**Evaluation:**

Objectives Met – Section 19:

Images referenced in the table are found in the pages following the table.

|  |  |  |  |
| --- | --- | --- | --- |
| Objective Number | Objective description | Evidence of Objective Being Met | Comments |
| 1 | The program will be a website which will find the optimal route from one tube station to another. | Image 1 | The CGI script in the file called “journeyplanner.pyw” completes this objective. When the form corresponding to this CGI script is run, Dijkstra’s algorithm is run on the input the user provides. This results in the optimal route between the 2 stations being displayed to the user. |
| 1.1 | The website should allow you to input your start location and end location with a dropdown menu. | Images 2, 3 | The HTML file called “journeyplanner.html” completes this objective. The “select” element is used and “option” tags provide the options which the user can input (these are the station names). The stations which may be used for the step-free journey planner include “Step-free” in the options so that the user knows which stations can be used in the step-free journey planner. |
| 1.2 | The website should find the optimal route based on the time taken to get between each station. | Image 4 | The CGI script in the file called “journeyplanner.pyw” uses a function called “dijkstra”. This function uses Dijkstra’s shortest path algorithm which finds the optimal route between 2 stations. It uses the values in the adjacency dictionary called “stations\_adjacency\_dict” (which are timings between stations). This achieves the objective. |
| 1.3 | The website should allow for a 5-minute change over to another line and the times between each station will be measured at an off-peak time, recorded, and inputted in the program. | Images 5, 6 | The CGI script in the file called “journeyplanner.pyw” has a dictionary called “stations\_adjacency\_dict”. Keys are stations and values are more dictionaries. These include keys which are stations you travel to and values which are the times between the stations. Substations are stations but with specific lines. To get between substations of a certain station, a 5-minute changeover time is added using the stations adjacency dictionary. For example, to get to Paddington Elizabeth line to Paddington District line, it takes 5 minutes. This achieves the objective. |
| 1.4 | The website should only focus on zone 1 of the London Underground, as stations outside of zone 1 have less routes that can be taken and, therefore, an optimal route may be the only route. | Images 5, 6 | The CGI script in the file called “journeyplanner.pyw” has a dictionary called “stations\_adjacency\_dict”. This includes information on the timings between the stations in zone 1 which can be traversed. This achieves the objective. |
| 1.5 | The website should display which lines you need to use in order to travel to your destination. | Image 1 | The CGI script in the file called “journeyplanner.pyw” completes this objective. When the form corresponding to this CGI script is run, Dijsktra’s algorithm is performed on the inputs and the route is output. This route includes the lines as the adjacency dictionary used for Dijsktra’s algorithm already contains the line names. |
| 2 | The program should use Dijkstra’s algorithm in order to find the shortest path between two tube stations. | Image 4 | The CGI script in the file called “journeyplanner.pyw” uses a function called “dijkstra”. This function uses Dijkstra’s shortest path algorithm which finds the optimal route between 2 stations. The achieves the objective. |
| 2.1 | The program should use a graph which is abstracted (only has the necessary information, such as station name, connections between stations, and times taken between each station) in order to represent the problem. | Images 5, 6 | The CGI script in the file called “journeyplanner.pyw” has an adjacency dictionary. This adjacency dictionary represents the graph which shows the connections between different stations (the graph below):  A picture containing chart  Description automatically generatedThis achieves the objective. |
| 2.2 | The program should represent the graph using an adjacency list (of some sort) in order for the problem to be solved. | Images 5, 6 | Objective achieved in the same way as objective 2.1. |
| 3 | The website should have multiple webpages which can be navigated by the user. |  | The HTML pages are linked together using a navbar which can be navigated by the user. For example, if the user wants to use the journey planner, they can use it by clicking on “journey planner” in the navbar. Therefore, the multiple HTML pages (and the CGI scripts corresponding to these) achieve this objective. |
| 3.1 | These webpages will be accessible from the navbar. | Image 7 | Each HTML file contains a navbar div which allows the user to access different HTML pages. This achieves the objective. |
| 3.2 | The webpages will include a homepage (a page with information on the website’s functionality and background), a map webpage (page where you can view the tube map), a journey planner webpage (a page where you input your tube stations and the program gives you the optimal route), a status updates webpage (a page where you can view status updates of each line), an arrival times webpage (a page where you can view the arrival times at a specified station), and a lift updates webpage (a page where you can see if there are any lifts which are not working at a specified station). | Image 8 | This objective is achieved by the HTML files:   * index.html – This is used as a homepage/information page. * map.html – This is used to display the tube map to the user. * journyeplanner.html – This is used to create a route for the user when they provide inputs. * updates.html – This is used to show the different status updates of a given line. * arrivaltimes.html – This is used to display the arrival times of trains at a given station. * liftupdates.html – This is used to display the lift disruptions (if any) at a given station. |
| 3.3 | The status updates, arrival times, and lift updates webpages will use the TfL API in order to inform the user of the information they have requested. | Image 9 | The python file called “tubeAPI.py” is imported into all python files which need to use the TfL API. It contains a class which has methods that can be used to access information from the TfL API. |
| 4 | The website should have a user interface which is easy to navigate. | Images 10, 11­ | The CSS file called “styles.css” is used to style all of the webpages. It is also used in the CGI scripts for when a form is submitted. This ensures that all of the webpages have consistent styling. The styling uses black, white, and grey in order to be easy to navigate. This achieves the objective. |
| 4.1 | The website should have a navbar, a header, and a footer. | Images 10, 11­, 12, 13 | Each HTML file contains a header, navbar, and footer div. These are all styled in the same way using the “styles.css” stylesheet. |
| 4.2 | The website should use CSS for styling each section. | Images 10, 11­ | Objective achieved in the same way as objective 4. |
| 4.3 | The website should have a drop-down menu to show each station when inputting the start and end locations in the journey planner to ensure that the user input is valid. | Images 2, 3 | Objective achieved in the same way as objective 1.1. This ensures that the user does not input any invalid stations, as the stations are already coded in for the user. |
| 5 | The program should have an option to adjust journey times based on what time of day the user is using it. Then, the program should output the estimated time taken to get to their desired destination from their start location. | Image 14 | The HTML file called “journeyplanner.html” has a form which includes an option to select “Yes” or “No” to the questions “Rush Hour?” and “Weekend?”. Depending on the inputs, the CGI script corresponding to this HTML file (journeyplanner.pyw) changes timings between stations. This achieves the objective. |
| 5.1 | If it is a rush hour, then all of the times should be multiplied by 2. | Image 15 | The python file called “journeyplanner.pyw”. The code runs the “djikstra” function and this iterates through the timings in the “stations\_adjacency\_dcit” and multiplies the values based on the user inputs for rush hour and the weekend. |
| 5.2 | If it is a weekend, then all of the times should be multiplied by 1.5. | Image 15 | Objective achieved in the same way as objective 5.1. |
| 5.3 | If it is a rush hour on a weekend, then all of the times should be multiplied by 2.5. | Image 15 | Objective achieved in the same way as objective 5.1. |
| 6 | The program should explain how the website can be used and what it can be used for. | Image 16 | The HTML file called “index.html” is used to display information about the tube, the website, etc. This allows the user to understand what the purpose of the website is and how they can use it. This achieves the objective. |
| 6.1 | In the current systems, the journey planners are quite complex and may be hard to understand for some users. Therefore, my program should have a clear description for how the program can be used and should not have any unnecessary or complex details such as texts in the way of anything. | Image 16, 10, 11 | The HTML page called “index.html” shows the information about the webpage clearly and uses the styling from the “styles.css” stylesheet. It includes clear instructions on how the program can be used. There is an image which does not cover any information or confuse the user. This achieves the objective. |
| 6.2 | The user interface should not be too confusing or overwhelming: it should be based on simplicity so that many different users are able to use it with ease. | Images 10, 11 | Objective achieved in the same way as objective 4. Although this may be subjective, the project uses minimal complexity in the sense that the styling uses only 3 colours (black, white, and grey) and the user does not need to concern themselves with how the program works. |
| 7 | The program should have a normal journey planner (including all stations in zone 1), and a step-free journey planner (with only step free stations in zone 1). | Images 5, 6 | The CGI script in the file called “journeyplanner.pyw” has 2 classes. One is called “JourneyPlanner” and the other is called “StepFreeJourneyPlanner”. The main difference between them is the adjacency dictionary that they use. In the normal journey planner, the adjacency dictionary contains stations which are in zone 1, but the step free journey planner has an adjacency dictionary with stations in zone 1 that can be traversed for a step free route. |
| 7.1 | The journey planner should have an option on the form which asks if they would like to use the step-free journey planner or not. | Image 17 | The HTML file called “journeyplanner.html” has a form which includes an option to select “Yes” or “No” to the question “Step free?”. Depending on the inputs, the CGI script corresponding to this HTML file (journeyplanner.pyw) finds the route using different classes (“JourneyPlanner” or “StepFreeJourneyPlanner”). |
| 7.2 | The step-free journey planner should allow for additional time while getting from the street to the platform and the platform to the street. | Images 5, 6 (these are snippets of the whole stations adjacency dictionaries. | The CGI script in the file called “journeyplanner.pyw” has 2 adjacency dictionaries (one for normal journeys and one for step free journeys). The step free adjacency dictionary contains an additional 5 minute change-over time when travelling from the underground to the street. The normal journey planner does not have this. |
| 8 | The program should output the route for the user in a way which is easy to understand. | Image 1, 18 | The “output\_everything” function in the python file called “journeyplanner.pyw” outputs each station (and the line used to reach the station) in the route on separate lines. This achieves the objective. |
| 8.1 | The outputs should include the station name and the line used. The line used must be included because otherwise, the user will be left to figure out the line they need by themselves which does not fulfil the needs of the project, as people using the tube are usually in a hurry and don’t have time to check the map. | Image 1, 18 | Objective achieved in the same way as objective 8. |
| 8.2 | The outputs should be formatted as below:  *Station name (line used) ->*  *Next station (line used) -> etc.* | Image 1, 18, 19 | The CGI script “journeyplanner.pyw” achieves this objective. The “output\_everything” function is used to output the route in html so that the user can see the route. The line name is incorporated into the strings by using the “correct\_lines” function. |
| 9 | The program should effectively output the TfL API data so that it is easy for the user to understand. | Images 20, 21, 22 | The python files called “arrivaltimes.pyw”, “liftdisruptions.pyw”, and “status\_updates.pyw” use the TfL API. The CGI scripts use for loops in order to output information efficiently. This achieves the objective. |
| 9.1 | The arrivals webpage should display each arrival in the order in which it is received, and it should display the line which is used, which direction the train will be going, and the time at which it will arrive. | Image 20 | Objective achieved in the same way as objective 9. |
| 9.2 | The status updates webpage should display an image of the stations on the line, along with the information received from the TfL API in the correct order it was received. | Image 21 | The CGI script called “status\_updates” outputs the API data for the user to see. Then, based on the line the user has inputted, an image of the line is outputted using HTML tags. This achieves the objective. |
| 10 | The program should use CGI in order for the client to communicate with the server and access the required content. | Image 23, 24 | The python files in the “cgi-bin” folder use CGI when the forms are submitted by the user. This achieves the objective. |
| 10.1 | The program should have a python web server file which is used to access the service. | Image 24 | The python file called “webserver.py” achieves this objective. The file is used to create a webserver for the website to run. |

Graphical user interface, text, application

Description automatically generatedImages:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated1.) 2.) 3.)

Background pattern

Description automatically generated with low confidence4.)

Text, letter

Description automatically generated5.)

Text, letter

Description automatically generated

6.)

Text, letter

Description automatically generated7.)

Graphical user interface

Description automatically generated8.)

Graphical user interface, text, application, email

Description automatically generated9.)

Text

Description automatically generatedText

Description automatically generated10.) 11.)

Text, letter

Description automatically generatedText

Description automatically generated12.)

13.)

Graphical user interface, text, application, email

Description automatically generated14.)

Text

Description automatically generated with medium confidence15.)

Graphical user interface, text, application, email

Description automatically generated16.)

Graphical user interface, text

Description automatically generated17.)

Text

Description automatically generated18.)

A screenshot of a computer

Description automatically generated19.)

Text

Description automatically generated20.)

A red and green text

Description automatically generated21.)

Text

Description automatically generated22.)

Graphical user interface, application

Description automatically generated23.)

Graphical user interface, text

Description automatically generated with medium confidence24.)

Interview with End-User and Feedback – Section 20:

The end-user for this interview is the same end-user in the interview from section 6 (Maryum Fahim). The end-user was not living in London at the time of the initial interview (in section 6), but at the time of this interview, the end-user was living in London. The end-user was shown the website and all of its functionality and then asked the following questions:

1. **Does the program achieve objective 4 (“the website should have a user interface which is easy to navigate”)?**

End-User: “Yes I was able to use the program without any difficulties. The navbar is useful for finding different webpages and it uses clear text.”

1. **Does the program achieve objective 8 (“the program should output the route for the user in a way in which it is easy to understand”)?**

End-User: “I think that it is easy to understand but it could be made easier to understand by combining the different stops and lines, for example, if someone is going from Bond Street to Queensway, instead of showing all of the stops in between, it could just say to get the central line Westbound train. I think that this objective was achieved, but could be improved so that it is achieved more clearly.”

1. **Does the program achieve objective 9 (“the program should effectively output the TfL API data so that it is easy for the user to understand”)?**

End-User: “Yeah, that is very easy to understand as it is output in a neat format. Even if the user misspells a station, the TfL data may still be outputted, which is a good feature.”

1. **What is the best feature of the website, in your opinion?**

End-User: “The best feature in my opinion is the status updates page. I like it because it shows an image of the line you have entered and gives you information on any delays and disruptions, which is useful information for someone travelling on the tube.”

1. **Are there any improvements that could be made?**

End-User: “As I said earlier, the route could be outputted in a more understandable way as it shows all the stations in between but it could be changed to only show the stations you are getting off at or changing lines at.”

1. **Does the website fulfil expectations?**

End-User: “Yes, the program meets the expectations as it can be used to plan journeys, look at status updates, and arrival times. The website has many purposes, so it could be useful for lots of different types of people.”

1. **Would you use this website in London if you needed help for your tube journey route or to see TfL API data?**

End-User: “If I was going to a station I didn’t know how to get to and was far away from me, I think I would use the journey planner. I would probably use the TfL API data if there were strikes in order to determine if I should use the tube or another mode of transport, or when it is a busy time, like at 6pm for example.”

Response to User Feedback – Section 21:

The main improvement that the end-user suggested was that the route that is output could be displayed in a way that involves only the stations where the user needs to get off of the tube. For example, the following outputted route could be outputted differently:

A screenshot of a computer

Description automatically generated

Instead, the output could be “Marble Arch – Central Line Eastbound to Liverpool Street, 8 stops away”. This would be helpful as the user would be less overwhelmed with the information being displayed to them. The program could be adjusted for this by checking if the stations in a route are on the same line, and if they are, only record the start station and end station and line. Repeating this as many times as necessary in order to get the route outputted in a more concise way. Therefore, if I had more time, I would change the code to account for this change. I believe that this change would genuinely help the users interact with the website since people who may not have used the tube before are potential end-users may be confused by the route that is outputted in the above image.

The end-user reported that the TfL API data outputted after the forms are submitted are very useful. The status updates page and the image of the line shown after the form is submitted was noted to be extremely useful and the end-user said that the information was outputted in a clear, concise way. This is advantageous as potential end-users of the website would be in a rush and they may not have enough time to read lots of text. Making sure information is presented in a straightforward, precise way is important for creating programs. I could improve the program based on this feedback by using more APIs from the TfL website and I would display more images (where necessary) alongside these.

Finally, the end-user reported that the objectives (which are based on the styling of the website and are the subjective opinions of the user) were met. They said that the way that the content is outputted to the user is good as it is easy for the user to understand. Therefore, the objectives 4, 8, and 9 were met (according to this end-user). Other potential end-users may not agree with this, but I believe that, on the whole, they are met to a good extent.

Improvements I Would Make – Section 22:

More Stations:

If I had more time to complete the project, I would include all stations in the London Underground and all zones. I think it would be more beneficial for the users if they were able to access all zones in London instead of being limited to only zone 1. Having more zones included in the journey planner would ensure that more lines can be used, and more journeys can be planned. The journey planner would need to be adjusted so that the stations adjacency dictionaries have more stations and the nodes that you can travel to from these stations would need to be added. Additionally, the time taken to get between the stations would need to be added to the station’s adjacency dictionary after the times between each station is recorded. The times would need to be collected on a weekday, and not at rush hour so that the times for the new stations are not anomalies.

More Efficient Algorithms:

The journey planner used Dijkstra’s algorithm in order to determine the shortest path between 2 stations. This has a time complexity of O (n2) as it uses a nested for loop. It can be argued that this is inefficient as the time taken for the algorithm to run increases very quickly as the size of inputs increases (i.e., if there are more stations, then the time taken to calculate a route increases). At the moment, this may not be a problem as the program runs very quickly, however, if more stations are added to the journey planner, this may become problematic. Therefore, other shortest path algorithms could be explored as an improvement for the project. For example, the A\* search algorithm can be used to find the shortest route between 2 nodes.

A\* Algorithm Vs Dijkstra’s Algorithm:

* Dijkstra’s algorithm calculates the shortest path between the start and end node but also between the start node and all other nodes in the process of doing so.
* The A\* algorithm only calculates the shortest route between the start node and end node, not all of the other nodes as well.

|  |  |
| --- | --- |
| A\* Algorithm | Dijkstra’s Algorithm |
| More time efficient as it only determines a route between the start and end nodes. | Less time efficient as it calculates the route between the start node and all other nodes too. |
| May be more memory efficient as it does not need to store the routes between the start node and all other nodes, only stores route between start and end nodes. | May be less memory efficient (space complexity) as it stores the distances between the start node and all other nodes, how to get to each node, etc. |
| Requires a heuristic function which means that the solution may not be optimal. | Gives the optimal solution by calculating all possible routes. |
| Requires a heuristic function which may not be easy to select. | Follows 4 steps so the algorithm is standard each time. |

(Information obtained from <https://isaaccomputerscience.org/concepts/dsa_search_a_star?examBoard=all&stage=all> )

Therefore, if I had more time to complete the project, I would explore other algorithms which can be used for the journey planner and determine which one is the best for this project.

More APIs:

Another improvement which could be made to the program is the addition of more APIs from the TfL website. There are a range of other APIs which can give lots of information to the user, such as crowding levels in different TfL stations, getting all valid routes for all lines, getting the timetable for a specified station, etc. These additions to the program would be very useful for users as they can provide them with information they may want to know before their journey. This would require more HTML pages to be created and CGI scripts which correspond to them. Furthermore, the API could interact with the journey planner, for example, if a specific line had lots of disruptions, then the journey planner could use this information to determine a route which avoids these delays. This would be very useful as the user would be given a route that is the optimal route for a certain time in the day.

Improved Timings for Weekends and Rush Hour:

At the moment, the program accounts for weekend and rush hour journeys by multiplying all of the times by a scalar. This may not be as accurate as it could be, therefore, more data could be collected for these times in order to make the journey planner times more realistic for certain journeys. Then, based on the user input (weekend and rush hour values), the program would use different adjacency dictionaries with different values to determine the optimal route. Since London is a fast-paced, large city, the implementation of more accurate times for rush hour and weekends may greatly benefit users of the program. Although, it may take a long time to collect the data and input it into the program.