RailVentures

By Robbert Sinclair

Introduction

My project is going to be a website that will help people who want to have a day out, but they don't know where they will want to go. I wanted to develop an app which can take the user's budget for a train ticket and shows them the places that they can go for their money. The app could also take the user's time that they want to travel and will show where the person can go for the time they have. The app will take data from APIs to show users where they could go in the UK for the money that they have.

This app was inspired by the fact that I wanted to go somewhere brand new, but I didn't know where I could go with the money I had. This app will help inspire people like myself to go have an adventure to a different part of the country that they would never had considered going to.

For this app to be successful, I will need to make sure that the app can give an overview of the places that the user can travel to. This app will encourage people to go off the beaten track and go to places where they would never have thought of going. This can help small towns get more tourist traffic which can improve the local economy.

Existing Projects

Kayak

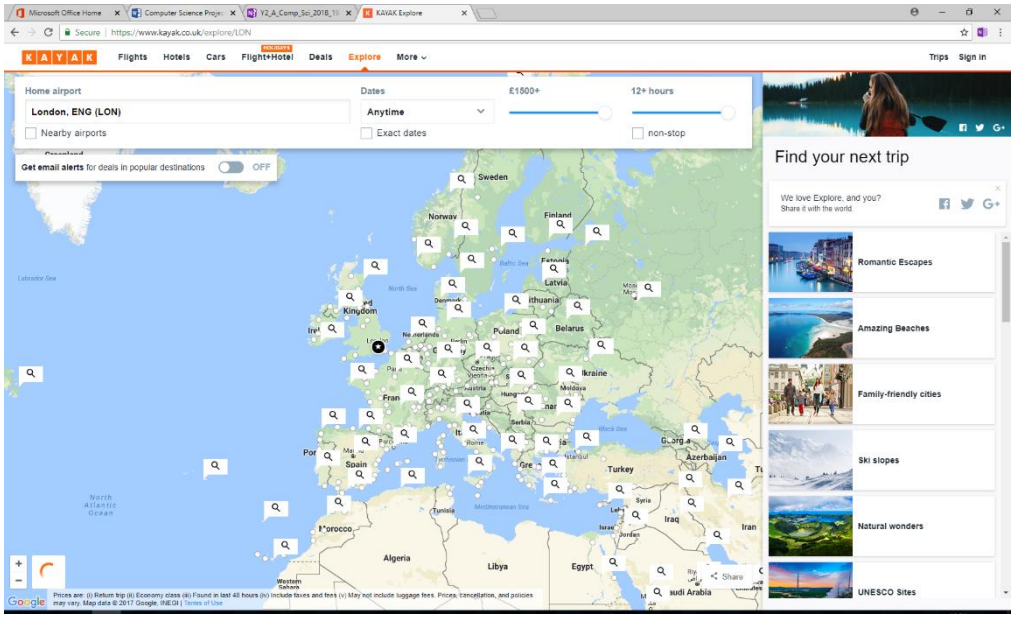


Figure 1:The first screen you get to on the Kayak discover tab

Kayak is a very popular site among people who want to get the cheapest flight possible. The great thing about Kayak is that the interface is very easy to use.

You simply put in your local airport and then the algorithm will show you the places that you can go to for the price that you have specified, and the time taken to get to the destination. There is also a checkbox which can get nonstop flights. I also like the function to the right-hand side of the screen where you can filter out the cities based on what you want to see when you are at the destination.

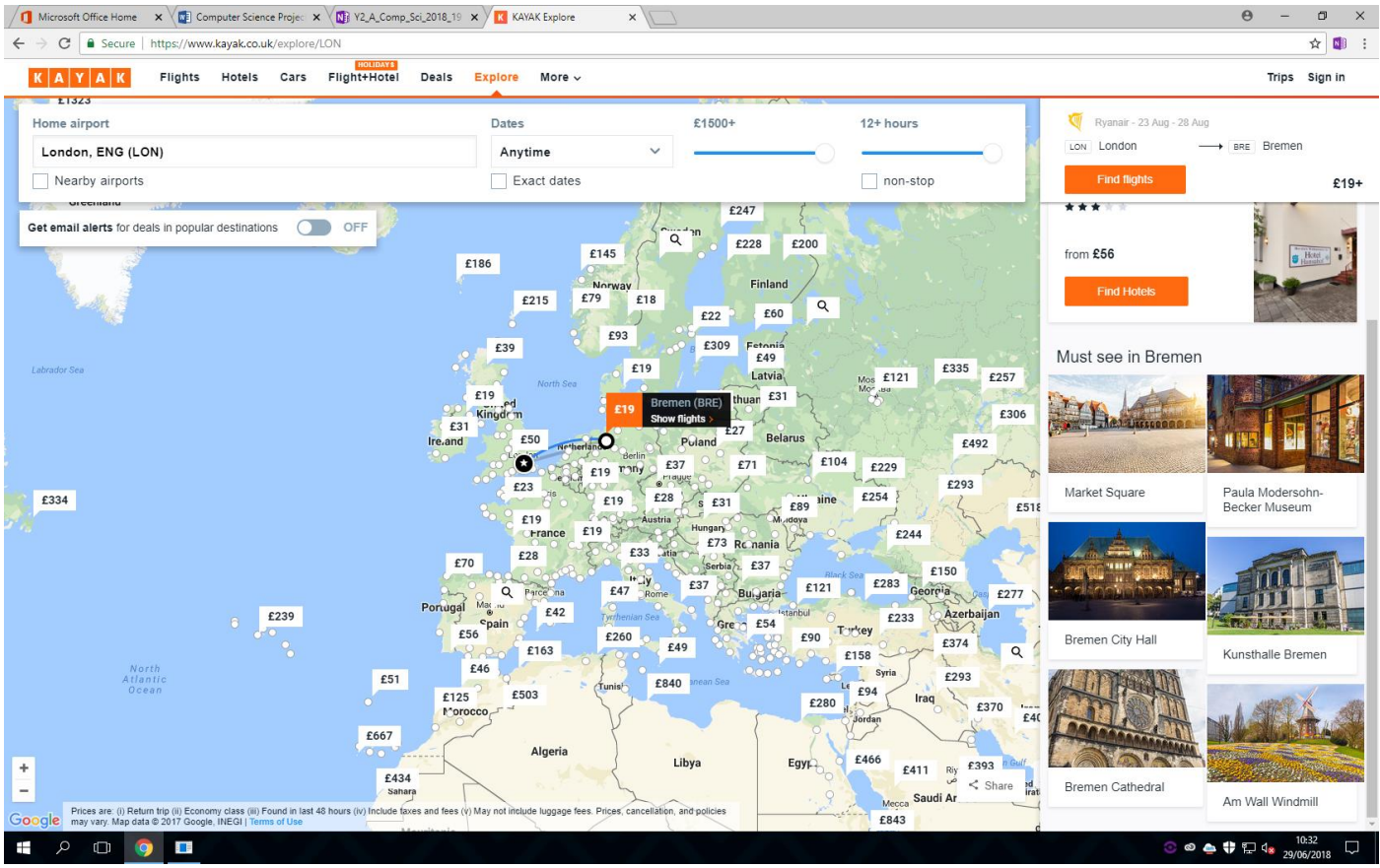


Figure 2: The Explore map when I select Bremen

Here is an example of what happens when you click on a destination in Kayak (in this case Bremen). It shows the cheapest price that you can get a flight to Bremen and on what day the cheapest flight is on. I also like the fact that it shows you the hotels that are available in the city and the main attractions which are in Bremen. One of the things that I am not a fan of is that it only shows the specific date when the airline ticket is at its cheapest and it doesn't show you what the average price of a ticket to Bremen may end up being.

Ryanair's Route Map

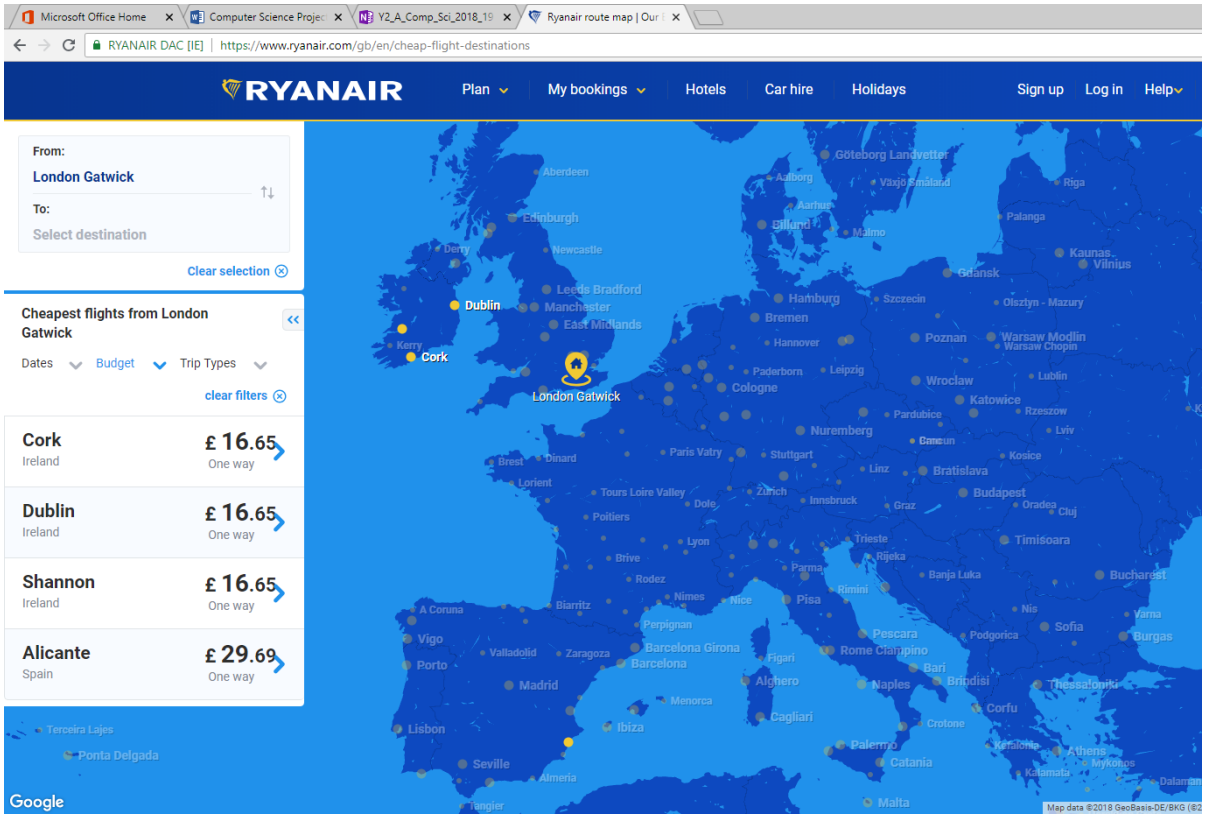


Figure 3: Ryanair's default state on the explore website

Ryanair also has a route map which is very similar to Kayak's map showing all the Ryanair destinations from a selected airport. This map is a lot simpler than the one offered by Kayak as it only has the route map with a list of destinations and the prices for a one-way journey. One of the things that I don't like about Ryanair's discover page is that as far as I can see, there is no way to filter the destinations by the type of holiday.

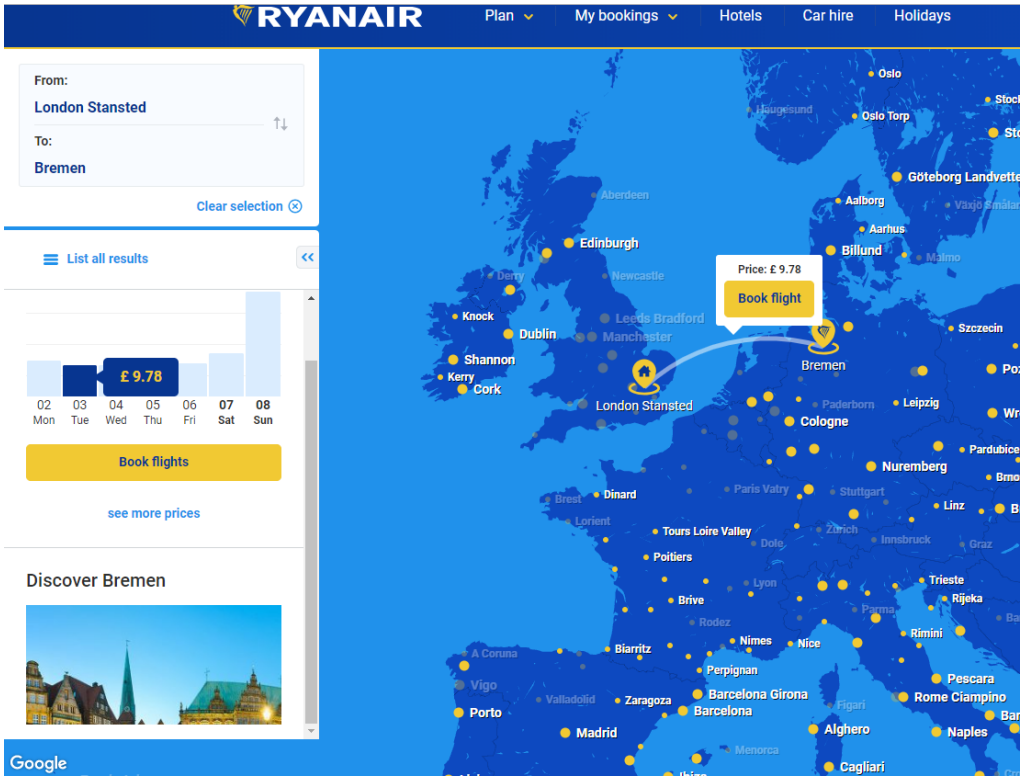
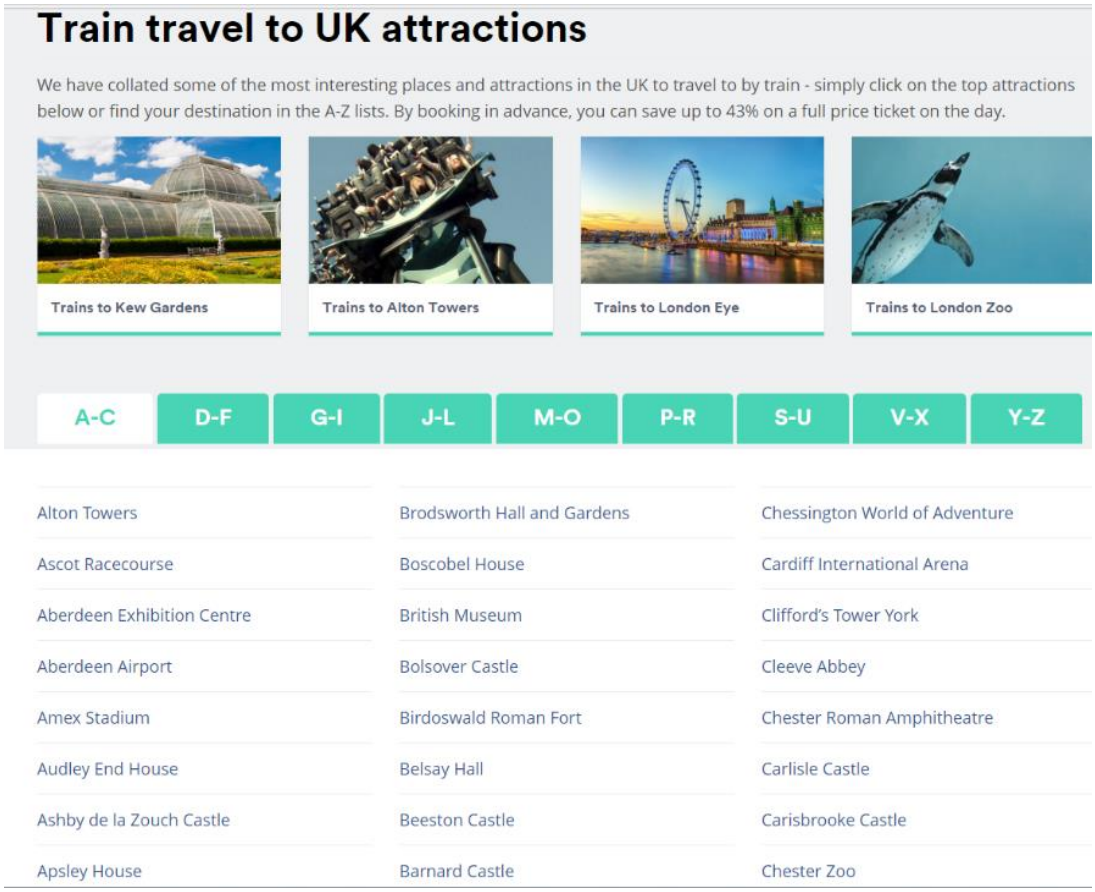


Figure 4: Ryanair website when I click on Bremen

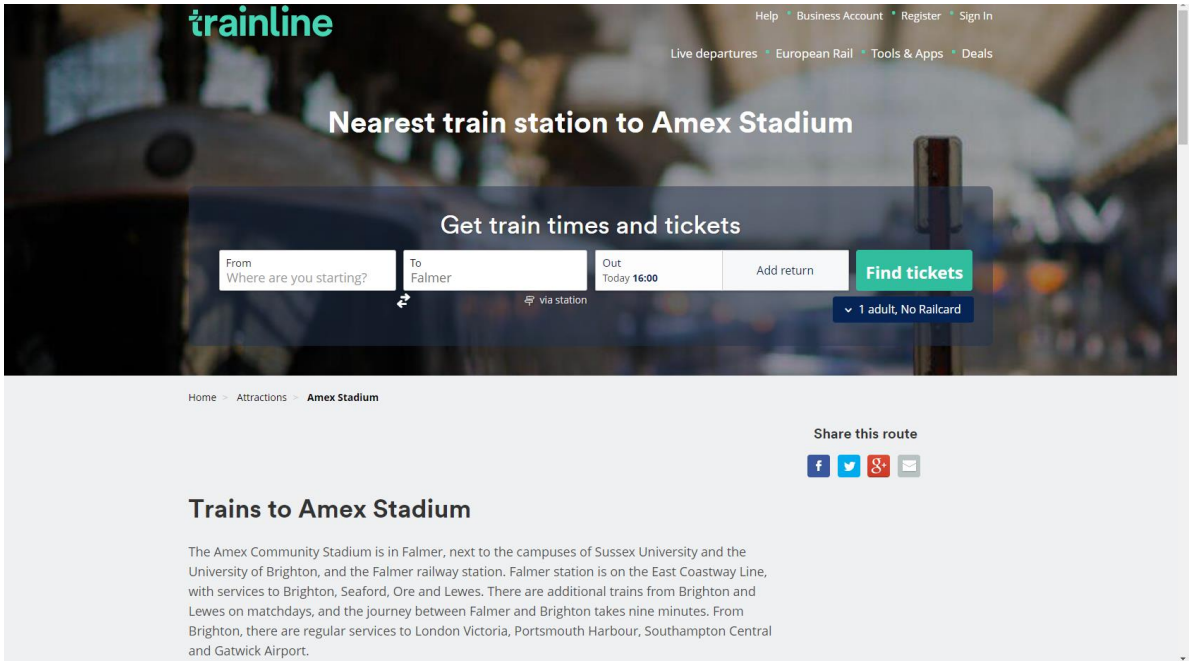
Here is a picture of what happens when I select Bremen on the map. I would say that compared to Kayak's explore sample, that this is rather disappointing, and I would say that it doesn't really inspire me to go to Bremen. The only thing that I really like is that it has a better view of when the best flights are as it is displayed in a nice bar chart. Another thing that I don't like is how unprofessional the "Discover" part of the website is. There was a very low-quality image of the city of Bremen that had been cut in half which was the same for every browser I tried it on. The worst thing is that there isn't a link to what attractions there is in Bremen compared with Kayak which had a great overview of the must-see attractions. I did manage to find an overview of Bremen when I went through the destinations tab of the website and managed to find a long piece of text buried under the flights to Bremen.

Trainline



Train line has a website where people can research a day trip within the UK. The page includes a list of notable attractions around the UK from A to Z with the links to further information on the website. This webpage is good for if you already have an idea of where you would want to go on a day trip but you don't know how to get there by train. In my project I will make sure that the user's budget is considered when they enter their details.

One of the things that I don't like about this website is that it only has a set number of attractions and it doesn't tell you what there is to do around certain stations which could mean that the stakeholders will only be limited to the top attractions in the UK. My app will ensure that people will go off the beaten track.



Shown above is the page that you are taken to when you click on an attraction. As you can see, it shows you where the nearest train station to the Attraction is and will give you a journey plan when you put in your local train stations. It also gives you an overview of what the attraction is and what can be done in the area around the attraction. The problem about this system however, is that it doesn't have all the major attractions and it just tells you how to get to these specific destinations by train. I feel that it would be better if you could be able to discover what towns you can get to by train which are more off the beaten track.

Stakeholders

My main stakeholders for the app are people who want to on a day out or a break within the UK but need inspiration to fully decide where they are going to go. My app will give the user an insight into what places they can go for the money that they have, and it will show them where they can go for their money. It could also be useful for people going on an interrail trip within the UK as it will show them all the places that are within the time that the person wants to go.

My main demographic will be families who will want to go on a day out but I will also be considering young people who are between the ages of 18 to 30. The main reason for choosing this age group is that it will hopefully give the young people a view of the places that they can go within the UK. Also, young people usually have limited budgets compared to older generations so it would

mean that they would really get a lot of use out of this app as they can find out where they can go in the UK without breaking the bank.

One of the reasons that I have put families as one of the demographics for this trip is because families are the most likely group that would be going on a day trip within the UK. Although people that aren't families do go on day outs it is more likely that families will have more use for the app as there is a greater proportion of families that will want to go on a day out. This will also help families discover places that they never would have thought of visiting.

I also chose people going on Interrail trips in the UK because it can show foreign tourists what the UK has to offer outside of Edinburgh and London. It could help these people see some parts of the UK which is more off the beaten track and can help inspire people on what else there is to see in the UK rather than just London.

The main reason why there is a need for this app is that there isn't really an app which can show the user where they could go for their money on the rail network or with any form of transportation. From my analysis of existing projects, there are only really apps which function in the same way as I plan to make the app for flights. The only rail related app I looked at was not very good in telling me where I could go to for the money that I have.

Success Criteria

Essential Features

- The app Must ask for the passenger's home station. This will be needed by the app to decide where the user will be, and it will be used as a basis for where the passenger can go. The input will be in the form of a text-based input with a drop-down list of suggestions.
- The app must take the journey times into account. This will mean that the user can limit how far they want to go because some people will not want to travel too far. (e.g. A user will probably not want to go from London to Inverness for a day). This would be done on a slider so that it is more user friendly
- The app must be able to give the user a list of available stations that the user can travel to with their money/time taken. This would hopefully be displayed on a map or will be put on a different page on the app.
- The app must have map support as that will be how the passenger can see where the stations around them are.
- The app should get the budget for the passenger so that the app can see what stations the user can get to for their price. The data will be inputted using a slider.

Non Essential Features

- The app could show a list of attractions around the station to give the passenger a more informed choice on where they can go. This will be in a separate page on the app to show what the main attractions are near to that station.
- The app could take railcards into account which is not an essential feature but it will be helpful as it could show the user that they can go to more towns with the money that they have for travelling.
- The app could also see whether the user wants to have any changes at stations or whether they just want a direct train to their destination. This should be inputted using a checkbox to only allow non-stop trains.
- The app could take multiple passengers into account. This will be in the form of a drop-down list with the numbers of adults and the number of children going on the trip. The app should then add up all the prices of all the people going on the trip.
- The app could let the user input the full name of the station. This is to make things easier for the user to tell the app which station they could travel to.
- The app could show the user the departures information from the local station to a selected station from the user's list.

Limitations

One of the main limitations to my app is that I will only be taking off peak ticket prices into account. This is because if I take peak tickets and advance singles into account I will be dealing with loads of prices. This will greatly increase the scope of the project and it will overcomplicate an already very complicated app. I will also not be considering advance singles as these can vary in time.
I will only consider the rail network in Great Britain and I will not be dealing with tickets abroad. This will be because I will not have the data to include stations that are outside the UK. I may just limit the map to the southern rail network as there would be a great scope in trying to see how many stations a person can go from every station in Great Britain.

Another limitation is that my app may only take the three-letter code to represent a station (e.g. BTN instead of Brighton). This is because the API that I am using uses the three letter codes to refer to stations on the network. If I have the time, I will try to implement a system that will let the person type in the full station name.

One of the limitations that I noticed when I was researching the API is that it doesn't yet have information for ticket prices. Therefore, my program will only be able to take the amount of time for someone to travel into account.
Another limitation that I have found from the API is that it only gives me the data of up to the next 25 trains from the station or the next trains in the

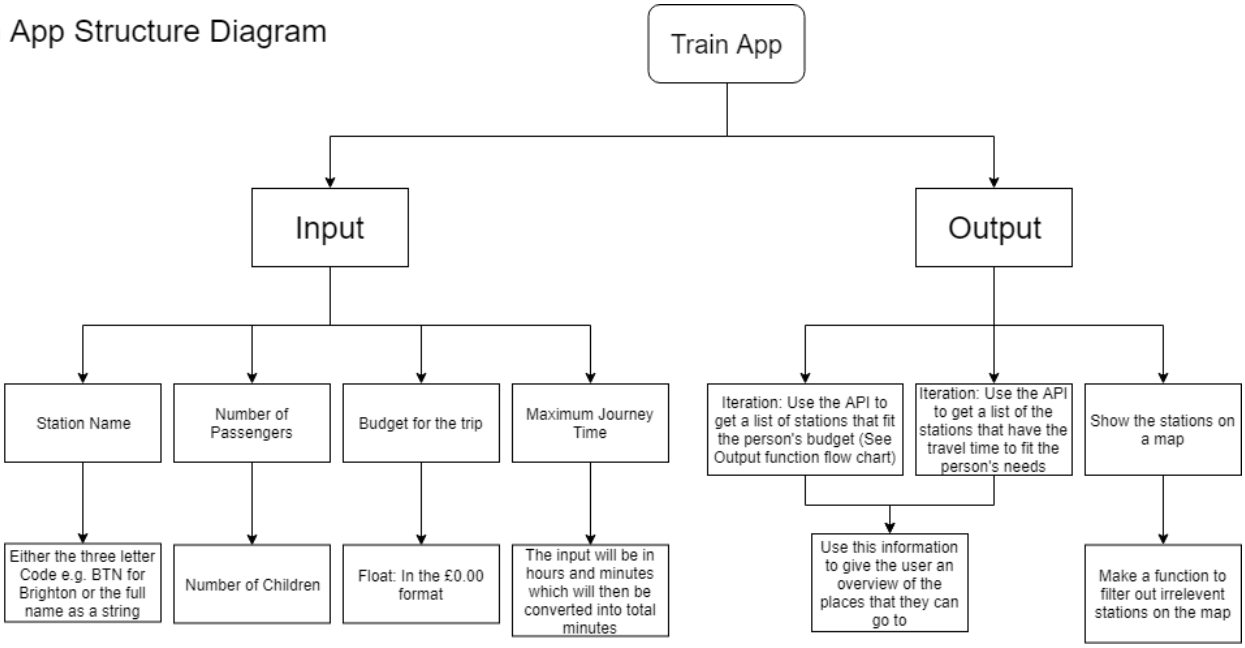
following 2 hours whichever limit is reached first. This may mean that the app may not be as useful for bigger stations like Clapham Junction as there are a lot more trains going through a station. On the other extreme, stations with only one train a day may not be well represented in this app.

There were some features that I would want to do but I am rather limited by the API. One of these is a feature which would distinguish between small suburb stations without any attractions to large city stations with loads of attractions. However, the API doesn't have the data to do that and my program will rely heavily on an API in order to get any data. Since my API doesn't have the data to do this it will be rather difficult and time consuming for me to categorize all 2,566 stations in the UK to suburbia stations and large city stations. Therefore, my app will just give the user all the stations that they can go to with a direct connection. If the user wants to find what is around the station, they can look at the map or search up the stations themselves.

As also stated above, the app will not show any attractions around the station unless it is shown on the map. This is because the API only has the data on stations, services or departures. If I want to show the attractions surrounding the station, I would have to look for another API which would give me the data and to find one which is free will be too much work. If I had more time than I am currently allocated, then I will look at making the app show the user the attractions around the area. I also feel that I would struggle to find the data to sufficiently give the user this information. Another thing is the fact that the large stations would bring with it a lot of stations so if I did manage to find an API to get the data then I would probably have run out of requests in a relatively short space of time. I feel that this will increase the scope of the project to the point where I wouldn't be able to cope with the amount of data.

Structure Diagrams

Train App Structure Diagram



The program will work by getting the input from the user. It will ask for details such as the age of the passengers (under 16s get a child ticket for example) and then it will ask for how many people will be going on the trip. Once the program has this information, it will ask for the maximum budget for the trip. This will hopefully will be represented on a slider. The next slider will show the maximum amount of time that the user wants to travel. Then the user will put in the station that they want to travel to. Once the program has found all of this information, it uses an API to search through all the stations to see which stations fit the criteria needed in terms of budget and time. Once that has happened then the program displays a list of stations in order of distance or price.

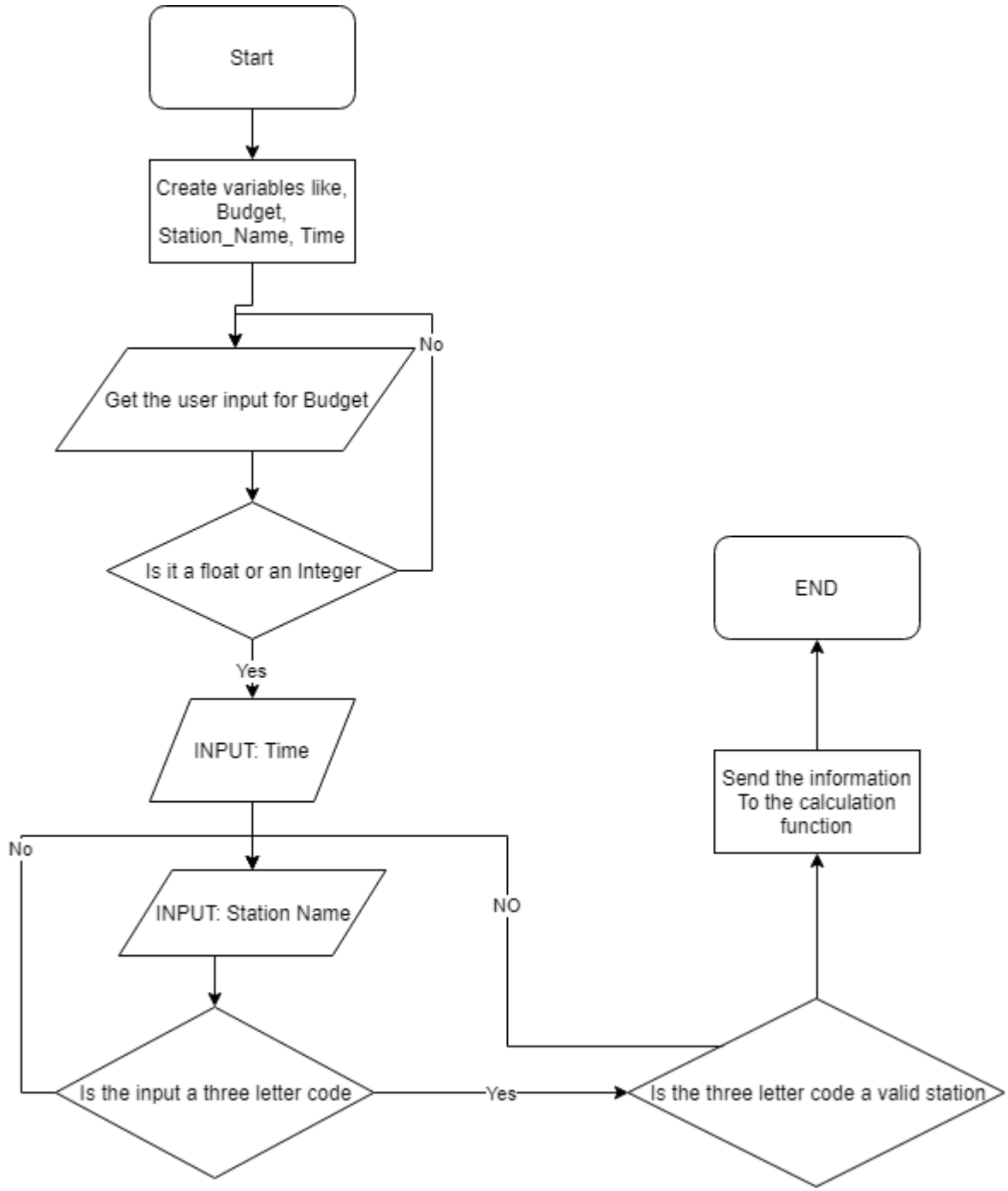
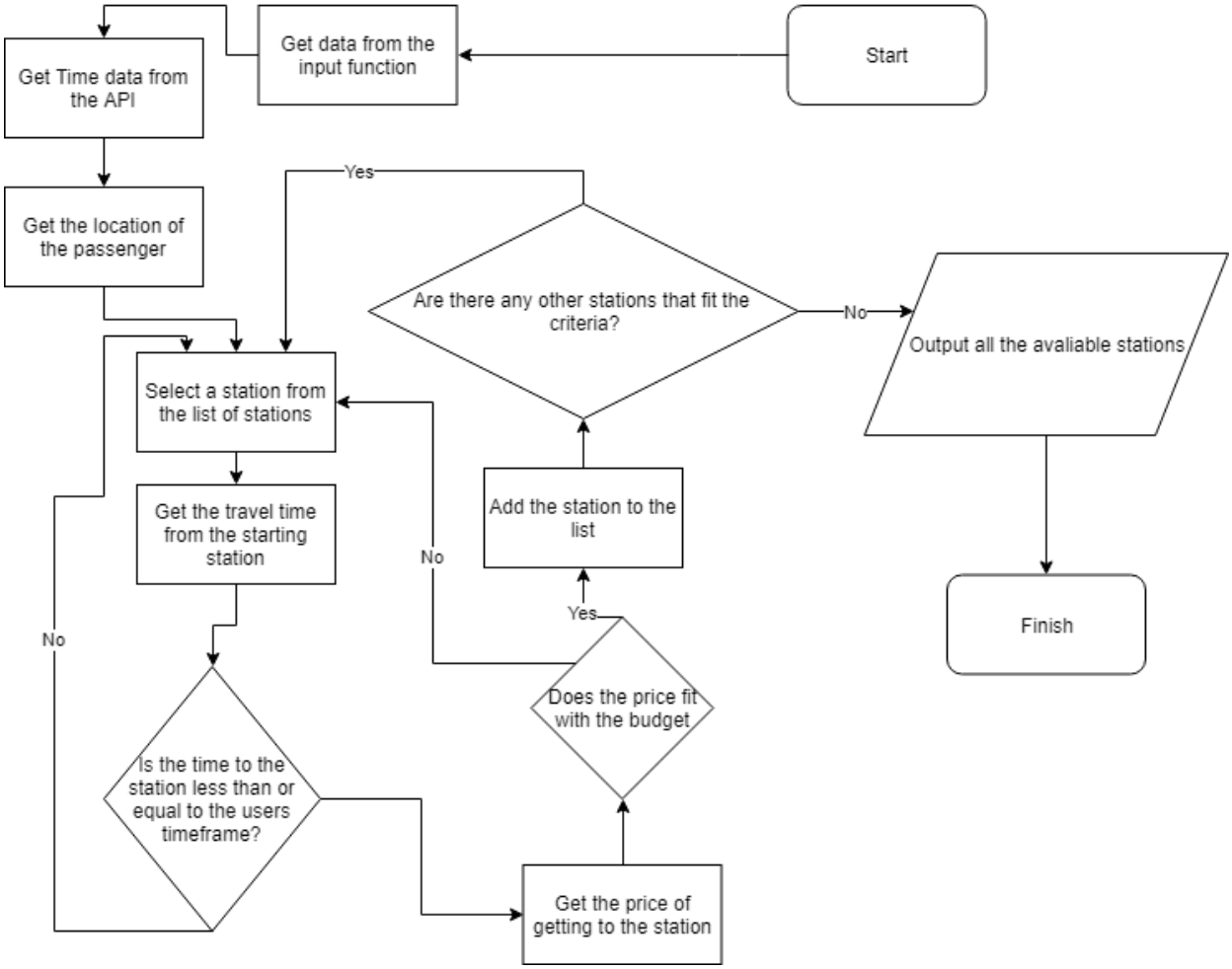


Figure 5: The flowchart for the output function



Proposed Programming Language

For this project, I will program in an object orientated language like Java. However, as this is going to be a web application, I will most likely be using JavaScript as it is easier to use when developing a website. For getting the information on stations, I will be using the transport API as it is easier to use than the national rail enquiries API. The transport API is also JSON formatted which means that I have the freedom to decide whichever language I want to use for my project. Another advantage of the transport API is that the API is free, so it will hopefully be easier to access.

Hardware and Software Requirements

For the user to use my website, they will need a computer which can run Chrome or other browsers. Therefore they will need the following requirements (taken from the Chrome requirements as this will be the browser that I will be developing in):

Windows

- Windows 7 or later
- An Intel Pentium Processor or later which is SSE2 capable

Mac

- OS X Yosemite 10.10 or later

Linux

- 64-bit Ubuntu 14.04+, Debian 8+, openSUSE 13.3+, or Fedora Linux 24+
- An Intel Pentium Processor or later that's SSE2 capable.

I feel that the user will have to have a good internet connection in order to ensure that the API can get the data in a faster time. The user also may need a decent amount of memory to handle the requests that I will be sending to the API. This is because I feel that my app will have lots of API requests in order to give the user the information.

Computational Approach

I feel that this problem is a computable problem. This is because there is a huge amount of stations, so it would take a human a long time to consider the entire rail network to make sure that the passenger can go to the best place for the best price. My app should hopefully use backtracking to solve the problem. For example, the app will check the price of getting to the station. Once the program has the information on whether the passenger can afford to go to the place, the program will backtrack to see which other stations the passenger can get to.
I will need to abstract my program to make sure that the program works to an efficient standard. One of the things that I can do to abstract the problem is making sure that I only take stations to account when creating a map functions so that I am not taking other geographical features into account. This means that the customers will only be seeing the train stations that they can go to. This will be better than a human approach to the problem is that they may not suggest places that are off the beaten track as they take other factors into account when giving the customer information. Therefore, a computable approach is suitable as it eliminates bias from other people.
Another reason why this app is best suited to a computer is that a person running the ticket office will not have the time to help someone decide where they would want to go from their station and would be a lot slower in helping the customer. A computer on the other hand will be able to go through the places that the customer can go in a fraction of the time that it would take a

human as the computer can go through all the options in a matter of seconds using simple rules: If it fits the customer's budget, keep it. If not, then don't include the station.
The app will also have enumeration in the fact that the app will go through all the specific options in terms of stations. Although this may be an inefficient approach, it is the best option because it means that all the possible stations are considered in the UK which will give the passenger a more accurate view of where they can go within their budget and their time.

A computer can also instantly change what stations the passenger can go do rather quickly to better suit the person's needs. This means that the passenger can easily see where they want to go within a few seconds. This is better than a human who will have to take a few minutes or an hour to find the stations that suit the person's needs. It means that the passenger can also spend less time planning and more time going to the place where they want to go.

Design and Development

Please note that all discussion of errors in this section will be headlined like this

Stage 1 – Create a basic website containing the input details.

Development Plan

Main aims for this stage:

- Create a simple web page
- Create the areas where the user can input data.
- Create an output button which will send the user to the output page
- Create a simple Output page with the information as text (This will be changed in later stages, this will just be used for testing in the early stages). At this stage it will just be a simple piece of text to test that the code works.

In this stage, I will create the basic outline of the input and output screen. The main aim of the stage at first is not to make the user interface look good aesthetically but to start to create the input which can be used to make the program function. I would much rather make a Minimum Viable Product first and then build on what I have made in earlier stages to make a more customer friendly product. This interface will form as the basis for basic testing of whether the program has taken in the input that it has been given.

Test Description	Test Type	Test Data	Expected Result
Make sure that the input modules are where they are meant to be	Valid, White Box	Input Screen	The widgets are in the right place.
Make sure that the output buttons work	Valid, White Box	Input Screen	The button takes the user to the output page

Ensure that the label with the slider has the correct value	Valid, White Box	Input Screen. Time slider	I will expect the website will display the correct values.

Implementation Stage 1

UI Design

Here is the initial design for the input function of the program. It won't look exactly like this but it will be the basis on what the options will be on the input page before I think about sprucing up the app. Below is an annotated version of the app. (See page 123 if the picture isn't loading).

screen of the program.

As you can see, I haven't decided to put in a name yet. This is because at this stage of the design, I didn't know what the name of the website was going to be. All I need to do now is to make a more aesthetically pleasing design to my website to make sure that people will stick around. (See page 127 if the picture is not loading).

Here is the potential output page for the website. In future iterations of the design, I will make sure that there is a different pointer to show the users location. To make it more user friendly, I have put some of the inputs to the left-hand side of the screen. This means that the user can use the slider and see how many extra destinations appear when the user changes the budget and Time of travel sliders. There are also the direct trains only checkbox which will filter out any changes of trains to only ensure that the passenger can only travel on trains which go direct from the station (This is especially useful for people going from large city stations like London Euston). I am not sure what else will be on the pop up from Brighton to tell the user the price of going to the place. If I have time, I may put in the attractions that you can find on the way.

This output screen will probably not be fully implemented until stage 5 when I plan to have everything else finished. Instead, the output function will just contain the necessary outputs needed to test the program. (e.g. checking whether the inputs have been logged onto the system and whether the inputs have been carried over to the other screen). Once I have gotten some of the more complicated algorithms out the way then I can get to work on this potential output screen.

Unfortunately, as mentioned on the mockup of the input screen, because the API doesn't have functionality for calculating fare prices. I won't be able to implement the budget function. Therefore, when I am actually creating the program, there will be no budget function and therefore the time function will be in the place of the budget function and the direct connections area will be further up as well.

Required variables

As this stage is mainly creating the website, I won't need as many variables. However, there are still some variables that I need to implement. This is most prominent in the age section. Here is the table of the required variables at the current moment.

Variable Name	**Variable Type**	**Purpose**	Validation
option	Object	The purpose of this variable is to refer to a new option when creating the age drop down list.	N/A
i	Integer	The use of this variable is to act as an iterator when creating the array of options. It will also be used to put the options into the box	When "i" gets over a certain amount, it will be able to stop the loop that is needed to create the dropdown list
button_pressed	Boolean	The main purpose of this variable is to check whether the button is pressed.	In this current stage there will be no validation. However, in later stages, there will be checks to see whether all the information that is provided is valid.
age	Object	The purpose of this variable is to refer to the drop down list concerning the age of the passenger	N/A

Value	Integer	The purpose of this variable is to store the value being input into the time slider. This is so I can show the user the number of minutes they are inputting	0 to 1440. The validation will be in a form of a slider which has a set maximum and a set minimum so that the user cannot go out of bounds
slider	Object	This variable will be used so that the JavaScript can refer to the time_slider in the HTML file	N/A
Hours	Integer	This will be used to give a more user friendly feel to the slider variable. This will output the amount of Minutes as different hours.	

As you can see, compared to other stages, the variable list is quite small. This is because the amount of coding that involves lots of variables is very minimal as I am only creating the form in the input section of the web application.

Development

Even though there isn't a lot of coding using variables, one of the algorithms that are needed at this stage is the algorithm that puts all the options for the age dropdown box. Below is the basic outline of what the code should look like.

```
INT i = 0
ACCESS train_app.html

OBJ age = <select name = "age"> IN train_app.html

FOR i TO 110
        OBJ option = NEW <option> IN age
        APPEND str(i) INTO option
ENDFOR
```

This is one of the first algorithms that I will need using JavaScript. The age variable will act as the <select> tag in my form on the website. This will make things easier when making comparisons in my code as it will make sure that I won't have to write out the command referring to the specific tag as it is a very long command to type all the time and could lead to confusion. Especially when my program is a lot more complicated. Underneath is the code as it appears in source code.

```
Age:<br>
<select id = 'Age'><br>
    <option value = "default">Please select an age range</option>
    <option value = "under_16">Under 16</option>
    <option value = "16-30">16-30</option>
    <option value = "30-45">30-45</option>
    <option value = "45-65">45-65</option>
    <option value = "over_65">Over 65</option>
</select><br>
Local Station:<br>
<input type="text" id = "local station" name="Local Station"/>
```

As you can see, it looks slightly different than the initial pseudocode. This is because I tried to make the code work with JavaScript, but I realised that it would be a lot easier to make the options using the <option> tag within the <select> tag which creates the drop-down box. Another reason for this change was that I thought that it would be a lot more user friendly to have a small list of age ranges rather than a long list of numbers from 0 to 110. Therefore, I felt that I was wasting time focusing on making a feature which wouldn't be very user friendly in favour of another option which was easier to implement and would be more user friendly as there are more complicated things which will need more time to implement rather than the age list.

The next major piece of code that I will need when implementing this stage of the program is to make the time taken slider more user friendly. One way that I will do this is I will make the website update the value that the slider is currently on. Unlike the last stage, I feel that I will need to use JavaScript to make this work. The pseudocode that I have written to make a brief outline is shown below.

```
ACCESS train_app.html

OBJ slider = <input id = "Time"> in train_app.html
INT value = "value" in slider
INT hours = 0

WHILE (TRUE)
        INT value = "value" in slider
        IF value / 60 >= 1 THEN
                hours = value DIV 60
        ENDIF
        OUTPUT hours, "hours", value - (hours * 60), " mins"
ENDWHILE
```

As you can see, it is a simple algorithm to code but it will make the website a lot more user-friendly as the user can have an idea of what value they are inputting into the app and so they will know that their time is correctly put into the program so they get stations tailored to their needs. One of the other additions is the addition of an hours label. This is because it is easier to read what time the person has got in terms of hours and minutes rather than how many minutes the passenger has especially when we are dealing with big numbers. This will make sure that the passenger can easily see what time they are inputting. Below is the Pseudocode implemented into JavaScript.

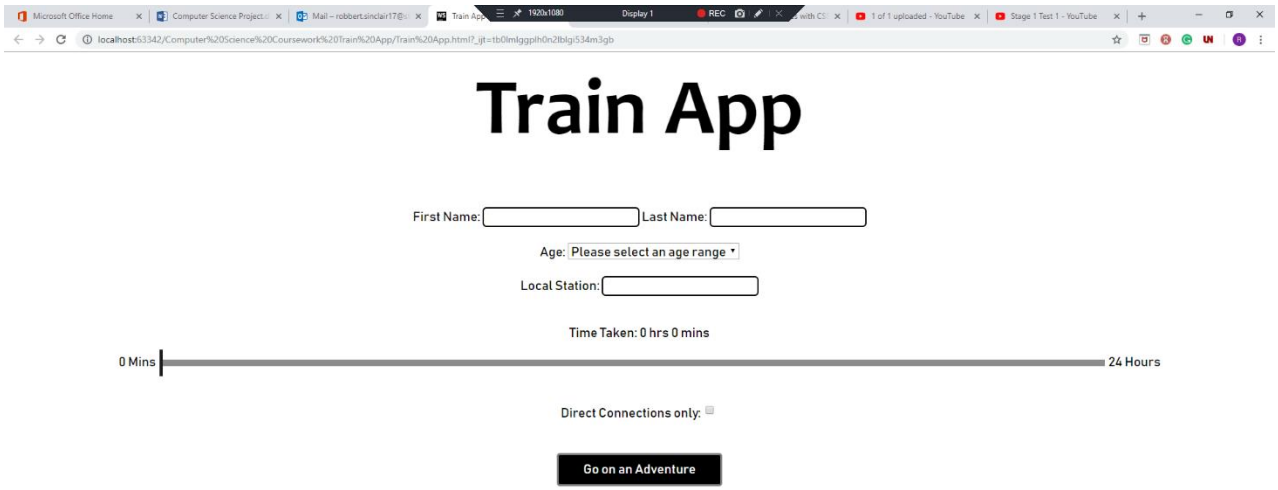
```
<script>
    let slider = document.getElementById("Time");
    let output1 = document.getElementById("hours");
    let output2 = document.getElementById("time_output");
    let hours = 0;
    output2.innerHTML = slider.value;
    slider.oninput = function(){
        hours = parseInt(this.value /60);
        output1.innerHTML = (hours);
        output2.innerHTML = (this.value - (hours*60));
    }
</script>
```

For coding this part of the slider, I made use of a helpful tutorial from www.w3schools.com which showed me how to implement the output on the webpage. I have also added the hours variable which would act as the basis of how many hours the user has input. I used the parseInt() function so that when I divide the value by 60, the hours variable would only have the full number part and it wouldn't have a lot of decimal places. Another thing that I needed to have is variables which were dynamic and didn't change as much. Therefore I made the variables contain the "let" keyword rather than the "const" keyword. This ensures that the variable will change value.

Testing

Test 1 - Make sure that the input modules are where they are meant to be

Now that I have implemented all the main plans for this stage, it is time to start testing all the features. The first test was to make sure that all the widgets that I wanted to use on the website were in place and to see whether the resulting page resembles the initial design.



From this screenshot, I can see that all the widgets that I wanted from the input screen are up and running as expected. **Test Successful.**

Test 2 - Ensure that the label with the slider has the correct value

The next test is to make sure that the "Time taken:" label was working properly. This is so that the user can properly see what time is being inputted.

Test Video Here:
https://www.youtube.com/watch?v=HOmZMO4T5aY&feature=youtu.be

From the video I can see that the slider worked as planned.

Error 1:

However, although it was not shown in the video, I found that when the website loaded up initially, the hour label didn't work properly as there seemed to be no number in the hours box.

Solution:

Thankfully, it was an easy fix as I had just forgotten to set the output before the user had used the slider. Below is the amended source code.

```
<script>
    let slider = document.getElementById("Time");
    let output1 = document.getElementById("hours");
    let output2 = document.getElementById("time_output");
    let hours = 0;
    output2.innerHTML = slider.value;
    output1.innerHTML = (hours);
    slider.oninput = function(){
        hours = parseInt(this.value /60);
        output1.innerHTML = (hours);
        output2.innerHTML = (this.value - (hours*60));
    }
</script>
```

After that small amendment to the web page, the slider label had both the hours variable and the minutes variables in their normal places. **Test Successful**

Test 3 - Ensure that the "Go on an Adventure" button sends the user to the output screen.

The final main test of this stage of the development cycle is to ensure that the "Go on an adventure" button would send the user to the output screen. This is to make sure that the website can send the user to the list of stations that they can visit. To see the test, please follow the link below to the video.

Test Video Here: https://youtu.be/gSiC0j9DaZI

As you can see in the test video, when I pressed the "Go on an adventure" button, it took me to the output page. The output page is only the words "It works" but in later stages, it will be changed into the output page showing the user all the stations that they can go to. **Test Successful**

Seeing that all my tests for this stage are complete, I am ready to move on to the next stage of development which is to make the data from the form that I have made be registered as an input.

Stage 1 – Review

Main aims:

- Create a simple web page **Fully met** (page 21)
- Create the areas where the user can input data. **Fully met** (page 21)
- Create an output button which will send the user to the output page **Fully met** (page 22)
- Create a simple Output page with the information as text **Fully met** (page 22)

I feel that I have managed to implement all the things that I was going to do with this stage. I have managed to get a presentable web page with all the form widgets needed for later stages. I also feel that I have managed to make some extra features which will help with the usability of the website. The most prominent of these features is the feature which shows the value that the user is inputting on the slider. I feel that overall I have succeeded on this part of development and I am able to move on to other stages.

Stage 2 – Get the user's personal details

Development Plan

Main plans for this stage:

- Get the passenger's name
- Get the passenger's age
- Determine whether the passenger is a child or an adult.
- Make sure that all the information is correct when the user presses “Go on an adventure” button.
- Get the user's local station
- Get the time that the user must travel

In this stage, I will be getting the personal details of the passenger that is travelling. This is needed in further areas of the development because it means that I can determine whether the passenger is an adult or a child. This is because children get discounted tickets on the network so it is important that I factor that in to give a more accurate overview of how far the passengers can go. In this phase. I will make text boxes for the name which will be stored in a string variable. In order to determine the age I will make a simple drop down arrow where the user can select the age. I will then make an algorithm which checks whether the passenger is an adult or a child. For example, if the passenger is less than 16 years old then they are a child, otherwise the passenger is an adult. Another thing is that I will limit the name to 30 characters to make sure that her is no excess in data.

This is also one of the most important parts of the input stage of the program. This is because the output function will need the time taken to travel is one of the most important factor when deciding the stations that the user can go to. The user's local station will at first be the three letter code for the station and if I have time the full station name will be used.

Test Description	Type of test	Test Data	Expected result
Put in the passenger name that is under 30 characters	Valid. Black Box test	Input the passenger name	The program allows the name to be used
Put a passenger name that is over 30 characters	Invalid, Black Box Test	Input the passenger name	The program asks the user to enter a shorter name
Set the age to something under 0	Invalid, Black Box Testing	Passenger age	The program should reject the age and give the user another try
Type in the three letter code of a valid station	Valid, White Box	Station name	The program should find the expected station.
Type in a station that does not exist.	Invalid, Black Box	Station Name	The program should tell the user to try again in inputting a station
Put in a valid time in which to travel	Valid, Black Box	Time	The program should accept the time to travel
Put in a time that doesn’t exist	Invalid, Black Box	Time	The program should reject the user’s time as it isn’t possible.
Input nothing on the first_name and last_name textfield	Invalid, Black Box	First_name, last_name	The program should tell the user that they have invalid data and they should try again.

Implementation Stage 2

Required variables

Compared to stage 1, this stage will need a lot more new variables to act as the input to store all the pieces of data that the user has put in to generate the list of stations. I will also need to start using the API to make the program work. Below is a list of the variables that I will need for this stage in development.

Variable Name	Variable Type	Purpose	Validation
direct_connections	Boolean	The purpose of this variable is to adapt the program if the direct connections box is checked	If the direct connections box is checked, then the variable will return true. Otherwise the variable will be false
First_Name	String	This is to make the program feel tailored to the user if the program refers to them as their name	Must have at least one letter as a name
Last_Name	String	This also will be used to make the program feel tailored to the user by having the program refer to the name of the client	Must have at least one letter as a name
Age	String	This is to determine whether the client is a child for if I add a budget function later	One of the age ranges from the drop down boxes created in stage 1. If the user puts in the “Under 16” function then the client is considered to be a child
time_taken	Int	This will determine how much time the	This is validated by the “time taken” slider

		user will have to make the journey. The variable is stored in minutes	created in stage 1 and so will have a minimum value of 0 and a maximum value of 24 hours.
local_station	String	This will be the basis of where the user will be travelling to and as a basis of where the program will start.	At this stage, the station must have the 3 letter ATOC code. The train station must be one of the 2653 stations in Great Britain. The variable will be validated using the Transport API. https://www.transportapi.com
correct	Boolean	To stop the user from continuing if they have put in invalid data	Allow the user to go onto the next page when the user has input all the correct details.
first_name_box	Object	The purpose is to refer to the first name text box for inputting the first name	N/A
last_name_box	Object	The purpose of this variable is to refer to the last name text box	N/A
local_station_box	Object	The purpose of this variable is to refer to the local station text box for inputting data	N/A

Submit_button	Object	The purpose is to refer to the submit button in the input page.	N/A
ourRequest	XMLHttP Request	The purpose of this variable is to make a request to the API in order to validate the type of station that is being used.	N/A

There will need to be a lot more variables in this stage than stage 1. This is because there will be more algorithms to make to ensure that the information that is being inputted is correct. The first stage is to just get values onto the output screen. Therefore, I will need to make additions to the Output page to ensure that I can easily test the code. Below is the Output screen now.

As you can see, the output screen is very basic, but it will be improved in later stages, this is just so that I can see that the variables are carried over to the output screen. I feel that I should start by planning an algorithm which takes the input of all the variables and then places them in the output site. Below is the pseudocode which will give you a rough outline of how it will work.

```
local_station_label.value = local_station
```

As you can see there aren't any real algorithms in this piece of code but there will be some algorithms later. This is mainly because at this stage, I just want to be able to transfer variables from one part to the other. Later on in the stage, I will start to validate the variables. Below is this stage in code.

As you can see, the actual code is exactly like the pseudocode and was initially a test to see whether I could output the variables on the input screen. It was here where I realised that I may not need to have an output screen. This was because when I clicked the submit button on the form, the form was replaced by the output values. Here is what I got as an output.

As you can see, this discovery made the separate output file redundant. This may also be more efficient as it means that most of the functions are all happening on one page rather than two pages. It also means that the page will be a lot easier to manage as the input and the output would all be happening on the same page.

The second part of creating the input values was to carry out the necessary validation to the text boxes. I will start with validating the name values which is the more simpler of the things that need to be validated. Below is the pseudocode that will be needed to fully run this part of the validation. Please note that this will be an extension of the input function therefore it will carry on some of the variables.

Here you can see the basic outline of how the validation of the first name and last name will work. The validation of the local station will be added after this but for now I want to work on the first_name and last_name variables. One of the main things that I will need to consider when programming this stage in the program is to make sure that the form doesn't disappear if the information is correct and to only output the text when it is fully validated. Therefore, I will probably replace the "return true" statements with a Boolean variable.

```
    }
```

Here is the algorithm in JavaScript. I decided that I would make the validation a separate function so that it will be easier to debug later in development. It also means that the validation will be simpler as it just meant that I needed to use if, else if, and else statements to make it work. Another thing I found was that when the alert message came up, it meant that the form would just reset and the website would not display the inputs. This meant that I didn't need to use a loop for this part of development. However, it completely resets the form which may not be very user friendly which may be something that I need to work on after I have done my final piece of validation. This will be to validate the local station. Below is the proposed pseudocode to show what it will look like.

Here is the code for validating the station. This will be my first use of the transport API so I will need to do some research on how the API works before I can get started. My initial idea is to see whether the API recognises the value of the station as input. If there is no output then it is obvious that the station does not exist. Below is the code for the validate_station function.

In order to make use of the API. I needed to learn a bit on how JSON and AJAX works. I also needed to get the specific URL of the API and the data from the API that I wanted to request. For this specific function I made use of the /uk/places URL. This is because it can also find the station based on the full name of the station rather than just the three-letter code. This makes the program more user friendly as initially, I thought that the API would only allow me to take the three-letter code into account. It is very simple to make a dynamic request from a url as it acts the same way as a string. Therefore all I had to do to make the API search for the station I wanted was to break the quotation marks and put in "+ station" in order for the correct call to be entered. In order to validate the station, I first needed to make sure that the data was in a JSON format. This is the reason why I put in the "JSON.parse"

function into the program. Below is an example of what the JSON output looks like when I put in BTN which is the three letter code for Brighton station.

Here is an example of the JSON data that I got back from the request which is vital to understanding some of the other variables in the code. The variable ourRequest refers to the AJAX request that I sent to the API which is the reason why the variable has an onload function. The variable ourData refers to all the JSON data that is returned from the request. The ourData.member.length variables refers to the amount of results that I got back from the API which means that I will be able to use it to determine whether a station exists in the UK. This is what the variable "l" is used for. I set "l" as a global variable so that the value for l could be used outside the on_load function. The idea being that when the "l" variable was set equal to 0 then the program would return false and would mean that the user would have to try again in terms of inputting a station.

Testing

Test 1 – Put in a passenger name under 30 characters

One of the first tests that I need to carry out is to make sure that I can put a passenger name under 30 characters and the last name box under 30 characters. The expected result is that I will be taken to the output page.

Test 2 – Put in a valid age

As I must go through all the text box inputs before submitting my form. This means that I will do all the first three tests in one go. As I had said in my test plan. My expected result is that the program will take in the valid age.

Test 3 – Put in a valid station

The penultimate test that I will be doing in one go is to input a valid station name. This is so that the user can get all the stations within a certain radius of their own station. For this test I will put in the station code for "Brighton" which is BTN.

Test 4 – Put in a valid time

The final test that will be done is to see whether the time function works properly. Therefore I will put in a valid time and I will see whether the program works.

Test video here: https://youtu.be/Chw1gDUf4J4

Test 1 – Results

When I put in my name into the text box and pressed submit it confirmed to me that I had put in a valid name as it took me to the output page. This means that the passenger can put their name in correctly. **Test Successful**

Test 2 – Results. As you can see from the video. I just used the dropdown list to select a valid age range. When I clicked submit, I was also sent to the output

page confirming to me that the input had been carried out and that it was valid. This is exactly as I expected it to work. **Test Successful**

Test 3 – Results. From the video I can see that when I put "BTN" into the program. The program then took the user to the output page confirming to me that the station is valid and that the user can use that station as their local station. This is what I expected the program to do. **Test Successful**

Test 4 - Results. From the video I can see that when I put in my input. The program takes the input and places it on my output screen. This is good because it is what I expected from the program. **Test Successful**

Test 5 – Put in a travel time that doesn't exist

For the next test I will check the time slider to see whether the program will reject the user's input when they put in a time under 0 minutes. This is to ensure that the program has a valid time in which to operate and it would not make sense for there to be a negative number as the time.

Test Video: https://youtu.be/Mh0R_xDWitU

Test 5 – Results

As you can see from the video. I realised that I was really struggling to move the slider below 0. This means that the user will not be able to get a value under 0 since it is not on the bounds of the slider. Therefore the program's time slider is working as planed. **Test Successful**

Test 6 – Put in a name that is over 30 characters

The next test is to see whether I can put a name that is over 30 characters long in both the first_name box and the last_name box. The main reason why I decided on having a cap on the amount of characters is that it would take up a lot of memory which could be used for more important things. Therefore, I will put a name which is 31 characters long and I will see what happens.

Test video: https://youtu.be/6vYxz9U5_UM

Error 2:

One of the things that I have noticed from this test is that when I put in a name that is over 30 characters, the program thinks that it is a valid input. This should not happen therefore I will need to amend the validate_values() function to make sure that this program works.

Solution:

It was a rather easy fix to do. I had just forgotten to put in an or statement in my if statements that made sure that the program would return false when the word length is longer than 30. Below is the amended code which will validate the values.

Now that I have made these amendments I will then carry out the test again to see whether I the changes have made any difference.

Test video: https://youtu.be/X6MTgEE7hEY

From the video I can see that once I had put in a name which is over 30 characters, the program rejects the input and asks the user to try again. This is exactly what I expected to happen. **Test Successful**

Test 7 – Put in an age that is less than 0

Like the time variable, it would not really make sense for the user to be able to put in a negative value for the age. Therefore I will make sure that the age value cannot be under 0 and I hope that the program will reject my input if I try to put in an invalid age.

Test video: https://youtu.be/-w564p2wOHo

Test 7 – Results.

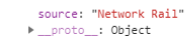
I can see from the test video that I cannot put in an age under 0 as there is no option to set the age to something less than or equal to 0. Also if I leave the drop down box on the default option, the website will simply not let me progress onto the output page. Therefore I feel that the age group system is working properly and there is no way to break it. **Test Successful**

Test 8 – Put in the three letter code of a station that exists.

For this test I will be putting in the three letter code of a valid station. For this test I will be putting in the three letter code for London Victoria station. This is VIC. My expectation is that the website would take me to the output screen. I will also output the console output for this.

Test Video: https://youtu.be/7oDHiYhPCTU

Below is the console output of the test.



Test 8 – Results

I can see that from the test video the website took me to the output page when I put in the three letter code for London Victoria. This is what I expected to happen. However, when I looked at the Output on the console from the test. I noticed that there were more results than I expected as along with "London Victoria" There were also other stations. However, I feel that this is because the API looks at both the station code and the name when looking for the stations. Therefore I feel that the test is still successful. In the next stage I will only take the first station on the list as it probably will be the most likely station that the user wanted. **Test Successful**

Test 9 – Put in an invalid station

For this test, I am going to see how the website reacts when I put in a station that isn't known to exist. Therefore I am just going to put in the station name as "Munich" as there is no station in the UK with the station name of Munich.

Test Video: https://youtu.be/GR8czeZIwqQ

Here is the console output of the test:

Error 3:

As you could see from the test video. The test was not as successful as I hoped. This is because, despite the fact that the length of the output is 0. The program is still letting me access the output page. This could be a problem as the API will not be able to do it's other calls. I think the problem is the fact that the onload function is not talking to the validate_station procedure correctly. Therefore in order to solve this error, I am going to have to work out a way of letting the validate_station access the result of the onload function.

Solution

After a lot of time trying to fix this solution, it was only when I decided to move on to stage 3 where I realised what the problem was. The main problem was that the onload function is an asynchronous function. This means that it will wait until the main batch of Code is run before it could run. This meant that the "l" variable was undefined because the function wasn't run until after the main function had worked. This meant that since there wasn't an assigned value to "l", the default would have to be true. It turned out that there was an easy solution to this problem and there had to be a way to make the onload function synchronous. There is a third parameter in the open function when creating an AJAX request which is a boolean function which decides whether the function is run with the main thread or whether there should be an asynchronous thread. All I needed to do was set this parameter to false. The screenshot below shows the amended code.

After this simple amendment, the program rejected my input of "Munich" meaning that I have to put in a valid station. This is how I expected the program to work. **Test Successful.**

Stage 2 Review

- Get the passenger's name **Fully met**
- Get the passenger's age **Fully met**
- Determine whether the passenger is a child or an adult. **Not met**
- Make sure that all the information is correct when the user presses "Go on an adventure" button. **Fully met**
- Get the user's local station **Fully met**
- Get the time that the user must travel **Fully met**

Looking back on this stage, I feel that I managed to make most of the most important parts of the stage. I have managed to successfully get the user's input and I have managed to validate the program to a good standard. One of the things that I would say however, is that I didn't manage to get the program to check whether the passenger was a child or an adult. This is because as I said on the limitations section (see page 9), I don't have the data to get the fare prices. Therefore there was no real need to let the program determine whether the passenger is a child or not. If I were to have more time and if I where to get the data for getting ticket prices, then I will make sure to implement this feature. I will list this in my Future Developments on page 114.

Stage 3 – Get a list of stations that the person can go with a direct train from their local station

Development stage

Main aims of this stage

- Check whether the user has checked the direct connections checkbox
- Find all the stations that can be reached directly from the station and which fits the other criteria (Whether the train can get to the station within the maximum time).

This is one of the first things that I want to do in terms of outputting what stations a person can go to. This is because having the amount of stations in terms of where they can go with one train is not as complex as when a certain amount of changes are needed. Therefore, I will make sure that I can get this part of the program working before I make the code take junction stations into account. Below are the tests that I will partake in for this stage

Test Description	Type of Test	Test Data	Expected Result
Check whether the stations that are displayed can be reached with one train if the direct connection box is checked	Valid, White Box	Direct connections box	There will be no stations on the results that cannot be reached with more than one train
Make sure that the stations are also within the time limit	Valid, White Box	Direct Connections box, Time taken	I will expect that all the stations that come up will be within the time limit
If the direct connections only box is not ticked, Make sure that there are stations that can only be reached with more than one train if the station is within the time limit	Valid, White Box	Direct Connections box, Time Taken	I will expect the stations to appear on the list if they are on the time limit but I won't expect them to appear if they are too far away.

Find the stations that can be reached from a busy station on the national rail network	Valid, White Box	Time taken. The station will be London Waterloo which is the busiest station in the UK	I will expect there to be a lot of stations within a certain amount of minutes
Find the stations that can be reached from the least used station on the national rail network	Valid, White Box	Time taken. The station that I will be testing this on is Barry Links which is the least used station in the UK	I will expect there to be not as much stations and I may not get any results due to the infrequency of the trains.

Implementation Stage 3

For this stage I will need to use a wide range of variables. This is due to the complexity of the algorithm. Below is a table of the variables that I have used or plan to use in this stage.

Variable Name	Data Type	Purpose	Validation
station_code	String	The main purpose of this is that I can't just refer to the station by name and I will need to use the three letter code. "i.e. BTN for Brighton". This is so that I can call the api to get the timetable.	It is taken from the places call from the API in stage 2. All I will need to do is get the station code from the results of the API
Request_2 and Request_3	XMLHttp Request	The purpose of this part of these variable is that they will be used to refer to the AJAX requests needed to call	N/A

		the timetable data and services data respectively.	
train_uid	Integer array of size 25	These are the unique codes which refer to a specific train service. This is so that I can access the calling points of these trains so I can make my comparisons in terms of time. The main reason why it has a size of 25 is because that is the maximum amount of results I get back per call.	These will be sourced from the timetable call from the API. I will go through each of the departures that I received from the local station to take the uid.
station_index	Integer	In order to get the amount of stations that the user can travel to. I will need to make sure that the array starts at the user's local station. Therefore the user can get a more accurate feel of what stations they can visit	Loop through the calling points from the services until the program reaches the local station.
Station_time	Integer	This is to keep track of the amount of time	This variable will be compared to the max time that

		the time takes to get from the local station to the current station being checked.	the user has inputted in stage 2 to determine how long the train takes to travel
HTML_Output	String	This will be the variable that outputs all the names of the stations that the user can travel to.	It will take the variable of the station name and the time it takes to travel.
data_2	JSON Array	The purpose of this variable is to store the results of the request to the API for the timetable of the station. This will also be needed to manipulate the information and to get the nessary data	N/A
data_3	JSON Array	The pupose of this variable is to store the results of the services requests to the API. It will give out the calling points of the trains.	N/A
start_time	String	The main purpose of this variable is to get the departure time of the train from the station.	Concatenate the aimed departure time and the aimed departure date together so that the data can

		This will be the aimed departure time and the aimed departure date concatenated	be turned into a departure time.
end_time	String	This is the same as the start_time variable but this time it gets the aimed arrival date and the aimed arrival time of the destination station	See the start_time validation
start	Integer	This variable is the UNIX time of the start_time	Use the start_time variable to get the UNIX time for the station
end	Integer	This is the same as the start_time instead it is for the end_time variable	Use the end_time variable to get the UNIX time for the station.
time_to_travel	Integer	This gives the travel time in minutes between the local station and the destination station	Subtract start from end
stations	String Array	The purpose of this array is to ensure that there are no repeats in the data. This is so that the user can look at the	Use a linear search to find a station within the array and if the station is not in the list then

		stations that they can go to.	append it to the end of the list
station_index	Integer	This stores the starting term of the services array. This ensures that the local station will not get into the array and also stations that come before the local station are omitted from the array as well	Loop through the list of calling points to get the index of the local station.

For this stage I will first make an algorithm that gets the timetable request and then accesses the calling points of the services. I decided for the first part of developing the stage, I will make it output all the stations that can be reached directly from the station. I will add time capabilities in the next part of development. The reason for this is so that I have a basic skeleton of what the program will do to get these stations before doing something more complex. Below is the pseudocode for the procedure.

```
ACCESS train_app.html
ACCESS transport_api

PROCEDURE get_stations(station)
        STRING ARRAY train_uid[25]
        INT station_index
        ARRAY data_3
        INT j
        REQUEST timetable AT station FROM transport_api
        ARRAY data_2 = timetable.results
        INT i = 0
        FOR i TO LEN(data_2)
                train_uid = data_2[i].uid
                REQUEST services WITH train_uid FROM transport_api
                data_3 = services.results
                j = 0
                FOR j TO LEN(data_3)
                        IF(data_3[j].station_name = station)
                                station_index = j
                        ENDIF
                ENDFOR
                INT j = station_index
                FOR j TO LEN(data_3)
                        OUTPUT(data_3[j].station_name)
                ENDFOR
        ENDFOR
ENDPROCEDURE
```

As you can see I will have to nest a request within a request. However although I haven't shown it here but I will also be making these requests within my initial request in order to create these requests. This is because of the way that AJAX works in order to put things onto a website. Below is the actual source code of the procedure.

```
get_request = function(station){
    let station_box = document.getElementById("station_box");
    let ourRequest = new XMLHttpRequest();
    ourRequest.open("GET", 'https://transportapi.com/v3/uk/places.json?query='+ station +'&type=train_station&app_id=8a247cdd&app_key
    ourRequest.onload = function() {
        var ourData = JSON.parse(ourRequest.responseText);
        var station_code = ourData.member[0].station_code;
        console.log(ourData.member[0].station_code);
        let second_request = new XMLHttpRequest();
        second_request.open("GET", "https://transportapi.com/v3/uk/train/station/"+ station_code +"///timetable.json?app_id=8a247cdd&
        second_request.onload = function() {
            var data_2 = JSON.parse(second_request.responseText);
            var j;
            console.log(data_2);
            let request_3 = new XMLHttpRequest();
            data_2.departures.all.forEach(function(listItem) {

                console.log("Request Sent");
                request_3.open("GET", "https://transportapi.com/v3/uk/train/service/train_uid:" + listItem.train_uid + "///timetable.
                request_3.onload = function () {

                    var data_3 = JSON.parse(request_3.responseText);
                    var j;
                    var station_index = 0
                    var htmlString;
                    console.log("loaded");
                    for (j = 0; j < data_3.stops.length; j++) {
                        if(data_3.stops[j].station_name == data_2.station_name)
                        {
                            station_index = j;
                            break;
                        }

                    }
                    for(j = station_index + 1; j < data_3.stops.length; j++)
                    {
                        /*htmlString = "<p>"+ data_3.stops[j].station_name + "</p>";
                        station_box.insertAdjacentHTML('beforeend', htmlString);*/
                        console.log(data_3.stops[j].station_name);

                    }


                }
                request_3.send();
                //console.log("Platform " + data_2.departures.all[i].platform + " for the " + data_2.departures.all[i].aimed_departur
```

As you can see there is a lot to the procedure. The first thing I did to make this procedure is I decided to make an AJAX request to get the station data again. Once I had done this, I decided to get the station code from the first station in the JSON data. This is because the timetable code needs the specific three letter code of the station in order to function. Once I had got the station code, I could now create a second AJAX request to get the timetable data from the station. The reason for making this call to the API is so that I could get each trains unique identification number. This is so that I could access the calling points of the stations. This will be needed later in development when I am getting the times between stations to tell the user which stations that they can go to within a certain amount of time. Once I had gotten the timetable data, I could then get the service data using the API. This is the request that gives me the stations that the train has called at.

In order to get all the stations that can be reached by one train, I decided that the app should only display the stations that the train calls at after the local

station. This is because the services function shows all the stops that the train will get to. To fix this problem, I decided to make the program find the local station using a for loop. When the local station is found then the index of the local station is set to the “station_index” variable. After that I could simply make a for loop starting at the next station from the station index and then run the code until the terminus of the station. This is repeated for every departure that I have found.

This procedure works but since it is just outputting the calling points of the services there seems to be a lot of repeats in the data. Therefore before I can work out the time between stations I will need to expand my procedure so that it only outputs a certain station once and not twice. Therefore I will need an array with all the stations that have been found. The Pseudocode below shows the amendments that I will have to make to ensure that the program does not repeat.

```
ACCESS train_app.html
ACCESS transport_api

PROCEDURE get_stations(station)
    STRING ARRAY train_uid[25]
    BOOLEAN flag
    STRING ARRAY stations[]
    INT station_index
    ARRAY data_3
    INT j
    REQUEST timetable AT station FROM transport_api
    ARRAY data_2 = timetable.results
    INT i = 0
    FOR i TO LEN(data_2)
        train_uid = data_2[i].uid
        REQUEST services WITH train_uid FROM transport_api
        data_3 = services.results
        j = 0
        FOR j TO LEN(data_3)
            IF(data_3[j].station_name = station)
                station_index = j
            ENDIF
        ENDFOR
        FOR j TO LEN(data_3)
            flag = FALSE
            INT k = 0
            FOR k TO LEN(stations)
                IF(data_3[j].station_name = stations[k])
                    flag = TRUE
                ENDIF
            ENDFOR
            IF(flag = FALSE)
                stations.append(data_3[j].station_name)
            ENDIF
        ENDFOR
        FOR j TO LEN(stations)
            OUTPUT(stations[j])
        ENDFOR
    ENDFOR
ENDPROCEDURE
```

From this Pseudocode I can see that I will need to make another for loop for this to work. Although it is not as efficient as it could be, I will make sure that the algorithm works first before I look at how I can make it more efficient. I will need to make sure that there isn't a limit to how big the array is so that I can store as many stations as I want it to.

```
console.log(data_3)
for (j = 0; j < data_3.stops.length; j++) {
    if(data_3.stops[j].station_name == data_2.station_name)
    {
        station_index = j;
        break;
    }

}
for(j = station_index + 1; j < data_3.stops.length; j++)
{
    flag = false;
    for(k = 0; k < stations.length; k++)
    {
        if(stations[k] == data_3.stops[j].station_name)
        {
            flag = true;
        }
    }
    if(flag == false)
    {
        stations.push(data_3.stops[j].station_name);
    }



}
```

Here is the code which gets rid of repeats in data. As you can see in order to fix the problem of repeats was relatively simple to code. All I needed to do was make a new array called stations. The idea of this is that the program will check whether the station name is eligible for being in the stations array. Another good thing is that JavaScript has a very useful function for this purpose as there is the push function which can put the station name into the main array. In order to check for repeats, I have the flag variable which is just a simple boolean function which checks whether there is a station that is already in the list. For this to work I have used a linear search for this purpose. Despite the fact that usually a linear search is not the most efficient form of searching algorithm, I felt that for this program it would be the best solution as a binary search would have meant that I would have had sort the stations into alphabetical order which could add to the amount of comparisons.

Now that I have an array containing all the stations that a person can get to from their local station. The next stage is filtering the array so that only stations within a certain amount of time are shown to the user. One of the things that I will need to take into account is that the expected time of arrival and expected time of departure is a string. Therefore, I will need to find a way of converting the string into a time function. From the time function I should be able to convert it into the time in minutes and therefore I can get the time it takes to get from one station to the other. The screenshot below shows the Pseudocode that is being used for this procedure.

```
ACCESS train_app.html
ACCESS transport_api
PROCEDURE access_time(start_time, end_time)
                INT time_travelled = end_time - start_time
                RETURN time_travelled
ENDPROCEDURE

FUNCTION get_stations
        //Most of these lines will be the same so I am just going to skip
        aheadto the relevant parts of the subroutine which will be changed.
        FOR i TO LEN(data_2)
                train_uid = data_2[i].uid
                REQUEST serivces WITH train_uid FORM transport_api
                data_3 = services.results

                FOR j TO LEN (data_3)
                        INT k = 0
                        FOR k TO LEN(stations)
                                IF(data_3[j].station_name = station)
                                        station_index = j
                                        start_time = data_3[j].departure_time
                                        BREAK
                                ENDIF
                ENDFOR
                FOR j TO LEN(data_3)
                        flag = FALSE
                        INT k = 0
                        end_time = data_3[j].arrival_time
                        FOR k TO LEN(stations)
                                IF(data_3[j].station_name = stations[k] OR access_time(start_end, end_time) > max_time)
                                        flag = TRUE
                                        BREAK
                                ENDIF
                        ENDFOR
                        IF(flag = FALSE)
                                stations.append(data_3[j].station_name)
                        ENDIF
                ENDFOR
        ENDFOR
        FOR i TO LEN(stations_
                OUTPUT(stations[j])
        ENDFOR
ENDPROCEDURE
```

As you can see, there is not a lot of difference to the procedure except for the addition of a new procedure which calculates the time taken to travel from one station to the other. It turned out to be a very simple algorithm to code thanks to the JSON formatted data of the API. The screenshot below shows the actual code that was written for this procedure.

```
calculate_time = function(start_time, end_time)
{
    var start = Math.round((new Date(start_time).getTime())/60000);
    var end = Math.round((new Date(end_time).getTime())/60000);
    var time_to_travel = end-start;

    return time_to_travel;


}
```

Here is the working time function for the program. The first thing that I had to do to make this work is to set the start and end variables into a unix time. A UNIX time is the amount of milliseconds away from 1st January 1970. In order to get this value I need to use the Date methods in JavaScript and specifically the getTime() method. For this I needed a specific string format which I will explain in the next screenshot. Since I wanted to work with minutes, I had to change the value in milliseconds to the value in minutes. Therefore, all I had to do was divide that value by 60000 as a second is 1000 milliseconds. Finally, I had to round both the start and the end to the nearest integer. After that was done, all I needed to do was subtract the start time with the end time and then the program would return the time taken to travel between the two stations.

```
for (j = 0; j < data_3.stops.length; j++) {
    if(data_3.stops[j].station_name == data_2.station_name)
    {
        station_index = j;
        start_time = data_3.stops[j].aimed_departure_date + ":" + data_3.stops[j].aimed_departure_time;
        console.log(start_time);
        break;
    }

}
for(j = station_index + 1; j < data_3.stops.length; j++) {
    flag = false;
    end_time = data_3.stops[j].aimed_arrival_date + ":" + data_3.stops[j].aimed_arrival_time;
    if(calculate_time(start_time, end_time) > max_time || data_3.stops[j].aimed_arrival_time == null)
    {
        flag = true;
    }
    else
    {
        for (k = 0; k < stations.length; k++) {
            if (stations[k] == data_3.stops[j].station_name) {
                flag = true;
                break;
            }
        }
    }
    if (flag == false) {
        stations.push(data_3.stops[j].station_name);
        station_codes.push(data_3.stops[j].station_code);
        journey_time.push(calculate_time(start_time, end_time));
    }


}
```

Here is the updated main code of the program. As you can see there are some minor changes to the code for this to work. For one thing I needed to make sure that I could make the strings for start_time and end_time. This is so that I could make the UNIX values which I discussed in screen 1. Since the JSON data that I am using for this entire code has the departure and arrival dates and times in strings it meant that I would have to make them into proper times that the computer could read which is explained up in the calculate time function. In order for the date methods to work I needed to make sure that the string was in the format YYYY-MM-DD:HH:MM and since the arrival time and day were both in separate areas of data, all I needed to do was concatenate the code and then send it to the calculate_time function. After that I just needed to add a few lines of code to ensure that the program would filter out stations in terms of time. The main reason for having a journey times array is so that I could make the user experience in the output page more useful by showing the time it would take to get the train. One final thing to point out is the piece of logic which says if(data_3.stops[j].expected_arrival_time = null). This is because there were some virgin trains services from London Euston

which had some expected arrival times as null. This caused problems as it showed the user that they could go to some stations that were over the time allowed for the user to travel to.

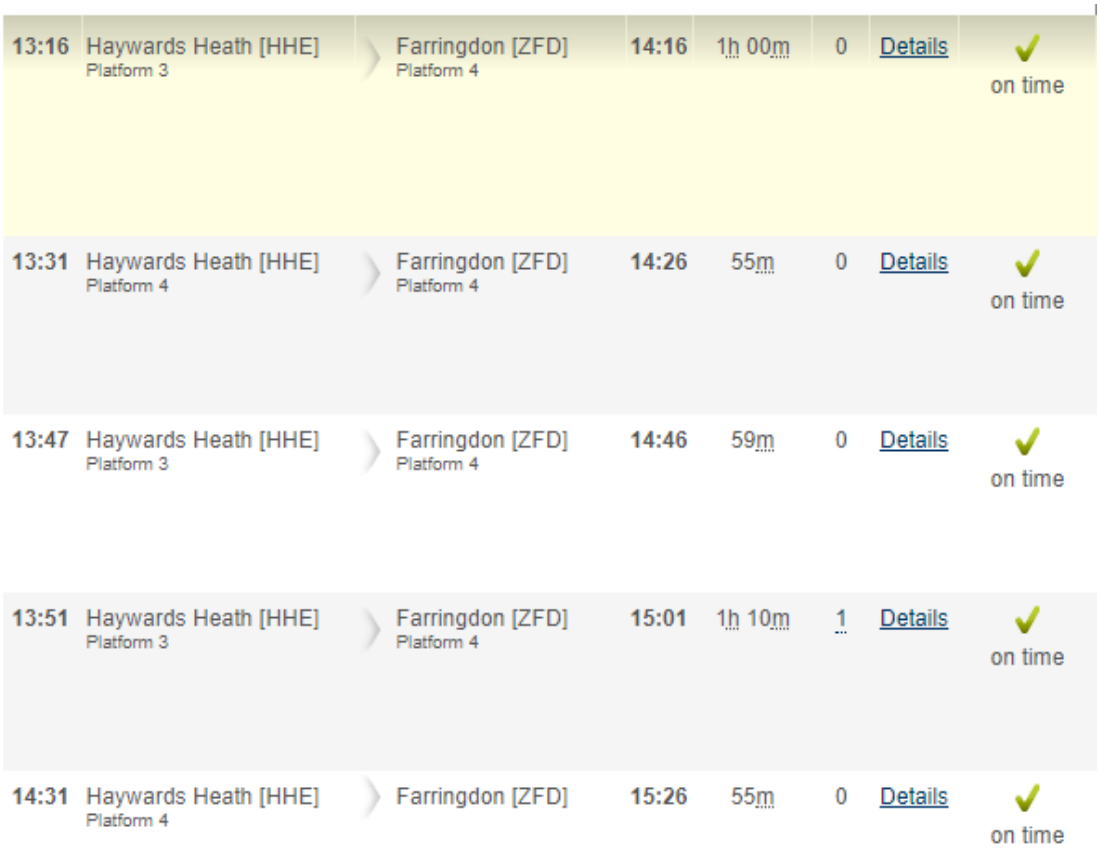
Testing

Test 1 and 2 – Ensure that the data is coming from one train and make sure that the journey time is within the time limit

The first test that I need to partake in is to make sure that the stations that come up are only stations with only one train when the direct connections box is checked. This is so that the user has the option of seeing the stations that they can only get to with just one direct connection. The next thing I needed to ensure was that the journey time was within the time limit.

Test video: https://youtu.be/iJ7ys2UqGoM

To ensure that the data is correct I loaded up the national rail enquiries data for a journey from Haywards Heath station and Farringdon station. This was because Farringdon was at the time limit for the amount of stations from Haywards Heath. The Screenshot below shows the data from National Rail enquiries.



13:16	Haywards Heath [HHE] Platform 3	Farringdon [ZFD] Platform 4	14:16	1h 00m	0	Details	on time
13:31	Haywards Heath [HHE] Platform 4	Farringdon [ZFD] Platform 4	14:26	55m	0	Details	on time
13:47	Haywards Heath [HHE] Platform 3	Farringdon [ZFD] Platform 4	14:46	59m	0	Details	on time
13:51	Haywards Heath [HHE] Platform 3	Farringdon [ZFD] Platform 4	15:01	1h 10m	1	Details	on time
14:31	Haywards Heath [HHE] Platform 4	Farringdon [ZFD]	15:26	55m	0	Details	on time

This data from national rail enquiries shows me that the time for getting to Farringdon is valid and the journey can be done with only one train. This is what I expected the program to do. However, I have noticed that some later journey's have shorter travel times. I think that the program has just taken data from the 13:16 service when there is a shorter service avaliable.

I will just make a quick test where I put 55 minutes as my maximum time to see whether Farringdon will come up in that test. The screenshot below shows the output.

Robbert Sinclair, here are th

Three Bridges Journey time: 8 minutes

Gatwick Airport Journey time: 13 minutes

East Croydon Journey time: 30 minutes

London Bridge Journey time: 44 minutes

London Blackfriars Journey time: 50 minutes

City Thameslink Journey time: 52 minutes

Farringdon (London) Journey time: 55 minutes

Plumpton Journey time: 10 minutes

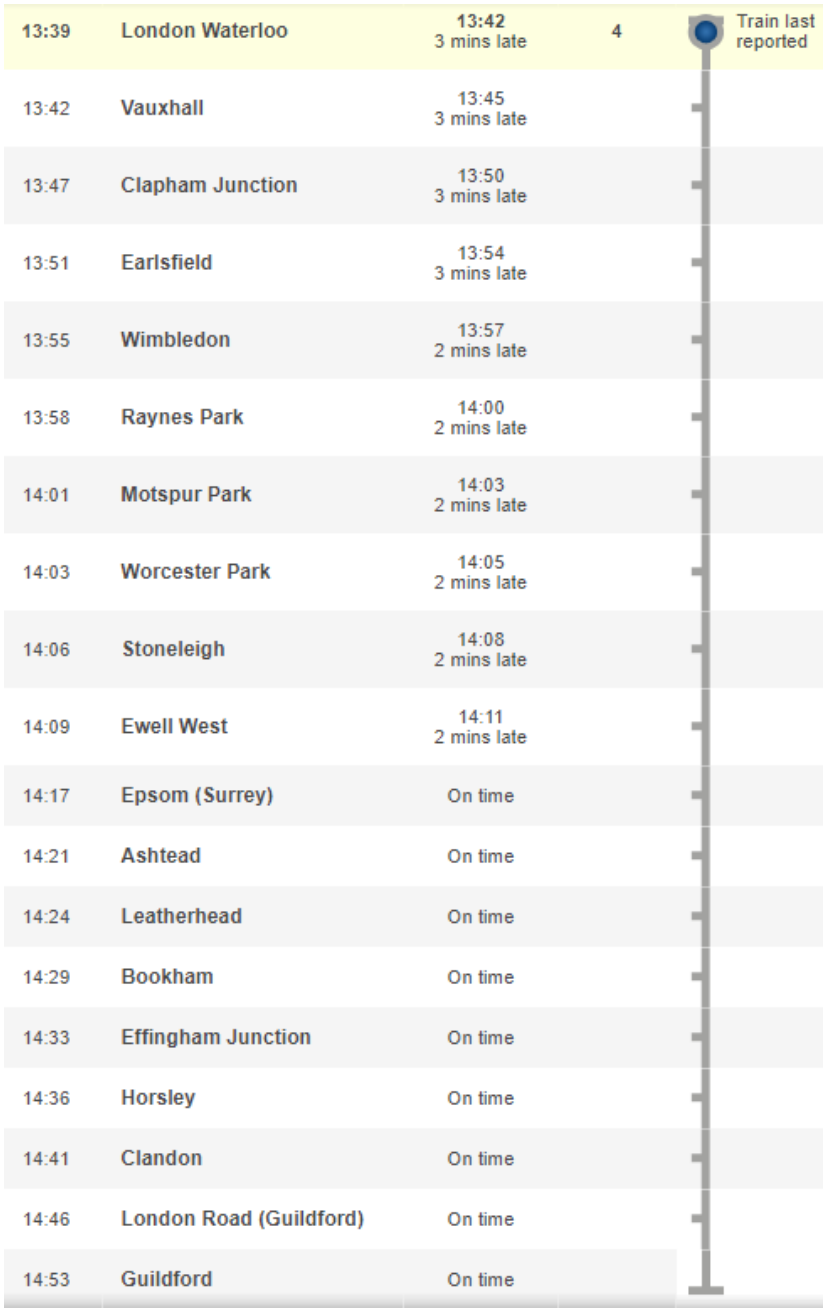
I can see from this run of the program that the program states that the journey time between Haywards Heath and Farringdon is 55 minutes. This is just as valid as the one hour journey time. In stage 5 I think I will try and find an algorithm that gets the quickest time instead of just the first time that fits the criteria. **Test Successful**

Test 3 – See how many stations you can get to from the busiest station in the UK

For this test, I am going to run the program to find the amount of stations from London Waterloo station. The reason why I have picked this station to run the program with is because London Waterloo is the busiest station in the UK. Therefore I will see how many stations I can go to within an hour of London Waterloo.

Test video: https://youtu.be/MaDoIqaHONo

As expected, a lot of stations came up from the stations. To ensure that this data is accurate, I am going to get the live departure board information from national rail enquiries and get a service at random. If these stations are on the list then the data will be accurate.



13:39	London Waterloo	13:42 3 mins late	4	Train last reported
13:42	Vauxhall	13:45 3 mins late		
13:47	Clapham Junction	13:50 3 mins late		
13:51	Earlsfield	13:54 3 mins late		
13:55	Wimbledon	13:57 2 mins late		
13:58	Raynes Park	14:00 2 mins late		
14:01	Motspur Park	14:03 2 mins late		
14:03	Worcester Park	14:05 2 mins late		
14:06	Stoneleigh	14:08 2 mins late		
14:09	Ewell West	14:11 2 mins late		
14:17	Epsom (Surrey)	On time		
14:21	Ashtead	On time		
14:24	Leatherhead	On time		
14:29	Bookham	On time		
14:33	Effingham Junction	On time		
14:36	Horsley	On time		
14:41	Clandon	On time		
14:46	London Road (Guildford)	On time		
14:53	Guildford	On time		

Here is the first randomly chosen service from national rail enquiries. From this information, I will expect that the last station to be on the list is Horsley. I will now look on the list of stations and check whether the stations up to Horsley are on the list of stations.

Vauxhall Journey time: 4 minutes

Queenstown Road (Battersea) Journey time: 7 minutes

Clapham Junction Journey time: 10 minutes

Earlsfield Journey time: 12 minutes

Wimbledon Journey time: 16 minutes

Raynes Park Journey time: 19 minutes

Motspur Park Journey time: 22 minutes

Worcester Park Journey time: 24 minutes

Stoneleigh Journey time: 27 minutes

Ewell West Journey time: 30 minutes

Epsom Journey time: 38 minutes

Ashtead Journey time: 42 minutes

Leatherhead Journey time: 45 minutes

Bookham Journey time: 50 minutes

Effingham Junction Journey time: 54 minutes

Horsley Journey time: 57 minutes

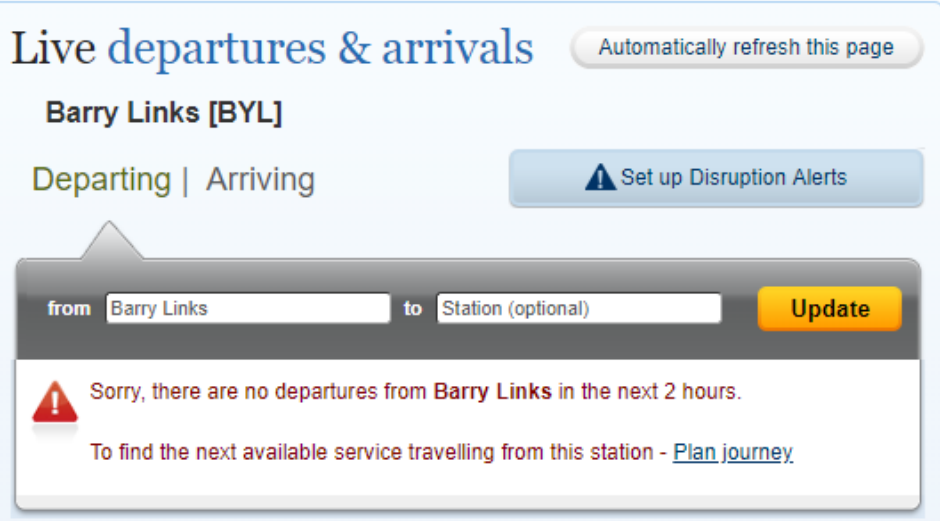
From this data I can see that I get all the stations form London Waterloo to Horsley and I don't get any more stations from this dataset. I can also see that the time between London Waterloo and Horsley is accurate. Therefore I can say that the program is accurate and works for the biggest station. Therefore it can work for smaller stations as well. **Test Successful**

Test 4 – Get the stations from the least used station in the UK

For this test, I am going to run the program for the lease used station in the UK. Currently that station is Barry Links in Aberdeenshire. This will show the other extreme and can show how the program reacts to a station with lots of trains compared to a station with a very limited service.

Test video: https://youtu.be/VivmdenE5aA

From the video I can see that there is a very different result to what I was expecting. This is because the program returned with no stations from that station. This unfortunately may not be a problem I can fix as the API only gives back the next 25 trains or the trains in the next 2 hours whichever limit is reached first. This goes to show that there is a very good reason why Barry Links is the least used station in the UK. This is because there is only one train a day. This means that this program may be limited with these remote stations that nobody really uses. To prove this, I have gotten the live departure board from Barry Links from national rail enquires.



This screenshot shows that national rail enquiries also has the problem that shows that there aren't as many departures from this station. Therefore this shows that this isn't a problem that I can avoid and it shows one of the limitations of the API. **Test Partially Successful**

Stage 3 Review

- Check whether the user has checked the direct connections checkbox **Partially met**
- Find all the stations that can be reached directly from the station and which fits the other criteria **Fully met**

I feel that this stage was successful in terms of development. I feel that I have managed to show the user the amount of stations that they can get to from one singular train. I have also shown that the information that the program shows is rather useful as it tells the user the station name and the amount of time it takes to get to the station.

One of the things that I will need to look at is the function that checks whether the direct connections button is pressed. I felt that since I this stage was only going to show the direct connections anyway and since I only made the program work for direct connections, I felt that I would get this part of the function working when I can get the function which gets stations that needs any change of trains. I feel that once I have done that, I can easily change the way the program gets the amount of trains with a simple if statement.

Stage 4 – Show the user additional information about the station

Development Plan

Main aims for this stage

- Get the user's input on what station that they want to see further information on.
- Get information from the API on what services you can get to get to that station.
- Make sure to only show the services that are under the time limit
- Indicate which service is the quickest route to the chosen station

The original idea of this stage was to get the amount of stations that the user can go to including stations where the user must change in order to get there. However, the problem was that I only have 1000 requests per day which may seem like a lot but considering that in order for the algorithms created in stage 3 to work, I need to make at most 27 requests. That means that I only have a certain number of tests that I could do. Therefore, I think that this is a better feature to make in terms of both user experience and in terms of API limitation to make this feature instead. Another thing that I will say about the feature not working is that I don't know how I would be able to make an algorithm which would tell me where I would be able to change trains. I feel that this will also help inform the user on what trains the user can take in order to get to that station.

Test Description	Type of Test	Test Data	Expected Result
Get the departure information of three stations and compare it to	Valid, White Box	Stations array	I will expect the app to show me the exact information that I need and for the information to

national rail enquiries			match that of national rail enquiries
Ensure that the departure information only shows trains that will be under the maximum time for the station	Valid	Stations array	I will hope to see that the services that I can take will be within the specified time limit
Check that the quickest route to the station is correct	Valid	Stations array	I will expect the algorithm to tell the user which service is quickest and that the station data is accurate.

Implementation Stage 4

Below is a list of variables which I may need for this stage.

Variable Name	Type	Purpose	Validation
request_4	XMLHttp Request	The purpose of this variable is to get the departure data from the local station which includes trains that call at the station that the user has selected	I will validate this by getting the station code of the local station and the station code of the selected station.
destination	String	This will be the station code of the destination station. The purpose of this variable is so that I can call the API with the data	I will validate this code by having the variable be set by the link the user clicks on in order to get to the program.

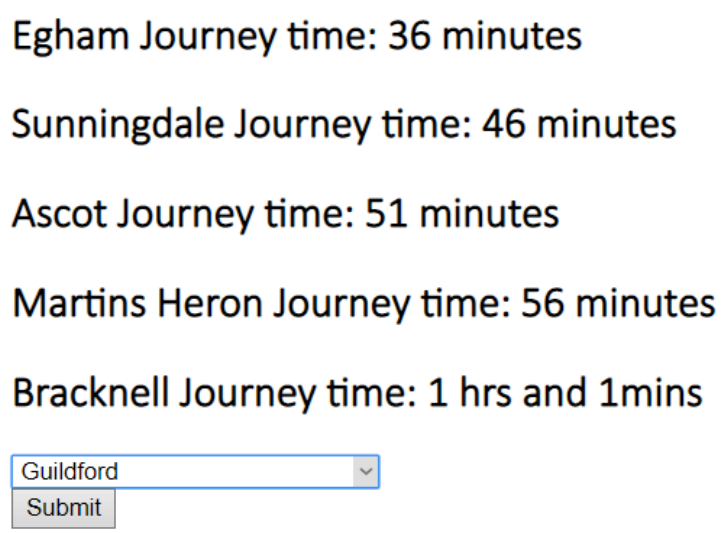
		that the user has requested	
data_4	JSON data	This is the data from request_4 but parsed into JSON so that I can easily manipulate the data inside	Get the data from request_4
HTMLString	HTML	The purpose of this variable will be to turn the data of the departure information into an h2 element which can be written onto the webpage and therefore displaying the data to the user.	Get the data from the departures API call to get this information
toc	String	This is will store the name of the train operating company so that the user will know what train operating company is running the service.	The variable will take the train operating company from the call to the API
Departure_time	String	This will be put into the HTMLString to let the user know which train they will need to get in order to get to their chosen station	Like most of these variables, the data will come from data_4 (The departure time from the API)

Arrival_time	String	This will show the user what time the train will arrive at	The data will come from data_4
time_to_travel	Integer	Stores the journey time in minutes in order to check whether the train is within the maximum time allowed	This will be calculated by getting the UNIX times of both departure_time and arrival_time and then subtracting the result
mode	String	This variable will store whether the mode of transport is a train or a bus. This is to give the user the warning that the train they may be getting is a rail replacement bus	This information will come from data_4
platform	Integer	This tells the user what platform they may expect their train to depart from	If there is a value of a platform from data_4 then the platform will be displayed in the app. If the platform then the platform will not be displayed

The first thing that I am going to do in this stage is I am going to first make a call to the API and get all the services that are serving the local station which call at the selected station. This is so that I can create a basic outline of the program and then I can modify the program so that it can fit my needs. Below is the pseudocode for the algorithm that I need in order for this to work.

```
ACCESS train_app.html
ACCESS transport_api

PROCEDURE get_services(local_station, destination_station)
    API_REQUEST request_4 = REQUEST departures WITH destination_station FROM local_station FROM transport_api
    DATA data_4 = request_4.results
    INT i = 0
    FOR i TO data_4.length
        OUTPUT "Platform: ", data_4.platform, " for the ", data_4.departure_time, " ", data_4.operating_company_name, " service to ", data_4.destination
    ENDFOR
ENDPROCEDURE
```

As you can see, there is not much to this piece of Pseudocode. This is because at this stage we are just getting the data from the API and are then just outputting it to the user in a way that is easy to understand. This is so that the user can find out what train they need to get in order to get to their destination. In later stages I will ensure that the information will only include services that gets the user to the destination within the maximum time that they specified. Below is the source code and additions to the GUI that I created to create an extra input to the program.



Here is the slight change that I made to the output screen in order to get this feature to work. I decided that the best way at the current moment to get the user's input on what station they would want to travel to was to create a drop-down list containing all the stations included in the list of stations in the output screen. I will make sure to make this drop down list more user friendly later on but I just thought that in order to test this new feature I should just make a small new drop down list to get a basic input. The screenshot below shows the new HTML that I created for this function.

```
        document.write("<form>");
        document.write('<select id = "stations">');
        for(m = 0; m < stations.length; m++)
        {
            document.write('<option value = ' + station_codes[m] + '>'+ stations[m] + '</option>');
        }
        document.write("</select>");
        document.write("</form>");
        document.write('<input type="submit" id="destination_finder" onclick = get_services(station_code)>')
    }
```

In order to create the new form in the output screen, I made use of the document.write function. This is so that I could easily put in some new HTML containing all the stations that the program had output. Another thing is that making the HTML using the document write function meant that I could make a dynamic set of options for the drop-down box. It also meant that I could simply set the value of the option to the three-letter code of each station which meant that it was easy to use the data in my fourth request to the API. The last thing that I had to do in this part of the program was to make a submit button which starts up the function which gives the user the services to that particular station. The get_services function takes in the parameter station_code to get the three-letter code of your local station. The screenshot below shows the get_services function in detail.

```
get_services = function(local_station){
    var destination = document.getElementById("stations").value;
    var m;
    var destination_name;
    for(m = 0; m < station_codes.length; m++)
    {
        if(station_codes[m] == destination)
        {
            destination_name = stations[m];
            break;
        }

    }
```

```
console.log(destination);
var request_4 = new XMLHttpRequest();
request_4.open("GET", "https://transportapi.com/v3/uk/train/station/"+ local_station + "///timetable.json?app_id=8a247cdd&app_key=9e9645bc611d43785491c64665bbe300&calling_at="
    + destination + "&train_status=passenger", false);
request_4.onload = function(){
    var data_4 = JSON.parse(request_4.responseText);
    console.log(data_4);
    document.write("<h1>Here are all the services to " + destination_name + " from " + station_name + "</h1>");
    for(m = 0; m < data_4.departures.all.length; m++)
    {
        if(data_4.departures.all[m].platform == null)
        {
            var htmlstring = '<h2>The ' + data_4.departures.all[m].aimed_departure_time +
                ' ' + data_4.departures.all[m].operator_name + ' service to '
                + data_4.departures.all[m].destination_name + '</h2>';
        }
        else
        {
            var htmlstring = '<h2>The ' + data_4.departures.all[m].aimed_departure_time + ' ' + data_4.departures.all[m].operator_name
                + ' service to '  + data_4.departures.all[m].destination_name
                + ' departing from Platform ' + data_4.departures.all[m].platform + '</h2>';

        }
        document.write(htmlstring);
    }

}
request_4.send();
```

Shown above is my code for the get services function. One of the first things that I had to do was create the “destination” variable which would take the value of the station that I had selected from the drop down list. This was the main reason why I decided to set each of the station’s values to their three-letter codes as it would mean that I could just put destination into the API call later on in the algorithm. For there to be a friendlier interface for the user, I used a for loop to make a linear search through the station_codes array and once it hit the code that was being used then the destination_name variable would be set to whatever was in the same index in the stations array. This means that the app will display “Here are the stations to Brighton from Haywards Heath” rather than “Here are the stations to BTN from Haywards Heath”.

Once that part of the function was working, it was time to make yet another AJAX request, this time I am using the call for the departures from the station, but I am only including services that stop at the station that the user has selected. That way the user can find out which train they would need to catch in order to reach their destination. Therefore I made use of the calling_at parameter to add in the destination variable. Once that AJAX request was working, I made sure that the user could see the reults of the request. Therefore, I made it loop through the JSON formatted data to output all the results from the request and then I made sure that if the platform number was null, then the code will not output “departing from platform XX” so that the user would not get confused over what platform they need to get.

Now that part of the programme is working, I need to extend it so that It only shows the services that fit within the time limit that the user has put on the app. This is because there is no point in showing the user all the services if

some of the services are going to take longer to get to the station than the maximum time that the user has specified. The Pseudocode below will show you how I plan to implement this algorithm.

```
ACCESS train_app.html
ACCESS transport_api

PROCEDURE get_services(local_station, destination_station, max_time)
        API_REQUEST request_4 = REQUEST departures WITH destination_station FROM local_station FROM transport_api
        DATA data_4 = request_4.results
        INT i = 0
        FOR i TO data_4.length
                API_REQUEST request_5 = REQUEST services WITH train_uid FROM transport_api
                DATA data_5 = request_5.results
                start_time = data_4.departure_time + data_4.departure_date
                end_time = data_5.arrival_time + data_5.arrival_date
                IF(calculate_time(start_time, end_time) < max_time) THEN
                        OUTPUT "Platform: ", data_4.platform, " for the ", data_4.departure_time, " ", data_4.operating_company_name, " service to ", data_4.destination
                ENDIF
        ENDFOR
ENDPROCEDURE
```

As you can see the Pseudocode for this part of the algorithm only has a few new features added to it in order for it to work. For one thing I will need to make hopefully the last AJAX request of the build in order to get the data on the services to check whether the train gets the user to their destination in the correct amount of time. For this algorithm I am going to make use of the calculate_time algorithm that I made in stage 3 as it should be able to work for this algorithm so there will be no need to make a new algorithm for this procedure.

```
var request_5 = new XMLHttpRequest();
request_5.open("GET", "https://transportapi.com/v3/uk/train/service/train_uid:" + data_4.departures.all[m].train_uid
    + "///timetable.json?app_id=8a247cdd&app_key=9e9645bc611d43785491c64665bbe300&darwin=false&live=false", false);
request_5.onload = function () {
    var data_5 = JSON.parse(request_5.responseText);
    console.log(data_5);
    var destination_index;
    var a;
    console.log(station_name);
    for(a = 0; a < data_5.stops.length; a++)
    {
        if(data_5.stops[a].station_code == local_station)
        {
            start_index = a;
            start_time = data_5.stops[a].aimed_departure_date + ":" + data_5.stops[a].aimed_departure_time;
            break;
        }
    }
    console.log(start_index);
    console.log(start_time);
    //let start_time = data_5.stops[start_index].aimed_departure_date + ":" + data_5.stops[start_index].aimed_departure_time;
    for (a = 0; a < data_5.stops.length; a++) {
        if (data_5.stops[a].station_code == destination) {
            destination_index = a;
            break;
        }
    }
    let end_time = data_5.stops[destination_index].aimed_arrival_date + ":" + data_5.stops[destination_index].aimed_arrival_t
    console.log(end_time);
    if (calculate_time(start_time, end_time) <= time.value) {
        train_uids.push(data_5.train_uid);
    }


}
request_5.send();
```

Here is the source code for my penultimate AJAX request of the project. One of the first things that I had to do was get the station index of the starting station.

This is essentially the same as the algorithm which searches through the calling points as was used in stage 3 to get the amount of stations that the user can get to. This is so that I could get the departure time from the station and so that I could concatenate both the departure data and the departure time. The next thing that I did which differentiates this code to the code from stage 3 is that I find the destination index instead of just going through all the stations to check whether the user could visit all those stations. Once I had found the destination from the data, I could then just get the arrival time and arrival date and then concatenate the variables. Now that I have the arrival time and the start time, I could then use the calculate_time function that I made in stage 3 to check whether the train gets the user to the station within the time that they were travelling to. If the train gets to the selected destination within the time that the user has to travel, then the specific train id is added to an array. The code below shows the updated output function for this feature of the app.

```
for(m = 0; m < data_4.departures.all.length; m++)
{
    console.log("Loop");
    var a;

    for(a = 0; a < train_uids.length; a++)
    {
        if(data_4.departures.all[m].train_uid == train_uids[a])
        {
            if (data_4.departures.all[m].platform == null) {
                var htmlstring = '<h2>The ' + data_4.departures.all[m].aimed_departure_time +
                    ' ' + data_4.departures.all[m].operator_name + ' service to '
                    + data_4.departures.all[m].destination_name + '</h2>';

            }
            else {
                var htmlstring = '<h2>The ' + data_4.departures.all[m].aimed_departure_time + ' '
                    + data_4.departures.all[m].operator_name
                    + ' service to ' + data_4.departures.all[m].destination_name
                    + ' departing from Platform ' + data_4.departures.all[m].platform + '</h2>';

            }
            document.write(htmlstring);

        }
    }
    document.write('</table>');
}
```

Using the train_uids array that I explained in the previous function, I could then search through the departures information for the trains by comparing the unique id of each train. I decided to use another linear search through the list as there could be multiple trains that fit the criteria that the user has set and so the user should be shown all the stations that fir their criteria.

The St Pancras Bug

When testing the algorithm on certain stations during the development of this feature, I noticed that there were some logical errors when I asked for information on trains to Paris or Brussels from London St Pancras. I found that when I asked for information on the next departures to these stations, it would give me the sentence “Robbert Sinclair here are all the departures to [destination] from London St Pancras” but it didn’t show me the actual departures that were from this station. Upon inspection of the code, I saw that the API was recognising the trains, however the API was referring the London St Pancras as “SPX” when the default station code is “STP”. This meant that the algorithm couldn’t find the code for St Pancras which meant that the start time is null so therefore the calculate_time function could not calculate the time between stations so therefore, the Eurostar departures were not being output.

Solution

In order to fix this problem, I had to make special if statements to check whether the train is going to Eurostar Destinations like Paris, Brussels etc. Once this check was carried out, the algorithm would then check whether the station code was “STP” and if that is true, then the station code would be changed into “SPX” so that the algorithm will be able to find London St Pancras. The interesting thing is that none of the other stations where Eurostar trains call at have this problem and have just one station code. I then also made sure that at the start of the algorithm that the station code would revert to “STP” if the station code is “SPX”. The screenshot below shows the code that I wrote to solve this problem.

```
    if(destination_name == "Bruxelles Midi")
    {
        document.write("<h1>De volgende vetrekken vanuit " + station_name + " naar Brussel Zuid</h1>");
        document.write("<h1>Le prochaine d&eacute;parts depuis " + station_name + " &agrave; Bruxelles Midi");
        if(local_station == "STP")
        {
            local_station = "SPX";
        }
    }
    else if(destination_name == "Amsterdam Cs")
    {
        document.write("<h1>De volgende vetrekken vanuit " + station_name + " naar Amsterdam Centraal</h1>");
        if(local_station == "STP")
        {
            local_station = "SPX";
        }
    }
    else if(destination_name == "Paris Nord" || destination_name == "Marne La Vallee")
    {
        document.write("<h1>Le prochaine d&eacute;parts depuis " + station_name + " &agrave; " + destination_name);
        if(local_station == "STP")
        {
            station_code = "SPX";
        }
```

Once I had managed to fix the St Pancras Bug, I wanted to introduce the feature that would tell the user which service they could take is the fastest. To do this, I feel that I am going to have to make use of arrays which stores the journey times and then it puts “Fastest Time” to the train which has the shortest time and if there are more than one train that gets the user to the place in the shortest time then I will go for the train that is departing first. The screenshot below shows the Pseudocode for the algorithm that will do this.

```
ACCESS transport_api
ACCESS train_app.html

PROCEDURE get_shortest_time(station, destination, max_time)
    INT shortest_time
    FOR a TO LEN(data_4)
        XMLRequest request_5 = REQUEST calling_points WITH train_uid FROM transport_api
        ARRAY journey_times
        DATA data_5 = request_5.response
        FOR i TO LEN(data_5)
            IF(data_5[i].name == station)
                STRING start_time = data_5[i].departure_date + ":" + data_5[i].departure_time
            ELSE IF(data_5[i].name == destination)
                STRING end_time = data_5[i].arrival_date +  ":" + data_5[i].arrival_time
            ENDIF
        ENDFOR
        IF(calculate_time(start_time, end_time) < max_time)
            APPEND calculate(start_time, end_time) TO journey_times
        ENDIF

    ENDFOR
    shortest_time = journey_times[0]
    FOR a TO LEN(journey_times)
        IF(journey_times[a] > shortest_time)
            shortest_time = journey_times[a]
        ENDIF
    ENDFOR
ENDPROCEDURE
```

As you can see from this screenshot, I am just making modifications to the existing code but instead now I am going to add in a feature that gets all the journey times and puts them into the array. Once the journey times are put

into the array and once all the information has been received, then the program will go through the journey_times array and then will see which of the journey times are the shortest. As said above, I will consider the train which is departing first to be the train with the shortest time if there are more than one train that has the fastest train. The screenshot below shows the source code for this part of the program.

```
            console.log(end_time);
            if (calculate_time(start_time, end_time) <= time.value) {
                train_uids.push(data_5.train_uid);
                journey_times.push(calculate_time(start_time, end_time));
            }


        }
        request_5.send();



}
shortest_time = journey_times[0];
for(m = 0; m < journey_times.length; m++)
{
    if(journey_times[m] < shortest_time)
    {
        shortest_time = journey_times[m];
    }
}
```

As you can see, there isn't a lot that has been changed in this part of the program. The only real difference is the addition of the journey_times array to get the journey times between the start station and the destination station. This was so that I could make a for loop which could go through all the times of the trains to that destination to see which train gets the user to their destination in the fastest possible time to get to their destination. This works so that I can amend the output area so that if the certain train gets the user in the shortest time then it will tell the user that that train gets them there in the shortest possible time. The screenshot below shows the output.

Here are all the services to Aberdeen from Glasgow Queen Street

The 15:45 Scotrail service to Aberdeen departing from Platform 6

The 16:45 Scotrail service to Aberdeen departing from Platform 4 SHORTEST TIME

As you can see, the program will give the user all the trains in the next two hours that the user can take in order to get to Aberdeen from Glasgow Queen Street. It has also shown the user that the 16:45 gets them to Aberdeen in a quicker time than the 15:45.

Testing Stage 4

Now that I have developed the features that I wanted to make in this stage, it is now time to test the features that I have created.

Tests 1 and 2 – Get the departures to one of the stations from a busy local station and then compare the data to National Rail Enquiries. Check whether the shortest time to the station is correct.

For this test, I will be looking at the depatures from Brighton to Gatwick Airport. My main reason for carrying out this kind of test is that there are loads of trains that run between Brighton and Gatwick each with varying speeds depending on what stations they stop at on the way. For this test I will expect the Gatwick Express services to be the fastest trains avaliable. This is because the Gatwick Express services don't stop at any other station. The following screenshot shows the results of the test.

Here are all the services to Gatwick Airport from Brighton (East Sussex)

The 13:08 Thameslink service to Cambridge departing from Platform 5

The 13:18 Gatwick Express service to London Victoria departing from Platform 4 SHORTEST TIME

The 13:26 Thameslink service to Bedford departing from Platform 6

The 13:33 Southern service to London Victoria departing from Platform 3

The 13:48 Gatwick Express service to London Victoria departing from Platform 3 SHORTEST TIME

The 14:03 Southern service to London Victoria departing from Platform 4

The 14:08 Thameslink service to Cambridge departing from Platform 5

The 14:18 Gatwick Express service to London Victoria departing from Platform 4 SHORTEST TIME

The 14:26 Thameslink service to Bedford departing from Platform 6

The 14:33 Southern service to London Victoria departing from Platform 3

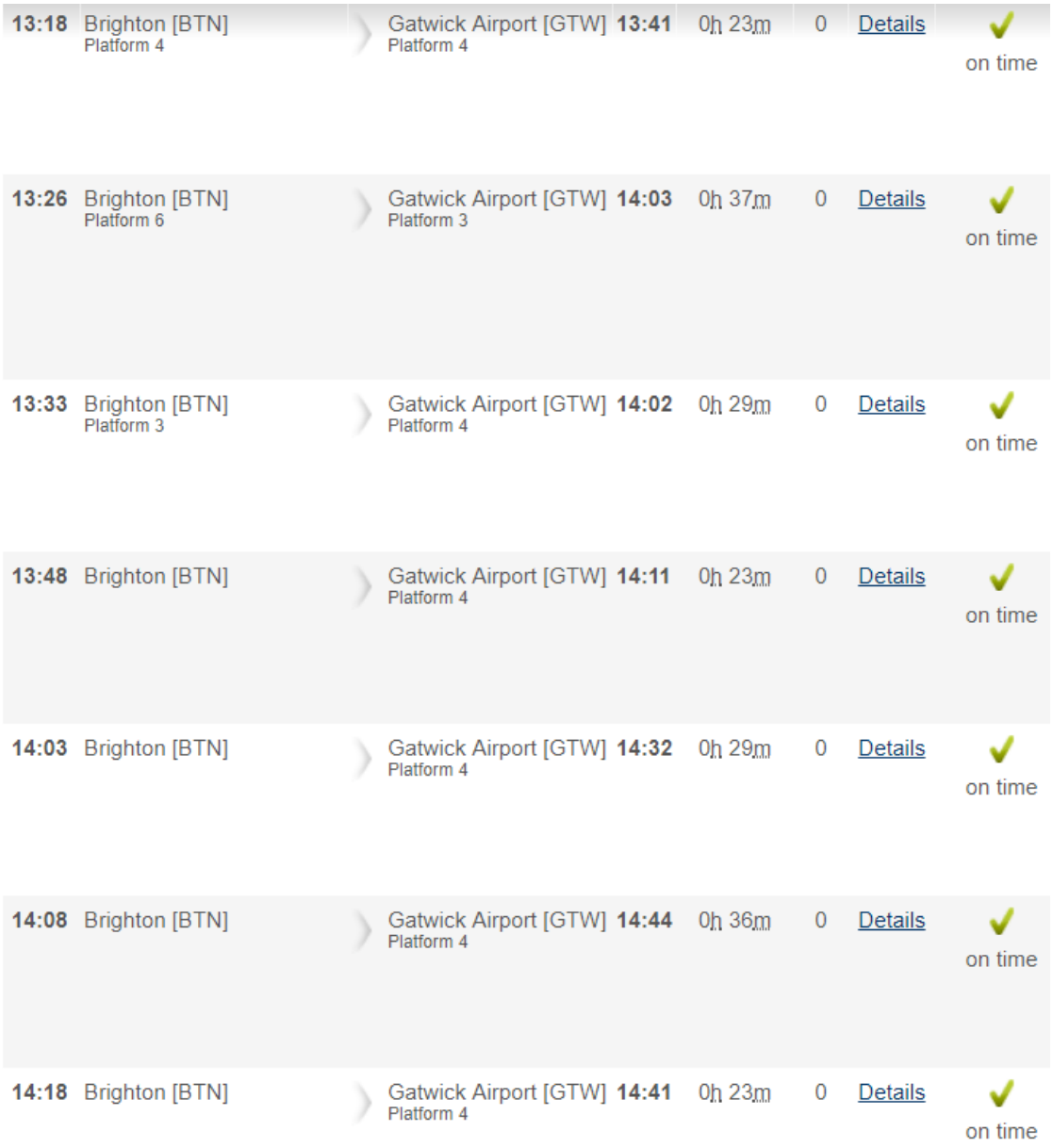
The 14:48 Gatwick Express service to London Victoria departing from Platform 3 SHORTEST TIME

The 14:56 Thameslink service to Bedford departing from Platform 6

The 15:03 Southern service to London Victoria departing from Platform 4

The 15:08 Thameslink service to Cambridge departing from Platform 5

From what I can see from the results, the program is saying that the Gatwick Express services have the fastest time to get to Gatwick Airport. This is what I expected to happen. However, I will get the same information on National Rail Enquiries to see whether the results are truly accurate.



13:18	Brighton [BTN] Platform 4	Gatwick Airport [GTW]	13:41	0h 23m	0	Details	on time
13:26	Brighton [BTN] Platform 6	Gatwick Airport [GTW]	14:03	0h 37m	0	Details	on time
13:33	Brighton [BTN] Platform 3	Gatwick Airport [GTW]	14:02	0h 29m	0	Details	on time
13:48	Brighton [BTN]	Gatwick Airport [GTW] Platform 4	14:11	0h 23m	0	Details	on time
14:03	Brighton [BTN]	Gatwick Airport [GTW] Platform 4	14:32	0h 29m	0	Details	on time
14:08	Brighton [BTN]	Gatwick Airport [GTW] Platform 4	14:44	0h 36m	0	Details	on time
14:18	Brighton [BTN]	Gatwick Airport [GTW] Platform 4	14:41	0h 23m	0	Details	on time

This screenshot shows me that the information is accurate. I think this because the information on national rail enquiries shows me that the train times are correct and it also shows me that the Gatwick Express services take the shortest time to get the user to Gatwick Airport from Brighton. Therefore this information shows me that the information is accurate. **Test Successful**

Test 3 – Ensure that the departures are within the time limit that the user has imposed

For this test I will get the stations from Brighton, but this time I am going to set my maximum time with the shortest time between the stations. For this test I will hope that only the Gatwick Express services will be shown. I will then

compare the data with national rail enquiries to ensure that the trains can get to the destination in that time and to ensure that the information is accurate.

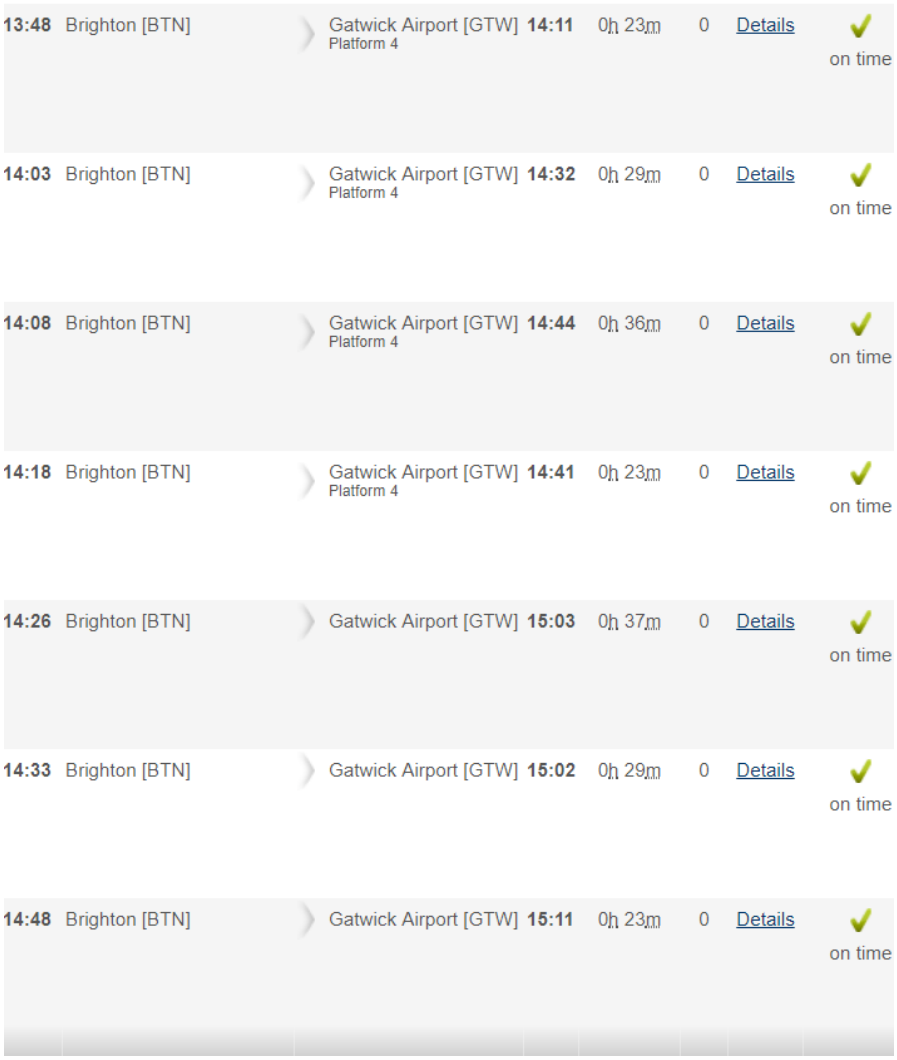
Here are all the services to Gatwick Airport from Brighton (East Sussex)

The 13:48 Gatwick Express service to London Victoria departing from Platform 3 SHORTEST TIME

The 14:18 Gatwick Express service to London Victoria departing from Platform 4

The 14:48 Gatwick Express service to London Victoria departing from Platform 3

The 15:18 Gatwick Express service to London Victoria departing from Platform 4



I can see that in this instance of running the program, there is a significant decrease in the amount of services that can be used in order to create the list of stations. I can also see that the trains that have had a journey time higher than the shortest time has been excluded from the list. This means that the user will only be able to choose between the trains with the shortest travel time between stations. Therefore the information is accurate. **Test Successful**

Stage 4 Review

Main Aims

- Get the user's input on what station that they want to see further information on. **Fully Met**
- Get information from the API on what services you can get to get to that station. **Fully met**
- Make sure to only show the services that are under the time limit **Fully met**
- Indicate which service is the quickest route to the chosen station **Fully met**

I feel that I have really managed to achieve what I set out to do in this part of development. I really feel that I have managed to make function that will be helpful for the users using this website. As I said at the beginning of the development of this part of development, I do feel that this is a much better feature to implement in terms of the limitations in the API. I think it shows the user not only where they can go to from their local station, but also which train to catch in order to get to their intended destination. However, one of the things I will say about this feature is that it is limited to only the departures to that station in the next 2 hours. Therefore, it may not be as useful if people are checking out when to travel in the next few days rather than in the next 2 hours. However, since this is a limitation of the API, I will not be able to really fix this problem.

Stage 5 – Create a better user interface

Main aims for this stage

- Add some features in order to make the users experience better

For this stage, I am going to add a series of features to the app to ensure that I can make the user experience as easy as possible. I will first implement features that I personally feel will add to the user experience and then I will ask some people to test my system to see what they would want to make the app useful to the public.

Feature 1 – Suggested Stations

Design section

One of the first feature that I personally want to implement is a function which gives the user a list of stations if the user puts in a string which returns more than one result from the API. Therefore I want to make a menu that will ask the user what station they meant in order to ensure that the user has the correct station. My main reason for this is that in order to get Brighton station, I must put in BTN otherwise I get New Brighton in the Wirral or when I put in "London" without specifying which station, I get a station called Lee in the suburbs. Therefore, I hope that this feature will help the user get the station they want rather than a station that is in a completely different part of the country. Below is the tests that I will need in order to ensure that the feature is working.

Test Description	**Type of Test**	**Test Data**	**Expected Result**
Put in a very specific name or three letter code for the station	White Box	Station	I will expect the program to just get the stations that the user can go instead of getting the user to the suggested stations feature
Put in a set of three letters which could be in a lot of stations	White Box	Station	I will expect the program to take me to the window which asks me which station I meant
Select a station from the list of suggested stations and click submit	White Box Valid	Station	I will expect the program to display the relevant information for that station
Try and input a station that is not on the list	Black Box Invalid	Station	I will expect the program to reject my invalid input and will ask me to try again.

For this feature I will need a wide range of variables to store the suggested stations. The table below shows the variables that I will need.

Variable Name	Type	Purpose	Validation
suggested_stations	String array	The purpose of this variable is to store all the names of the results of the places API. This is so that the user can see what stations are suggested	Get the stations from the first API call
suggested_stations_code	String Array	The purpose of this variable is to store the three letter code of the stations. This is so that I can set the value of the drop down box term can be the three letter code so that I can easily just put the three letter code into the depatures call	Get the station_code from the JSON response from the places API call
Suggested_label	HTML	This is a variable so that the JavaScript can refer to the suggested stations label. This is so that I can remove it once I don't need it anymore	N/A
Suggested_choice	HTML	This variable will refer to the drop down list	N/A

button	HTML	This variable will refer to the button variable	N/A

Here is the Pseudocode for how this function will work. My idea is that I will extend the validate stations function in order to make this program work. This is becuase I already have a variable which keeps track of the amount of stations.

```
ACCESS transport_api
ACCESS train_app.html

PROCEDURE validate_station(station)
    ourRequest = REQUEST train_stations WITH station FROM transport_api
    data_1 = ourRequest.results
    INT l = LEN(data_1)
    IF(l > 1)
        OUTPUT "We received " + l + " results. Which station did you mean"
        DROPDOWN suggestions = new DROPDOWN
        FOR(i TO LEN(data_1))
            suggestions.add(data_1[i].name)
        ENDFOR
        SUBMIT button = new SUBMIT
        IF(button.pressed = TRUE)
            station = suggestions.value
    ELSE IF(l == 1)
        station = data_1[0].name
    ELSE
        OUTPUT("INVALID STATION")
    ENDIF
ENDPROCEDURE
```

As you can see from the pseudocode above, I will be making amendments to the validate station function that I created in order to ensure that the user can select the station that they meant when putting in a broader input instead of a specific station. Below is the source code for the function.

```
if (validate_values(fname, lname, age) == false) {
    alert("INVALID DATA");
}
else if(len == 0)
{
    alert("Invalid station");
}
else {


        document.write("<!DOCTYPE html>");
        document.write("<link rel=\"stylesheet\" href=\"output.css\"/>");
        document.write("<h1 id ='suggested_label'>We found " + len + " results. Please select the station you meant from the list</h1>");

        document.write("<select id=station_choice>");

        for(i = 0; i < len; i++)
        {
            document.write("<option value =" + suggested_stations_code[i] +">"+ suggested_stations[i] + "</option>");
        }
        document.write("</select>");
        document.write("<br><br>");
        document.write("<button type = 'button' id = 'station_choice_submit' >Click to submit</button>");
        document.getElementById('station_choice_submit').addEventListener("click", function() {set_station(fname.value, lname.value, time.value);
        var suggested_label = document.getElementById('suggested_label');
        var suggested_choice = document.getElementById('station_choice');
        var button = document.getElementById('station_choice_submit');
        suggested_label.parentNode.removeChild(suggested_label);
        suggested_choice.parentNode.removeChild(suggested_choice);
        button.parentNode.removeChild(button);
        }, true);
```

Above is the amended code in the getValues() function. The main changes to this function is to add some more HTML to the website. This is to display the stations that came up in the API search for places in a dropdown box so that the user can get the station that they meant. For the button, I have created an event listener which will run the set_station function which I will explain later and then it gets rid of the HTML that has just been made as the user will have made their choice of station. The main reason for this was that I tried to let the user dynamically change their code, but it was causing some logical errors with the amount of services to the user's destination station, this was because it wouldn't show the depatures from two stations which don't have a direct connection. The screenshot below shows the amendments that I have made to the validate_station function.

```
validate_station = function(station) {
    let count = 0;
    let ourRequest = new XMLHttpRequest();
    ourRequest.open("GET", 'https://transportapi.com/v3/uk/places.json?query='+ station +'&type=
    ourRequest.onload = function() {
        var data_1 = JSON.parse(ourRequest.responseText);
        console.log(data_1);
        l = data_1.member.length;
        for(i = 0; i < l; i++)
        {
            suggested_stations.push(data_1.member[i].name);
            suggested_stations_code.push(data_1.member[i].station_code);
        }


    }
    ourRequest.send();

    return l;

}
```

As you can see, there is a bit of change to the validate_station. One of the first changes that I have made is that I have made it so that it returns the variable "l" instead of a boolean variable based on the length of l. I have also made two arrays called "suggested_stations" and "suggested_stations_code". This is so that I could easily create the dropdown box discussed in the amendments to the getValues function. This is so that I can easily get the values needed so that I can give the user the stations that they can go to. This leads to the next function that I want to talk about which is the set_stations function. The screenshot below shows the code for it.

```
set_station= function(fname, lname, max_time)
{
    document.write("<h2>Loading</h2>");
    let station_suggestion = document.getElementById("station_choice").value;
    for(i = 0; i < suggested_stations.length; i++)
    {
        if(suggested_stations_code[i] == station_suggestion)
        {
            station_name = suggested_stations[i];
            station_code = suggested_stations_code[i];
        }

    }

    get_request(station_suggestion, fname, lname, max_time);
}
```

This is the only new function that I have needed to make when implementing this feature. This function is called when the user presses the submit button when the program returns the number of results. The first thing it does is get the value of the drop down box which was set to the three letter code of the station. Once it has this value, it goes through the suggested_stations array to set the station_name and station_code variables. This is so that I can easily get the station names and the three letter code as I already know that these stations are legitimate so I don't need anymore validation. Finally, the function calls the get_request function. The main reason for this is that I have also made amendments to the get_request function in order to make it work. The screenshot below shows the source code for get_request.

```
get_request = function(station, fname, lname, max_time) {
    let station_box = document.getElementById("station_box");
    let flag = false;
    var start_time;
    var end_time;
    console.log(station);
    console.log("Yes");


    document.write("<h1>" + fname + " " + lname + ", here are the stations that you can get to from " + station_name + " in the next two hours</h1>");

    let second_request = new XMLHttpRequest();
    second_request.open("GET", "https://transportapi.com/v3/uk/train/station/" + station + "///timetable.json?app_id=8a247cdd&app_key=9e9645bc611d43785491c64
    second_request.onload = function () {
        var data_2 = JSON.parse(second_request.responseText);
        var i;
        let k;

        let request_3 = new XMLHttpRequest();
        data_2.departures.all.forEach(function (listItem) {


            request_3.open("GET", "https://transportapi.com/v3/uk/train/service/train_uid:" + listItem.train_uid + "///timetable.json?app_id=8a247cdd&app_ke
            request_3.onload = function () {
                var data_3 = JSON.parse(request_3.responseText);
                var j;
                var station_index = 0
                var htmlString;
                console.log(data_3.stops.length);
                console.log(data_3);
```

Here is the amendments that I had made to the get_request function. One of the only things that I changed in this function is that I removed the first AJAX

request which got the station name. The main reason for the removal of this AJAX request is that I have already validated that the stations exist and I have made it so that the user has to go through the suggested stations menu even if only one station is found. Also I have made it so that the value of the dropdown box is the three letter code of the station. Therefore, I can go straight to the AJAX request which gets the departures from that station. This also means that I have more requests to play with as it gets rid of redundant requests. Therefore I can test my program more.

Testing – Feature 1

Test 1 – Put in a very specific three letter code to the local station box and see what happens.

For this test I am just going to put in the full name of a specific station. The idea is that if the station is very specific then the program will go directly to the stations that the user can get to or the suggested stations page will only have one result.

Test Video Here: https://youtu.be/4LXL96N-7Bo

From the test video I can see that when I put in a very specific kind of station I can see that I am taken to the suggested station page and there is only one result. Although this may not be the best solution, it was the easiest way that I could get the function to work without having to make the same AJAX request twice in a row. Therefore I feel that this is the best solution at the current moment. **Test Successful**

Test 2 – Put in a broad term into the local station box and see what stations come up when I put it in.

For this test, I am going to input the word “London” this is because there are loads of stations which have the word “London” in it. Therefore I will expect that the program will give me a list of stations to choose from to get what the user requested.

Test Video: https://youtu.be/YKd5P8WJ-nU

From the test video I can see that there is a lot of stations which come up when I look for a station with the word “London”. This was what I expected to happen. This also means that the user will be able to select the station that they meant from the list to ensure that they can easily get the data from their local station.

Test 3 – Select a station from the list and then see what happens

For this test I will just be looking at what comes up when I select a station from the results of the search for a station. When I click the station name, I will expect the program to gives the user the amount of station that you can get to from that station. I will also expect that the program will remove the form when selecting the stations so that the user cannot select a station when the user has already selected the station. This is because when I tried to allow the user to select a new station while the data for one station is being displayed, the stations would get mixed up as it would include the stations from the previous station and the current station. Also when I selected a station from the current station, the program would get the departures from the previous station and then there wasn't any data. Therefore I hope that the program will get rid of the form for the suggested station.

Test video: https://youtu.be/Kq3KU9yKggo

From the video I can see that the suggested stations area was removed as soon as the data for London Paddington showed up. Therefore the user can see the information for stations coming out of Paddington and they cannot change their local station afterwards. This means that the user can use the program without there being any bad glitches in the code. **Test Successful**

Test 4 - Try and input a station that wasn't on the suggested station list.

For this test I am going to get the suggested stations that comes up when I type in the word "London". In this test I am going to try and get a station that wasn't on the list.

Test Video: https://youtu.be/R3H_YIe8xkE

From the video I can see that I cannnot select a station that is not on the list. I even tried typing the name of a station that is not on the list and I could not get any other station. Therefore I can see that the feature is being executed well enough to only limit the user's choice to the suggested stations list. **Test Successful**

Stage 5 Feature 1 Review

I feel that this feature has been successful, I have managed to get the program to work so that I can put a full name station or a broad term and get the station suggestion feature to give the user all the stations that came up in the results. One of the things I would say is that I should have a feature that if the

station name only sent one response, then the program should just go straight to the output feature without going to the suggested stations window. But other than that, I feel that this part of the program was very successful.

Feature 2 – Multilingual signs for stations served by Eurostar Services.

The main aim of this feature was to appeal to tourists from different countries especially for Eurostar services. Therefore, I have made it so that the label for the services to the destination station will display the information in English and French for services to Paris and Dutch for services to Amsterdam and Brussels. I thought that this would be useful as there will be a significant amount of people using the Eurostar services that would want to have the information given to them in their own language. The screenshot below shows the code that I utilised in order to create this feature.

```
document.write("<h1>Here are all the services to " + destination_name + " from " + station_name + "</h1>");
if(destination_name == "Bruxelles Midi")
{
    document.write("<h1>De volgende vetrekken vanuit " + station_name + " naar Brussel Zuid</h1>");
    document.write("<h1>Le prochaine d&eacute;parts depuis " + station_name + " &agrave; Bruxelles Midi");

}
else if(destination_name == "Amsterdam Cs")
{
    document.write("<h1>De volgende vetrekken vanuit " + station_name + " naar Amsterdam Centraal</h1>");

}
else if(destination_name == "Paris Nord" || destination_name == "Marne La Vallee")
{
    document.write("<h1>Le prochaine d&eacute;parts depuis " + station_name + " &agrave; " + destination_name


}
```

As you can see, there is only a small amount of code that was needed in order to implement this feature. However, there is one thing of note that I must address. This is the codes of the accents. This is because, you cannot just put in the accent into the sentence and expect the website to render that accent. This is because the characters are running on ASCII letters which means that it only has 128 characters to play with. If you want to have Unicode characters like ‘é’ you must put in a code like “à”. Once you have put in this notation then the Unicode value will come up in the website. The screenshot below shows an example of what the website looks like when you ask for services to Paris.

Here are all the services to Paris Nord from London St Pancras International

Le prochaine départs depuis London St Pancras International à Paris Nord

As you can see from the screenshot, the website first gives the user the information in English and then it gives the user the information in French. I feel that this will improve the user experience because as I said above, the Eurostar services have a lot of foreign traffic so by giving people the option to see their information in their preferred language will really help.

Feature 3: Map support

Main aims:

- Get the first 30 stations of the stations list on the map in the right location
- Get a pop-up window to tell the user where the stations on the list

In the required features section in the success criteria, it states that I should have map support to make the user experience a lot better. Therefore I need to find a new API in order to create a map. For this feature I have decided to use the "Location IQ" API. This is because it is free and has the exact same functionality as Google Maps. This is because Google Maps is expensive to use and I would much rather use something that is free with all the features rather than having to pay for a well known brand.

Test Description	Type of Test	Test Data	Expected Result
See whether the location of the stations are correct	Black Box Testing, Valid	Location of the stations	The stations will be in the correct places on the map
See whether the popup windows have the correct information	White Box Testing, Valid	Station Price, Time taken	The windows should show the station name as well as the time taken to get there.

Implementation Feature 3

For this feature to work, there will need to be a few variables that I will need in order to make sure that this feature will work. The table below shows the variables that I will need.

Variable Name	Type	Purpose	Validation

map	Leaflet map	This is the variable which initialises the map. This will be using the leaflet library which is part of the location IQ API	
Key	String	This variable will store the key that I have to the location IQ API	N/A
local_lat	Integer	This stores the latitude of the local station so that I can center the map on the station	Get the data from the places call of the API
local_long	Integer	This stores the longitude of the local station so that I can center the map on the station	Get the data from the places call from the API
stations_lat, stations_long	Integer array	The purpose of these arrays is so that I can make markers on where the other stations will be. This is because it stores the latitude and longitude of the stations	Get the data from the places call from the API.
request_7	XMLHttpRequest	This variable will get the data from the transport API to get the latitude and	N/A

		longitude of the stations to put onto the map	
data_7	JSON Data	This variable will store the response from request_7. This will turn the response text into JSON	Response Text from request_7
Local_icon	Leaflet icon	The main part of this variable is to define the icon which shows the user the picture for the local station	N/A
calling_at_icon	Leaflet icon	This variable will act in the same way as local_icon but it will store the picture for any station which is calling at another station	N/A

Now that I have planned my variables. I needed to plan out my algorithm. The Pseudocode below shows how the code was going to work.

```
ACCESS train_app.html
ACCESS transport_api
ACCESS leaflet.js
ACCESS location_iq
ARRAY stations[] //This was intialised in an earlier stage and is global so I can refer to
it in the procedure //
PROCEDURE create_map()
        // Get the local_latitude and the local_longditude
        ARRAY stations_lat[], stations_long[]
        XMLHTTPRequest request_7 = REQUEST train_stations WITH local_station FROM transport_api
        JSON data_7 = request_7.response
        INT local_lat = data_7.latitude
        INT local_long = data_7.longitude

        //Get the coordinates for the next stations
        INT i = 0
        FOR i TO LEN(stations)
            request_7 = REQUEST train_stations WITH stations[i] FROM transport_api
            data_7 = request_7.response
            stations_lat.APPEND(data_7.latitude)
            stations_long.APPEND(data_7.longitude)
        ENDFOR

        //Create the map
        OBJECT map = NEW leaflet map, center = [local_lat, local_long]
        OBJECT marker = NEW leaflet marker, coordinates = [local_lat, local_long], text = local_station, ADDTO(map)
        i = 0
        FOR i TO LEN(stations_lat)
            marker = NEW leaflet marker, coordinates = [stations_lat[i], stations_long[i], text = stations[i], ADDTO(map)
        ENDFOR
ENDPROCEDURE
```

As you can see, there is a lot of extra requests that I am making to the transport API. This is because the departures and train services calls don't give me the latitude and longitude. Therefore I will need to call the places requests for every single station. This is something that I will touch on when I show the source code for the create_map function. But before that, I made some amendments to the validate_stations function to get the local latitude and longitude.

```
validate_station = function(station) {
    let count = 0;
    let ourRequest = new XMLHttpRequest();
    ourRequest.open("GET", 'https://transportapi.com/v3/uk/places.json?query='+ 
    ourRequest.onload = function() {
        var data_1 = JSON.parse(ourRequest.responseText);
        console.log(data_1);
        l = data_1.member.length;
        for(i = 0; i < l; i++)
        {
            suggested_stations.push(data_1.member[i].name);
            suggested_stations_code.push(data_1.member[i].station_code);
            suggested_lat.push(data_1.member[i].latitude);
            suggested_long.push(data_1.member[i].longitude);
        }


    }
    ourRequest.send();

    return l;

}
```

I didn't do too much to the validate_stations function but the one amendment that I did do was create two new arrays called suggested_lat and suggested_long so that once the user has selected their stations, the local_lat and local_long variables will be set to their terms in the suggested_lat and suggested_long arrays.

```
create_map = function()
{

    console.log(stations);
    var request_7 = new XMLHttpRequest;
    if(stations.length < 30)
    {
        for(i = 0; i < stations.length; i++)
        {

            request_7.open("GET", 'https://transportapi.com/v3/uk/places.j
            request_7.onload = function()
            {
                var data_7 = JSON.parse(request_7.responseText);
                if(data_7.member.length > 0)
                {

                    console.log(data_7);
                    stations_lat.push(data_7.member[0].latitude);
                    stations_long.push(data_7.member[0].longitude);
                }


            }
            request_7.send();

        }
    }
    else
    {
        for(i = 0; i < 30; i++)
        {
            request_7.open("GET", 'https://transportapi.com/v3/uk/places.j
            request_7.onload = function()
            {
                var data_7 = JSON.parse(request_7.responseText);
                if(data_7.member.length > 0)
                {

                    console.log(data_7);

                    stations_lat.push(data_7.member[0].latitude);
                    stations_long.push(data_7.member[0].longitude);
                }

            }
            request_7.send();
        }

    }
```

Here is the code for when I get the latitude and longitude of each of the calling points. One of the first things you may have noticed is that I have placed a check on whether the length of the stations array was over 30 or not. This is because I only have a certain amount of calls with my API that I get in a day and so therefore, I decided to put an arbitrary cap on the amount of stations that could be shown in order to show which stations the user can go to. If I had unlimited requests, I would make sure to include all the stations that the user could go to. Once I had made that check I made sure that for every item in the stations array up to the 30th term in the array, the program would request the places information from the API and then it would take the latitude and longitude values and append them to the stations_lat and stations_long arrays which will be used to create the map.

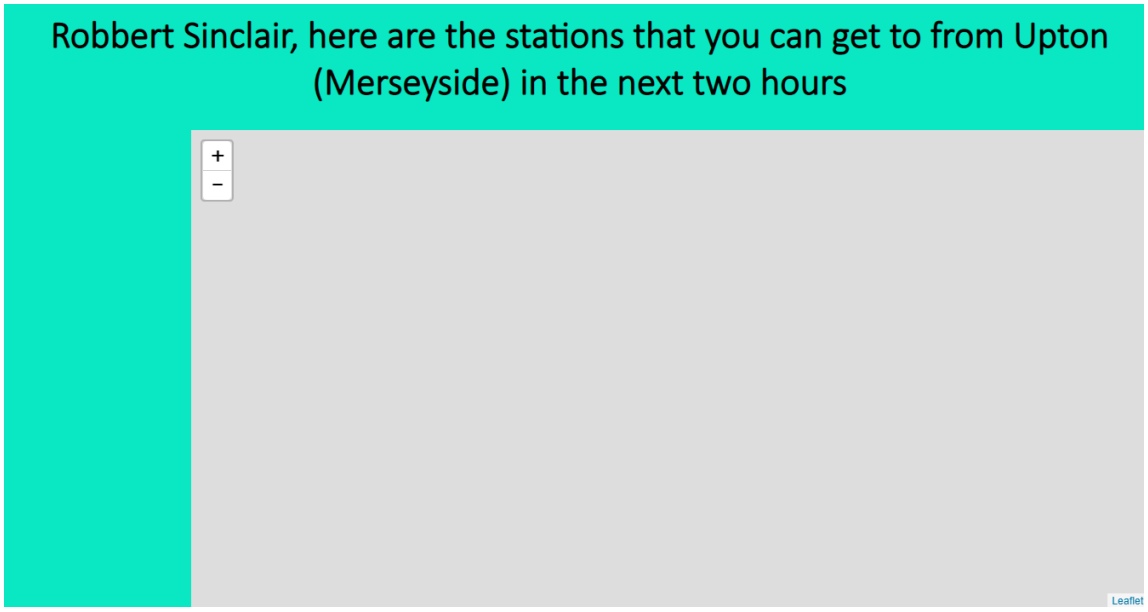
```
const key = '5e8368d318410e';
const streets = L.tileLayer.Unwired({key: key, scheme: "streets"});
let map = L.map('map', {
    center: [local_lat, local_long],
    zoom:10,
    layers: [streets]

});
let marker = L.marker([local_lat, local_long]).addTo(map);
marker.bindPopup(station_name);
for(i = 0; i < stations_lat.length; i++)
{
    marker = L.marker([stations_lat[i], stations_long[i]]).addTo(map);
    marker.bindPopup(stations[i]);
}

L.control.scale().addTo(map);
```

Here is the code which actually creates the map. The first thing I did was I set the “key” variable to my API key to the location IQ API. Once I had initialised that I then needed to create a tile layer. The way leaflet maps work is that you can have both street maps or satellite maps depending on what layer you want. As you can see it is pretty easy to set up as you only need the API key and the streets. Once I had the layer, it was time to set up the map. In my HTML I have created a tag which tells the website where the map is going to go. I gave this tag an id called “map”. Therefore all I had to do was put map at the start to tell the browser where this map was going to go. Once that was done I set the centre to the latitude and longitude of the local

station and I set the zoom level to 10. In leaflet, there is a scale of 1 – 18 in terms of zoom level. For example 1 is the entire planet and 18 refers to a specific building. I felt that zoom level 10 was the right choice as it would show me a good view of the surrounding area and most of the stations would be on this map. Finally I set up the markers. These markers will point out to the user where the stations would be. First I put a marker in the local coordinates and then using a loop, I put all the markers in the places where the other stations would be. The purpose of the “bindPopup” function is to create a popup to show the station name.

Error 5 – The Map Creation Error

While creating the map, I had noticed an error which was different to the test file that I had made to see whether the map would work. I didn’t know what the problem was at first as the code that I had written for the main project was almost identical to the code that I had in my test. I even tried looking at the leaflet documentation to see what was going on. The screenshot below shows what my code looked like when the code wasn’t working.



As you can see, I only had a grey box with a control scale for zooming in to the map and the stations that you can go to weren't there. When I checked the console. I found this error.

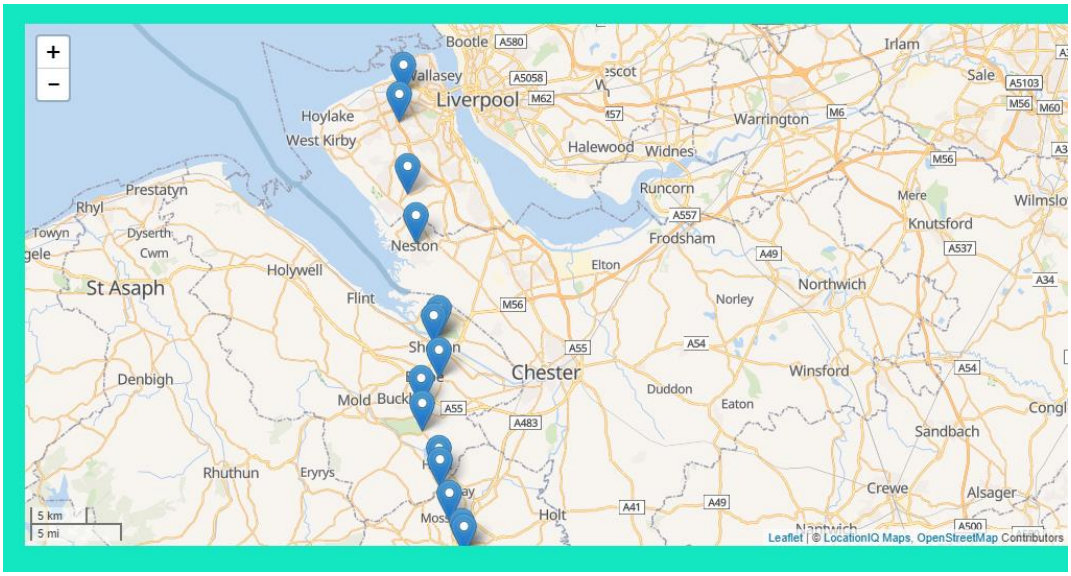
```
▶Uncaught TypeError: t.replace is not a function                    leaflet.js:5
    at Object.template (leaflet.js:5)
    at e.getTileUrl (leaflet.js:6)
    at e.createTile (leaflet.js:6)
    at e._addTile (leaflet.js:6)
    at e._update (leaflet.js:6)
    at e._setView (leaflet.js:6)
    at e._resetView (leaflet.js:6)
    at e.onAdd (leaflet.js:6)
    at e._layerAdd (leaflet.js:6)
    at e.whenReady (leaflet.js:6)
```

As you can see, the error is coming from the file “leaflet.js”. This caused me a lot of confusion as it was saying that the error was coming from the library from the map API. This meant that I didn’t know what was causing this reaction as the console wasn’t giving me any hints as to where the problem was coming from.

Solution

After many hours comparing my test file and the project code, I finally noticed the problem I was happening. It turned out to be a very simple fix as in the L.tilelayer line shown above, I had forgotten to put in the phrase “Unwired”. Apparently for the map to work, I needed to make sure that the tile layer was unwired which I hadn’t done. This is because this type of maps comes from a company called unwired labs which creates the map API so therefore I needed to specify that it was an unwired map.

One of the things that I noticed when I was making the map was that the default pointer meant that it was hard to differentiate between pointers. The screenshot below shows what the map looked like before I made custom icons.



As you can see, this version of the map isn't very useful in differentiating the local station with all the other stations. Therefore, I decided that I would draw my own pointers. The pointers are shown in the table below:

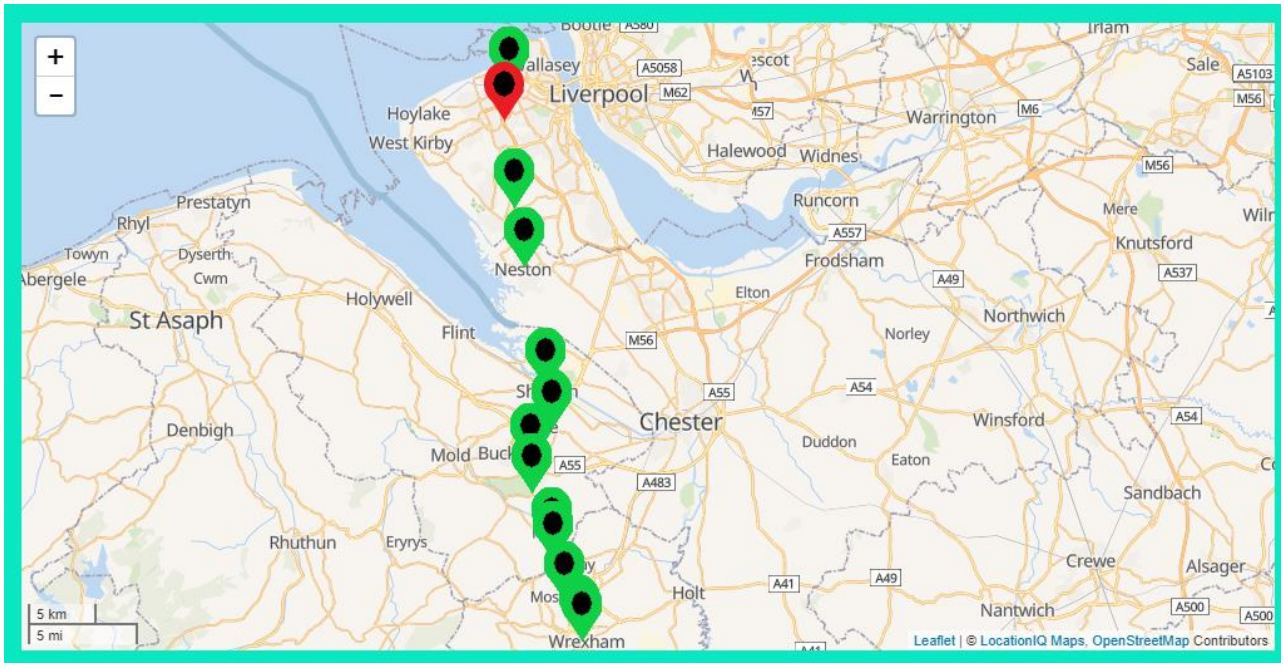
	
This pointer was created to show the local station.	This pointer was created to show the stations in which the trains called at.

Now that I had my new pointers, I then had to put these pointers into the map. The source code below shows the code which was needed in order to put the pointers in the map.

```
let map = L.map('map', {
    center: [local_lat, local_long],
    zoom:10,
    layers: [streets]

});
var local_icon = L.icon({
    iconUrl: 'local_station.png',
    iconSize: [32, 48],
    iconAnchor: [16, 48],
    popupAnchor: [0, -48]
});
var calling_icon = L.icon({
   iconUrl: 'calling_at_station.png',
    iconSize: [32, 48],
    iconAnchor: [16, 48],
    popupAnchor: [0, -48]
});
let marker = L.marker([local_lat, local_long], {icon:local_icon}).addTo(map);
marker.bindPopup(station_name);
if(stations_lat.length > 0)
{
    for(i = 0; i < stations_lat.length; i++)
    {
        if(stations_lat[i] == 0 && stations_long[i] == 0)
        {
            console.log("No Pointer");
        }
        else
        {
            marker = L.marker([stations_lat[i], stations_long[i]], {icon:calling_icon}).addTo(ma
            marker.bindPopup(stations[i]);
        }

    }
}
```

As you can see, there is quite a bit to making the code work. For one thing, I need to intialize the icons. The first thing that I need to do is make sure that the pointer image is set to the “local_station.png” image or the “calling_at_station.png” image. The next thing that I had to do was set the size of the image. Since my image was 32 by 48, I put that in as the size. The next thing to do was set the “iconAnchor”. This was the part of the code which sets where the pointers were going to be placed. The main reason why I put 16 and 48 was because those where the relative coordinates that I wanted in order fo the pointer to point at the station. Finally, all I had to do next was to set where the popup was which I set to the very top of the picture.

The next thing that I had to do was to ensure that the right pointer had the correct image allocated to it. Therefore, all I had to do was adjust the marker line in the code to set the icon to “local_icon” or “calling_at_icon” to show the correct pointer. The screenshot below shows the final map image.

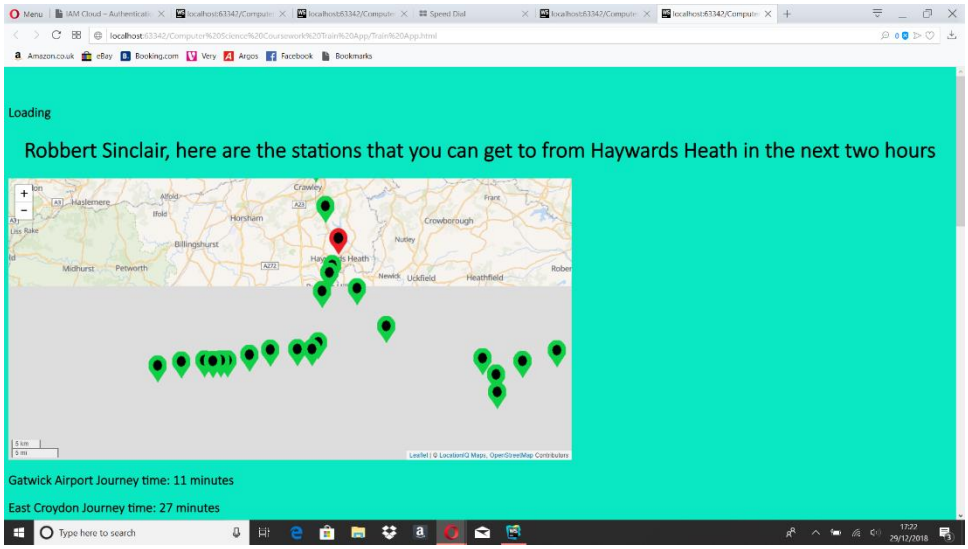


As you can see from the screenshot above, there is a significant difference from the initial screenshot. I feel that this is an improvement from the initial screenshot as there are specific pointers to differentiate the local station and the stations that you can get to from that station. With the default icons, it was hard to find out where the stations were compared to the local station. With these pointers it is easier to see where the stations are compared to the local station.

Testing Stage 5 Feature 3

Test 1 – Check the location is correct

For this test, I am going to look at the map and I will select stations at random to see whether the train station is in the right place. For this test I will look at the stations going from Haywards Heath station in West Sussex. The screenshot below is the results of that test.



Error 6 – The map doesn't load properly.

One of the things that I had noticed when trying to test this map is that the map for some reason is only loading half of the map. The problems seemed to start when I was creating the CSS for the website and it seems to be that afterwards the map hadn't loaded properly.

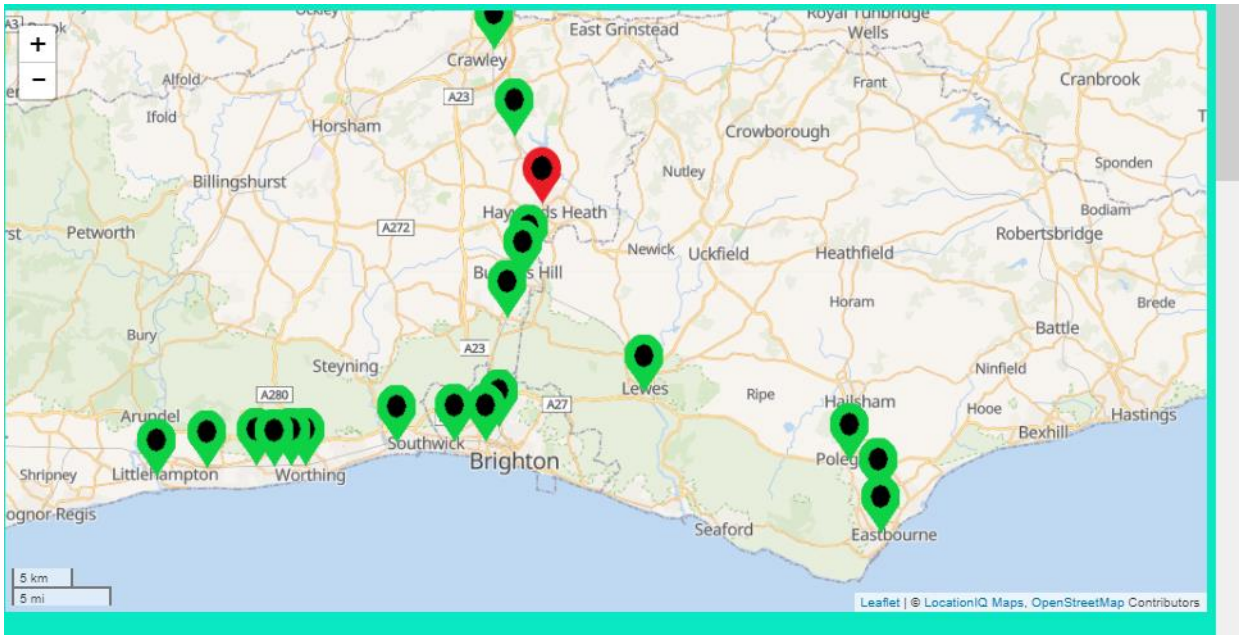
Solution

It turns out that it was a very simple fix. I had a look at the leaflet documentation and I realised that all I needed to do was add a function called invalidateSize. What this does is that after the size of the map is changed, this function can then adjust the map. This will ensure that the map will be optimised after the size is changed. The code below shows the amendments that I had to make.

```
document.write("<style> #map{height: 500px; width : 1000px; float : right;} </style>");
map.invalidateSize(true);
```

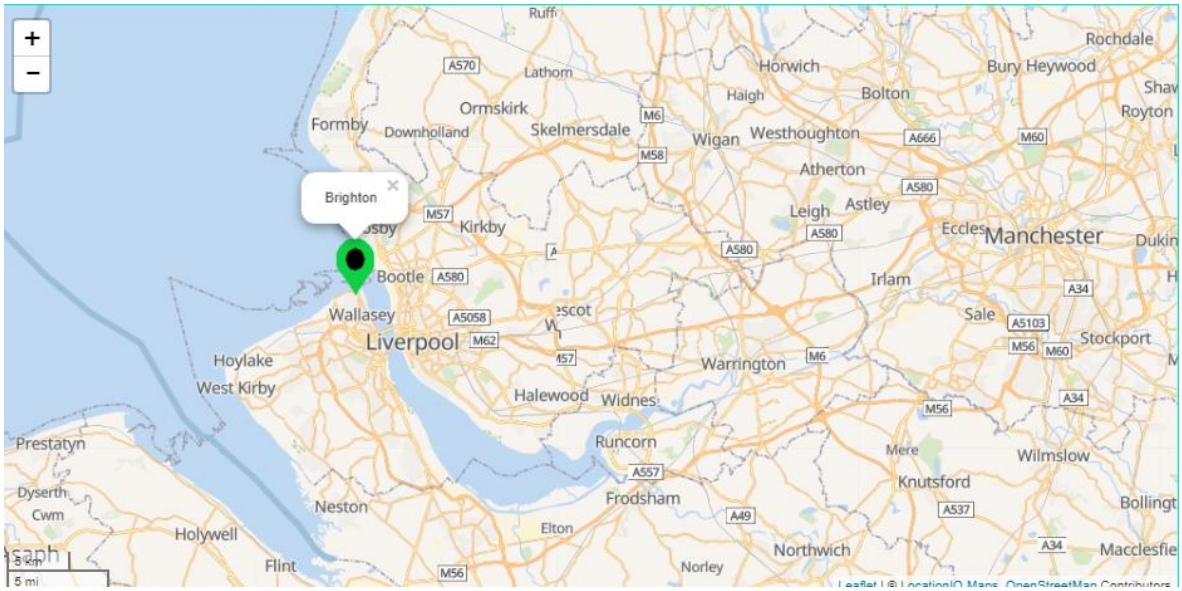
As you can see, there isn't an awful lot of code needed to fix this problem. One thing that I will have to note is that in order to make the invalidateSize method to work, I had to make sure that the CSS for the map was implemented first before the function worked properly. I found out that when I had the CSS for the map in my stylesheet, the function was not adjusting the map and I would still have the same problem that I had shown beforehand. Therefore, I decided that I would just write a <style> tag into the website which would set up the styling for the map. After I had done that the map loaded properly.

Now that I have fixed that small bug. I will now carry out the test for checking whether the location of certain stations is correct or not. The screenshot below shows the second attempt of the stations.



Error 7 – Some locations are wrong

I can see from the screenshot that most of the stations that you can get to from Haywards Heath are in the correct places on the map. However, I noticed that Brighton station was not pinpointed to the map. On further inspection of the map, I noticed that the pointer for Brighton was pointed to a town near Liverpool. Here is the screenshot.



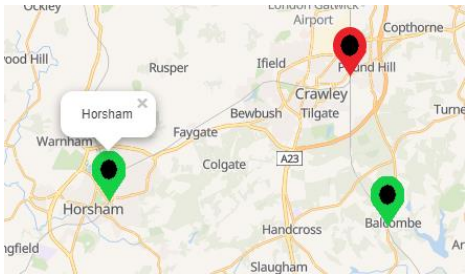
When I found this problem, I knew that this was a general problem with the places API call which inspired me to make the suggested stations part of the code. The main reason for this is that I would make the program look at the first item in the search which for 90% of the stations will be fine. However, whenever I would test with Brighton, I would always have to put in BTN. This is because I would always get New Brighton which is what has happened in this map. The most annoying thing about this is that if I tried to base the program on station codes, then I will have problems with other station.

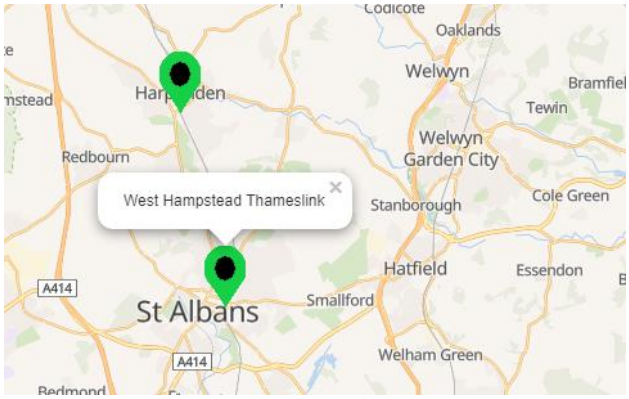
Solution

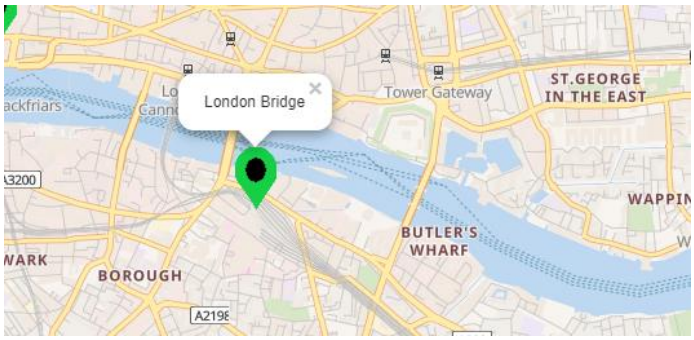
For this error I am not sure how to solve this problem on my own, I feel that I can fix this for Brighton station. I will make it so that if the station name is Brighton, it will get the transport API to call the place for BTN. However, I will need to find a way to fix this if I had more time. Therefore for now, I have created a separate if statement for Brighton. If I had more time, I will have a look at the code. **Test Partially Successful**

Test 2 – Look at the popup windows and see whether the information on the popup windows is correct

For this test, I am going to be checking 3 random stations when testing the website when getting the stations from Three Bridges. I am going to see whether the stations are in the right place. This is so that I can ensure that the information provided is accurate. The screenshot below shows the result of this test.





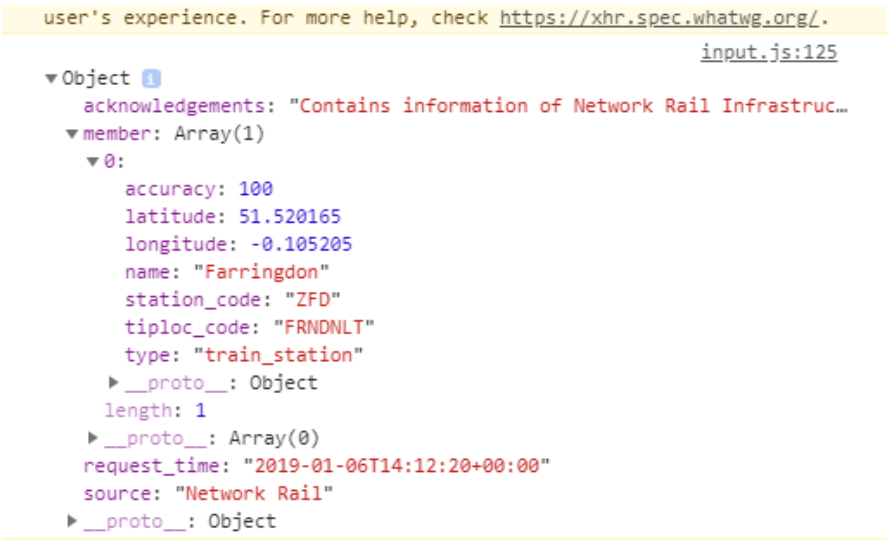


Error 8 – Popup Windows have the incorrect information

I can see from these tests that the function works well for the stations south of the River Thames. However, when I was looking at the stations towards the north of the thames, I could see that something had gone wrong with the locations of the stations. From what I can see, the program has seemed to ignore Farringdon when getting the latitude and longitude of the stations. Therefore, I need to find a way of getting the program to ignore stations where there is no information on latitude and longitude coordinates.

When I checked the console, I found that the program had given me this output.

I can see from this screenshot that not only has the program skipped out Farringdon, but it didn't even request Farringdon to the API. Therefore I tried testing the program to see how it would react to having Farringdon as the local station. Here is the result of this test.



As you can see, when I type in Farringdon into the program. For some reason it gets the latitude and longitude of the station. However, when creating a map, for some reason the app doesn't recognise the existence of Farringdon. When I had another look at the map, the program referred to the station as "Farringdon (London)". Therefore, I decided to input "Farringdon (London)" into the program. The screenshot below shows what I got as an output.

This screenshot made me think that the API was trying to get information on Farringdon but was finding no information. Therefore, I felt that in order to solve it I may just have to put in a new if statement to remove any stations which had any calls which had no results.

Solution

When I had another look at my code, I realised that I had already made an if statement determining whether there is any results in the API call. Therefore, I decided I would then use an if statement instead and have a function which sets both the latitude and longitude to 0 (I.e. null, null). The principle was to set stations which created 0 results to be sent to these coordinates so that the program will know that those stations will not be valid and will be excluded from the map. The screenshot below illustrates my thinking.

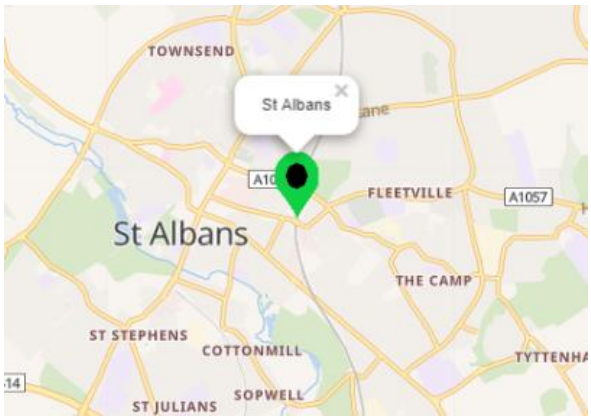
```
if(data_7.member.length > 0)
{

    console.log(data_7);
    console.log(data_7.member[0].latitude);
    console.log(data_7.member[0].longitude);

    stations_lat.push(data_7.member[0].latitude);
    stations_long.push(data_7.member[0].longitude);
}
else {
    stations_lat.push(0);
    stations_long.push(0);
}
```

Here is the code for the updated map generation. As you can see, if there is a length which is greater than 0 then it will get the latitude and longitude of the stations. However if the call gets no results then the latitude and longitude will

push 0 into their respective positions. The screenshot below shows the amendments to the code which determines the places where the map goes.

```
if(stations_lat[i] == 0 && stations_long[i] == 0)
{
    console.log("No Pointer");
}
else
{
    marker = L.marker([stations_lat[i], stations_long[i]], {icon:calling_icon}).addTo(map);
    marker.bindPopup(stations[i]);
}
```

Now that I had all the invalid stations set to the coordinates [0, 0]. I can simply make a new if statement which can then check whether the stations latitude and longitude are 0 and if they both are, it will just ignore that station and go to the next station. All that remains now is to check whether the stations are in the correct positions. Here are the same three pointers from the same station that I initially tested to see whether there are any improvements. The following screenshots shows the results of this test.



As you can see, the station in St Albans has now got the right pop up and also London Bridge has the right pop up as well. Therefore, the problem has been fixed and now the program does what I expected it to do **Test Successful**.

Feature 3 Review

- Get the first 30 stations of the stations list on the map in the right location **Partially Met**
- Get a pop-up window to tell the user where the stations on the list **Fully Met**

I feel that I have managed to make an effective map function in the time that I had made. For one thing, I feel that I have managed to make an effective map with a good selection of stations displayed on the map. I have also managed to make a feature which gives the users more unique pointers dependent on whether there is a local station or calling at station. However, there were some things that I would improve, hence me saying that I have partially met some of the aims for the feature. One of the main problems with the map feature is that I base it on the top result of the station when the places request is called. As I had said in page 89, there are a small amount of stations where the API puts the stations in the wrong position. I still haven't managed to find a solution for all stations, but I have fixed it for specific stations like Brighton. I would also like to make it so that the program gets the map data for all the stations from a specific station. However, due to API limitations, I can only really show all the stations available from a local station when the amount of stations that you can get to is less than 30.

Post Development Testing and Evaluation

Robustness Testing

Now that I have developed the code, I will now test the code to see whether the code is truly robust enough to carry out the tasks that I have set it. These will help me in my evaluation and will show me how the project holds up on a technical level. The table below shows the list of tests that I am going to carry out.

Test Description	Test Type	Test Data	Expected Result
Test the website on five different browsers (Internet Explorer, Microsoft Edge, Google Chrome,	Valid	The Entire Website	I will expect the website to function correctly in all five browsers

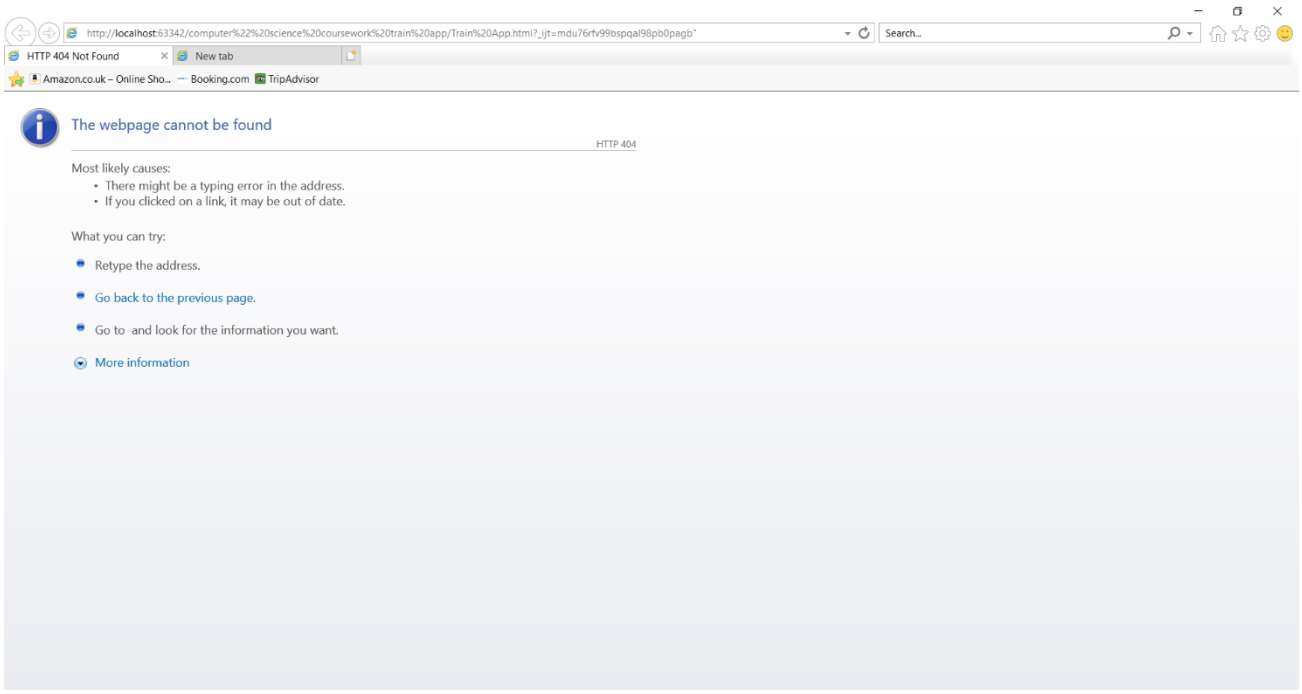
Mozilla Firefox and Safari)			
Put in an invalid station 20 times to see what happens	Valid, Black Box	Stations box	I will expect the website to reject the user's input each time if they put in an invalid station

Robustness Test 1 – Run the website on five different browsers to see whether the website runs properly.

For this test, I will be running the website on five different browsers. The browsers I have chosen to test the program on are:

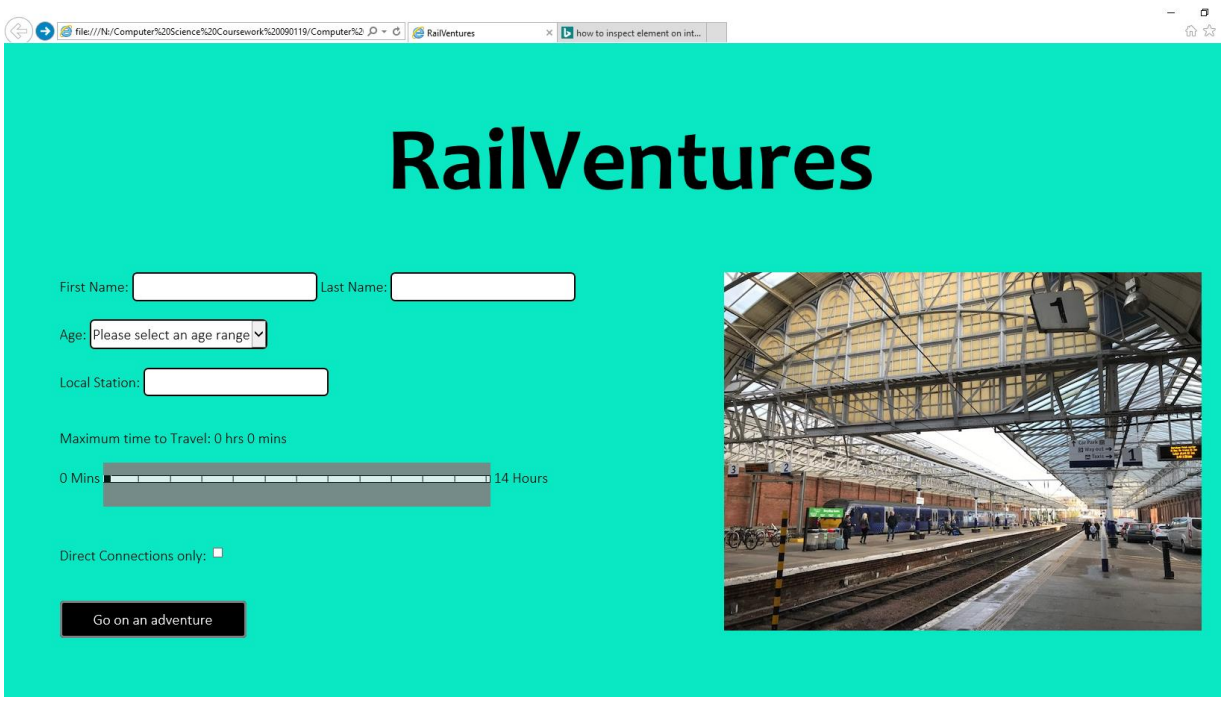
- Internet Explorer
- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Safari

The reason why I have chosen these browsers specifically is because these browsers are the most used browsers and so that I can see whether the website is accessible to everyone. Also since I tested my website on Opera, I need to make sure that the website works on all the other major web browsers. The first browser I decided to run the website on was Internet Explorer. Shown below are the results:

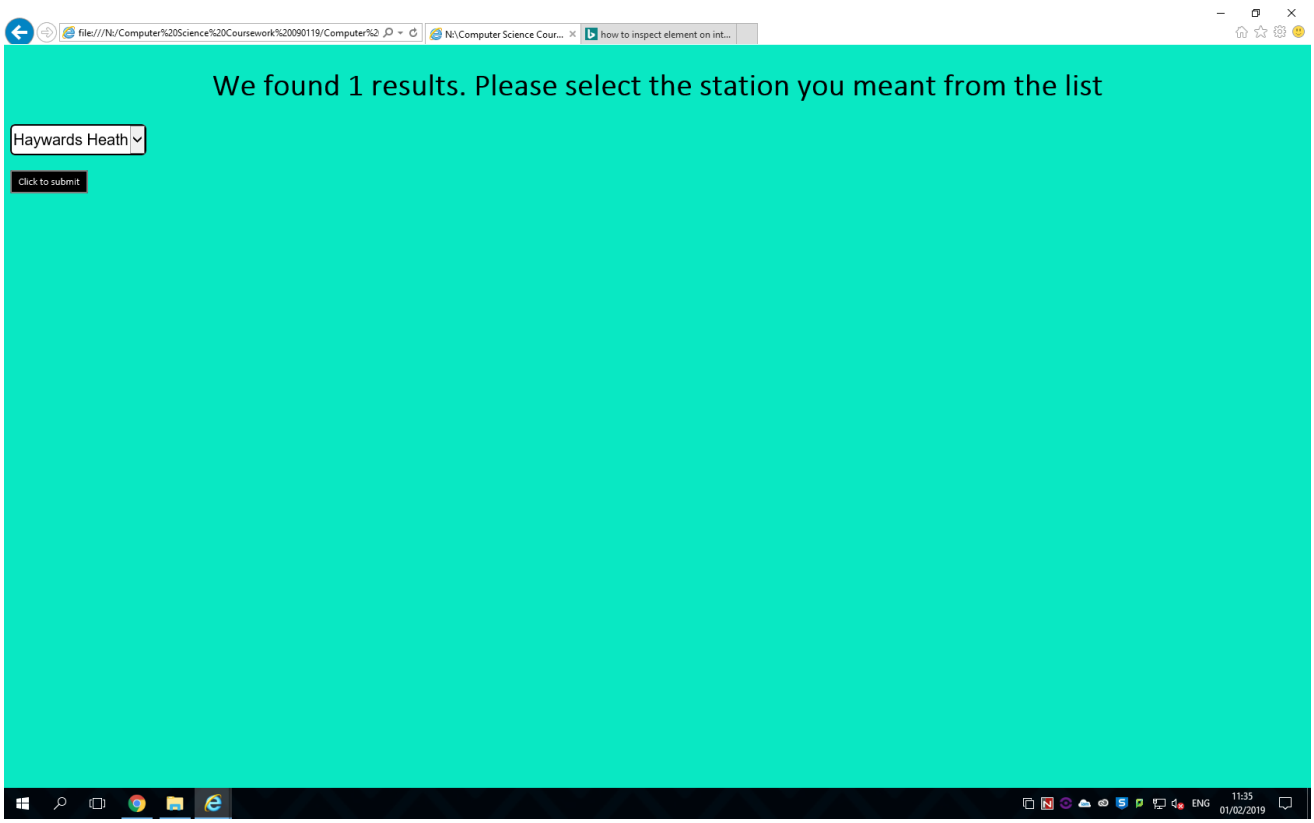


As you can see, Internet Explorer hasn't responded well to my website. For one it seems to give an HTTP Error 404 Not found. It could be that my website has code that Internet Explorer couldn't read. If I had more time to develop the website, I would try to make the code so that it is compatible with Internet Explorer.

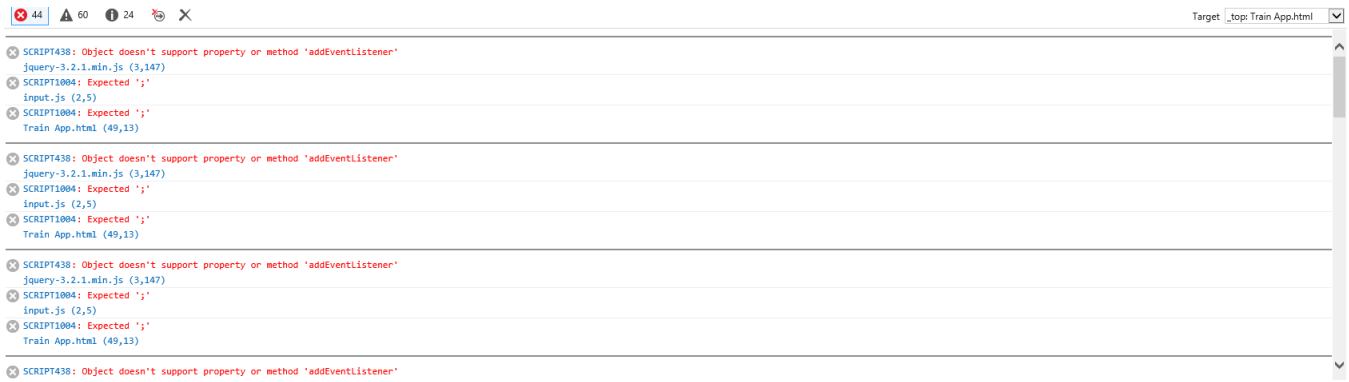
UPDATE: A few weeks after the time that I did my robustness testing, I found a way to run my website on Internet Explorer. I think that the reason why Internet Explorer didn't work is because I was trying to access the website from an external hard drive as I managed to access the website using Internet Explorer when I put the project in the external hard drive. With that being said, here is the input screen as seen in Internet Explorer.



I can see from this screenshot that the website looks pretty good. Especially considering that the layout hasn't been compromised to load up the website. The only thing I would say is I don't like how the time slider looks as it seems to have created a hybrid between my CSS and the default look of a slider on Internet Explorer. Another thing is that the numbers on the maximum time to travel label don't change as I move the slider. The next screenshot shows what the suggested stations screen looks like on Internet Explorer.



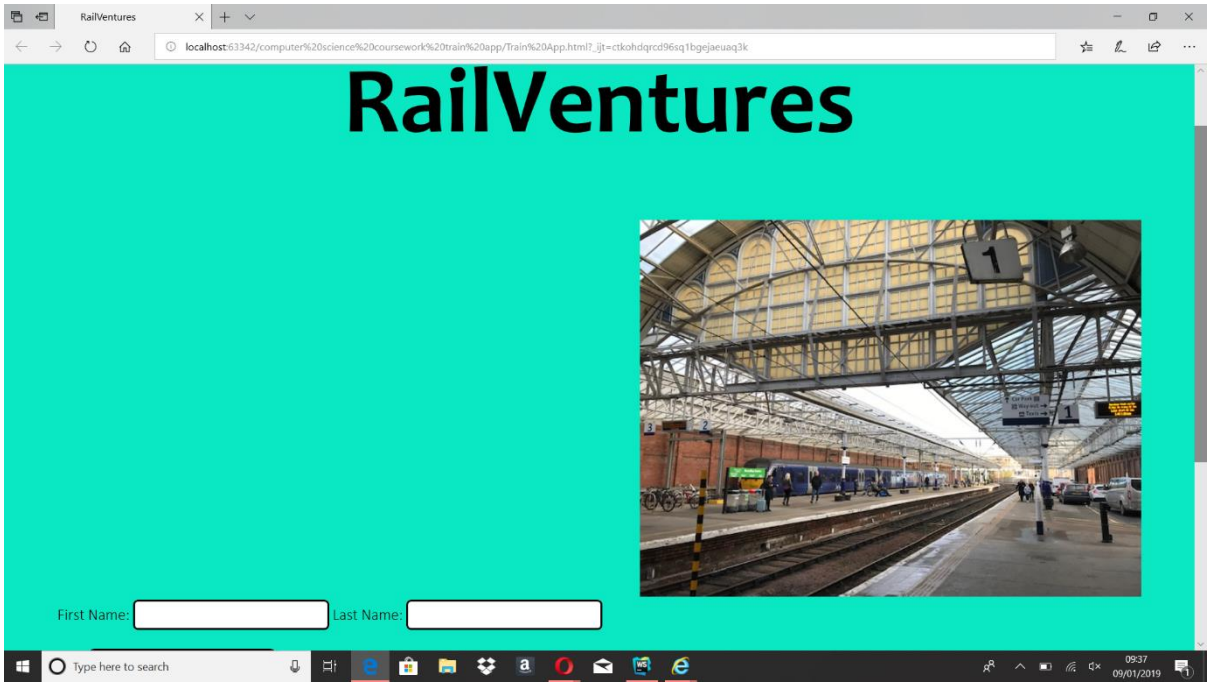
I can see from this screenshot that the suggested stations window looks like I intended the app to work. However, one thing I will say is that I couldn't get any further as when I tried to click on the submit button, I realised that the app wasn't letting me get to the main output screen. When I looked at the console, I got this result.

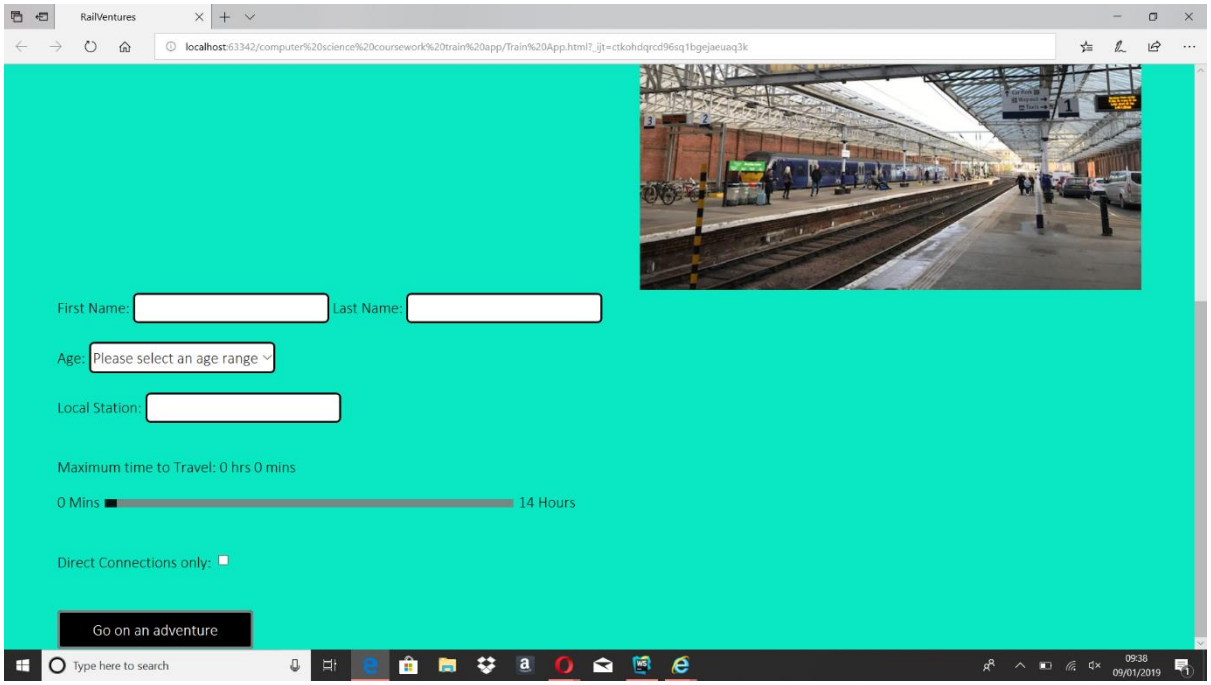


Microsoft Edge

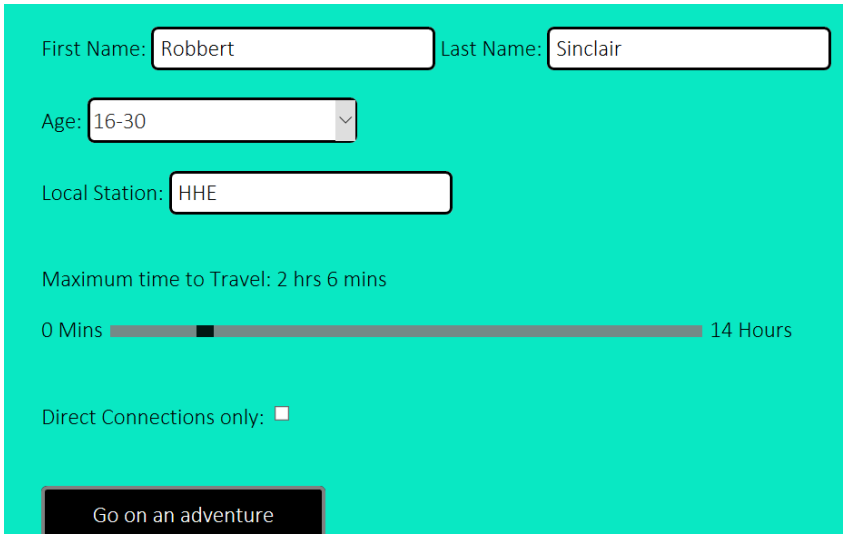
Since I have started with a Microsoft Internet Browser it only makes sense for me to try out Microsoft's newest browser; Microsoft Edge. I hope that since

Microsoft Edge is a more modern browser that it will load my website. The screenshot below shows the results.

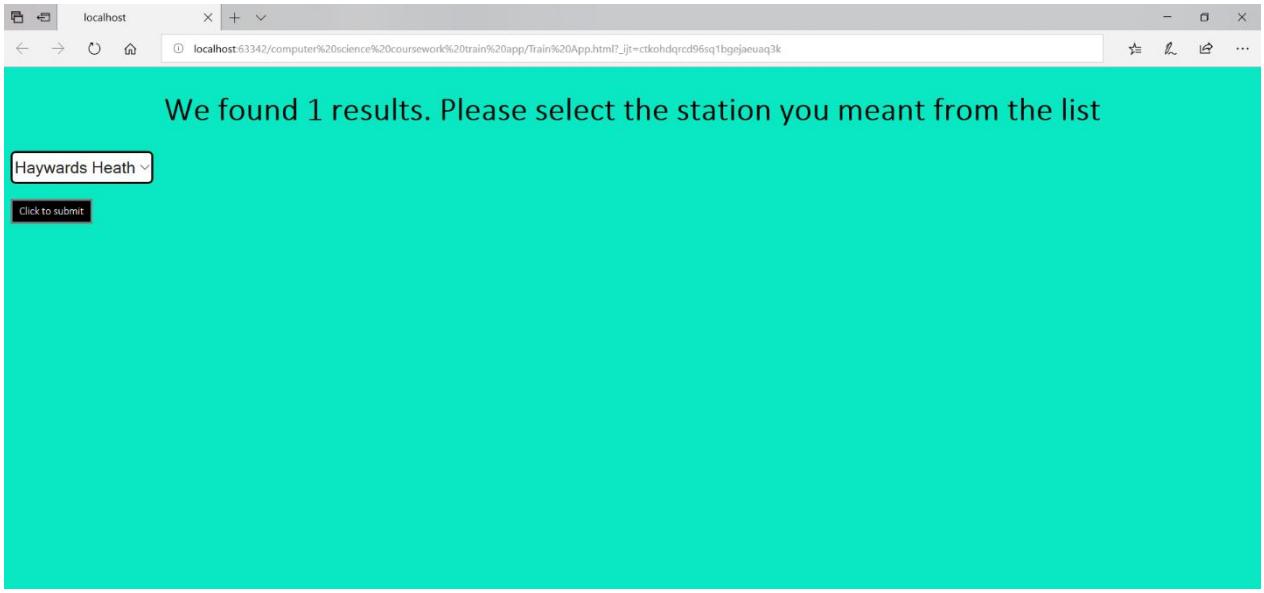




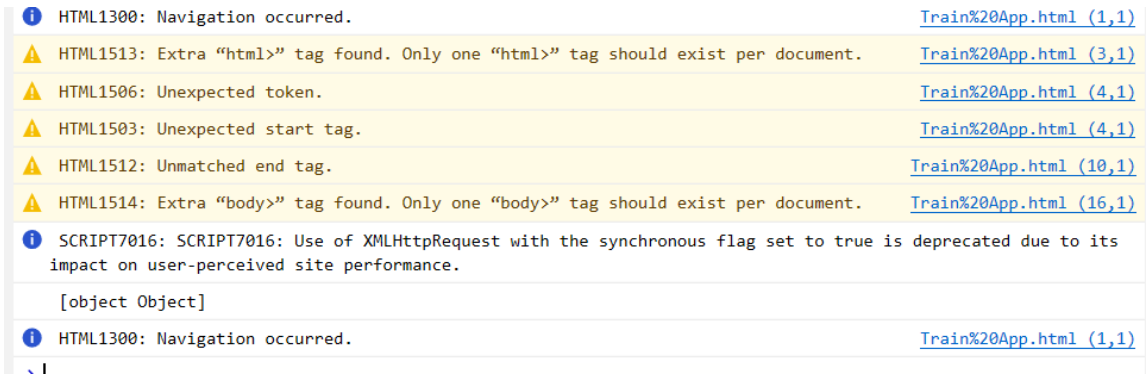
My first impressions of this result is that is a lot better than that of Internet Explorer. However, I have noticed that my CSS has been affected in a different way with this browser. For one thing the gap between the picture and the form is very different as the entire form section is further down the page. Now that I know that the home page has loaded, I am now going to see whether the website works in Edge. Therefore, I am going to run the code and see what happens. Shown below is the test data that I am putting in.



Here is the results:



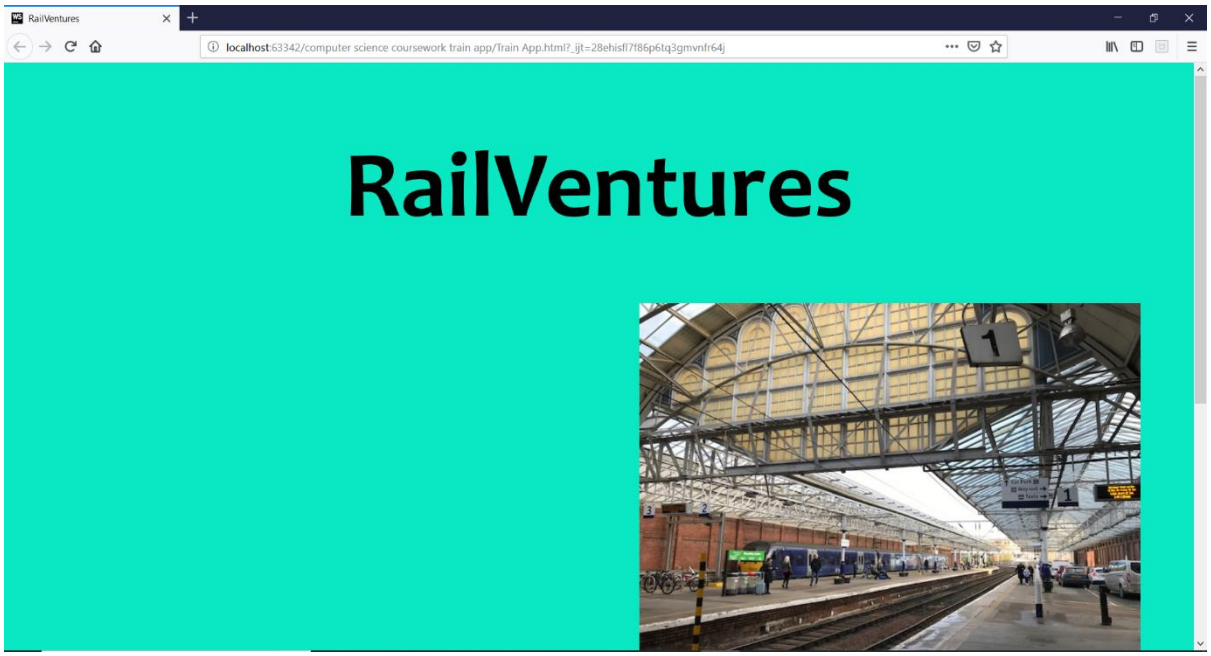
As you can see, I managed to get to the results page. However, when I tried to go further into the results page, I realised that the website was not letting me go any further. I decided to have a look at the console and see what is going on. Here is the console:

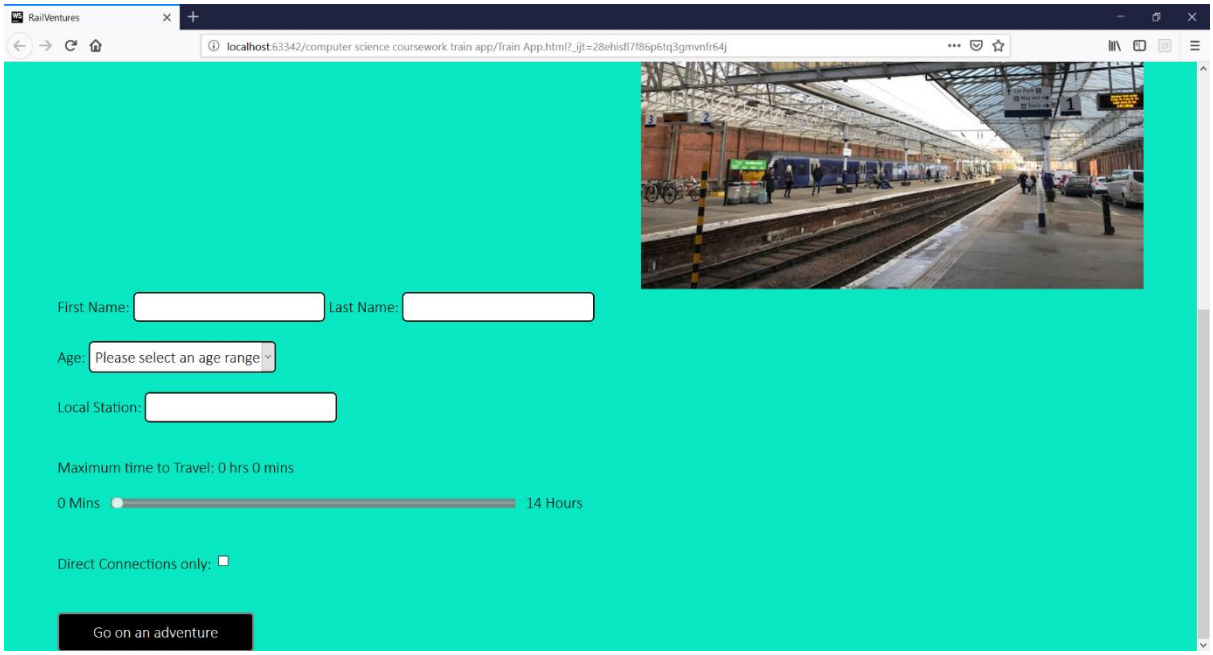


From what I can see from the console, it seems to be that there are some parts of the HTML file that the browser doesn't like. I have the feeling that the program doesn't like the fact that I have "document.write" functions. Therefore I will need to look at compatibility with some browsers.

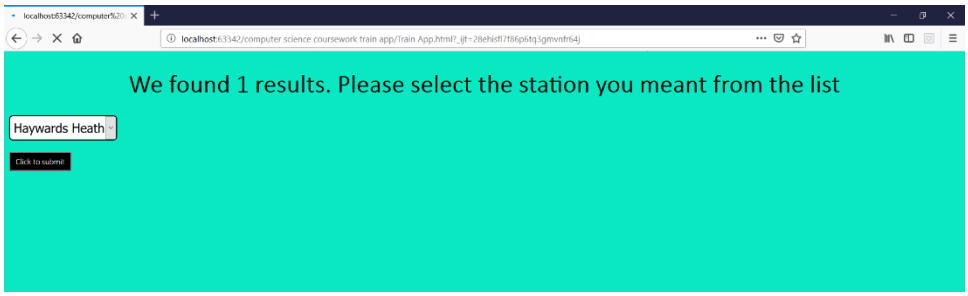
Mozilla Firefox

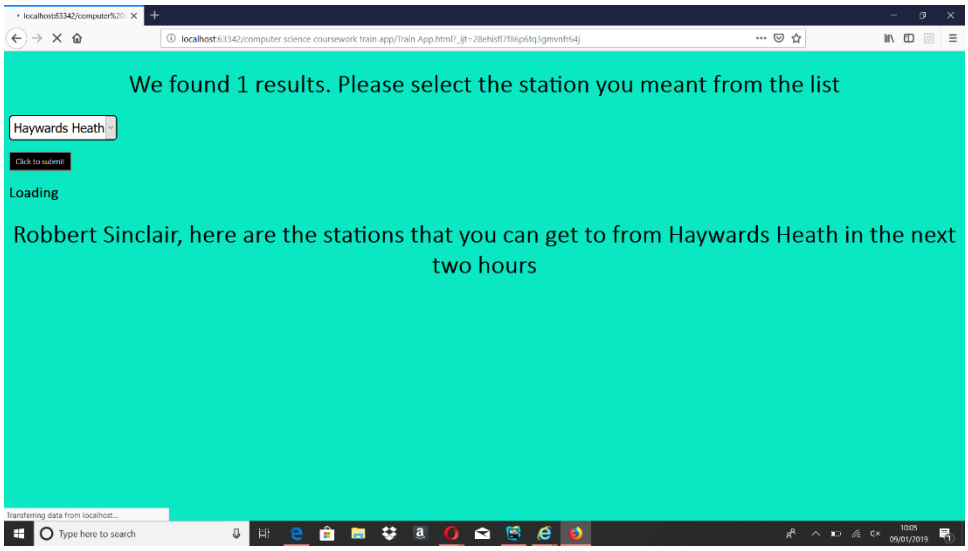
The next browser to test is Mozilla Firefox. I feel that since Firefox is more widely used that the browser will be similar to Chrome. Although I do remember having to make specific CSS for Firefox back in stage 1 so it may be interesting to see how Firefox will react to my website. The screenshot's below show the results at the home page.





I can see here that Mozilla Firefox loaded the page in a similar way to Microsoft Edge. This is where the form is under the picture of the station. This could mean that Firefox works in a similar way as edge. I also have noticed that the slider has changed slightly as most of my CSS on the slider is removed in favour of a default slider in Firefox. Shown below is the results when I ran the program with the same dataset.

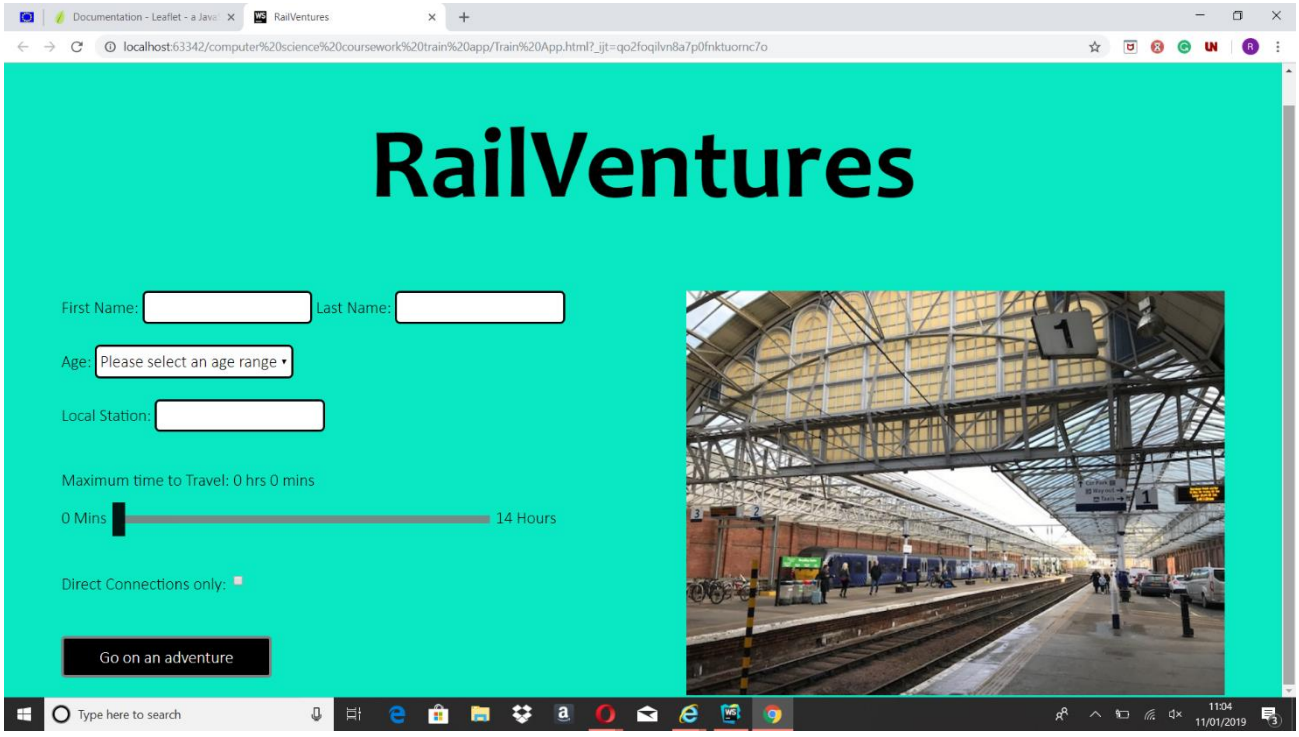




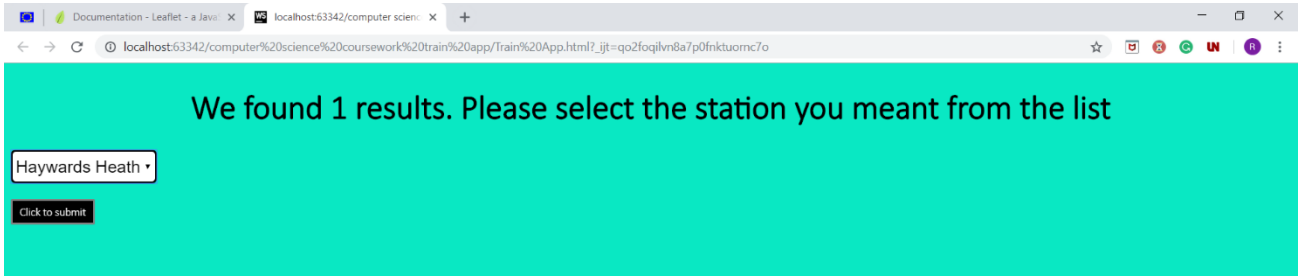
As you can see, Firefox reacted in a similar way to Microsoft Edge. However, I did manage to get the message to load this time. The thing that I find is quite bizarre is that it let me get to the suggested stations window but as soon as I had clicked on the submit button to see the stations that I could go to, I was presented by a few errors. This could be that the code isn't optimised for firefox and it might be that if I had more time, I would have to look into how I can get the program working for other browsers like FireFox. I should also have a look into this considering that of the three browsers that I have tested so far, none of them have loaded the website without there being a problem with the layout.

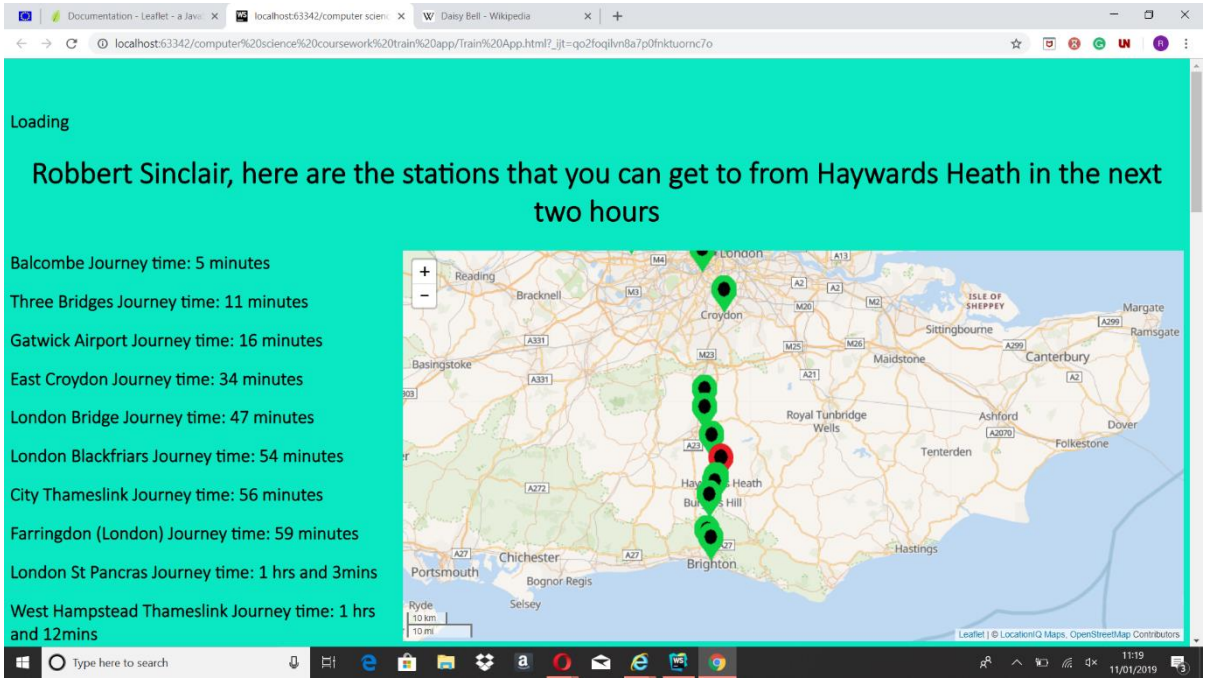
Google Chrome

The next browser will be the most used browser, Google Chrome. I have high hopes that my website will work as intended on this browser as I have been developing my code on Opera. This is because from experience, Opera works in a very Similar way to Chrome. It also has the same inspect window as Chrome. Therefore, I hope that the website will work. The screenshot below shows the home screen of the website running on Chrome.



I can see that this is a significant improvement from the previous browsers. For one thing, I can see that the layout is the same as on Opera. This really supports my hypothesis that Chrome will do better than the other browsers as it seems that I won't have to make specific changes to suit the browser. The next test however, is seeing whether the website will actually work in this browser. The following screenshot shows the results of what happened when I ran the website in Chrome.



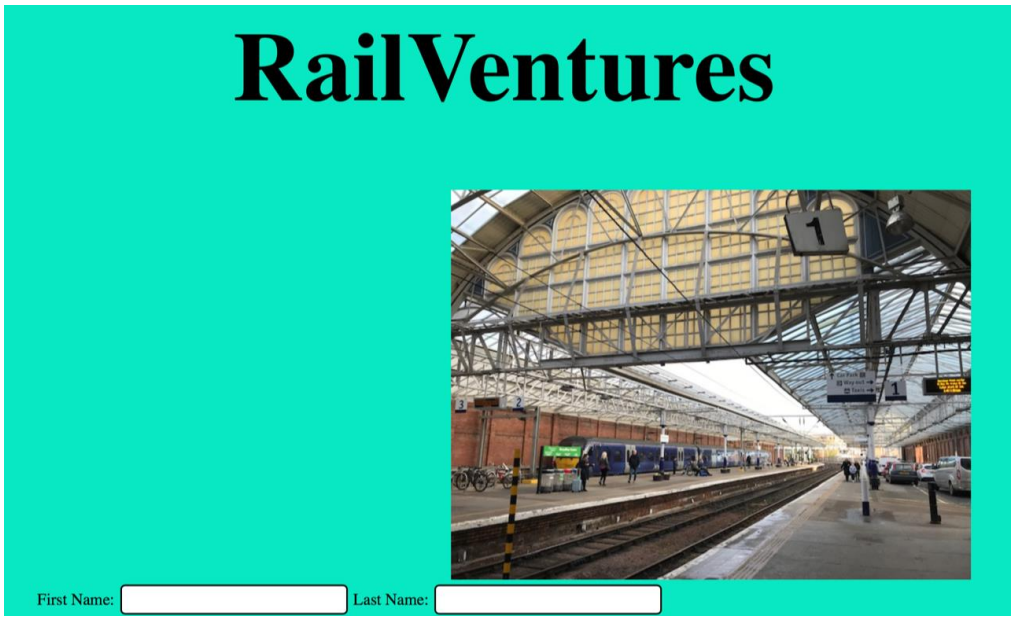


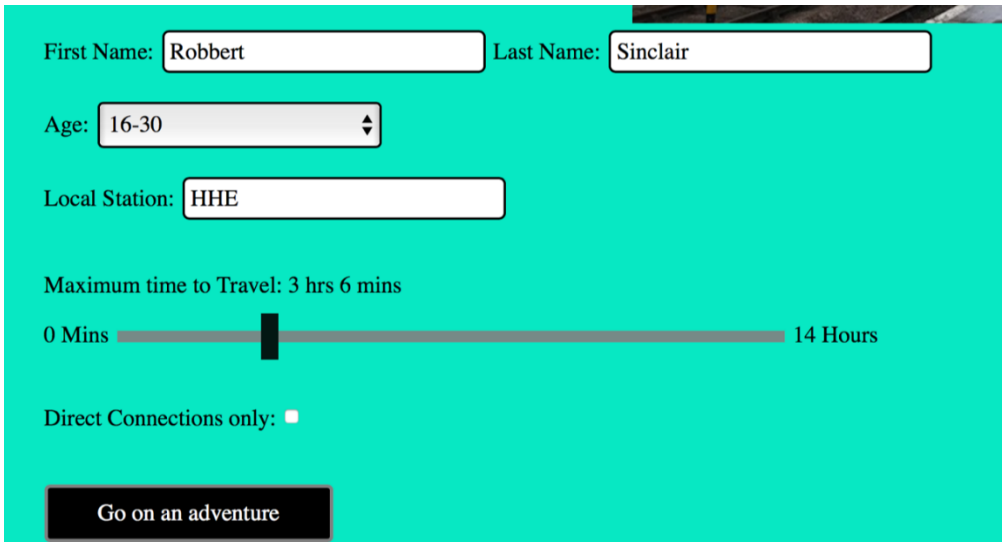
As you can see, the website functions very well in Chrome and there were no significant problems. The one thing that I would say is that the font sizes and the size of the elements in the forms are slightly bigger. However it doesn't affect the layout of the website so that really isn't a problem. Overall I feel that the website has fared well on Google Chrome and so therefore, the website at least works for another browser other than Opera. Also, I feel that I will be reaching out to the majority of my Stakeholders as Google Chrome is by far the dominent web browser. I think the reason why both Google Chrome and Opera are working so well is because both browsers work on Chromium which is

Googles open source web browser tool. This may explain why the two browsers are so similar.

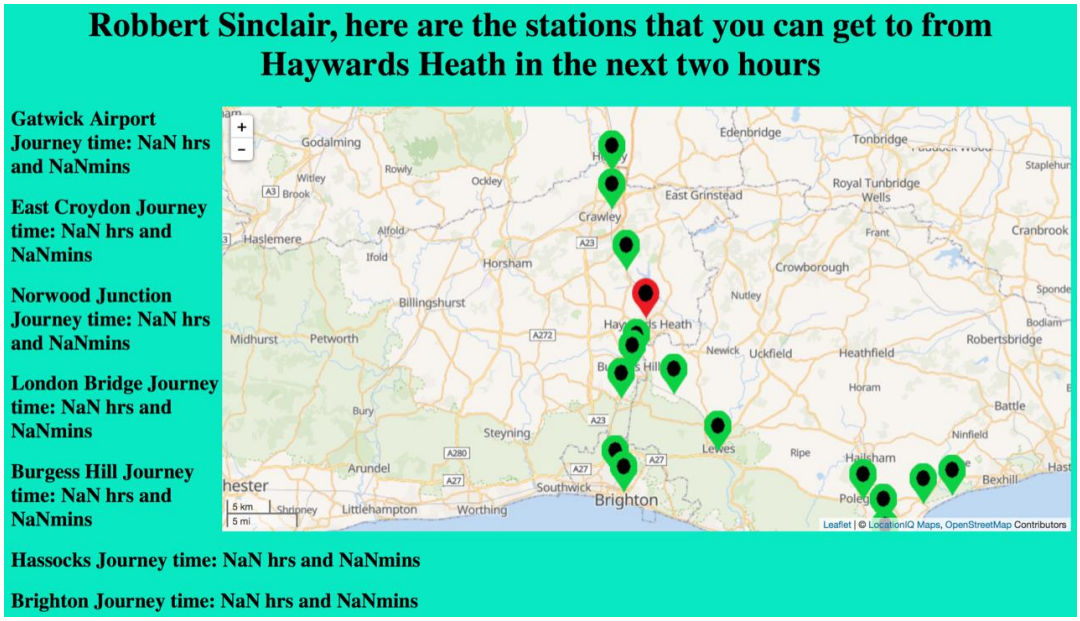
Safari

The final browser that I wanted to test was Safari. My reasoning for this is that I have developed this website on computers that run windows. Therefore I wanted to see what the experience would be like for a user using a Mac. Therefore, I borrowed a mac to see how the website runs on Safari. I don't know how Safari would react with the website as it seems to be that it works especially well on the browsers that are based on Chrome. The first thing that I had to do to see what the input screen looked. The screenshot below shows the results.





I can already see from the screenshots that I have experienced the same problems that I had with Edge and Firefox. This being the layout of the website being off. I have a feeling this could be because all the browsers have different default sizes for these elements and that if I had more time to develop the website further I should make sure that there is a specific size to ensure that I don't have this problem. I also noted that the font that I had given to the website isn't recognised on this browser. If I had more time to develop this website, I would make sure that I would have a font that I knew would work on all browsers. The next thing to do was to run the website with the same data that I had inserted with all the other browsers. The screenshots below shows the results of this test.



I can see from these screenshots that this is the best performance that I have had from a browser which isn't based on Chrome. However, there are still some problems with the website. For one thing I have noticed that the website isn't displaying the actual time it takes to travel. Instead it came up with "NaN" where there should be a number. One of the other things that I have noticed is when I asked for all the stations to Gatwick Airport from Haywards Heath. I noticed that I wasn't getting any data. I have no idea why this is happening. Therefore I decided that I would have a look at the console to see what the console was outputting.

I can see from the output that the browser is getting the start time however, I feel the problem is that I can't find the finish time. I have no idea why it isn't calculating but considering the poor performance from Edge and Firefox and the non-existent performance of Internet Explorer, I at least got an output and I got some data from it. One of the other smaller things that I have noted is that when the website was loading the data, I got the mac loading graphic on my mouse which was something of note considering that I didn't get a chance to make a graphic to make the user know that the website is loading data.

Robustness Test 1 – Conclusion

I feel that testing these five browsers really showed me that certain browsers work in different ways and hence I would have to code differently to make sure that the code works for all types of browsers. I can also see that my code works the best on Chromium browsers. I feel this is because I developed and tested the code on Opera which is Chromium based. I felt disappointed that the code didn't work on all browsers or were very limited in the amount of stuff that did work. However, since Edge will be moving to a Chromium build in 2019 I may need to test Edge again when that happens to see whether my hypothesis that the code works well on Chromium browsers is correct. But overall I would say that the test was partially successful and if I had more time I would make sure to ensure that the code will work on all the major browsers.

Robustness Test 2 – Put in invalid stations 20 times to see whether the program will withhold.

The main idea for this test was to test the robustness of the input form. The idea being that if I cannot proceed with the incorrect data for 20 times then I should never be able to get through with the incorrect data.

Test Video: https://youtu.be/qPjbpHOpBm8

I can see from the video that every time that I had put an invalid station or invalid data, the website just rejects my input and tells me to try again. But I did see that if I put nothing in the local station box, I would get some stations

come out on the suggested stations. Therefore, I may need to slightly fix the website so that the program will not take a blank input on the stations box. Despite this small problem I feel that the user will not be able to put in the wrong information for the majority of uses of the website. Therefore I can say that the test is partially successful.

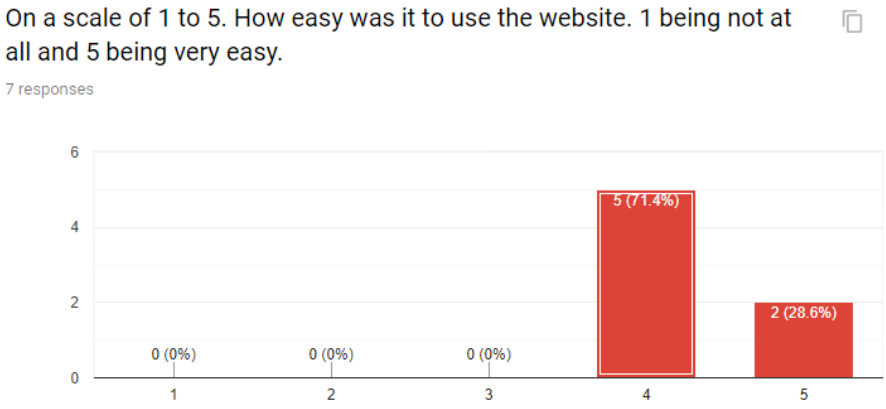
Beta Testing

For the Beta testing, I decided that I would let the stakeholders use the website and I would let them fill out a survey based on their overall experience that they had when using their website. For these tests I wanted to make sure that the stakeholders would find the website easy to use and to see whether the users would be able to use the app. On the questionnaire, I asked these questions:

- On a scale of 1 to 5. How easy was it to use the website. 1 being not at all and 5 being very easy.
- Do you have any comments on how easy the website was to use?
- On a scale of 1 to 5. With 1 not being useful at all to 5 being very useful, how useful did you find the station suggestion feature?
- Do you have any additional comments about the suggested stations feature?
- On a scale of 1 to 5. How useful was the map function?
- Do you have any additional comments about the map?
- On a scale of 1 to 5. How would you rate presentation of the station list?
- Do you have any additional comments about the presentation?
- On a scale of 1 to 5. How easy was it to find departures information from your local station to your selected station?
- Do you have any additional comments about the departures information?
- Do you have any additional comments about the app as a whole?

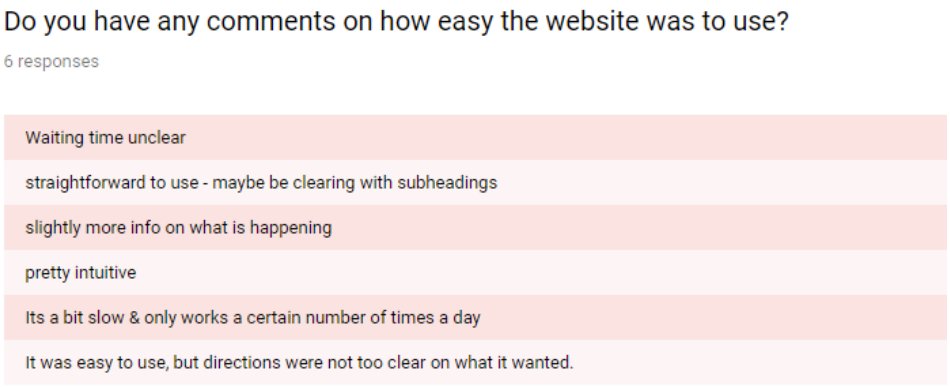
These questions will help me paint a picture on the usability of the website and will tell me what I can do to improve if I had the time. I managed to get 7 responses from the Beta testing The following images show me the results of these beta tests.

On a scale of 1 to 5. How easy was it to use the website. 1 being very difficult and 5 being very easy.



I can see from the data that I have received that the majority of people found it to be easy to use the website. However I can see that two 5 out of 7 of the people using this website rated the usability as a four out of ten. This shows me that I will need to ensure that the website is slightly easier to use. The next question told me what people would want to see from the website.

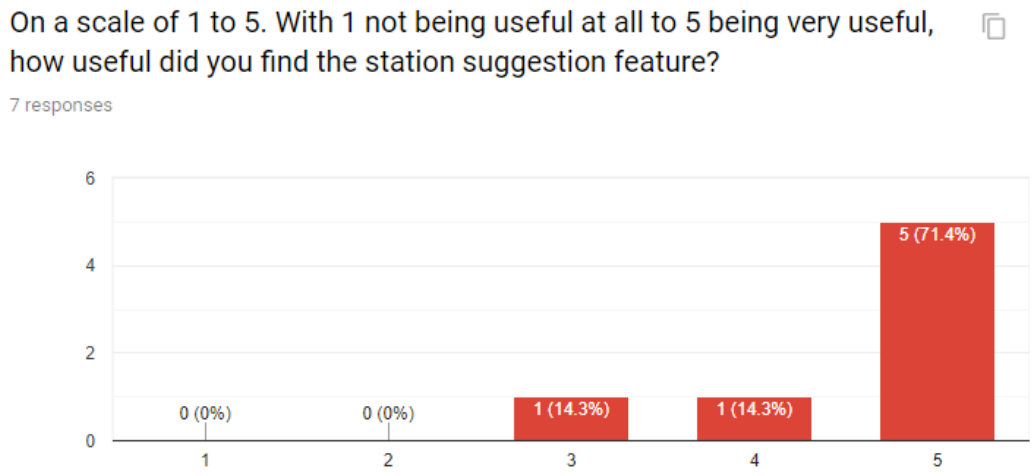
Do you have any comments on how easy the website was to use?



For this question, there was a total of 6 responses, I can see that the consensus was that the website was rather straightforward. There were a few comments on the speed of the program. Two out of 6 said that they didn't like the fact that it was a bit slow and that they were a bit unclear of how long the program would take to load. This is definitely a problem that I tried to do. However, I will not be able to make the app faster as the website works on asynchronous requests (See page 32 for a full discussion) in order to work which takes some time to work. I will definitely be able to make a loading screen and I wish I did

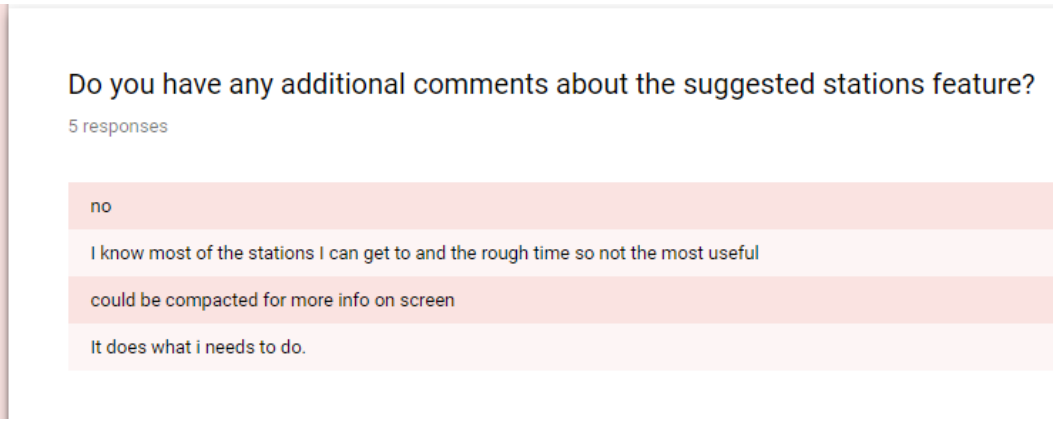
manage to implement this feature. Half of the respondents said that some of the directions were unclear. Therefore, I will need to have a look at the instructions that I have given in the app to see how I can fix that. I feel that there are some places where I myself felt that the headings were unclear.

On a scale of 1 to 5. With 1 not being useful at all to 5 being very useful, how useful did you find the station suggestion feature?



For this question the majority thought that the station suggestions feature (development is discussed from pages 65 – 78) was a very useful feature with 5 out of 7 of the respondents giving this feature a 5 out of 5. There was also 1 person who gave it a 4 out of 5 and another person who gave the feature a three out of ten. The next question will show me the additional comments about the suggested stations feature.

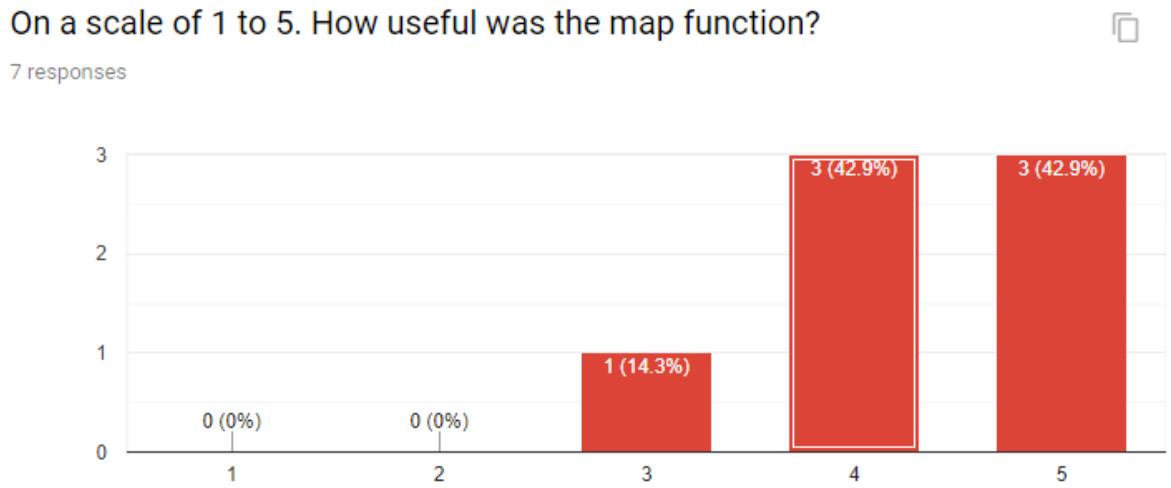
Do you have any additional comments about the suggested stations feature?



From this question I can see that people didn't have as much to say about this feature. One of the people who did respond said that "It does what it needs to

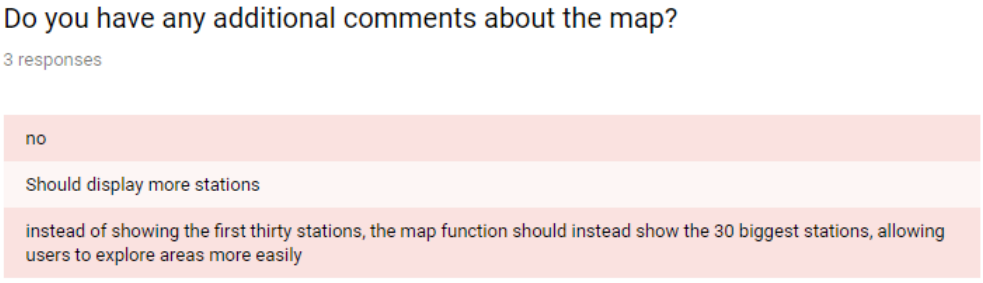
do". I feel that this shows the general consensus of how people feel about this app. Some of the other comments were not even about the suggested stations feature which could mean that I should have been clearer when writing this Beta Test Questionnaire. Overall I feel that the people like this feature.

On a scale of 1 to 5. How useful was the map function?



For this question there seemed to be a wide range of views on the map feature (development discussion on pages 74 – 88). I can see that 6 out of the seven people rated it a 4 and above out of 5. There seemed to be a 50/50 split with the people who answered between 4 and 5 out of 5. Therefore I have a feeling that the majority of people found that the map function was useful. The screenshot below shows the additional comments that people had about the map.

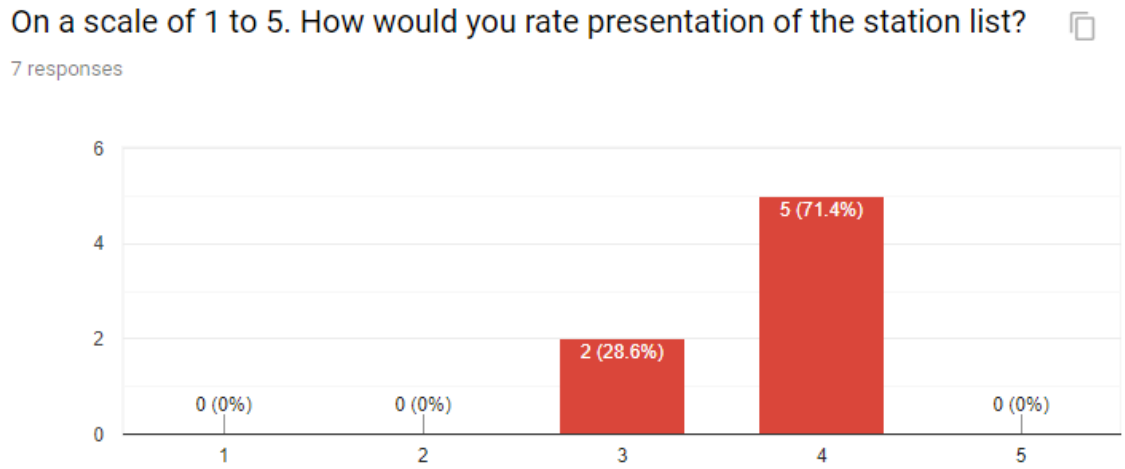
Do you have any additional comments about the map?



I can see from these comments that there isn't as much to work in compared to the rest of the questions. However the feedback that I did get was based on the amount of stations on the map. Some people were asking for me to display more stations. The problem with the map feature is that as I have mentioned

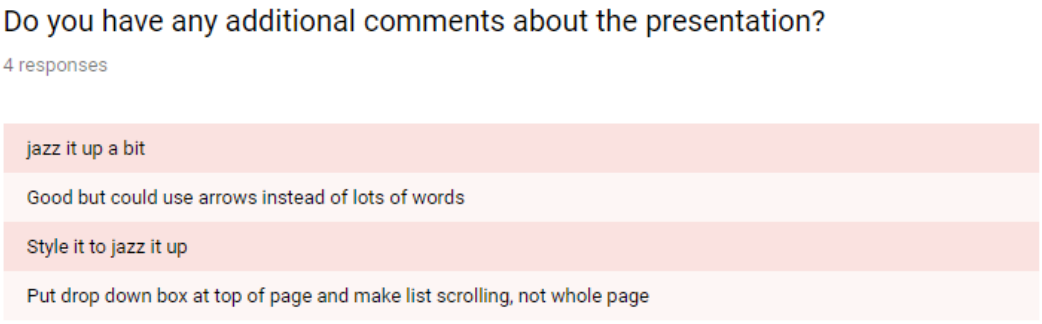
many times before is that I only have 1000 API requests per day under my current plan with the Transport API. Therefore if I were to include more stations on my map, I will have to pay money to get more requests. Another interesting comment that I had was that instead of the first 30 stations they asked for the 30 biggest stations. Unfortunately the Transport API doesn't have the data for me to do that. However, if I did have the data (I.e. Passenger numbers) then I would have considered creating this feature. Overall I feel that people did like the map feature.

On a scale of 1 to 5. How would you rate the presentation of the station list?



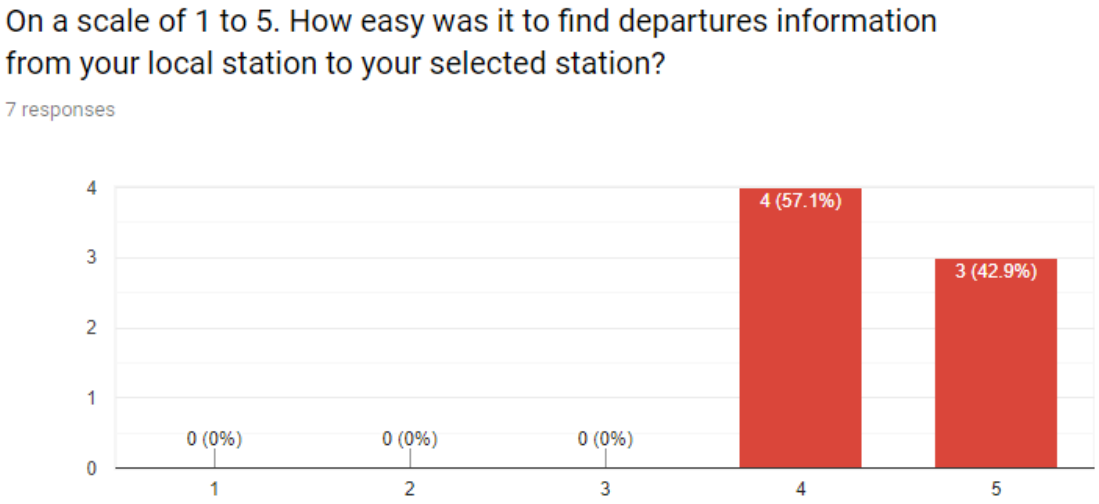
I can see from this question that there seemed to be a generally positive reaction with some having mixed feelings. As you can see 5 out of 7 rated the presentation a 4 out of 5. This seems to be the on only question where I didn't get anyone rating it a 5 out of 5. This gives me the impression that I will need to make some improvements to the presentation. The screenshot below shows the comments that I have received for the presentation.

Do you have any addition al comments about the presentation?



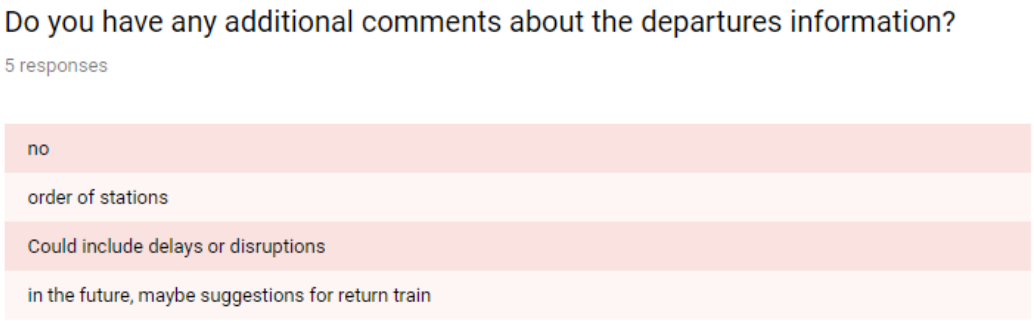
Here are the comments on the presentation. From what I can see from these comments, a lot of people wanted the presentation to be more pleasing to the eye. Some people also said that the drop box containing the departures information could have been put to the top of the list instead of the bottom. I also see that people wanted less words in the website and maybe have it so that it has more arrows or other symbols to make the website work better. I must say that I agree with what the people have said as I could have looked at better ways to make the presentation of the information more pleasing and simple to understand. One of the things that some people did say was that the order in which the stations are outputted was a bit awkward and it should have been changed to being alphabetical order or ordered by time.

On a scale of 1 to 5. How easy was it to find departures information from your local station to your selected station?



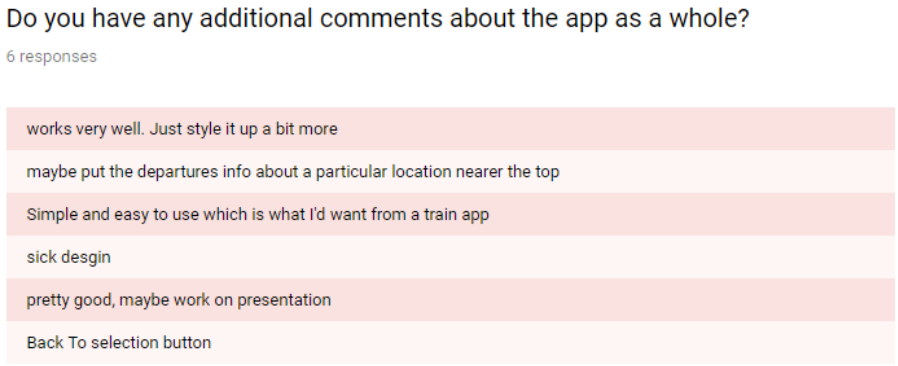
From this question, I can see that everyone that I asked rated the departures information 4 or above out of 5. There is a slight majority giving the rating a 4 out of 5. This shows me that the majority found it to be easy but could be improved to make it easier. As I had mentioned in the presentation question in the beta testing, many had told me to put the dropdown list which asks the user to select a station at the top of the stations list. I also remember having to tell people where the list was. This shows me that I need to make this feature a lot clearer to people so that they would be able to find the feature without me telling me where the feature is. The screenshot below tells me the additional comments.

Do you have any additional comments about the departures information?



I can see from the comments about this feature in the app that people were wanting more real time updates to the departures. Some people said it would be useful to have a look at whether there are any delays on the trains or if there is any cancellations and the reasons for the delays or cancellations. For this to work I would have had to use the “live” API request instead of the “departures” request. However I will take onboard what people have been saying and I feel that having more in depth departures information would have been rather useful. Some people also told me that they didn’t like having to scroll down on the website in order to see the departures and they said that they would much rather have it so that the departures information was put somewhere which was a lot easier to see. One of the other interesting things that I noticed is that people had asked to see the return trains for coming back to their local station. I would do that, however my API only gives me the departures in the next 2 hours so I would need to find a way of getting data beyond that time.

Do you have any additional comments about the app as a whole?



For the final question, I just asked the people who I was beta testing with what their comments where about the app as a whole. I managed to get a lot of constructive feedback which is both good and bad. Quite a few people said that they thought that it was easy to use and a lot of people said it worked very well. One of the more interesting things that people had said that they would want an easy back to selection button which refreshes the website. I think that this will be a good idea because not everyone knows about the refresh button and it would be handy if there was that feature so that every user that is making use of the app will be able to easily go back to the home page.

Evaluation

In this section, I am going to look at the success criteria that I have talked about in my analysis and I will discuss whether I think I have fully met, partially met or haven't met this section of the success criteria. For this section, I will start with the essential features.

Success Criteria feature	Partially, Fully or not met	Comments
"**The app Must ask for the passenger's home station**"	Fully met	I feel that I have managed to let the user input their home station in a very simple and easy to use way. My Beta Testers said that they had no problems about the home screen and thought that it was easy to use. I also feel that I have effectively used the API. To see the test for this part of the code please refer to stage 2 starting at page 29. One of the things I would say in order to improve this feature is to make it so that there are dropdown suggestions while the user is typing

		the station name. This will be a lot better than the solution that I currently have which shows the user the stations that they can get to.
"The app must take the journey times into account."	Partially met	For this part of the app. I definitely feel as though I have met this part of the success criteria. I feel that I managed to manipulate the data that the API gave me to successfully filter out any stations that the user couldn't get to in the maximum time. However, one thing I will note is that I didn't fully manage to meet the concept that I had on what the GUI would look like. For example I had no area where you could change the time to see what other stations I could go to. It just didn't seem feasible to do this function as it would take a long time to recalculate the journey times for every station on the list. (See page 46-48 for evidence) The main reason why I decided to say that I partially met this feature is that the feature doesn't work very well with remote stations that only have one train a day or one train every few hours. As I mentioned in page 50, the API only gives me the trains that are departing in the next two hours so if there isn't a train departing that station in the next 2 hours then the program will not come with any departures. This means that the station must have a consistent and frequent service in order for the algorithm to work properly.
"**The app must be able to give the user a list of available stations that the user can travel to with their money/time taken.**"	Partially Met	I know that I can show the list of available stations from the time taken (see page 45 for evidence of this). However, one of the things that I have said before is that I could only do it for the direct connections from the local station. I was originally planning on looking at what stations you could go to with a change over. However, I had the problem that I wouldn't have had enough API requests and after working with the API for a while, I realised that I couldn't think

		of how I would possibly implement that functionality. According to the Beta Testing, a lot of the respondents told me that the presentation of the list should be improved. Therefore, I will also need to have a look at the presentation in order to improve the customer experience. (For more discussion the beta testing turn to page 105) Another thing that I knew that I couldn't do for a fact is that I couldn't get a list of stations based on budget as the API is still yet to get data on rail fares. This is something I will explain more in depth in one of the other essential features in my success criteria.
"**The app must have map support**"	Partially met	Overall, the map function is working well for the majority of the stations that I put on it. However, there is one small logical error on the map that I couldn't quite get fixed (Turn to page 89 for a more Indepth discussion of this problem). This was the problem that some of the stations are in the wrong location. As I said in page 89, I knew what was causing the problem but I didn't really know how to fix it. This was because it was a problem of the API registering the first station on the list of stations it gets back when you input a station name to the "places" call. As I also said, I only put in an extra if statement for Brighton, but I didn't do that for any other station which I really wished I could have done but I had no idea of how to fix the problem without breaking it for the 9 out of 10 stations where this wasn't a problem. Another thing which could be improved on this feature in the future is that I should make it so that the map pinpoints all the available stations on the map. However since I only had 1000 API requests when developing this website, I could only put the first 30 stations as a way of showing how the map function would work. However I

		had to make sure that I could make the app while still working to the limitations of the API. I have some good evidence of the map function working on page 86-87.
"**The app should get the budget for the passenger**"	Not Met	Unfortunately, I couldn't implement this feature because the API at the time of developing this app didn't have the functionality of rail fares in it. When I initially looked at the API when trying to find data to create this app. It told me that there was going to be data on rail fares and that it said that it was "coming soon". Six months later at the time that I am writing this evaluation, the API website is still showing the same message that the fares information is coming soon. Therefore I had to drop the stage where I was going to implement this feature and instead I implemented the function which gets the departures information from the local station to one of the stations on the list. Maybe later when the API has the information then I may revisit the app to add this feature. Please see the limitations section at pages 9-10 for more discussions about this topic.

Essential Feature Evaluation – Conclusion

I can see from the essential features that I had said that I had to put into the app, I have managed to at least implement some of what the success criteria stated. I am really pleased to see that there was only one feature which I didn't manage to implement. The only reason why I couldn't implement that feature was because of the limitations of the API. For example I couldn't get the fare information. However, I am happy with the implementation of the features that I did manage to implement out of the essential features. I feel with a bit of improvement to some of the features, I will be able to fully fulfil the success criteria.

Non-Essential Features

Now that I have evaluated the Essential features. It is now time for me to have a look at the non-essential features. The table below shows my evaluation on these features.

Success Criteria	Partially, Fully or Not Met	Comments
"**The app could take railcards into account**"	Not met	While I feel that this would have been rather easy to implement. The reason why this feature is not met is that I couldn't get the list of stations based on budget meaning that I wouldn't be able to make this feature. (Turn to page 110 for a more Indepth analysis of this).
"**The app could also see whether the user wants to have any changes at stations or whether they just want a direct train to their destination**"	Partially met	The main reason why I say that this part of the success criteria is partially met is that I managed to get the website working with direct connections only. However, I never managed to program the stations where you must change at another station in order to get to that station. Therefore, in order to find what stations you can get to from a major interchange, you will have to set that station to the local station. One of the other reasons why I decided that I wouldn't be creating this feature is the API. Yet again the API was too limited for this stage to come into full fruition. I also felt that if I just set it to stations with direct connections it would keep the app from being too complicated. I also discussed this in the planning section of stage 4 (See pages 49-50 to see more information).
"**The app could take multiple passengers into account**"	Not met	Like a lot of the other features in the non-essential list on my success criteria, this feature relied on the budget feature. If you want to see a more Indepth discussion see page 110.
"**The app could let the user input the full name of the station**"	Fully Met	When I initially planned out my app, I was originally going to only let the user input the three-letter code of the station. So I was really surprised and impressed to see that the API could find stations based on the full name of the station or by the three letter code. I have a full discussion

		of the development of this feature on page 26 and I have testing on page 29.
"The app could show the user the departures information from the local station to a selected station from the user's list"	Fully met	When I realised that I was probably not going to be able to get the stations which require a change in trains (see pages 49-50 for more information), I decided that the better alternative would be to code a function which gets the departure times from the local station to the destination station. I would say that I am happy at how this feature turned out. I really like the fact that I managed to get the departures based on the max time to travel. (See page 56 for development on this). I feel that I have some very good evidence of this part of the website working on pages 61 – 64.

Future Developments

I feel that overall I have made a very functional program. However there are some things that I would want to do in order improve the program. The table below shows these features.

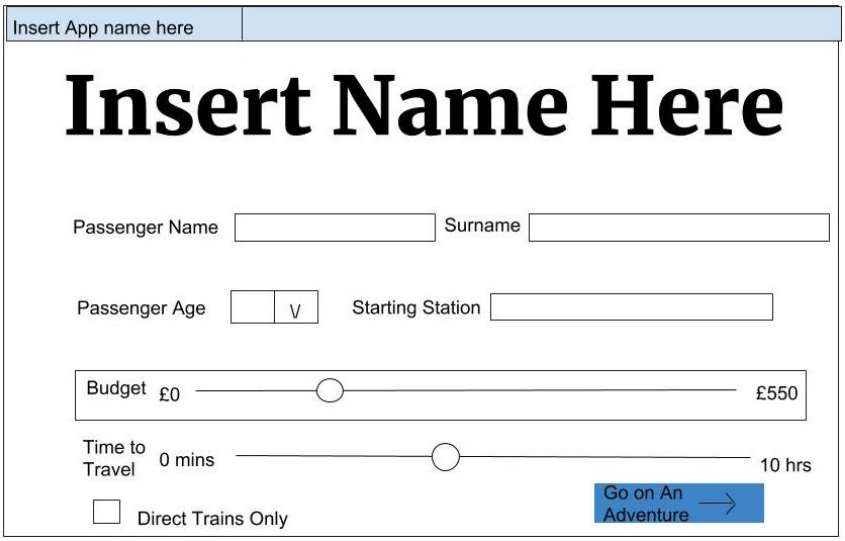
Proposed Feature	Comments and Explanation.
"**The app could show a list of attractions around the station**"	This was one of the original ideas that I had for the app. However when researching the API, I noticed that I couldn't really do it within the scope of my project. I mentioned the main problem with this feature in the limitations section on page 10. One thing that I will be able to note is that the map does show the user the surrounding areas around the station so that could technically be a feature which shows the attractions around some of the calling at stations. If I where to develop this part of the program. I will have to find a database on what attractions there are. I could maybe give the user a survey on what kind of thing the user would want to do when on a day out. Then I would then show the user the best stations suited for what the user wants to do.
"**The app could also filter the**	The main problem with this proposed feature is that isn't that feasible to code for. While a human will be able to tell the difference between a large central station from a small

stations to only city center stations" and "**The app could then filter out stations that doesn't have touristy attractions**"	village station, a computer unfortunately cannot make that distinction. Also considering that there are in all 2,566 stations in Great Britain, it would take a ridiculously long period of time telling the computer which stations serve a major tourist attraction and which stations don't have anything to offer and then you get into what the person is into. Therefore, I decided that I wouldn't develop this part as after further consideration, I felt that this wasn't something that you can have a computational approach towards. That being said, I feel that if I had the time, then I could try and find another API which would give me this kind of information. I could for example, try and find a way of incorporating the Office of Rail and Road Statistics to see which stations were used by different people. For example one of my Beta Testers said that they would rather the map showed the 30 biggest stations on the route rather than just the first 30 stations that came up from the running of the API (See pages 104 – 105 to see more analysis of what the beta testers thought of the map). If I had the ORR statistics then I would be able to implement this feature. In terms of stations that have more attractions, I could make it so that there would be an attractions count with the data from the "attractions function" that I mentioned above on page 114. If the attractions count is equal to 0. Then filter out the station.
"The app could allow the user to see the stations that they could go to change train"	As I mentioned in my evaluation (page 112) and in my development section (page 50), I couldn't really implement this feature. As I mentioned in page 50, I really wanted to implement this feature but I just didn't have the API requests available in order to make this feature. Therefore, in order to make this feature, I would have to pay to have more requests in the API. I would also need to make sure that I can work on creating an algorithm which could go down one line when there is a station in which the user can change trains at. After that the algorithm will then backtrack to another path from the junction station.
"The app should get the budget of the user so	As I have mentioned in many parts of this project report, my original idea for this app was to get the user's budget. In order to develop this program, I will either have to wait for the API to release the information.

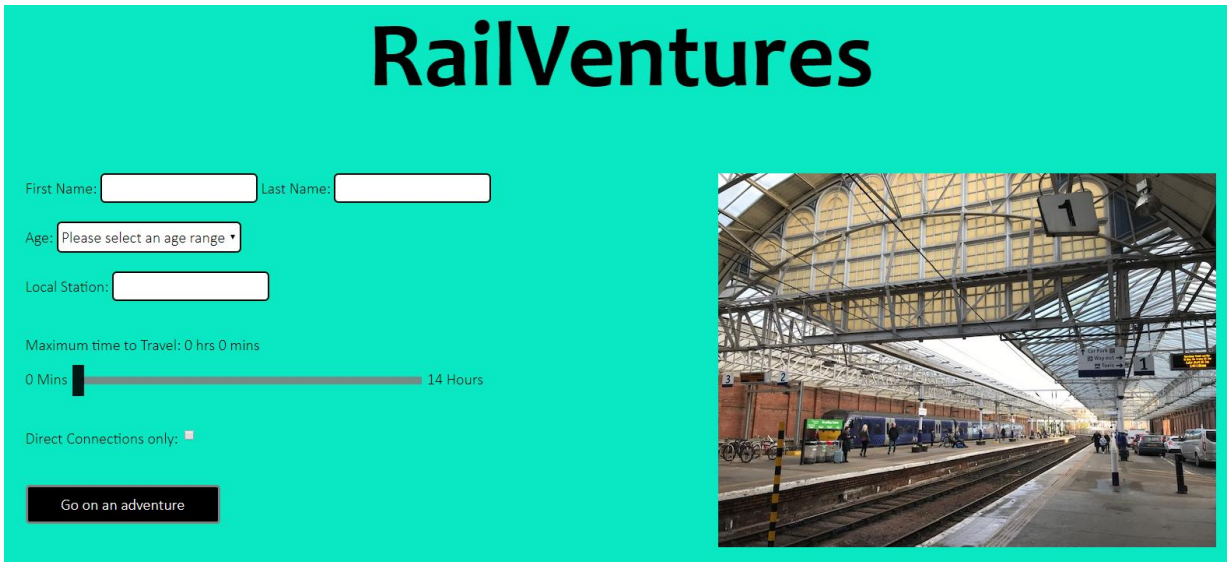
that they can find out where they can go to for their money"	As I mentioned in page 110, the API at the time of writing this report still hasn't got the functionality required to get ticket prices. Therefore, I will have to wait for the functionality to be released before getting to work on ticket prices.
Functionality on other browsers	One of the biggest problems with my app which I found in my robustness testing (please see pages 90-100) was that the website only worked on Chromium browsers. If you want to see my overall thoughts on this problem. Turn to the conclusion of robustness test 1 on page 100.
Loading Screen Graphic	A lot of my beta testers requested that I should have some sort of graphic which shows the user that the program is loading all the data to show the user. If you want to see more analysis of this topic turn to page 117. In order to make this part of the program, I will need to find a way of getting the program execute some code while the data is loading. I will then need to make the graphic disappear when the program is loaded.
Information for remote stations	As I mentioned in my testing in page 50. The website is not very friendly on stations in remote areas with a very infrequent service. This is yet again a limitation of the API. If I had more time and if I had the data, I would have made it so that the program will find the closest departure times whether that be in 5 minutes time or 5 hours' time. This will mean that people whose local station is in one of the least used stations in the country will be able to see what stations they can get to without having to wait until 2 hours before the train departs.
The app could be expanded to show the stations that someone can get to from Northern Irish Stations	Even though I have the API working for all the stations in Great Britain, I could expand the app further to make it work for train stations in Northern Ireland. That way I can say that the app works for every station in the UK. I also feel that I could make the app work for stations in European nations. (Like the Netherlands). When I was intially planning my app, one of the things I had considered was to make the app work for Dutch trains rather than British Trains. This is because the Dutch railways have a fixed fare system. Therefore I would look into finding APIs to get data from other countries to make this work.

Usability Criteria

In this section I am going to compare what I designed with the GUI and what the actual GUI ended up looking. For reference, here is the input screen as I had originally planned it. (Full annotation is on page 15).



And shown below is the final input screen (As seen on Google Chrome).



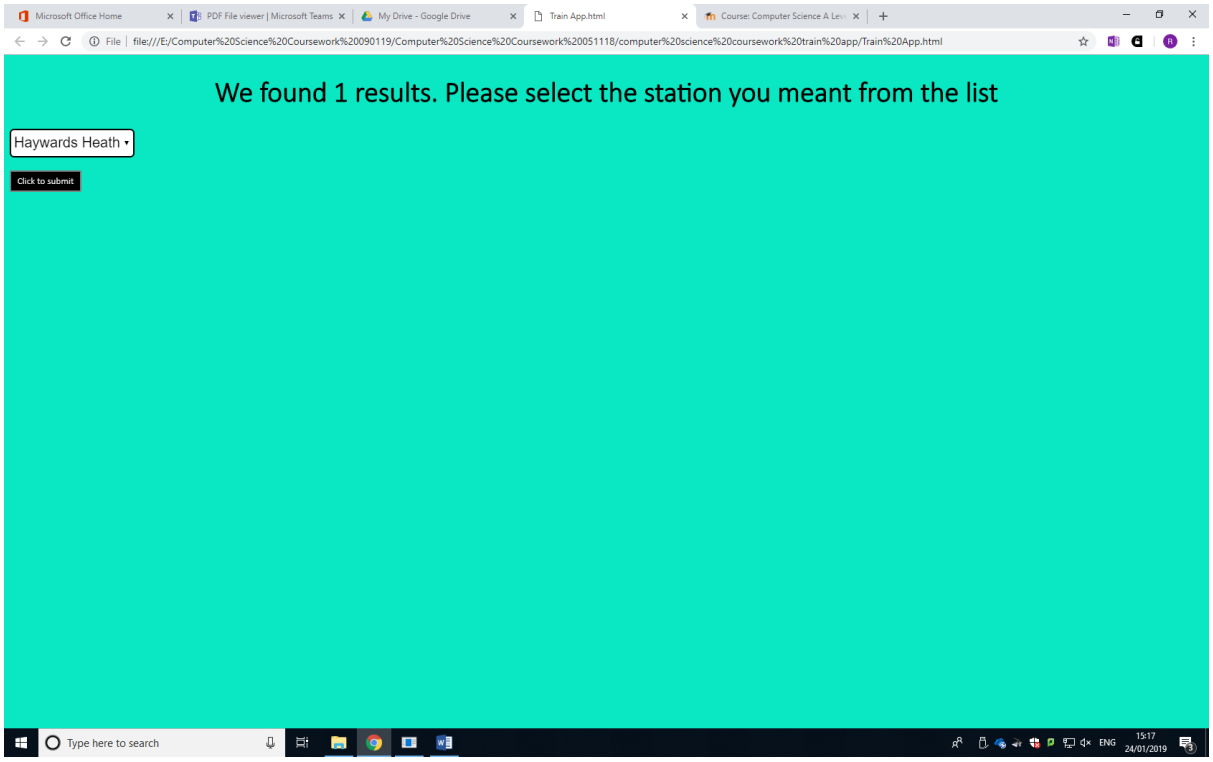
Looking at the planned input form compared to the actual input form, I feel that I actually made an improvement stylistically. For one thing I have given the input screen a much nicer colour than the white background that I had originally. I also feel that the picture to the right hand side of the form really shows the user what the app is meant to be about. This input screen was also rather popular with my beta testers as all of them rated it a 4 and above out of 5. Although some people did say that they thought that the directions could be

clearer. One thing that I will say is that when I was showing this project to people while developing this project is that they would usually go straight to the “Go on an adventure” button when the slider said “Time to travel” instead of “Maximum time to travel”. This is the reason why I changed the words on the slider. Overall I feel that this GUI design works well because it is simple and to the point.

One of the things I would say about this input screen is that some of the widgets in the form have been made redundant considering that the API didn’t have the data on certain things. For one thing the age dropdown box doesn’t really have a use later on in the program. This was because I was planning on having the program check that the user is a child which would have affected ticket prices. However, since I never managed to get the data on ticket prices, it means that I didn’t have a use for the age input. Another thing I will note as well is that the name widgets aren’t necessary for this kind of app. The only use for it seems to be to give the user a more customized message at the output screen. I feel that I would have gotten rid of it if I had more time. The last redundant widget of this screen is the “Direct connections only” widget. This was put in when I thought that I was going to make it so that the program would check stations in which the user could change trains so that they could see all the stations that they could get to within the maximum time to travel.

Station suggestion screen

Since this part of the program wasn’t really planned when I initially designed the app. I didn’t draw an initial design of this feature (see page 66 for the reasons why I decided to add this feature). The screenshot below shows the GUI of the stations suggestion feature.

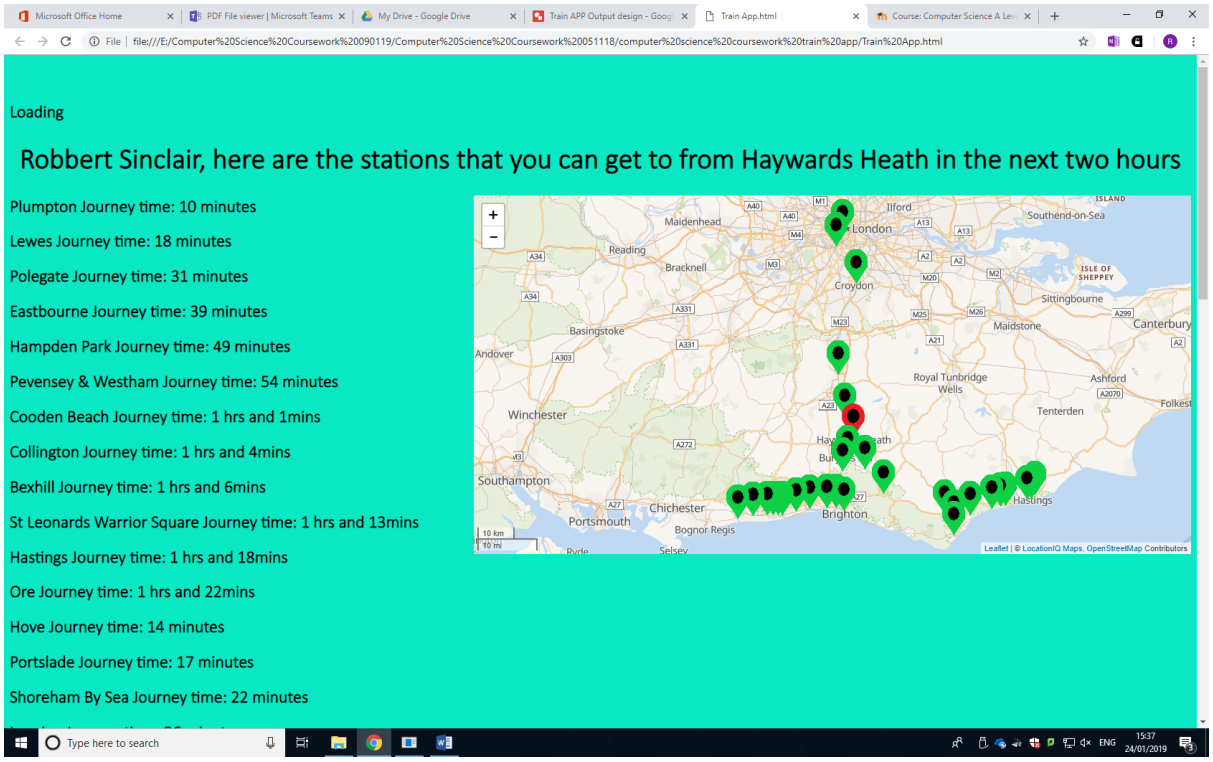


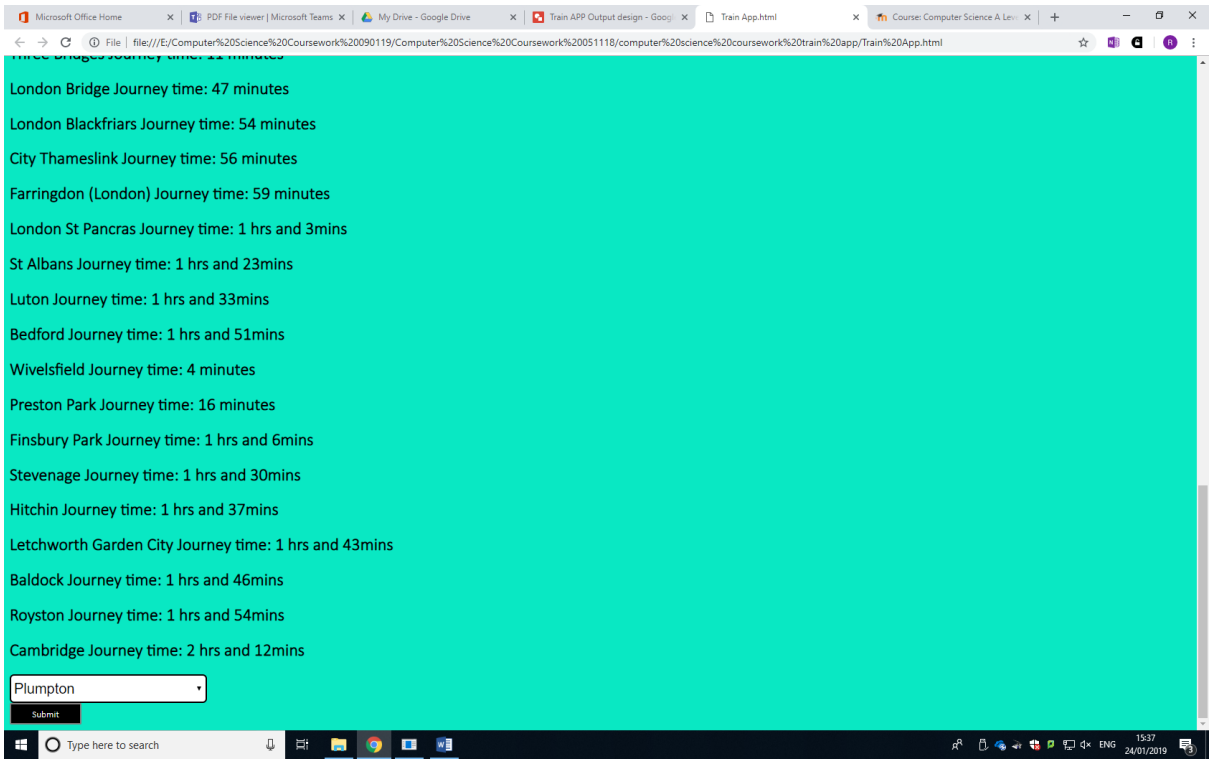
Looking at this screenshot I can see that this window is pretty bare bones but it does the job that it gives to the user. As one of my beta testers said ("It does what it needs to do"). One of the things I think I can do to improve this feature is find a way to make the program show the suggestions to the user as the user is typing like on national rail enquires. However, since this was a last minute addition to combat a bug that I was having with the code, this is a short term solution to the problem. If I had more time I would make it so that it would be more helpful to the user.

One of the things that the beta testers did say about this part of the screen is that there wasn't any indication to the user that the website was loading something. This is because the website does look like it has crashed when it is loading up the data to show the user. This is one of the problems of using Asynchronous AJAX requests (see page 33) where it takes longer to load up the program. For all the major browsers with the exception of Safari (see pages 99 and 100) which has an automatic graphic when the website is loading, no other web browser gave an indication of the app loading. A lot of my beta testers did complain that the app was "a bit slow" and that the "waiting time" was "unclear". Therefore if I am to develop this app further, I will make sure that there is a clear graphic to give the user the indication that the program is loading and to reassure the users that the app hasn't crashed.

The final part of the GUI that I needed to look at is the output screen. Shown below was the original idea for the output screen in the app. (Full annotations shown on page 16)

On the next page is the final Output screen.





Out of the input and the output screens in the app, I feel that the output screen had the most radical change from the original design. For one thing I didn't manage to make the mini form that was meant to be towards the left of the screen. The closest thing to that would be the dropdown list at the bottom of the screen which gives the user the services between the local station and a specific station that the user has specified. I feel that the main reason why I

decided not to put a second form on the output form was that I wanted the input to be dynamic so that the map function would automatically update with all the extra stations. I feel that with the way my website works it would just take a whole lot longer just to update the map.

One of the things I am happy about though is the map function. 6 out of 7 of my beta testers rated the map a 4 or above. The main feedback that I received from this part of the GUI is that they would much rather have the drop down list at the top of the station list rather than right at the bottom. I would agree with these comments as I would always have to tell the people using the app to scroll down to the bottom to get to this feature. I feel that for this GUI to be more user friendly, I should be able to just let the user use the app without me helping them guide them through the app.

One the other things my beta testers told me was that they thought that the ordering of the station list was confusing and that they would much rather have the station names in alphabetical or order in terms of time. I would definitely say that I would want to improve the code in that way as the current way that it is ordered in. As shown on the code screenshot on page 41, the program only puts the station into the list if it fits the criteria but it doesn't do anything to really sort it into a more logical order. I feel that if I had more time, I would have implemented a sorting algorithm to help passengers find the station that they wanted.

The last detail that I didn't put in from my initial design of this screen was that I didn't put the time on the pop up window (See page 89 for an example). All the times for the stations are placed in the text to the left of the map. I feel that looking back I should have made the effort to put the time in the popup especially considering the confusing layout of the station list.

One of the last parts of my app is the screen which shows the trains that can get the user from their local station to one of the stations on the list. The screenshot below shows the screen that shows me the trains from Haywards Heath to Gatwick Airport.

I feel that this part of the GUI does the job that it set out to do. However, I would personally make some improvements to make this screen even better. For one thing, I would want to give the time it takes to get to the station on the label. I feel that this would be a lot more useful for the consumer as it means they can stay on this website and not have national rail enquiries on another tab. A lot of the beta testers said that they would want to see information about delays and disruption. I feel that this could be useful for this feature, however I feel that the information that I am providing is what stations you can get to from your local station and the trains you need to get to that station.

One of the other things that I would change is I will have a proper table which stores the departure information. One of the things that I would do is get the logos of all the train operating companies and put them in the train operating companies field. I feel that this would make this part of the website more visually appealing.

Another part of this window is the multilingual information for international trains. Here is the variant of this part of the departures window.

I feel that this feature is quite useful for people who want to know what trains they can get to in international areas in their local station. I could also improve this feature to include information from the airport. I could also make a feature in which the user can decide between English and Welsh. I feel that this would be very useful for people in local areas as there aren't as many transport applications in the UK which doesn't have any information in Welsh.

Evaluation – Conclusion

Overall I feel that I have managed to create an easy to use and functional app for my stakeholders. I feel that I have managed to put enough functions to give a passenger an informed view of what stations they can visit within a certain amount of time. However, I will say that there are a few small things that I would want to change. As mentioned in my future development section, there were a few things that I would want to change to make the app appeal more to tourists who want to use it for a day out like having more information on things to do while there. I feel that I have managed to fulfil all of the most important parts of the success criteria and the main reason why I haven't done some of the success criteria is that there wasn't the available data on the API so I would have to return to the app. I will also have to look at bugs to make sure that this app runs perfectly and without fail.

Appendix - Code Listings

Train_App.html

```
<!DOCTYPE html>
<html lang="eng"></html>
<html>
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet"  href="input_style.css"/>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4=" crossorigin="anonymous"></script
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css" />
    <title>RailVentures</title>
</head>
<script src="input.js">
</script>
<header>
<h1 id="demo"><strong>RailVentures</strong></h1>
</header>
<body>
<section>
<script src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
<script src="https://tiles.unwiredmaps.com/js/leaflet-unwired.js"></script>
<div id = "station_img">
    <img src = 'image1.jpeg'/>
</div>
<form name= "input_form" >
    First Name:
    <input type="text" id = "first_name" name="First Name"/>
    Last Name:
    <input type="text" id = "last_name" name = "Last Name"/><br><br>
    Age:
    <select id = 'Age' name = "age"><br>
        <option value = "default">Please select an age range</option>
        <option value = "under_16">Under 16</option>
        <option value = "16-30">16-30</option>
        <option value = "30-45">30-45</option>
        <option value = "45-65">45-65</option>
        <option value = "over_65">Over 65</option>
    </select><br><br>
    Local Station:
    <input type="text" id = "local_station" name="Local Station"/><br><br>

    <p>Maximum time to Travel: <span id = "hours"></span> hrs <span id = "time_output"></span> mins</p>
    0 Mins
    <input type="range" min = "0" max = "840" value = "0" class="slider" id = "Time"/>
    14 Hours<br><br><br>
    Direct Connections only:
    <input type="checkbox" name = "direct_connections" value = "direct_connections"/>
    <br><br><br>
    <input type="submit" id = "submit_button" value="Go on an adventure" onclick="getValues()"/><br>
    <script>
        let slider = document.getElementById("Time");
        let output1 = document.getElementById("hours");
        let output2 = document.getElementById("time_output");
```

```
        let hours = 0;
        output2.innerHTML = slider.value;
        output1.innerHTML = (hours);
        slider.oninput = function(){
            hours = parseInt(this.value /60);
            output1.innerHTML = (hours);
            output2.innerHTML = (this.value - (hours*60));
        }
    </script>


</form>
<script src = "input.js"/>
</section>


</body>

</html>
```