	Open Closed
East London Line _____	Open
Hammersmith & City Line _____	Open
Jubilee Line _____	Open
Metropolitan Line _____	Open
Northern Line _____	Open
Piccadily Line _____	Closed
Victoria Line _____	Open
Waterloo & City Line _____	Open
DRL _____	Closed

Annotations:

- Each line's status is displayed
- Drop-down list allows admin to change the status of a line
- Button returns the admin to the admin menu
- Commits changes to line statuses to the database

Admin – New Admin Account

Here is where new admin accounts are created. This panel must not be available to any ordinary user, so is accessed from the admin menu.

For an account to be created, the information entered must be valid:

- The username must be unique to all existing usernames
- The username must be ≤ 20 characters
- The username must not be blank
- The password must be between 8 and 20 characters (inclusive)
- The password must be the same as the confirming password.

The screenshot shows a web form titled "New Admin Account" with a light blue background. The form contains three text input fields and two buttons. Annotations with arrows point to each element:

- Username:** A text input field containing "Dave Douglas". An arrow points to it with the text "Text field for username to be entered".
- Password:** A text input field filled with 10 dots. An arrow points to it with the text "Text field for password to be entered".
- Confirm Password:** A text input field filled with 10 dots. An arrow points to it with the text "Text field for confirming password to be entered".
- BACK:** A red button with white text. An arrow points to it with the text "Button returns the admin to the admin menu".
- Create Account:** A red button with white text. An arrow points to it with the text "Button create a new admin account using the details entered".

Algorithms

Dijkstra's algorithm

I am using this algorithm as my path finding solution. It will allow me to get the fastest route between two stations and return it to the user. I have chosen to use this algorithm because it allows me to find the fastest route on a node and edge graph. This means I can use stations as nodes and tube lines as edges to make up the London Underground map as a graph that can be traversed by the algorithm.

Why not A*

A* pathfinding builds on Dijkstra's algorithm but also considers straight line distances between each node and the end node, prioritising nodes that are the quickest to reach (like in standard Dijkstra's) but also the nodes that head in the correct general direction of the end node. This form of pathfinding is quicker than Dijkstra's as it checks less nodes before finding the shortest path.

However, I have chosen to use the standard Dijkstra's algorithm rather than A* pathfinding because I don't have the real-world distance information between each pair of stations, which is needed for A*. Another reason is that real world distances do not necessarily relate to travel time on the Tube, because Tube lines are not direct, and have different numbers of stops and changes. The London Underground network is also not so big that there would be a significant performance boost from using an A* pathfinding algorithm.

Code:

```
package tuberoutefinder;

import java.util.ArrayList;

public class DijkstrasAlgorithm {

    private final ArrayList<GraphNode> graphNodeArray; // all nodes in the graph
    private final ArrayList<GraphEdge> graphEdgeArray; // all edges in the graph
    private final int[][] adjacencyList;
    private final GraphNode startNode;
    private GraphNode endNode;
    private final boolean minLineChanges;
    private final int timeToChangeLines; // time to change from one line to another in mins

    public DijkstrasAlgorithm(ArrayList<GraphNode> graphNodeArray, ArrayList<GraphEdge> graphEdgeArray,
        int[][] adjacencyList, GraphNode startNode, GraphNode endNode, boolean minLineChanges) {
        this.graphNodeArray = new ArrayList<>(graphNodeArray);
        this.graphEdgeArray = new ArrayList<>(graphEdgeArray);
        this.adjacencyList = adjacencyList;
        this.startNode = startNode;
        this.endNode = endNode;
        this.minLineChanges = minLineChanges;
        this.timeToChangeLines = 5;
    }
}
```

```
public ArrayList<GraphNode> getRoute() {
    // set timeFromStartStation, of starting station, to 0
    graphNodeArray.get(startNode.getIndexInGraphNodeArray()).setTimeFromStartStation(0);

    // create an arrayList acting as the priority queue of graphNodeArray, ordered in terms of
timeFromStartStation of each node
    ArrayList<Integer> queueOfGraphNodeIndexes = new ArrayList<>();
    queueOfGraphNodeIndexes.add(startNode.getIndexInGraphNodeArray());

    // recursively runs until the time to get to the end node has been confirmed
    recursiveAlgorithm(endNode.getIndexInGraphNodeArray(), queueOfGraphNodeIndexes, adjacencyList);

    // back tracks from the end GraphNode (using previousNode of each GraphNode) to the starting GraphNode
    // then returns an array of GraphNodes on the route
    return backtrackToFindRoute(endNode.getIndexInGraphNodeArray());
}

private void recursiveAlgorithm(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
    graphNodeArray.get((queueOfGraphNodeIndexes.get(0))).setVisited(true);
    while (graphNodeArray.get(queueOfGraphNodeIndexes.get(0)).isVisited()) {
        queueOfGraphNodeIndexes.remove(0);
    }

    // if the end node is not first in queue, carry on recursion
    if (graphNodeArray.get(queueOfGraphNodeIndexes.get(0)).getStationID() !=
(graphNodeArray.get(endNodeID).getStationID())) {
        recursiveAlgorithm(endNodeID, queueOfGraphNodeIndexes, adjacencyList);
    }

    // the end node may be at a different platform to the end node at the start
    endNode = graphNodeArray.get(queueOfGraphNodeIndexes.get(0));
}

private ArrayList<Integer> updateQueue(ArrayList<Integer> queueOfGraphNodeIndexes, int[][] adjacencyList) {
    // gets the index of the node currently first in the priority queue (the index of the node in graphNodeArray)
    int nodeIndex = queueOfGraphNodeIndexes.get(0);

    // loops through all the edges of the first node in the priority queue
    int edgeNum = 0;
    while (edgeNum < adjacencyList[0].length && adjacencyList[nodeIndex][edgeNum] != -1) { // -1 means null
and therefore no more edges
```

```
// gets the details of the edge being checked
GraphEdge currentEdge = graphEdgeArray.get(adjacencyList[nodeIndex][edgeNum]);

// gets the index of the node in currentEdge connected to the node currently first in the priority queue
int connectedNodeIndex;
if (currentEdge.getNode1().getIndexInGraphNodeArray() == queueOfGraphNodeIndexes.get(0)) {
    connectedNodeIndex = currentEdge.getNode2().getIndexInGraphNodeArray();
} else {
    connectedNodeIndex = currentEdge.getNode1().getIndexInGraphNodeArray();
}

if (!graphNodeArray.get(connectedNodeIndex).isVisited()) { // check to see if connected node has already
been visited (had its travel time confirmed)
    // time currently in 'timeFromStartStation' in neighbour node:
    int previousTime = graphNodeArray.get(connectedNodeIndex).getTimeFromStartStation();
    // time to neighbour node through current node being checked:
    int possibleNewTime = currentEdge.getTravelTime() +
graphNodeArray.get(nodeIndex).getTimeFromStartStation();

    // if the current node is the start, travel time between line changes should take 0 time
    if (nodeIndex == startNode.getIndexInGraphNodeArray() && currentEdge.getLine().getLineID() == 0) { //
lineID 0 is a line change
        possibleNewTime = 0;
    } else if (currentEdge.getLine().getLineID() == 0 && minLineChanges) {
        // time for a line change is set to a huge value to deter route finder if
        // minChanges check box has been ticked
        possibleNewTime = graphNodeArray.get(nodeIndex).getTimeFromStartStation() + 50;
    } else if (currentEdge.getLine().getLineID() == 0) {
        // time to travel a line change is set
        possibleNewTime = graphNodeArray.get(nodeIndex).getTimeFromStartStation() + timeToChangeLines;
    }

    if (possibleNewTime < previousTime) {
        // update TimeFromStartStation of node
        graphNodeArray.get(connectedNodeIndex).setTimeFromStartStation(possibleNewTime);
        // records the previous node that gave this node its time
        graphNodeArray.get(connectedNodeIndex).setPreviousNode(graphNodeArray.get(nodeIndex));
        // records the line that gave this that gave this node its time
        graphNodeArray.get(connectedNodeIndex).setPreviousLine(currentEdge.getLine());

        // insert node into queue (do not need to remove it from the queue if it is already in the queue
        // as when duplicates get to the front they will just be removed)
        queueOfGraphNodeIndexes =
insertIntoQueue(graphNodeArray.get(connectedNodeIndex).getIndexInGraphNodeArray(),
queueOfGraphNodeIndexes);
    }
}
edgeNum++;
```

```
    }
    return queueOfGraphNodeIndexes;
}

// uses a binary search to insert the new node in the priority queue based on its timeFromStartStation
private ArrayList<Integer> insertIntoQueue(int nodeIndex, ArrayList<Integer> queueOfGraphNodeIndexes) {

    int left = 0; // this pointer marks the index of the left most value (smallest value) that hasn't been checked
    int right = queueOfGraphNodeIndexes.size() - 1; // this pointer marks the index of the right most value
(largest) that hasn't been checked
    int mid = queueOfGraphNodeIndexes.size() / 2; // this pointer marks the mid point of the left and right
pointers, and is the index of the next value to be compared

    // continues until position for nodeIndex to be inserted has been found
    while (left <= right) {
        if (graphNodeArray.get(nodeIndex).getTimeFromStartStation() <
graphNodeArray.get(queueOfGraphNodeIndexes.get(mid)).getTimeFromStartStation()) {
            right = mid - 1;
        } else if (graphNodeArray.get(nodeIndex).getTimeFromStartStation() >
graphNodeArray.get(queueOfGraphNodeIndexes.get(mid)).getTimeFromStartStation()) {
            left = mid + 1;
        } else {
            break;
        }
        mid = (left + right) / 2;
    }
    queueOfGraphNodeIndexes.add(mid + 1, nodeIndex);
    return queueOfGraphNodeIndexes;
}

// prints out route
private ArrayList<GraphNode> backTrackToFindRoute(int endNodeIndex) {

    ArrayList<GraphNode> routeArrayList = new ArrayList<>();

    GraphNode graphNode = graphNodeArray.get(endNodeIndex);

    while (graphNode.getPreviousNode() != null) {
        routeArrayList.add(0, graphNode);
        System.out.print(graphNode.getName());
        System.out.print(" <-- ");
        graphNode = graphNode.getPreviousNode();
    }
    System.out.println(graphNode.getName());
    routeArrayList.add(0, graphNode);
    return routeArrayList;
}
}
```

Autocomplete

I have created autocomplete for my text fields so that when a user types in a station name they don't need to spend time typing out the whole name. This makes finding a route a quicker process. It also means the user doesn't need to know the correct spelling, just the first few letters.

Code:

```
private int charsTyped = 0;
private int length = 0;
static String[] stationNames;

private void autoComplete(JTextField textField) {
    charsTyped++;

    // 'charsTyped' is reset if the length of the unselected text has been reduced
    try {
        if ((textField.getText().length() - textField.getSelectedText().length()) <= length) {
            charsTyped = 0;
        }
    } catch (Exception e) {
        if ((textField.getText().length()) <= length) {
            charsTyped = 0;
        }
    }

    // local variables used to apply autocomplete
    String text = textField.getText();
    String newText = "";

    // checks all station names for a match with the text entered
    for (String stationName: stationNames) {
        if (text.length() < stationName.length() && text.equalsIgnoreCase(stationName.substring(0, text.length())) &&
            charsTyped > 0) {
            newText = stationName;
        }
    }

    // apply autocomplete
    if (!newText.equals("")) {
        textField.setText(newText); // updates textfield to autocompleted text
        textField.select(text.length(), newText.length()); // highlight autocompleted text
    }

    // Sets 'length' = to the length of the unselected text in the textfield:
    try {
        length = textField.getText().length() - textField.getSelectedText().length();
    } catch (Exception e) {
        length = textField.getText().length();
    }
}
```

```
}  
}
```

Priority Queue

I will use a priority queue in my Dijkstra's algorithm to hold the indexes of the next nodes to be searched by the path finding algorithm. They will be ordered by the travel time it takes to reach each node. This leads to the destination node receiving its travel time only when it is certain it is the quickest possible path.

Code:

The priority queue is an ArrayList called 'queueOfGraphNodeIndexes'. All the code for it is included previously in the algorithm section on [Dijkstra's algorithm](#).

Calling parameterised Web service API and parsing JSON

I'm calling an API provided by Transport for London because it will be useful for the users to know the live arrival times of the trains they are looking to catch. My API calls are parameterised with the 'NaPTAN' unique key of each station. This makes sure that the response returns information about train arrivals for the specific station that the user will be leaving from.

Below is some code for a test project I have made to print out all train arrival times for a particular station, along with the platform each train is leaving from. This test program prints out this information in the console, but when I add this feature to my project, it will display on the 'Route Details' Panel.

Code:

```
package tflapi_test;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
  
public class TflAPI_test {  
  
    public static void main(String[] args) {  
        long timeWhenFindRoutePressed = System.currentTimeMillis();  
        jsonURL("940GZZLUCWR"); // this naptanID represents Canada Water Station  
        System.out.println("");  
        System.out.println("Time taken for request: " + (System.currentTimeMillis() - timeWhenFindRoutePressed) + "  
ms");  
    }  
}
```

```
}

public static void jsonURL(String naptanID) {
    try {
        // url give json response with data about the trains arriving at the station of the given ID
        String url = "https://api.tfl.gov.uk/StopPoint/" + naptanID + "/arrivals";
        URL URLobj = new URL(url);

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

        int responseCode = con.getResponseCode();
        System.out.println("\nSending 'GET' request to URL : " + url);
        System.out.println("Response Code : " + responseCode); // returns response code so I know what the error
is (e.g. 200, 404..)
        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
            }
        }

        // JSONArray to store the different JSONObject from the response, there are different objects for each train
        JSONArray JSONtrainObjectArray = new JSONArray(response.toString());

        // Loops through all JSONObject from the URL request
        for (int i = 0; i < JSONtrainObjectArray.length(); i++) {
            System.out.println("");

            JSONObject Jobject = new JSONObject(JSONtrainObjectArray.get(i).toString());
            //System.out.println(Jobject.toString()); // prints the whole Object
            String arriavlTime = Jobject.getString("expectedArrival"); // gets the value stored under 'expectedArrival'
in the JSONObject (this is the arrival time, e.g. 2020-05-24T14:34:43Z)
            String platform = Jobject.getString("platformName"); // gets the value stored under 'platformName' in
the JSONObject (this is the platform name and direction, e.g. Westbound - Platform 1)
            // System.out.println(arriavlTime);
            System.out.println(platform);

            dueTime(arriavlTime.replace("T", " ").replace("Z", ""));
        }
    } catch (IOException | JSONException e) {
        System.out.println("ERROR:" + e);
    }
}
```



```
static void dueTime(String arrivalTime) {
    DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
    Date date = new Date();
    try {
        date = dateFormat.parse(arrivalTime);

    } catch (ParseException e) {
        System.out.println(e);
    }

    // calculates due time in mins
    long dueMins = (date.getTime() - System.currentTimeMillis()) / 60000;

    // if time zone is currently GMT rather than BST, times will appear 60 mins to big:
    if (dueMins >= 60) {
        dueMins -= 60;
    }

    System.out.println("Due: " + dueMins + " mins");
}
}
```

Hashing

I plan to use a SHA256 hash to store admin passwords securely in my database. I am using SHA256 as it is one of the most secure hash functions and it is more secure than using MD5 or SHA1. Below is the code I am using to implement this hash function.

Code:

```
// 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[] digest = messageDigest.digest(password.getBytes()); // apply hash to password

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

Cross-table SQL

I plan to use cross-table SQL when fetching zone pricing data from my database. I will use a JOIN to link the tblZonePricing table with the tblZone table. When fetching this zone pricing information. I will also need to use a UNION to get two records of zone details from tblZone for every record in tblZonePricing.

SQL statement I will use to get zone pricing information:

```
SELECT * FROM APP.tblZonePricing
JOIN APP.tblZone ON APP.tblZonePricing.zoneID1 = APP.tblZone.zoneID
UNION
SELECT * FROM APP.tblZonePricing
JOIN APP.tblZone ON APP.tblZonePricing.zoneID2 = APP.tblZone.zoneID
```

Parameterised SQL

I will use parameterised prepared SQL statements when inserting admin accounts into my database. I will also use them when checking whether an admin account exists for the username and password entered. In both cases I am using user-entered data in an SQL statement so I am using parameterised prepared statements to prevent SQL injection attacks.

Admins also have the ability to update zone pricing information and line statuses in the database. However, neither of these cases require prepared statements because the pricing information is validated via a regular expression and the line statuses are edited with drop-down lists so there is no user entered variables.

Below is the code I will use to insert new admin accounts and to check if an admin account exists, both using parameterised prepared SQL statements.

Code to insert new admin account:

```
// 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  
}
```

Code to check if admin account exists:

```
// 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  
}
```

Aggregate SQL functions

I will use the MAX function so that I am able to assign a unique primary key to a new admin account. Using this method means there will never be any duplicate IDs.

Below is the code I will use to create a new unique primary key (adminID) for a new admin account.

Code:

```
// 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;
}
```

Merge Sort

I am using a merge sort to sort station objects into alphabetical order based on the UNICODE value of their name. I am sorting the stations so that I can use a binary search to find a station entered by the user. This sorting allows the path finding to be quicker as I don't then need to use the slower linear search method.

I have chosen to use merge sort as it is much faster than other sorting algorithms such as insertion sort. Below is the code I will use to merge sort the stations.

Code:

```
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
public void merge(ArrayList<Station> stationArray, int leftIndex, int mid, int rightIndex) {

    ArrayList<Station> tempStationArray = new ArrayList<>(); // the two halves will be merged into this temporary array
    int firstHalfIndex = leftIndex; // index of next element to be considered in the first half
    int secondHalfIndex = mid + 1; // index of next element to be considered in the second half

    // while firstHalfIndex and secondHalfIndex haven't reached the end, add the alphabetically first into tempStationArray
    while (firstHalfIndex <= mid && secondHalfIndex <= rightIndex) {
```

```
// compares the two strings
int checkNum =
stationArray.get(firstHalfIndex).getName().compareToIgnoreCase(stationArray.get(secondHalfIndex).getName());

if (checkNum < 0) {
    tempStationArray.add(stationArray.get(firstHalfIndex));
    firstHalfIndex++;
} else {
    tempStationArray.add(stationArray.get(secondHalfIndex));
    secondHalfIndex++;
}
} // while ends when all of one half has been added to tempStationArray

// add any remaining items from the first half into tempStationArray
while (firstHalfIndex <= mid) {
    tempStationArray.add(stationArray.get(firstHalfIndex));
    firstHalfIndex++;
}

// add any remaining items from the second half into tempStationArray
while (secondHalfIndex <= rightIndex) {
    tempStationArray.add(stationArray.get(secondHalfIndex));
    secondHalfIndex++;
}

// override the non-merged stations (leftIndex to rightIndex) in stationArray with the new merged stations in
tempStationArray
for (int j = 0; j < tempStationArray.size(); j++) {
    stationArray.set(leftIndex + j, tempStationArray.get(j));
}
}
```

Recursion

I have used recursion in my merge sort, and in my implementation of Dijkstra's algorithm. I have used recursion in both cases as it is a more elegant solution than iteration making it easy to view my code and understand what is happening.

Code:

The code for both these implementations of recursion are shown previously in my algorithm sections about [Merge sort](#) and [Dijkstra's algorithm](#) respectively.

Binary Search

I am using a binary search to search for a station name entered by the user in an array of stations. I am using this search algorithm as it is much faster than linear search.

Code:

```
// binary search - used by GUI to confirm that the station name entered exists
public GraphNode binarySearchForNode(String stationName) {

    int left = 0; // this pointer marks the index of the left most value (towards A) that hasn't been checked
    int right = graphNodeArray.size() - 1; // this pointer marks the index of the right most value (towards Z) that
    hasn't been checked
    int mid = graphNodeArray.size() / 2; // this pointer marks the mid point of the left and right pointers, and is
    the index of the next value to be compared

    // continues until the the stationName is found, or it is found to not exist
    while (left <= right) {

        String currentStationName = graphNodeArray.get(mid).getName();

        // compareToIgnoreCase() returns 0 if strings are equal, negative number if the second
        // string appears after the first in UNICODE, and positive if it appears before
        int checkNum = currentStationName.compareToIgnoreCase(stationName);

        if (checkNum > 0) {
            right = mid - 1;
        } else if (checkNum < 0) {
            left = mid + 1;
        } else {
            return graphNodeArray.get(mid);
        }

        mid = (left + right) / 2;
    }
    return null;
}
```

Regular expression

I am using a regular expression to check that the prices entered by admins are in the correct format. I am using a regular expression for validation as it can validate the strings in very few lines of code.

Regular expression I have created for checking price format:

```
(^£?\d*\.\d{1,2}$) | (^£?\d+\.\d{0,2}$)
```

The method for where this regular expression is used is below. TRUE is returned if the string is valid, and FALSE if not valid.

Code:

```
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
}
```

Reading from files

I am reading from CSV files so I can import my station and connection information into my database tables rather than typing it all in manually. I need to use CSV files because the datasets I am using from GitHub are in this format. I am also reading an image file so I can set the icon image of my application, instead of having the default java logo.

Below is my code for reading the image file and then reading one of the CSV files.

Code for reading image file:

```
// set icon of GUI
Image icon = Toolkit.getDefaultToolkit().getImage("Files\\\\\\\\Images\\\\\\\\UndergroundIcon.png");
setIconImage(icon);
```

Code for reading CSV file:

```
try {
    // the CSV file
    String file = "Files\\\\\\\\CSVdata\\\\\\\\Stations.txt";

    // Create filereader object
    FileReader filereader = new FileReader(file);

    // create csvReader object
    CSVReader csvReader = new CSVReader(filereader);
    String[] nextRecord;

    // reading data line by line
    while ((nextRecord = csvReader.readNext()) != null) {
        // here can look at all fields in this line
        // nextRecord contains all fields in this record/line
    }

} catch (Exception e) {
    System.out.println("ERROR : " + e);
}
```

Libraries and APIs

Imports

```
com.opencsv.CSVReader;

java.awt.CardLayout;
javax.swing.Color;
java.awt.Component;
java.awt.Dimension;
java.awt.Image;
java.awt.Toolkit;

java.io.BufferedReader;
java.io.File;
java.io.FileReader;
java.io.InputStreamReader;
java.io.IOException;

java.net.HttpURLConnection;
java.net.URL;

java.math.BigInteger

java.security.MessageDigest;
java.security.NoSuchAlgorithmException;

java.sql.Connection;
java.sql.DriverManager;
java.sql.PreparedStatement;
java.sql.ResultSet;
java.sql.SQLException;
java.sql.Statement;
java.sql.Time;

java.text.DateFormat;
java.text.ParseException;
java.text.SimpleDateFormat;

java.util.ArrayList;
java.util.Arrays;
java.util.concurrent.TimeUnit;
java.util.Date;
java.util.regex.Matcher;
java.util.regex.Pattern;
java.util.Scanner;

javax.swing.JLabel;
javax.swing.JPanel;
javax.swing.JTextField;
```

Java Libraries

Derbclient.jar

- This will be used when creating and using my database

java-json.jar

- This includes libraries that allow me to parse my API responses from plain text into JSON objects. This allows specific fields of data to be easily accessed

opencsv-4.1.jar

- Used to read data from CSV files. I will need to do this when first setting up my database

commons-lang3-3.11.jar

- This library is also needed along with opencsv to parse data from CSV files

APIs

I will use the Transport for London API to get train arrival times.

I will also use this API to get station NaPTAN IDs when I first enter them into my database.

```
org.json.JSONArray;
org.json.JSONException;
org.json.JSONObject;
```