

COMP 1828

Designing, developing and testing a journey planner model for the London Underground system.

Table of Contents

Customer How-To Guide and Any Assumptions Made	3
Simplified How-To Guide	3
Step 1.	3
Selecting Your Origin	3
Step 2.	3
Selecting Your Destination	3
Step 3.	3
Selecting Your Departure Time	3
Step 4.	3
Searching for Your Route	3
Understanding the Results	3
Software Setup Guide	4
Environment Installation and Setup	4
Self-Setup and Installation	4
Pre-Configured Installation / Real World Example	4
Software Configuration	5
Software Assumptions	5
Performance Evaluation of the Structures and Algorithms	6
Data Importing	6
Double Linked List	6
Dijkstra's Algorithm	6
Test Data Discussion and Tests Performed	7
Testing Discussion	7
Appendix 1	9
Testing Data	9
Manual Testing	9
Automated Testing	13
Designed Django Tests	13
Test Execution and Build Overview	14
Travis-CI Test Passed Summary	14
Travis-CI Log Example	14
Travis-CI GitHub Integration	14
Travis-CI Failed Test Summary	15
Travis-CI Failed Test Logs - Outdated Test	15
Travis-CI Failed Test Example	15
Appendix 2	16
Graphic Designs / Concepts	16
Homepage Design	16
Dropdown Route	17
Appendix 3	18
Code Snippets for Evaluation of the Structures and Algorithms	18
Data Importing	18
Double Linked List	18
Dijkstra's Algorithm - Sample Section	18
References & Licenses	19
References	19
Licenses	20

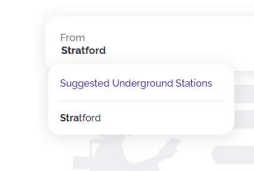
Customer How-To Guide and Any Assumptions Made

Simplified How-To Guide

Step 1.

Selecting Your Origin

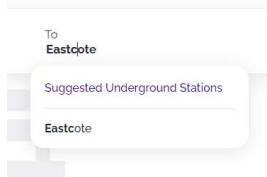
Begin by entering your origin station; the name of the underground station that you wish to begin your route. You can do so by filling out the “**From**” input box. Once you start typing, a list of suggested stations will appear; you can click one of these to automatically complete the station name. Alternatively, press **TAB** or **Enter** to auto-fill the suggestion. If your desired station isn’t recommended, you can continue typing the full station name until it appears.



Step 2.

Selecting Your Destination

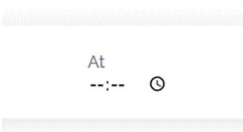
Next, please enter your destination station; the name of the underground station that you wish to end your route at. You can do so by filling out the “**To**” input box. Once you start typing, a list of suggested stations will appear; you can click one of these to auto input it into the field. Alternatively, press **TAB** or **Enter** to auto-fill the suggestion. If your desired station isn’t recommended, you can continue typing the full station name until it appears.



Step 3.

Selecting Your Departure Time

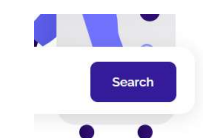
Enter your desired departure time into the “**At**” input box. It is important to specify the time of departure because the Bakerloo Line trains will be travelling between stations at twice the speed from **9am-4pm** and from **7pm-Midnight**. This is part of a new infrastructure rollout to get you to your destination as quickly as possible!



Step 4.

Searching for Your Route

Finally, click search.
It really is that simple!



Station	Line	Total Travel Time
Waterloo	Waterloo & City Line	0 mins (+0 mins)
Bank	Northern Line	5 mins (+5 mins)
Moorgate	Circle Line	8 mins (+3 mins)
Barbican	Circle Line	11 mins (+3 mins)
Farringdon	-	13 mins (+2 mins)

Journey Summary

1. Waterloo Station to Bank Station - **Waterloo & City Line**
2. Change at Bank from the **Waterloo & City Line** to the **Northern Line**
3. Bank Station to Moorgate Station - **Northern Line**
4. Change at Moorgate from the **Northern Line** to the **Circle Line**
5. Moorgate Station to Farringdon Station - **Circle Line**

Total Travel Time: 13 Minutes



Dijkstra Route Calculation (Seconds): 0.0270719528
Route Structure Formatting (Seconds): 0.0001816750

Understanding the Results

After clicking search, your calculated route will be displayed in a table. The first column is the stations on your route; these are displayed in a linear order from your origin to your destination. The second column shows the train line that you need to be on at the specified station, and the third column displays the travel time between stations (including the increase of time from the previous station). The example shown on the left shows the route from Waterloo to Farringdon that utilises three different train lines.

In the event you wish to have an overview or an explained version of your route, a journey summary will also be provided. This displays the route you are taking, any changes in train lines and the total travel time for your journey.

Software Setup Guide

Environment Installation and Setup

Self-Setup and Installation

Installation Time

~2 Minutes

Minimum Python Requirement (As Tested)

Python 3.8.0

PIP 20.1.1

Compatible Operating Systems (As Tested)

- Windows 10 Pro (Version 2004)
- Windows 10 Home (Version 2004)
- MacOS Catalina 10.15.7
- Ubuntu 20.04
- CentOS / CloudLinux 7

Software Setup & Installation Steps:

- 1) Perform installation of required software dependencies.
 - a) Execute `pip install -r requirements.txt` to install PIP requirements.
- 2) Perform automated tests to ensure the software is working as expected in your environment.
 - a) Execute `python manage.py test` to execute automated tests.
 - i. If tests fail, please ensure PIP dependencies installed correctly and the minimum system requirements are met.
- 3) Launch the application.
 - a) Execute `python manage.py runserver` to start the Django process.
- 4) Launch your preferred browser and navigate to <http://localhost:8000> to view the site.

Pre-Configured Installation / Real World Example

Installation Location

London, United Kingdom (~1ms Latency from University of Greenwich campus)

Configured Proxy Service

Cloudflare

Hosted Environment Operating System

CloudLinux 7

Hosted Environment Requirements

Python 3.8.1

PIP 20.2.4

Please Note: The code hosted in this environment will not be altered by any member of the team after past the deadline date; for transparency purposes, it is still **strongly** recommended that the self-setup version is utilised for the preliminary review. This instance of the software should only be used or referenced in the event of severe difficulties with the installation or access through the manual method.

URL: <https://comp1828.universitycourse.work/>

Software Configuration

Configuration File Description

Configuration of the software can be performed through the `route_planner_configuration.json` file; a pre-populated copy of this file will be provided alongside this submission to ensure the software is configured in compliance with the coursework specification of module COMP1828 and necessary requirements (such as Bakerloo line route speed factors) are made readily available. Alterations can be performed to this file if needed to alter the behaviour of the software. To reset the included configuration file back to “factory settings”, please delete or rename this file while the application is in a stopped state, and a fresh copy will be initialised on the next start-up procedure.

Please Note: The generation of geocoded data on first start-up after a configuration reset may take up to 30 seconds dependant on API rate limits applied by the Google Cloud Platform. A console log will be provided for you to track the progress of this process.

Configuration File Breakdown

Parameter	Type	Example	Description
route_data_file	string	<code>"Data File.xlsx"</code>	Defines the name of the .xlsx file that contains all of the route data.
route_geocoding	boolean	<code>true</code>	Enables or disables the geocoding processing. This will toggle the map visibility.
route_geocoding_api_key	string	<code>"UENEG1347ADF325"</code>	Google API Key (Maps JavaScript API, Geocoding API)
route_geocoded_data	dict	<code>"Woodside Park": [-0.185339, 51.6180026]</code>	Dictionary containing the names of the station, and a list containing the co-ordinates in the order of [Long, Lat].
route_speed_factors	dict	<code>"Bakerloo": { "applied_times": [{ "end_time": 16, "start_time": 9 }, { "end_time": 0, "start_time": 19 }], "factor": 0.5 }</code>	The dictionary can store a speed factor that is applied to a specific underground at certain times. The applied times list contains dictionary elements stating the start and end time; multiple dictionaries can be stored within the list. The factor is applied to the route in comparison to the base speed during the applied times - a 0.5 factor will reduce the times by half.
train_run_time	dict	<code>"start": 5, "end": 24</code>	Define the start and end times for the opening hours of the underground. For 24/7 operation, set start to 0 and end to 24.

Software Assumptions

- 1) Three routes failed the initial manual validation due to it being identified that there were no direct routes between the stations; Harrow-on-the-Hill to Finchley Road, Harrow-on-the-Hill to Wembley Park and Moor Park to Harrow-on-the-Hill. This validation failure has been overwritten and disregarded on the basis that TFL class the routes as valid and do run additional direct trains between the stations, as confirmed in the data provided on the 20th May 2013 in Freedom of Information request FOI-0175-1314, in addition to more recent evidence.
- 2) The project specification states that “you may assume that trains operate from 05:00 until midnight each day”; an additional assumption has been made that the times specified are to be considered as “inclusive”. This assumption allows for train routes to start at precisely 05:00 and for routes that end at 00:00 to be classified as valid, instead of terminating operation at 23:59.
- 3) It has been assumed that the route column layout provided within the specification is to be used as a reference, not a fixed structure. The format has been slightly modified to be more user friendly and to boost user experience by combining the columns, displaying the running total of the route and the difference between stations (including wait) in brackets to the right-hand side.
- 4) The use of Django has been used in the creation of the software, resulting in a breakdown of programming languages to result in Python (64%), JavaScript (18%), HTML (11%) and CSS (7%). It has been assumed that while the specification stated that it was a requirement to use Python, other languages are permitted to be used throughout the project in addition to it.
- 5) It has been assumed that due to the software being public facing, a cleaner and more user-friendly interface is more important than it being written purely in Python code. Leveraging Django that makes use of HTML, JavaScript and CSS over a Python library such as Tkinter or PyQt allows for a much nicer looking GUI and a more cleanly integrated user experience.

Performance Evaluation of the Structures and Algorithms¹

Data Importing

Importing the Excel spreadsheet given with the Underground data into a pandas dataframe. **Pandas** was preferred over **xlrd** due to the need to sanitise the given data, and as such, being able to store the excel data in a dataframe made manipulating the data more practical. The method also covers ways to sanitise the data to ensure the stability of the project. It traverses through the dataframe and checks for extra white spaces on the station and line names, to avoid entering duplicate stations or routes. It also checks for gaps in the data, such as missing names of lines or stations, and corrects them based on context.

Double Linked List

As required per coursework specifications, Each train station is stored as a node in a double-linked list in alphabetical order. The double-linked list also keeps a record of the number of stations beginning with a specific letter. The station handler structure that handles the double linked list operations, also works in a bi-directional manner, to ensure that when adding a new connection from the previously cleansed data, the association is added to both station A and B, so when calculating the routeing table using the Dijkstra's algorithm, it will work on both directions and give consistent results.

A method (**get_optimal_direction_of_travel**) was implemented to determine the best way to iterate through the list, taking advantage of the usage of a double linked list. The station_handler object keeps track of a dictionary with the first letter frequency of the stations on the list, information that will be used for determining the optimal starting point of transversion throughout the list.

This way, when trying to insert a new station or search for one in the list, if the cumulative frequency to the left of the station's beginning letter is less, then the program will traverse the double-linked list via the head; otherwise, the tail, resulting in better performance than a linear search ($O(n)$) on average, since the usage of this method will make it not needed to iterate through the whole list, bringing the actual performance close to $O(n/2)$ for the worst-case scenario. Adding all the stations accounted for less than 0.001 second, meaning that the initialisation of the front-end interface is fast and smooth, due to an exemplary algorithm implementation.

Dijkstra's Algorithm

The algorithm was implemented as the most efficient way to find the distance between two points, in this case, two different stations in the underground. For our implementation of Dijkstra's algorithm, we decided to use nested dictionaries to store the Dijkstra's table, after going through the double linked list to build the table.

The algorithm also accounts for the added time for changing train lines and for extra stations if the station is not the final destination, as mentioned in the coursework specification. Instead of adding those additional values after, it does it before the calculation, ensuring that those variables are accounted for when calculating the routeing table and finding the optimal route. Those settings can be changed on the configuration file.

As mentioned in the double-linked list section, the usage of the optimised search algorithm makes the creation of the table faster, as iterating through the list is $O(N/2)$. The time complexity is $O(E * \log(N))$, where E stands for the total number of edges, in this use case, would be the number of different stations that are only connected to another singular station, and N is the number of nodes.

It takes the algorithm $O(\log(N))$ to calculate the weight of each adjacent node, and it has to do this operation E times to calculate all the possible routes to form the routeing table, resulting in the aforementioned $O(E * \log(N))$ time complexity. From testing, calculating the route between Amersham and Stratford, two stations on far ends of the underground map, took approximately 0.01 seconds, resulting in smooth usage of the system.

¹ Appendix 3

Test Data Discussion and Tests Performed

Testing Discussion

We have carried out a range of tests to ensure that our software is both fully functional but also has a relatively decent user experience. Such tests that have been conducted include a mixture of automated tests and manual methods.

It was decided early on throughout the development process that it would be nice to have a way to detect if any alterations or additions to the core algorithms changed the response provided. Travis CI was selected to carry out this duty; a tool that integrates seamlessly into GitHub to help the development team to test code on commit or pull request. This tool works by cloning the repository at a specific point into a virtual environment to execute multiple automated Django tests crafted by the developer. Travis-CI was unified into our workflow pipeline with commits that passed the checks being authorised for a merge into the main branch and commits that failed being blocked with an electronic notification being provided. This style of testing is not a direct replacement for other testing techniques but supports developer testing by ensuring that syntax, start-up and logic errors related to the core functions are flagged should they be committed. This method can also be used to test unique use cases by generating test scenarios to verify that responses or more advanced requests are executed correctly.

At the denouement of the project, this automated testing was bulked out with an additional few checks added to ensure that any optimisations made to code made did not alter confirmed outputs. A full list of the tests used that are available at the conclusion have been included in **"Appendix 1, Section Automated Testing"**.

Manual testing has been carried out, both after the completion of the project to ensure that functionality worked as expected, but also throughout by the development team. Such examples of experiments include entering various different data types to check for unexpected outputs or unhandled errors. The testing of software that will be interacted with by a user is essential, ensuring that to the best of the current understanding, all known potential errors are handled gracefully. Automated testing can use more extensive data sets, however, may not always be the most reliable method as some problems may slip through the cracks should a specific use case not be tested against. The majority of the testing carried out is performed before a release is prepared for user testing. The developer should also be continually testing in the development environment while writing their corresponding code or algorithm; this is an essential step to ensure that they are working in the right direction. It is easy for small errors to be overlooked when working with advanced algorithms such as Dijkstra's and catching them early on in the development cycle will save significant time later on down the line.

One critical area of testing is the backend requests; when searching for a route, the application creates a GET request to pull the sanitised response from the application. This backend system will be interacted by the graphical user interface, but any request executed by the user should be deemed to be insecure and unreliable. As such, it requires additional validation to be performed to sanitise the input of these requests. It is easy for the end-user to change the GET request link to have different values and have the result printed in a new tab so is a necessity that the correct expected outputs are still being provided even when using this method and that any potential errors are handled gracefully.

Manual testing of the user interface started with the checks that the system would calculate the route correctly with valid, formatted data. Tests are required to ensure that the path that has been displayed to the user has been generated correctly with no anomalies. It is critical, in this instance, to ensure that the route is generated correctly in both directions to validate that the data has been inserted correctly; this was a significant problem during developer testing that was caught before a release version was generated whereby some routes were not connected in the opposite direction correctly.

Throughout the project, an array of different tests have been conducted on the functionality of the software, evidence of these tests can be seen in **"Appendix 1, Section Manual Testing"**. The first set of

tests were focused on the input data validation on the user interface; these tests are crucial to assure that the correct routes are being calculated efficiently for the end-user. Such tests consisted of verifying that all station names were valid and entered correctly without errors such as extra blank spaces or symbols that could interfere with the calculation of the users desired route. Test inputs such as “**B\$kerloo**” and “**West Brompton**” were inserted into the fields to check the correctness of the error handling. If an error is generated by the system, it is a requirement that is caught correctly and relays the problem gracefully to the user - this stage includes the testing of both invalid and valid data to check the system handles all inputs as expected. Alongside the validation of void data, tests were executed on blank or missing fields. An example of this is, if data is missing from the form such as the time of the departure, an error should be shown to prompt the user to enter the required information; if a manual request is made with a blank or missing parameter, the system should still gracefully display a message.

It is expected that not every test passed the first time around; if it does, you either have an inhuman developer that makes no mistakes, or your testing wasn't comprehensive or rigorous enough. Through our testing of valid error messages, we observed a Python error that wasn't handled when executing the request manually, raising a flag to allow for small tweaks to be made to increase validation on that specific request.

When the user begins to enter a station name into the input box, completed station names are automatically suggested for them. This feature must function correctly to get an accurate route calculation. The main tests conducted on this feature were ensuring that when you press “**TAB**” or “**ENTER**” while typing a station name, it automatically inserts it for you. As well as this, we needed to take into account different languages; for example, if somebody entered the station name in Arabic or Chinese, no route names should be suggested. Some examples of test data we inputted to test the languages are “**بورو**” and “**自治市鎮**” - both translate to the English word Borough. Unfortunately, due to time constraints, multi-language support has been unable to be implemented so these must be returned as invalid data.

Proceeding deeper into testing, we carried out checks on the current service restrictions in place on the underground trains. Trains run between **05:00** and **00:00**, this means that we needed to ensure that users could only view routes that could be completed within these times. This was supplemented by further checks on the timing to ensure that trains on the Bakerloo line were travelling between stations at twice the speed between **9am-4pm** and from **7pm-12am**. This was completed by selecting a route between these times and making sure that the total travel time is double the amount that it would typically be during regular hours; additional verification was required at this stage to ensure that the route calculation factored in these speed increases when selecting a route, in addition to only applying the speed increases for any section of the route within the given time frames. Similarly, we had to ensure that the correct amount of waiting time between each station was being added to the total route time, this was completed by comparing the total route time to the data in the excel spreadsheet. Wait times had to be verified as applying correctly, as a route with only a single stop shouldn't have a wait applied - as an example.

If the configuration file for the Geocoding gets corrupted or is missing upon start-up, the system will gracefully handle this by generating a new default file ready for use. Likewise, if geocoding data is missing from the file, it will communicate with the Google GeoCoding Cloud Service API and pass the result for each request to the configuration file to allow the data to be retained for future reference.

Appendix 1

Testing Data

Manual Testing

ID	Test Case	Actions Performed	Inputs	Expected Results	Actual Results	Result
User Interface Validation Testing						
1	Check to ensure that a valid station name when entered in full in the "From" box is accepted.	Type valid origin station.	"Stratford"	Input accepted with no error.	Input accepted with no error.	PASS
2	Check to ensure that an invalid station name when entered in the "From" box is marked as incorrect and a warning is presented to the user.	Type invalid origin station.	"Stretfird"	Error box appears	Error box appears	PASS
3	Check to ensure that a dropdown list of available stations is provided when a valid station name is started to be typed by the user in the "From" box.	Type partial valid origin station.	"Toot"	Dropdown with related options appears.	Dropdown with related options appears.	PASS
4	Check to ensure that a dropdown list states that there are no available stations when an invalid station is typed by the user in the "From" box.	Type partial invalid origin station.	"Asdf"	No related options appeared.	No related options appeared.	PASS
5	Check to ensure that pressing the tab character in the "From" box will allow for the recommended autocompleted word to be filled.	Tab complete valid origin station.	"Toot" + TAB	Station name is autocompleted	Station name is autocompleted	PASS
6	Check to ensure that pressing the enter character in the "From" box will allow for the recommended autocompleted word to be filled.	Enter complete valid origin station.	"Toot" + ENTER	Station name is autocompleted	Station name is autocompleted	PASS
7	Check to ensure no error message is displayed to the user if the website viewer clicks into and out of the origin station without typing in the "From" box.	Cancel input of blank origin station.	"" and Click Other Area of page.	No error appears	No error appears	PASS
8	Check to ensure a warning message is displayed to the user if they fail to complete a section or entry of a valid station name in the "From" box.	Cancel input of partial valid origin.	"Toot" and Click Other Area of page.	Error message is displayed	Error message is displayed	PASS
9	Check to ensure that a dropdown list states that there are no available stations when the user inputs extra blank spaces in the "From" box.	Tab complete station with extra blank spaces.	West Brompton (double spaced) + TAB	No related options appeared.	No related options appeared.	PASS
10	Check to ensure that a dropdown list states that there are no available stations when the user inputs special characters in the "From" box.	Tab complete station with special characters.	B\$nk + TAB	No related options appeared.	No related options appeared.	PASS
11	Check to ensure that a valid station name when entered in full in the "To" box is accepted.	Type valid origin station.	"Stratford"	Input accepted with no error.	Input accepted with no error.	PASS
12	Check to ensure that an invalid station name when entered in the "To" box is marked as incorrect and a warning is presented to the user.	Type invalid origin station.	"Stretfird"	Error box appears	Error box appears	PASS

13	Check to ensure that a dropdown list of available stations is provided when a valid station name is started to be typed by the user in the "To" box.	Type partial valid origin station.	"Toot"	Dropdown with related options appears.	Dropdown with related options appears.	PASS
14	Check to ensure that a dropdown list states that there are no available stations when an invalid station is typed by the user in the "To" box.	Type partial invalid origin station.	"Asdf"	No related options appeared.	No related options appeared.	PASS
15	Check to ensure that pressing the tab character in the "To" box will allow for the recommended autocompleted word to be filled.	Tab complete valid origin station.	"Toot" + TAB	Station name is autocompleted	Station name is autocompleted	PASS
16	Check to ensure that pressing the enter character in the "To" box will allow for the recommended autocompleted word to be filled.	Enter complete valid origin station.	"Toot" + ENTER	Station name is autocompleted	Station name is autocompleted	PASS
17	Check to ensure no error message is displayed to the user if the website viewer clicks into and out of the origin station without typing in the "To" box.	Cancel input of blank origin station.	"" and Click Other Area of page.	No error appears	No error appears	PASS
18	Check to ensure a warning message is displayed to the user if they fail to complete a section or entry of a valid station name in the "To" box.	Cancel input of partial valid origin.	"Toot" and Click Other Area of page.	Error message is displayed	Error message is displayed	PASS
19	Check to ensure that a dropdown list states that there are no available stations when the user inputs extra blank spaces in the "To" box.	Tab complete station with extra blank spaces.	West Brompton (double spaced) + TAB	No related options appeared.	No related options appeared.	PASS
20	Check to ensure that a dropdown list states that there are no available stations when the user inputs special characters in the "To" box.	Tab complete station with special characters.	B\$nk + TAB	No related options appeared.	No related options appeared.	PASS
21	Check that routes are slower from 05:00-08:59	Input Kenton to South Kenton at 08:59	Kenton South Kenton 08:59	Total travel time is 2 minutes	Total travel time is 2 minutes	PASS
22	Check that routes are faster on the bakerloo line between 09:00 and 16:00	Input Kenton to South Kenton at 09:00	Kenton South Kenton 09:00	Total travel time is 1 minute	Total travel time is 1 minute	PASS
23	Check that routes are slower on the bakerloo line between 16:01 and 18:59	Input Kenton to South Kenton at 16:01	Kenton South Kenton 16:01	Total travel time is 2 minutes	Total travel time is 2 minutes	PASS
24	Check that routes from 19:00-00:00 are faster.	Input Kenton to South Kenton at 19:00	Kenton South Kenton 19:00	Total travel time is 1 minute	Total travel time is 1 minute	PASS
25	Check to ensure that the inputted time is during opening hours (05:00-00:00)	Input time of 02:00	Bank Waterloo 02:00	Error box appears due to the time being out of open hours.	Error box appears due to the time being out of open hours.	PASS
26	Check to ensure that the route will be completed during open hours (05:00-00:00)	Input time of 23:59	Bank Waterloo 23:59	Error box appears because the route will finish after open hours.	Error box appears because the route will finish after open hours.	PASS
27	Check to ensure that an error is displayed when you click search with nothing entered into the fields.	Inputting nothing	Null	Error message is displayed.	Error message is displayed.	PASS
28	Check to ensure that an error is displayed when you click search with only the origin of the route defined.	Input only the origin	Bank	Error message is displayed	Error message is displayed	PASS
29	Check to ensure that an error is displayed when you click	Input destination only	Waterloo	Error message is displayed	Error message is displayed	PASS

	search with only the destination of the route defined.					
30	Check to ensure that an error is displayed when you click search with only the time of the route defined.	Input time only	18:00	Error message is displayed	Error message is displayed	PASS
31	Check to ensure that an error is displayed when you click search with the "From" and "To" boxes defined but not "At"	Input only Origin and Destination	Bank Waterloo	Error message is displayed	Error message is displayed	PASS
32	Check to ensure that an error is displayed when you click search with the "From" and "At" boxes defined but not "To"	Input only Origin and time	Bank 18:00	Error message is displayed	Error message is displayed	PASS
33	Check to ensure that an error is displayed when you click search with the "To" and "Time" boxes defined but not "From"	Input only Destination and Time	Waterloo 18:00	Error message is displayed	Error message is displayed	PASS
34	Check to ensure that the time between each station adds up to the total travel time.	Input all fields	Bank Waterloo 10:00	Time between each station should add to 17.	Time between each station adds to 17	PASS
35	Check to see that the total time of the route is the same when the route is reversed..	Input all fields	Waterloo Bank 10:00	Time between each station should add to 17.	Time between each station adds to 17	PASS
36	Check that each station between the origin and the destination of the route is correctly listed	Input all fields	Bank West Ham 10:00	Bank Liverpool Street Bethnal Green Mile End Stratford West Ham	Bank Liverpool Street Bethnal Green Mile End Stratford West Ham	PASS
37	Check that the journey summary tells you to change train lines correctly.	Input all fields	Bank West Ham 10:00	Change to the Jubilee Line at Stratford	Change to the Jubilee Line at Stratford	PASS
38	Check that the map shows the correct route and colors on the map.	Input all fields	Bank West Ham 10:00	The map shows the route correctly and the line colour changes from red to grey at Stratford.	The map shows the route correctly and the line colour changes from red to grey at Stratford.	PASS
39	Check that there are no wait times between stations when only travelling one stop.	Input all fields	Baker Street Great Portland Street 17:00	Total Time 2 Minutes	Total Time 2 Minutes	PASS
40	Check that there is a 1 minute wait time when travelling between 3 stations.	Input all fields	Baker Street Euston Square 17:00	Data set time 4 minutes, calculated route time 5 minutes	Data set time 4 minutes, calculated route time 5 minutes	PASS
41	Check that there are no wait times between stations when only travelling one stop with the route reversed.	Input all fields	Great Portland Baker Street Street 17:00	Total Time 2 Minutes	Total Time 2 Minutes	PASS
42	Check that there is a 1 minute wait time when travelling between 3 stations with the route reversed.	Input all fields	Euston Square Baker Street 17:00	Data set time 4 minutes, calculated route time 5 minutes	Data set time 4 minutes, calculated route time 5 minutes	PASS
43	Check that when you enter a second route that the first one clears away and the new one is displayed.	Input all fields	Euston Square Waterloo 17:00	New route will be displayed, and the old route will be cleared.	New route will be displayed, and the old route will be cleared.	PASS
44	Check that there is no error when you put the same station as the origin and the destination.	Input the same origin and destination	Bank Bank 10:00	Error should tell you to enter a different origin or destination.	Python error appears.	PASS ⁽¹⁾
45	Check that the time section doesn't let you input letters.	Input an invalid time with letters.	Bank West Brompton 10:aa	The field should not let you enter letters.	The field does not let you enter letters.	PASS
46	Check that the time section doesn't let you input symbols.	Input an invalid time with symbols.	Bank West Brompton 10:\$	The field should not let you enter symbols.	The field does not let you enter symbols.	PASS

47	Check that there are no suggestions when entering station names in Arabic. (Because Arabic write from right to left)	Input data in Arabic	بورج	There should be no suggestions	No suggestions appear	PASS ⁽²⁾
48	Check that there are no suggestions when entering station names in Chinese simplified.	Input data in Chinese	自治市镇	There should be no suggestions	No suggestions appear	PASS
49	Check that routes calculate on the mobile view of the website.	Input all fields	Euston Square Waterloo 17:00	The route should calculate as normal	The route calculates as normal.	PASS
50	Check that all content is still visible when in mobile view.	Input all fields	Euston Square Waterloo 17:00	All route data should be visible	All route data is visible.	PASS
Direct GET Request Testing						
51	Check that correct errors appear when station names are misspelled in the direct "GET" link.	Misspell a station name in the GET link	"West Brompton" Bank	Invalid message should appear	Python value error appears.	PASS ⁽¹⁾
52	Check that routes are correctly calculated when using the direct "GET" link.	Input all fields correctly.	West Brompton Bank 10:00	Route information and times should be printed onto the page.	Route information and timings printed onto the page.	PASS
53	Check that route information is not displayed when time parameters are missing from the direct "GET" link.	Input all fields except for time	West Brompton Bank	Missing parameters error should appear.	Missing parameters error appears.	PASS
54	Check that route information is not displayed when the origin parameter is missing from the direct "GET" link.	Input all fields except for origin	Bank 10:00	Missing parameters error should appear.	Missing parameters error appears.	PASS
55	Check that route information is not displayed when the destination parameter is missing from the direct "GET" link.	Input all fields except for destination	West Brompton 10:00	Missing parameters error should appear.	Missing parameters error appears.	PASS
Start-up Testing						
56	Check to ensure that the start-up configuration file is generated if missing or deleted.	Delete configuration.	N/A	Generate file based on template.	Generated file based on template.	PASS
57	Check if Geocoding data is regenerated if missing but enabled in the configuration.	Remove geocoding data.	Configuration - Geocoding Enable	Executes generation of geocoding.	Executed geocoding for all stations.	PASS

1) Initial test failed, has since been corrected and verified as passing. Actual value remains response from failure.

2) Alters the formatting of message - to review in later version.

Automated Testing

Designed Django Tests

Test Class	Class Tested	Test Name	Description	Check Type
StationHandlerTester	StationHandler	adding_stations	Executes two add_station_alphabetically commands to insert two new records into the handler. Compares against a pre-defined list that the stations have been created correctly.	assertEqual
		adding_duplicate_station	Executes add_station_alphabetically command to attempt to add the duplicate station. Checks to see if an exception is thrown on insertion attempt.	assertEqual
		get_valid_station	Executes get_station_node_by_name to fetch node (instance of the class) from the handler using a valid station name. Checks if the response is an instance of the class Station.	assertIsInstance
		get_invalid_station	Executes get_station_node_by_name to fetch a node from the handler using an invalid station name. Checks if the response is None.	assertEqual
		adding_station_connection	Executes add_station_connection to create a unidirectional connection after fetching the station nodes of two existing records. Checks to see if the exception is thrown on insertion attempt.	assertEqual
		station_counting	Calls getter to check that the station count returns the expected value of 4.	assertEqual
		optimal_direction	Executes get_optimal_direction_of_travel to fetch the direction of travel determined for the station name. Checks if direction matches the expected value of 4.	assertEqual
RoutePlannerTester	RoutePlanner	get_short_route	Executes calculate_route using the small sample data generated at the start of the test to determine the route of a simple route. Check if the hops match expected output of 2.	assertEqual
		get_long_route	Executes calculate_route using the small sample data generated at the start of the test to determine the route of a slightly longer route. Check if the hops match expected output of 2.	assertEqual
		time_differences	Executes two calculate_route commands for the same route at different times of the day, one during the Bakerloo speed up allocation. Checks to ensure that the total travel time for the tested route is not the same.	assertNotEqual
		route_time_selection_difference	Executes two calculate_route commands for the same route at different times of the day, one during the Bakerloo speed up allocation. Checks that the route chosen for the selected route is not the same (factors time speedup into calculation).	assertNotEqual

Test Execution and Build Overview

Travis-CI Test Passed Summary

CurrentBranchesBuild HistoryPull Requests> Build #264

More options

✓ main Merge pull request #41 from metallicgloss/timing-and-clean

#264 passed

Restart build

Add timings, display errors for invalid route, small UI tweaks.

Ran for 38 sec

21 hours ago

Debug build

Commit c9aa812

Compare 95841cc...c9aa812

Branch main

William Phillips

Python: 3.8

AMD64

DJANGO_VERSION=3.1.2

Travis-CI Log Example

Remove logRaw log

3.8 is not installed; attempting download

1 worker information

6

7 Build system information

161

162

163 Downloading archive: https://storage.googleapis.com/travis-ci-language-archives/python/binaries/ubuntu/16.04/x86_64/python-3.8.tar.bz2

164 \$ curl -sSf --retry 5 -o python-3.8.tar.bz2 \${archive_url}

165 \$ sudo tar xjf python-3.8.tar.bz2 --directory /

166

167 Installing SSH key from: default repository key

168 Using /home/travis/.netrc to clone repository.

170

171 \$ git clone --depth=50 --branch=main https://github.com/metallicgloss/COMP1828-Coursework.git

181

182

183 Setting environment variables from .travis.yml

184 \$ export DJANGO_VERSION=3.1.2

185

186 \$ source ~/virtualenv/python3.8/bin/activate

187 \$ python --version

188 Python 3.8.0

189 \$ pip --version

190 pip 19.3 from /home/travis/virtualenv/python3.8.0/lib/python3.8/site-packages/pip (python 3.8)

191 \$ pip install -r requirements.txt

236 \$ python manage.py test

237 -----

238 Program Initialisation In Progress...

239 -----

240 Creating test database for alias 'default'...

241 System check identified no issues (0 silenced).

242 Method: test_get_long_route.

243 .Method: test_get_short_route.

244 .Method: test_route_time_selection_difference.

245 .Method: test_get_long_route.

246 .Method: test_adding_duplicate_station.

247 .Method: test_adding_station_connection.

248 .Method: test_adding_stations.

249 .Method: test_get_valid_station.

250 .Method: test_get_valid_station.

251 .Method: test_optimal_direction.

252 .Method: test_station_counting.

253 .

254 -----

255 Ran 11 tests in 0.008s

256

257 OK

258 Destroying test database for alias 'default'...

259 The command "python manage.py test" exited with 0.

260

261

262 Done. Your build exited with 0.

Travis-CI GitHub Integration

Add timings, alter display of errors.

metallicgloss requested review from AdnanT-ADN, It-is-Dan and gabrielnetz 21 hours ago

Test correction.

It-is-Dan approved these changes 21 hours ago

metallicgloss merged commit c9aa812 into

335e0f2

ba664ed

All checks have passed
2 successful checks

Travis CI - Branch — Build Passed
Details
4w changes

Travis CI - Pull Request — Build Passed
Details

Revert

Travis-CI Failed Test Summary

✖ timing-and-cleanup Add timings, alter display of errors.

🔗 Commit 335e0f2

🔗 Compare a3b912e...335e0f2

🔗 Branch timing-and-cleanup

#260 failed

🕒 Ran for 42 sec

📅 21 hours ago

🔄 Restart build

🔍 Debug build

👤 William Phillips

🖥️ Python: 3.8

🏠 AMD64

📦 DJANGO_VERSION=3.1.2

Travis-CI Failed Test Logs - Outdated Test

```
237 -----
238 Program Initialisation In Progress...
239 -----
240 creating test database for alias 'default'...
241 System check identified no issues (0 silenced).
242 EMethod: test_adding_duplicate_station.
243 .Method: test_adding_station_connection.
244 .Method: test_adding_stations.
245 .Method: test_get_valid_station.
246 .Method: test_get_valid_station.
247 .Method: test_optimal_direction.
248 .Method: test_station_counting.
249 .
250 =====
251 ERROR: setupclass (underground_route_planner.tests.test_models.RoutePlannerTester)
252 -----
253 Traceback (most recent call last):
254   File "/home/travis/virtualenv/python3.8.0/lib/python3.8/site-packages/django/test/testcases.py", line 1123, in setupClass
255     cls.setUpTestData()
256   File "/home/travis/build/metallicgloss/COMP1828-Coursework/underground_route_planner/tests/test_models.py", line 245, in
257     setUpTestData
258     self.route_planner = RoutePlanner(
259   TypeError: __init__() missing 1 required positional argument: 'train_run_times'
260 -----
261 Ran 7 tests in 0.007s
262
263 FAILED (errors=1)
264 Destroying test database for alias 'default'...
265 The command "python manage.py test" exited with 1.
266
```

Travis-CI Failed Test Example

```
228 =====
229 ERROR: underground_route_planner.tests (unittest.loader._FailedTest)
230 -----
231 ImportError: Failed to import test module: underground_route_planner.tests
232 Traceback (most recent call last):
233   File "/opt/python/3.8.0/lib/python3.8/unittest/loader.py", line 470, in _find_test_path
234     package = self._get_module_from_name(name)
235   File "/opt/python/3.8.0/lib/python3.8/unittest/loader.py", line 377, in _get_module_from_name
236     _import_(name)
237   File "/home/travis/build/metallicgloss/COMP1828-Coursework/underground_route_planner/tests/__init__.py", line 1, in <module>
238     from test_models import *
239   File "/home/travis/build/metallicgloss/COMP1828-Coursework/underground_route_planner/tests/test_models.py", line 3, in
240     <module>
241     from underground_route_planner.models.route_planner import RoutePlanner
242   File "/home/travis/build/metallicgloss/COMP1828-Coursework/underground_route_planner/models/route_planner.py", line 1, in
243     <module>
244     from station_handler import StationHandler
245   ModuleNotFoundError: No module named 'station_handler'
246
247 -----
248 Ran 1 test in 0.000s
249
250 FAILED (errors=1)
251 The command "python manage.py test" exited with 1.
252
```

Appendix 2

Graphic Designs / Concepts

Homepage Design

TFL Route Planner

AboutContactHelp

Lorem is my Ipsum.
It really do be like that.

From
West Brompton

To
Fulham Broadway

At
18:30

Search

Worry less, travel more.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Viverra elit in etiam auctor facilisis. Ultrices nunc egestas ut consectetur imperdiet vulputate auctor.

Worry less, travel more.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Viverra elit in etiam auctor facilisis. Ultrices nunc egestas ut consectetur imperdiet vulputate auctor.

Dropdown Route

From


West Brompton ▾

To

Fulham Broadway ▾

At

18:30



Search

Station	Line	Total Travel Time
Station one	Bakerloo	0 min
Station two	Bakerloo	2 min
Station three	Hammersmith & City	4 min
Station four	Hammersmith & City	6 min
Station one	Bakerloo	8 min
Station two	Bakerloo	10 min
Station three	Hammersmith & City	12 min
Station four	Hammersmith & City	14 min

Journey Summary

Bakerloo: Station one to Station two

Hammersmith & City: Station three to station four

Total time: 12 minutes

NB: The reference designs provided above included GT Haptik font; this has since been replaced with **Montserrat** in our final version of software due to restrictions in the license owned.

Appendix 3

Code Snippets for Evaluation of the Structures and Algorithms

Data Importing

```
# Import provided station data.
def import_station_data(self):
    # Reading the Excel Sheet, and renaming the DataFrame Columns.
    data = pd.read_excel(
        os.path.abspath(self._software_configuration['route_data_file'])
    )
    data.columns = ["Line", "Origin", "Destination", "Time"]
```

Double Linked List

```
def add_station_connection(self, station_node: object, time_taken: int, train_line: str):
    station_name = station_node.station_name

    if station_name in self._connected_stations.keys():
        # Check connection does not already exist
        if self._connected_stations[station_name]["TRAIN_LINE"] == train_line and \
            self._connected_stations[station_name]["STATION_NODE"] == station_node:
            # Connection already exists.
            raise exception (
                "Attempted to create a duplicate connection."
            )
```

Dijkstra's Algorithm - Sample Section

```
# Calculate Dijkstra table
starttime = time.time()
while len(remaining_stations) != 0:
    not_none_stations = sorted(
        not_none_stations, key=lambda x: self._route_calculator[x]["SHORTEST_TIME"]
    )
    if not_none_stations == []:
        none_type_stations = list(
            filter(
                lambda x: self._route_calculator[x]["SHORTEST_TIME"] is None, remaining_stations
            )
        )
        current_station_name = (
            not_none_stations + none_type_stations
        )[0]
    else:
        current_station_name = not_none_stations[0]

    # Freeing no longer needed memory
    del not_none_stations

    # Cycle Dijkstra Algorithm
    current_station = self._station_handler.get_station_node_by_name(
        current_station_name
    )
    current_station_shortest_time = self._route_calculator[
        current_station_name
    ]["SHORTEST_TIME"]

    current_time_in_minutes = self._route_calculator[
        current_station_name]["TIME_REACHED_STATION"]
```

References & Licenses

References

Brown, H., 2009. *How do I check that multiple keys are in a dict in a single pass?*. [Online]
Available at: <https://stackoverflow.com/questions/1285911/how-do-i-check-that-multiple-keys-are-in-a-dict-in-a-single-pass>
[Accessed 19 11 2020].

Django, 2020. *Managing static files (e.g. images, JavaScript, CSS)*. [Online]
Available at: <https://docs.djangoproject.com/en/3.1/howto/static-files/>
[Accessed 20 10 2020].

Echo, 2016. *Getting 404 for all static files during WSGI setup with django*. [Online]
Available at: <https://stackoverflow.com/questions/33168308/getting-404-for-all-static-files-during-wsgi-setup-with-django/45558652>
[Accessed 15 11 2020].

GetBootstrap Team, n.d. *Bootstrap Tables.*. [Online]
Available at: <https://getbootstrap.com/docs/4.1/content/tables/>
[Accessed 02 11 2020].

Google Developers, 2020. *Client Libraries for Google Maps Web Services*. [Online]
Available at: <https://developers.google.com/maps/documentation/geocoding/client-library>
[Accessed 23 10 2020].

Google Developers, 2020. *GeoCoding API*. [Online]
Available at: <https://developers.google.com/maps/documentation/geocoding/start>
[Accessed 24 10 2020].

Google Developers, 2020. *GeoCoding API - Get Key*. [Online]
Available at: <https://developers.google.com/maps/documentation/geocoding/get-api-key>
[Accessed 23 10 2020].

Google Developers, 2020. *Styling your map*. [Online]
Available at: <https://developers.google.com/maps/documentation/javascript/styling>
[Accessed 23 10 2020].

Morris, D., 2015. *Typeahead custom template Without Handlebars*. [Online]
Available at: <https://stackoverflow.com/questions/29223516/typeahead-custom-template-without-handlebars>
[Accessed 25 10 2020].

pandas development team, 2020. *IO tools (text, CSV, HDF5, ...)*. [Online]
Available at: https://pandas.pydata.org/docs/user_guide/io.html
[Accessed 26 10 2020].

Sambol, M., 2014. *Dijkstra's algorithm in 3 minutes — Review and example*. [Online]
Available at: https://www.youtube.com/watch?v=_IHSawdgXpl
[Accessed 25 10 2020].

Shahbaz, 2014. *Understanding Time complexity calculation for Dijkstra Algorithm*. [Online]
Available at: <https://stackoverflow.com/questions/26547816/understanding-time-complexity-calculation-for-dijkstra-algorithm>
[Accessed 19 11 2020].

Licenses

Django

Version: 1.0

URL: <https://github.com/django/django/>

Copyright: © 2013-2020 Django Software Foundation and other contributors.

(<https://github.com/django/django/graphs/contributors>)

Bootstrap

Version: 4.5.2

License: MIT

URL: <https://github.com/twbs/bootstrap/>

Copyright: © 2011-2020 The Bootstrap Authors (<https://github.com/twbs/bootstrap/graphs/contributors>)

jQuery

Version: 3.5.1

License: MIT

URL: <https://github.com/jquery/jquery>

Copyright: © 2007-2020 JS Foundation and other contributors.

typeahead.js

Version: 1.3.1

License: MIT

URL: <https://github.com/corejavascript/typeahead.js>

Copyright: © 2013-2020 Twitter, Inc. and other contributors.

Google Fonts (Raleway)

Version: 1.1 Update 5

License: SIL Open Font License (OFL)

URL: <https://fonts.google.com/specimen/Raleway>

Copyright: © 2012-2020 Matt McInerney and other contributors.