Group 12

COMP 1828 - Designing, developing and testing a journey planner model for the London Underground system

Team members:

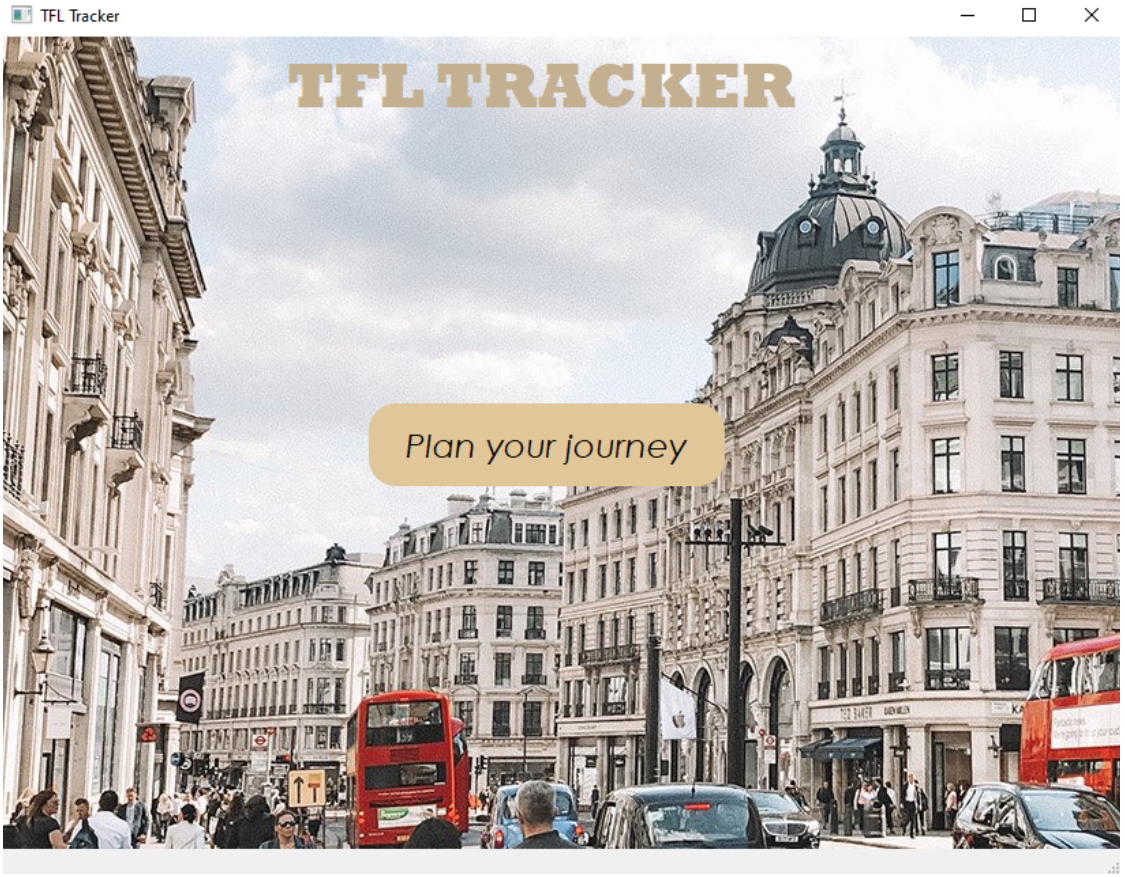
Francesco Romano - 001086733-3

Hadil Bati - 001090152

Mohamed Saif Meeran Mohamed Mohideen - 001081662

Yuhaï Namugunga - 001080121

1. **“HOW TO guide” for a customer and any assumptions made (max 3 pages) [10 marks]**



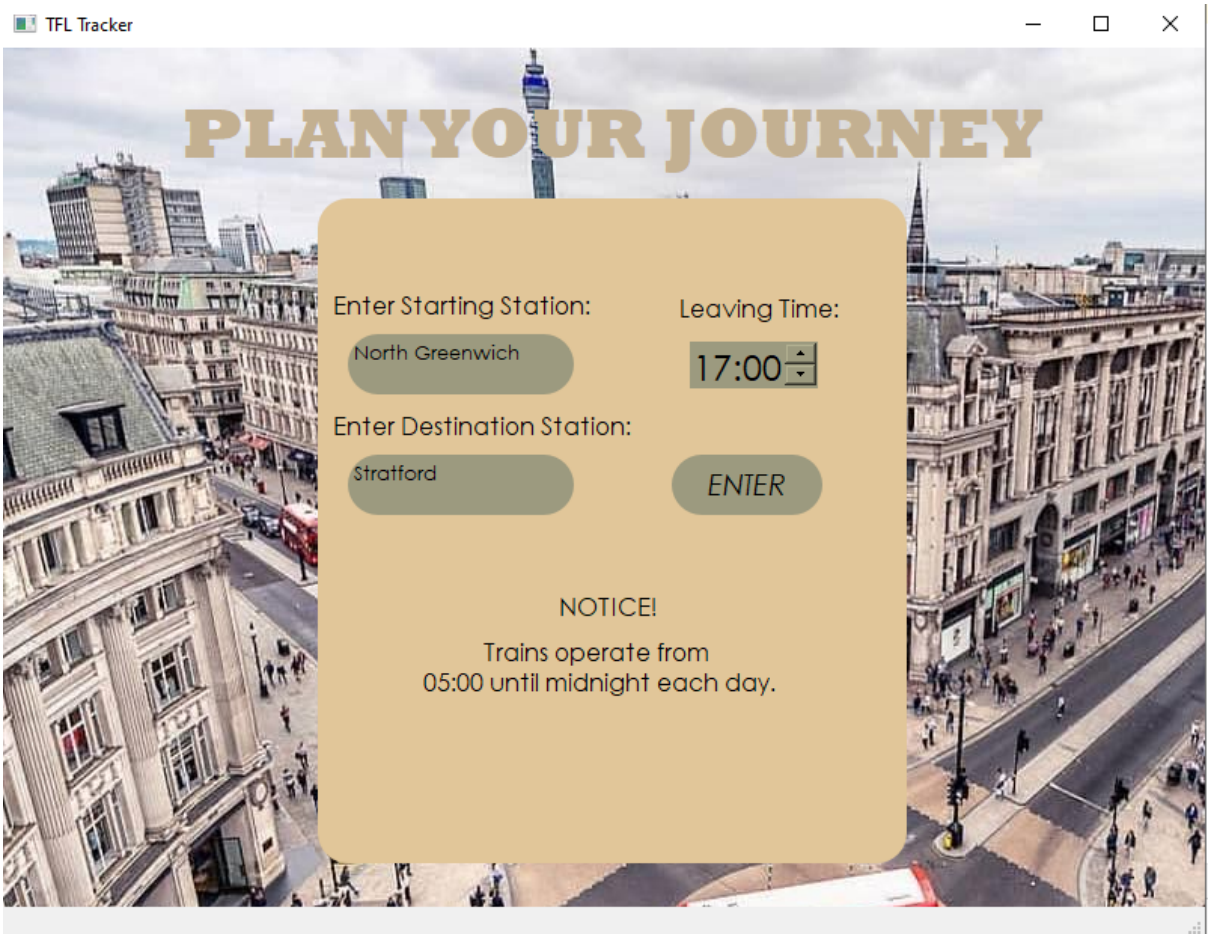
Step 1- Click ‘Plan your journey’

When first opening the TFL Tracker, there is a button which is labelled ‘‘Plan your journey’. Upon clicking the button, it will take you to the window where you enter your journey stations.

ASSUMPTIONS:

- The program only looks for the quickest path and it does not take into account the number of changes. Nevertheless changing lines has a considerable cost in terms of minutes so the program will not change incredibly often.
- Trains wait for 1 minute at each station in order to allow passengers enter or exit the train. This will affect the total time of travel, the only exception being the last station where the traveller will disembark the train.
- It is assumed that the user will always get onto the correct train and follow the exact same route provided by the application.
- Trains will never be affected by delays, disruptions or technical difficulties, and the carriages will always have enough room for the passenger.

- Trains will run every five minutes from 5:00 to 23:00 regardless of the day of the week, peak and off-peak times, line and time of the day.
- Between 9am-4pm and from 7pm-12am, the Bakerloo line will use a new system of sequencing lights that will allow trains to travel at twice the original speed, making journeys on the Bakerloo line half as long only during that specified time. During those time slots trains on the Bakerloo line will run every 2.5 minutes, but will continue stopping for one minute at each station.



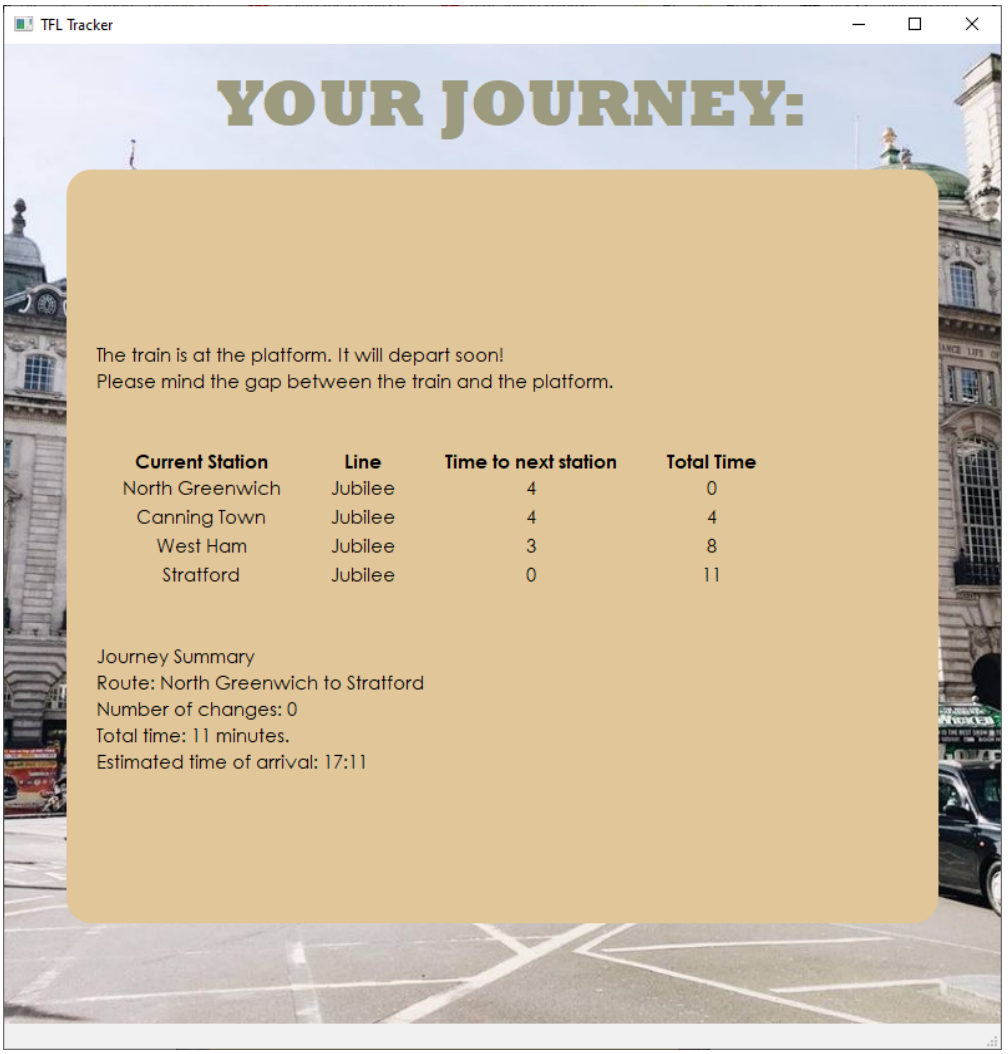
Step 2- Enter your Start & End stations

Next, you are required to enter your start and end stations. You can do this by typing them in the required fields. (E.g: from starting at ‘North Greenwich’, and ending at ‘Stratford’). You will also be required to input the time that you are leaving to achieve the most accurate results of journey time. After entering the stations and leaving time, you can click the ‘Enter’ button which will achieve your journey result. In this window, there is also a displayed notification with entails of the train operating times.

ASSUMPTIONS:

- The time to the next station displayed includes the minute to let passengers (dis)embark.

- The time to the next station displayed when changing lines includes:
    - Three minutes, corresponding to the time we assumed it takes to walk from one platform to another one.
    - The time spent waiting for the incoming train.
    - One minute for letting passengers (dis)embark.
- The Overground and DLR lines were split into multiple lines due to the fact that these two lines consist of a network of separate lines and many trains run through them following different paths. This was causing some problems because the program would think that a single train traveled to each station in the line but in fact, this does not happen.



Step 3- Receive Your Journey Results

Now that you have entered your start and end stations, you are ready to start your journey as you are given your journey results! The journey results include the exact stations from starting destination to end destination. Furthermore, it will display the underground line from each station as well as the time till arrival. Your journey summary will include your route, the number of changes you have had to take and the total time in minutes of your accumulated journey.

2. Critical evaluation of the performance of the data structures and algorithms used (max 1 page) [15 marks]

The functions 'get_lines', and 'link_same_line' have a big O notation of O(N), as they both have a for loop that goes through the entire excel file one time. This means that the amount of operations carried out is going to go up by 1 when one row is added to the file.

The function 'link_all_station' instead will take $O(N^2)$. This is due to the fact that for every single node in the Doubly Linked List the algorithm will look through every element in the list 'seen_stations' which contains every station in order to link the different Nodes.

The Doubly Linked List we used to store the information about the stations uses a linear search to find Nodes, so the complexity is O(N). The search functions will start from the head of the DLL and will go through it until it finds the corresponding element. In the worst case scenario the element is either the last one in the DLL (tail) or it is not present in the DLL, meaning that the algorithm will check every node in the list.
Appending an element to the doubly linked list will take O(1) because the program will quickly jump to the tail and add an element instead of looking through all the elements.

The time complexity for Dijkstra's algorithm is slightly more complex.
When setting the distance to every node the program goes through the list of stations and sets the distance to infinity. This has a time complexity of O(N) as the number of operations will rise in a linear way along with the number of nodes.

The core of the algorithm will try to find a path by analysing every node and its links to other nodes, so this has roughly a complexity of $O(N^2)$ in the worst case.

So, the whole algorithm has a complexity of $O(N + N^2)$

3. Discussion for the choice of test data you provide and a table detailing the tests performed (max 2 pages) [10 marks]:

The method of testing we had used to test our software was White Box Testing. In this test we had chosen inputs to exercise paths through the code and then determine the expected outputs.

S. No	Description	Input	Expected Output	Actual Output	Result	Remarks
1	Not entering one of the two stations or misspelling the station.	None or misspelt station.	"Check the name of the stations (start/end). They might be spelled incorrectly."	"Check the name of the stations (start/end). They might be spelled incorrectly."	Pass	Initially our program could not handle blank inputs so we added an if clause in order to prevent the program from throwing errors due to blank or misspelled station names.
2	Entering leaving time between midnight to 5:00am	Leaving Time: 00:00-4:59am	A message notifying the user that trains do not run at that time.	“Trains run from 5:00 to 00:00.”	Pass	An if clause determines whether the travel is possible before the underground shuts down or not.
3	Entering the same station	‘**Stratford**’ ‘**Stratford**’	A message that tells the user that they entered the same station twice.	"You are already there!"	Pass	An if clause detects if the two stations (leaving station and destination) are the same and if so it returns a message to notify the user.
4	Entering leaving time from 9am-4pm and from 7pm-12am on Bakerloo line.	‘**Baker Street**’ to ‘**Paddington**’ at 3pm	“Estimated time of arrival: 15:11”	“Estimated time of arrival: 15:11”	Pass	The algorithm successfully calculated the travel time between the stations on the bakerloo line.
5	Testing if the program can correctly calculate the waiting time for the next train.	‘**North Greenwich**’ to ‘**Canning Town**’ at 18:58	Waiting time 2 minutes	Waiting time: 2 minutes	Pass	The algorithm is able to correctly calculate the waiting times.

6	Testing if the program can correctly calculate the time it takes to change lines.	**'Marylebone'** to '**Great Portland Street**' 15:00	A total of 8 minutes is expected for the change from Baker street (Bakerloo) to Baker Street (Circle)	8 minutes.	Pass	The program is able to calculate the time it takes to change lines. The user was at Baker street on the Bakerloo line at 15.02 and it takes 4 minutes to walk to the other platform, and 4 minutes to wait from 15.06 to the next train at 15.10.
7	Trying to run the program twice	**Any path that involves the Bakerloo line.**	//	//	Fail	The program will have some issues when run more than once in the same session if the path involves the bakerloo line. This is due to the fact that the algorithm will half the times in the bakerloo line and when run again it will half them again, making the numbers imprecise.

A lot of our ideas for how to test the software to make sure the code is implemented in a method of accuracy, was by making sure we looked through the assumptions that were created. We found that utilising those assumptions into tests would indicate if we were correctly implementing the software that was required. As you can see from the table of testing that was made, a lot of the results were passed as we had programmed the software to make sure the requirements were met. Whilst we found that many of our testings had passed due to relatively good programming- from the error handling down to the timings, we still found that some tests failed. You can see that in 7 when we tried to run the program twice. We found that the minutes in the output turn into imprecise floats at times, and with further inspection and analysis we understood that this was due to the algorithm for the bakerloo line that runs at twice the speed so therefore the times will be halved again, if run in the same program twice.

4. Individual contribution by each team member and reflection (max 2 pages) [10 marks]

Name	Allocation of marks agreed by team (0-100)
Francesco Romano	25
Hadil Bati	25
Mohamed Saif Meeran Mohamed Mohideen	25
Yuhaï Namugunga	25

Our group had the advantage of working closely together to understand any of the implementations that were commenced by individuals. Having previously worked with Python we had the advantage of getting started and understanding the essential idea of the software we were required to create. We had studied data structures previously which further expanded our knowledge and experience by implementing those concepts into the TFL Tracker program. Furthermore, on the same guidelines we had also previously worked with implementing a GUI so we gained further understanding as we programmed.

The Backend implementation in this coursework was definitely a challenge as we all felt equally tentative about how we were going to put this software together. With finding that some team members knew better on how to manipulate Python and create an idea of the program- we wanted to make sure that every team member understood how and tried to troubleshoot together the errors that appeared as we went through every stage of our journey tracker program. We were still unable to completely continue the coursework until we understood Dijkstra's algorithm, therefore when we learned the implementation of it in the lecture as well as the Doubly Linked List, much became clearer on how to enforce it.

This is why in terms of reflection, we found that the team was very good- as we continuously tried to work together to put an effort in the very least understanding how to work through the coursework. When Hadil had troubles with understanding Dijkstra's algorithm implementation, Francesco's explanations in the team meetings on the code that was implemented was just as important as it made sure she had an idea on how to move forward with it. Likewise when Saif wanted to understand more about the functionality of the UI, Hadil was able to give in depth analysis about the specifics of the code. Furthermore, when Yuhaï and Saif had found and questioned the errors and omissions that they found in the database, the team work made sure that the team entirely understood and knew of those.

As a team, we found that we were least knowledgeable about the Tkinter module for implementing the GUI, therefore we came to the conclusion early on that PyQt5 was a useful module for us to utilise. Whilst we had grasped an idea for the Backend program implementation and had roughly created it, we grasped an idea as to how we wanted our Front end to look.

Although we had a useful advantage with some brushed knowledge on how to use Qt Designer, we still found that after we created the GUI windows, manually programming the GUI in Python was better. We consistently manipulated and learned as we went - creating and continuously testing the windows to add functionality - from the push buttons down to

the output window. It was entirely time consuming but we learned a lot more as we delved deeper into each of the parameters and how we manipulate them to create a pristine UI.

Conclusively, the advantage cooperation of our group is that with every change to the backend or frontend software, we would notify and explain the added functionality to make sure everyone in the group would understand. Whilst each person in the group had a small or large piece of understanding and input to offer, the teamwork and cooperation of the team overall excelled with implementing the final program.

INDIVIDUAL CONTRIBUTIONS:

Francesco Romano: I mainly worked on the backend. I started working on the project by understanding how to retrieve the data from the excel file into python. After analysing and correcting the data, I learned about DLLs and how to implement them in python. I was then able to create a Doubly Linked List and append all the nodes obtained from the excel file to it. Furthermore I worked with the other members of the group on integrating Dijkstra's algorithm in order to find the shortest path between two nodes. Finally, we all spent hours on team calls testing the code looking for bugs to fix and features to implement.

Mohamed Saif Meeran Mohamed Mohideen: I primarily worked as a Tester for the project. I had checked for errors in the excel spreadsheet by manually checking the data with the corresponding Tfl line maps and corrected any errors that I had come across, thereby ensuring the accuracy of the data that will be used to generate the nodes in the Double Linked List. I had worked with the rest of the team to implement Dijkstra's algorithm to determine the shortest path between two stations. I was also testing each iteration of the code to ensure that it works as desired and fixed any bugs that I came across. I had quality checked the results of the program and ensured that they meet the requirements of the specification.

Hadil Bati: I was primarily focused on the GUI. I began through recollecting the slight past experience I have had with UI integration and continued to build my knowledge. I used the PyQt5 module to manually integrate a GUI design with functions that are well suited to the TFL Tracker. I found it challenging, creating three windows and widgets that were suitable according to the backend functions that we integrated. I found my solution of using QLabel widget to display the backend outputs on the third window. The GUI was time consuming for me as there was continuous testing that took place. I worked alongside the team in Dijkstra’s algorithm integration, finding the challenge successful as we were able to find the shortest path between two nodes. The team spent countless hours on calls, going through all features and bugs to perfect.

Yuhaï Namugunga: I have searched for errors on the excel spreadsheet, I have contributed to this coursework by quality checking the excel spreadsheet on errors and omissions. This was to make sure that the provided data did not cause any errors in our program. I have also helped test the code, to ensure that there were no bugs in it. Finally, I have also helped with integrating Dijkstra's algorithm, in order to find the shortest route between two stations.

APPENDIX:

ERRORS/ OMISSIONS:

We started from analysing the excel document in order to find errors/omissions we could sort out. Firstly, we checked that the lines contained every station and that they were linked correctly. This led us to discovering some mistakes in the Metropolitan and Northern line:

1. Harrow-on-the-hill was linked to Wembley Park. (Metropolitan Line)
2. Harrow-on-the-hill was linked to Finchley Road. (Metropolitan Line)
3. Moor Park was linked to Harrow-on-the-hill. (Metropolitan Line)
4. We added the links from Moor Park to Harrow-on-the-hill, as they were missing from the original file. (Metropolitan Line)
5. Euston was linked to Camden Town. (Northern Line)

Then, we tried to run the dijkstra's algorithm through all the lines from one end to the other, and it came to our notice that the program would pick very odd paths to reach destinations. We found out that this situation was being caused due to several cells having a space at the end, which was causing the 'link_all_stations' function to not work properly, as it was having problems with matching the strings.

We handled this issue by creating a loop that iterates through the entire excel file which looked for spaces at the end of every station name. If this occurred it would print the name of the station, making it extremely easy to detect and correct it.

```
for index, row in file.iterrows():
    a = file.iloc[index][1]
    if a[-1] == " ":
        print(file.iloc[index][1])
```

This loop allowed us to thoroughly analyse the excel file and found a large number of errors in a significantly small amount of time, which would have taken a much longer time to carry out manually.
The spacing issue was found to be in the following stations:

- Central line: Epping, Loughton, Snaresbrook, Northolt
- District Line: Richmond
- Northern Line: Morden
- Circle Line: Royal oak, Westbourne Park, Ladbroke Grove,Latimer Road, Wood Green, Shepherd's bush market, Goldhawk Road, Hammersmith, Edgware Road

In some other instances, a few stations had the wrong name:

- Finsbury park on the Victoria Line is called "Finsbury Park Victoria" instead of "Finsbury Park".
- On the Piccadilly line Holborn was called "Holborn Central", and we corrected it back to "Holborn".
- On the central line "Tottenham Court Road" was called just "Tottenham", so we added in "Court Road" to it.

FRONT-END ERRORS:

With the approach of implementing a suitable GUI, we came across several bumps which formed errors that we fixed. As we were using the **PyQt5** module to integrate it, some classes were essentially foreign in usage. Our first encountered error was after having turned the **ui**. file into a **py.** file- and running the program, we found that image files were not visually present. With having set a Widget label with a path to the picture, we overcame the error by having to create a defined image path which was called when setting the **QPixmap** class. This allowed the image files to therefore be read in.

One of the most important errors that we encountered was the attempt to connect the second window's, **'Start station:'** and **'End station:' QTextEdits** which essentially allow the user to input their selected stations furthermore display it across the third window's **QLabel** as an output. We had already integrated this through the backend and therefore were required to manage to display it in the front end.

- To do this we added a self. parameter in the **'enterWindow'** function which connects the **'ENTER' QPushbutton** widget to the third window. The parameter allows us to pass the argument which includes the **start_station, end_station** and **TimeEdit** as well as the function which changes them into plain text.

When implementing the GUI, it went through countless testing phases as we continuously manually altered the functionality as well as, consistently added widgets and connections to see how they would change the front end and improve it. We also had different versions saved to make sure that errors could be found easily and overcome when knowing what implementation and improvement had caused them.

Another error that we were able to fix was the cut off display of the output. The error essentially was that when clicking enter, the output destination journey information would be displayed on the **QLabel** in the output window. However what happened here is that some of the longer destination journeys would have the information cut off as the **QLabel** was not large enough to accommodate the entire output. After testing with smaller fonts, and trialing with a Vertical scroll bar widget, we still had this persistent issue.

- Conclusively, to fix this issue we added another widget called **QScrollArea**, and added the defined **QLabel** inside the ScrollArea. With this, we also added a self. parameter in the Qlabel functionality to call the ScrollArea contents. With this, we found that when testing with shorter destinations- the set of the **QLabel** was as it was before; displaying outputs correctively. Furthermore, when inputting a longer journey, the ScrollArea would allow the output display to vertically (or horizontally if required) scroll down the contents- making sure the entire journey details and the journey summary to be displayed.

REFERENCES:

Amitabha Dey (2019). *Dijkstra's Algorithm in Python Explained*. *YouTube*. Available at: https://www.youtube.com/watch?v=Ub4-nG09PFw&t=1206s
[Date Accessed: 21 Oct. 2020].

LucidProgramming (2018). *Data Structures in Python: Doubly Linked Lists -- Append and Prepend*. *YouTube*. Available at: https://www.youtube.com/watch?v=8kptHdreaTA&t=683s.

Qt (2020).*Py for Python Documentation: QPixmap--*
Available at: https://doc.qt.io/qtforpython/PySide2/QtGui/QPixmap.html
[Date Accessed: 23 Oct. 2020]

ily Code (2018). *Python PyQt5 -4- Create your first label with QLabel and display an image on it. YouTube.*
Available at: https://youtu.be/dDCiiVRhWUc
[Date Accessed: 18 Oct. 2020]