Technical Report

Journey Organiser

CO600 Group Project

Supervised by

Dr Peter Rodgers

Mateusz Maly
mfm9@kent.ac.uk

Jan Gucwa
jg404@kent.ac.uk

Filip Borowiak
fb225@kent.ac.uk

Karol Baran
kb440@kent.ac.uk

Dawid Janelli
ddj4@kent.ac.uk

Abstract

Journey Organiser is an Android app and a live cloud (AWS) based Java service to help users to plan their journeys. Currently, Google fails to provide indicative travel costs. The primary novelty of this service is the integration of the pricing API with the Google mapping and routing service. The server utilises a pricing Rome2Rio API with multiple mapping and routing Google APIs, to gather the various data associated with each route.

On the app, the user selects the origin and destination locations, the travel dates and the preferred transport modes. The available transport modes are walking, cycling, coach, public transport and driving. The user is then presented with a series of travel routes each with a price, journey distance and travel time.

1. Introduction

The application allows the user to search for travel options to their destination. It is a collection of APIs implemented together to return accurate information on desired journeys. The idea was born when one of us thought of an all-in-one app which would guide the user in their travels. We wanted to create an application which is user friendly to use but reliable and accurate.

Collection and handling of data is the most challenging part of creating such application. Journey Organiser uses APIs to collect live data from Google and Rome2Rio and returns parsed information to the user. The server is permanently hosted on AWS (Amazon Web Services) and can be accessed at any time from the mobile app or the internet browser.

2. Background

At present there are many applications designed to find information on a given journey. Through our research, we have found that this information is not always complete. For example, you could find route details, time, duration and a map on Google Maps but it might not always return the price for such journey. Thus the idea of implementing our own system which would collect the same information that all the other similar applications do, but it will return as much information as is available.

One of the other motives for taking on this project is because it was challenging. We didn't know if we would be able to fully implement the application until we have completed our research. None of the group members had previous experience in data collection and handling. Therefore, we aimed to gain a great deal of knowledge and new skills through the project.

3. Aims

The main aim of the project is to deliver a fully functioning, easy to use application which is implemented on both Android and as a website. The design and functionality of the app and the website must be consistent. The user will be able to search available travel options by choosing an origin and destination locations, time, date and a type of transport. The application should return all available routes and display information, such as: map with the route of travel, journey duration, price and the distance. The user selects the transport mode they wish to use or select 'any', which will display options for each transport mode. The user can sort the results by choosing the cheapest or fastest travel option.

Both the app and the website need to implement a simple design and be easy to use. Error handling system has been put in place to make the application user friendly. Errors guide the user to a solution such as an empty string error points to which field needs to be completed.

In the project scope, an optional features section details some aspects of the project that could have been implemented if there was some spare time at the end. One of such features was the user database, which

could have allowed the user to save their details, favourite journeys and transport options. Implementation of the database has been abandoned because of delays in completion of the basic functionality.

In terms of our intangible goals, the initial plan was to learn as much about data collection as possible. None of the group members had previous experience in data mining or handling, therefore this project was a brand new experience to all of us. We have all gained a great deal of knowledge throughout the project and are all much more confident in developing such applications.

4.1 Server

The purpose of the server is to:

- Receive user input sent from the app and the website
- Parse the input and request results from relevant APIs.
- Send the results to the app and the website.

The server is the most important and complex aspect of our application, both the app and the website connect to it in order to send and retrieve data. It is written in Java and consists of seven classes. There might be better languages to write a server in, such as Python, however, after careful consideration of other options, we have decided to stay with Java as we all know it and it would be too time consuming to learn a new language. Furthermore, it was also very important to get the server working as soon as possible because, it was the main part of the system, without which the application can't be run.

The communication between the server and both the app and the website is shown in figure 4. The server serves as a receiver and sender of data. It will also request particular information from the APIs. When the user fills out the information needed to find a journey and clicks search, the app or the website will put all that information in form of an XML String and send it to the server. The information received is then parsed by the server and dealt with accordingly, for example if the user has provided the desired transport mode of their journey then the server will pass that data to the APIs, which will provide results based on that data set. The communication between the APIs and both the app and the website is the most important job of the server. It requests results from Google and Rome2Rio APIs, which take the information supplied by the user and supply the results for each route. The output includes the price, distance and time for that journey, as well as the addresses of the locations and departure and arrival dates.

The server is equipped with many error handling methods which will attempt to pick up any errors that might occur. One of such errors is the location not found error which occurs when the location supplied by the user is not recognised by the Google geocoding API. Another error that the server is prepared to handle is the date in the past error which is triggered by the user providing a date which is in the past. For both these errors the server will send an error coding message to the app or the website.

The most difficult part of implementing the server was connecting it to the app and the website. It has been very difficult to distinguish whether it is the code that is wrong or it's the port forwarding or firewall that is not letting the connection through.

Server is deployed to a .jar file so it can be run from any kind of computer which has port forwarding. This is the back up in case the main server goes down. We can easily host the server on another machine and keep the application running. The main server is hosted on the AWS (Amazon Web Services) and is available all the time. Hosting of the server was established at a last minute notice, however, it works perfectly. The speed of the application is not affected by the cloud hosting which is what we were concerned about when we first hosted the server.

4.2 Android App

The purpose of the app is to:

- Enable the user to easily input data into the system.

- Send the user data to the server in an XML format.
- Receive the results from the server and display them to the user.

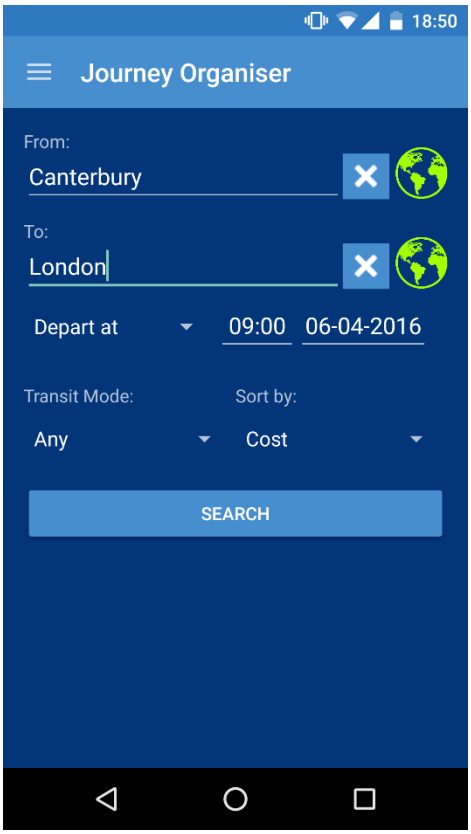

Figure 1 Main Interface

The app allows the user to input the information needed to plan their journey. The user is required to provide origin and destination of the journey. These can be either entered manually by the user, or chosen from an interactive map provided by Google API. Address search is also implemented using Google API, as well as auto complete to make it easier for the user. Time, Date and transport mode are optional and don't have to be supplied by the user. If the user leaves them blank, then the app will search for the next available routes using any available transport. If the date is filled out then the app will include them in the message to the server, which will then respond with available routes on the date provided by the user. The users can choose which transport mode they wish to use, if public is their preference then only that option will be displayed in the results. Figure 1 shows the first screen which the user will see after running the app.

Figure 2 Results Page

After the user makes their choice, all the results are displayed in a new activity which consists of multiple rows with different routes. Each row contains an image of the transport mode, departure and Arrival time and date, duration of the journey, distance from location A to location B and the indicative price for that journey. The user can use those details to compare the routes and choose the preferred one.

The sorting of the page is set by default to distance, however, the user can change that in the main page, or change it on the results page, using the multi toggle button at the top of the page. The sorting is done prior to displaying each view, therefore, there is no loading when the user changes the view, which makes it more user friendly.

The Android app has been developed simultaneously with the server. Android studio has been used as an IDE. The app uses API level 21 (Lollipop), and above. Originally the app was using API level 16, however, the problems with the outdated API

seemed to outweigh the benefits of using the older version. We were having problems with the UI of the app, especially the colour schemes. From our research we found out that most Android devices use newer versions of the API so there was no point in limiting the app for the sake of few devices. Changing to API level 21 has greatly improved the performance and fixed a lot of the issues with colours.

When the user decides on their preferred route, and clicks on one of the rows, another activity is opened. The last Activity contains more details about the journey, such as the addresses of locations A and B, as well as an interactive map with a plotted route. Figure 3 shows the journey details from Canterbury to London when the user selects one of the driving options.

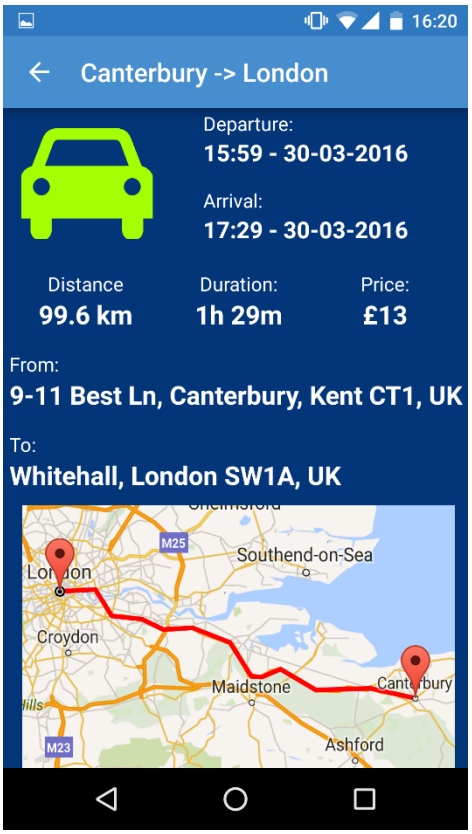

Figure 3 Detailed Results Page

During the integration of the map API for the app we came across many problems. The first version had the map display an accurate route from point A to B. Google API was sending us encoded coordinates, which we then decoded on the server and sent to the app. However, this method has not worked properly because the encoded message was different to the one the server decoded. As a workaround we used a version with no encoding. That seemed to work, however, if the points were too far away from each other, for example if the user chose a journey from Canterbury to Manchester, the app would crash because the message being sent was too long. A solution to that problem was to use encoded coordinates from Google API, but not to decode it on the server. Instead the encoded message is sent from the server and decoded on the app. As a result the app receives a smoothed encoded line which is an approximation of the route.

Another problem that has occurred during the implementation of the app is the issue with the journey search. The first time the search is run it works perfect, but after the user goes back to the main screen again and tries to search again, the search button has to be clicked twice for it to work. This issue has since been resolved and it turned out that it was the internet connection that was causing the problem. The app worked fine when mobile internet was used, but when Wi-Fi was turned on, with the server connected to the same network, it was causing problems.

We also had a problem with the hash maps, which we have used to store the results received by the app from the server. The problem with that was that the order of these results changed every time the hash map got resized. That became particularly frustrating when we have implemented the sorting of the results. Our solution was to split the hash map into two hash maps. In the first hash map we are storing the generic information about the chosen journey such as origin and destination. This information will be the same for every individual result. The second hash map contains mappings of the results indexes to hash maps with details about each route. When the application is run and the user searches for a journey, the hash map with the results will become a list of integers and a list of hash maps. Straight after, these lists are again swapped into hash maps. That way we are preserving the order of the results that we put into the hash maps. This solution has since been changed and an easier and more efficient has been put in place. Array lists are now being used instead of hash maps. The difference between them is that array lists store items inside in an

ordered fashion, whereas hash maps don't guarantee the same ordering.

4.3 Website

The purpose of the website is to:

- Enable the user to easily input data into the system.
- Send the user data to the server in an XML format.
- Receive the results from the server and display them to the user.

The website is a browser representation of the app. It is consistent with the app in terms of colour schemes and images and it also requires the user to provide same input as in the app. The website has been written in HTML and styled in CSS, communication with the server is being implemented in PHP whereas any dynamic functionality was done using JavaScript.

Figure 5 shows the home page of the website, this is where the user makes the choice of origin and destination location and decides on the optional search criteria. The design is being consistent to the one of the app by having same buttons and images for transport icons.

Members of the team that were responsible for developing the Website found connecting it to the server to be most problematic and time consuming. They found it challenging to keep adapting to the constantly changing server.

Figure 6 shows the results page, which displays the results for a given journey and allows the user to decide on their preferred route.

Whilst working on styling and design of the website, the web development team has run into some difficulties. Their main approach was keeping the CSS and JavaScript clean and organised so that any future changes could be implemented quickly and efficiently. Adapting the style of the website to match that of the android app proved challenging. For example, finding the libraries for date and time pickers which looked similar to the ones used on the android app was difficult. Browser compatibility has also caused some issues. Certain styling features might appear different depending on what internet browser the client is using. Some design tags were completely unsupported, mainly by Internet Explorer, in which case alternative appearance had to be created. Same goes for the accurate visual representation of the website at different resolutions and zoom ratios. This required frequent mini-tests during the development to ensure that the styling does not 'break' for variety of different possible displays.

The team also faced problems with JavaScript applied to checkboxes, which they have resolved by using CSS3 only. Showing and hiding map tabs resulted in maps with no content shown. This was another case for which the team could not find an acceptable solution. Their workaround was to resize the maps to almost invisible size instead of hiding them.

5. APIs

To provide the user with prices for the journeys we have used the Rome2Rio API which enables us to extract relevant information, including indicative cost of that journey. Rome2Rio offered the best API for our project as it gave us a price for every type of route that we searched using Google API. Therefore, we have decided to keep using that API as it was already late in the project. Another alternative would be screen scrapping the prices for each route from websites such as TFL. However, without any prior knowledge on this topic we have decided against such approach as it seemed to be very time consuming to implement and validate.

Google has provided us with multiple useful APIs that we have used throughout the project. Implementing these APIs has been difficult at first, however, after we have learned how to do so, it was easy. Google provides useful guides and tutorials that we have used to be able to implement the APIs on our application. One of the difficulties we came across occurred right at the start of the project. Google API was giving us only one result for each journey and we struggled to find a solution. Only after a thorough research

we managed to find a solution. It occurred to us that we have been using a wrong method, which called for one route only. Changing that method resolved our problem and we were able to continue.

6. Testing

As a team we have adopted agile software development and testing has been a big part of our project. Our software has been tested throughout the project, however, most tests have been performed at the end of the project. The tests have proven to be very useful as a number of big errors have been found. For example, the apps results were not displaying accurate data which was caused by ordering of the Linked Hash Maps.

Some of the error messages that are produced by the server are: 'invalid request', 'location not found', 'is in the past', 'no route found'. Invalid request deals with the XML send by the app of the website, which is not in the valid format. The user does not have direct access to the XML output so this error should only occur when the user tries to alter the message to the server. Location not found is displayed when the server can't find the location that the user has submitted in the input boxes. Is in the past error message is displayed when the date or time entered by the user is in the past. No route found is displayed when the server is unable to find a route from the user's origin to destination locations.

7. Quality Assurance

Certain rules have been put in place to make sure that the code and documents are of good quality. Quality Assurance document has been drafted to outline those rules and then QA form has been made to make sure that each piece of code and documentation has been checked. This method proved to be very time consuming, however, it highlighted the areas that need improvement, such as code without comments and documents with wrong style.

Using the feedback from the Quality Assurance forms, we were able to make the necessary changes to the relevant items. Following the first iteration, a lot of changes has been made to both code and documents because they have not met the quality assurance standards. The most commonly occurring mistakes were the lack of comments in the code and wrong style for the documents. These errors have been fixed and documented.

8. Project Management

Managing of this project proved to be very challenging, especially towards the end of the project when a lot of work had to be completed in a very short time. Overall the group did well, however, the differences in work style and contribution of its members proved to make a big difference in the final outcome of the project. Different coding abilities meant that multiple people had to work on the same elements of the application. As a result, additional parts of the implementation were abandoned due to lack of time.

The first phase of the project went really well, with good input from all its members, however, that contribution has soon came down to a minimum and had only recovered in the second term of the project. A number of methods has been put in place by the project manager to make sure everyone in the team is contributing and completing their work on time. First email and verbal warnings have been given in the first weeks of second term due to lack of contribution to the project. However, these proved to have little effect, only after a discussion with the project supervisor, the members have started to contribute more to the project. This did not have a lasting effect as the contribution has then fallen to a minimum. All of this can be seen through the number, frequency and quality of commits on the GitHub repository. [1]

8.1 Planning

A number of project plans were drafted in an effort to highlight the key areas of the project that need implementation. These project plans were consistently

uploaded into the group repository by the project manager which enabled every member to see what needs to be done. Although the members were required to follow the project plan, the number of deadlines missed, meant that the plan was put offside and not used often by the group members. Other ways of ensuring the group completed their work were put in place, such as emails with deadlines. These proved to be more successful at making the group complete their work on time.

The group held many meetings throughout the year to make sure everyone is up to date and completing their work. Weekly meetings were established and meeting notes completed by the project manager. Furthermore, social network communication was much more often used and preferred. A number of group coding sessions was also established, mainly to get everyone up to date, especially because the website team had problems with connecting the website to the server. Although these sessions proved to be useful in terms of progress on the project, the general feedback received from the group members has been that the sessions have been useless and they could have done the same on their own.

9. Conclusions

We have managed to create a working application which is able to retrieve the information passed by the user, parse it and respond to the user with relevant data. The user is able to use an Android app as well as a website. The app and the website are consistently designed and implemented, with same images and logo used. Both the app and the website have been put through thorough testing and all bigger errors have been fixed or workarounds found. Any errors or user input missing is communicated through various error messages and hints in the UI. Resource usage has been minimised since the first version release by implementing more efficient methods such as a ‘smooth map’ API from Google which encodes the locations coordinates and makes the message substantially shorter.

Our application performs well in comparison with other projects such as this. It is difficult to compare it to other travel search engines such as Google because we are using Google APIs in our project, but we are also adding cost of journeys that Google does not always provides prices on. This makes our project more unique. Although the prices are not always completely accurate, they are an indication of the cost of travel and should give the user a good idea of the actual price.

The best advice we can give to others doing similar projects is to conduct your research thoroughly. If we could redo our project we would take a different approach to our research, we could have completed more tasks if we didn’t spend that much time on fixing problems that were either caused by our lack of knowledge in this area or wrong resources being used. The difficulty in our project was that we actually didn’t know what data we can get, we found lots of APIs that we could possibly use, however, without trying them we had no idea if they were going to work with our project.

Our initial project scope included a user database which could be used to save favourite journeys. However, we have underestimated the time it will take us to finish other tasks in the project. Fixing errors has been particularly time consuming. We had numerous problems with the APIs which also made the project rushed in the final weeks. Another future improvement would be to add flight routes, this was in our initial scope, however, we had to remove it because through our research we found that Google API doesn’t include flights. This has since changed as Google does currently include flight routes in their search engine, therefore adding the flight information would be quite easy.

The last improvement to the application is to add search options not only for the UK but the rest of the world. Our application allows the user to search in the UK only, however there is an option of expanding it because Google API allows you to search a route pretty much everywhere. The difficulty in this though is the foreign signs that will come with areas outside the UK. That is one of the reasons we decided to stay in the UK, at least until we can get all other functionality implemented.

Acknowledgments

First of all, we would like to thank our supervisor, Peter Rogers, who has guided us through the project and proved to be extremely helpful in every aspect of the project. It is fair to say that without him we would not be able to complete the project.

I would also like to thank my good friend, Navrit Bal, who has advised us on multiple occasions and also implemented the loading spinner.

References

[1] Groups repository on GitHub

https://github.com/dajposkakac/CO600_TransportProject

[30/03/2016]

Appendices

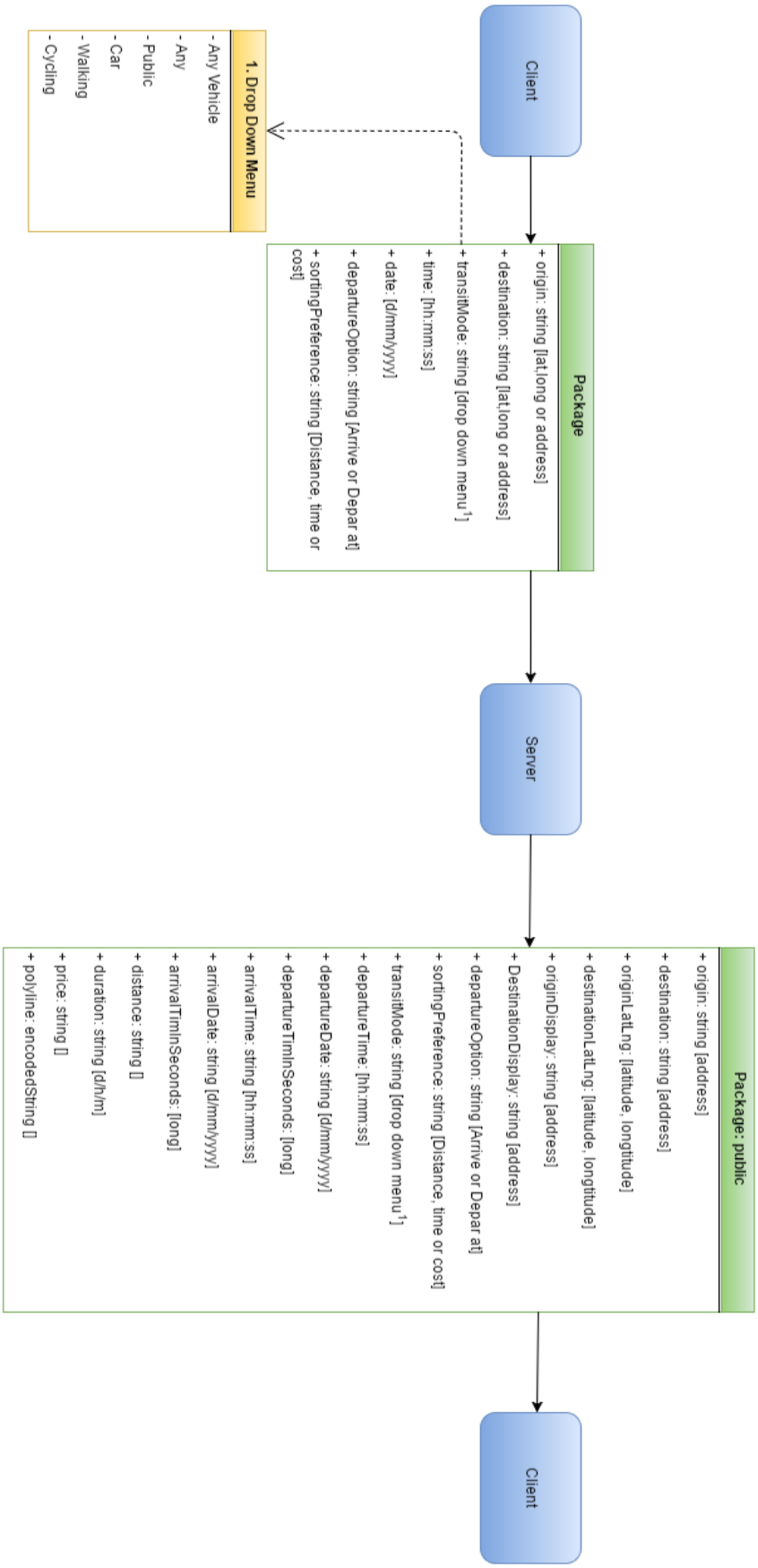

Figure 4 Client to Server communication

Figure 5 Website home page

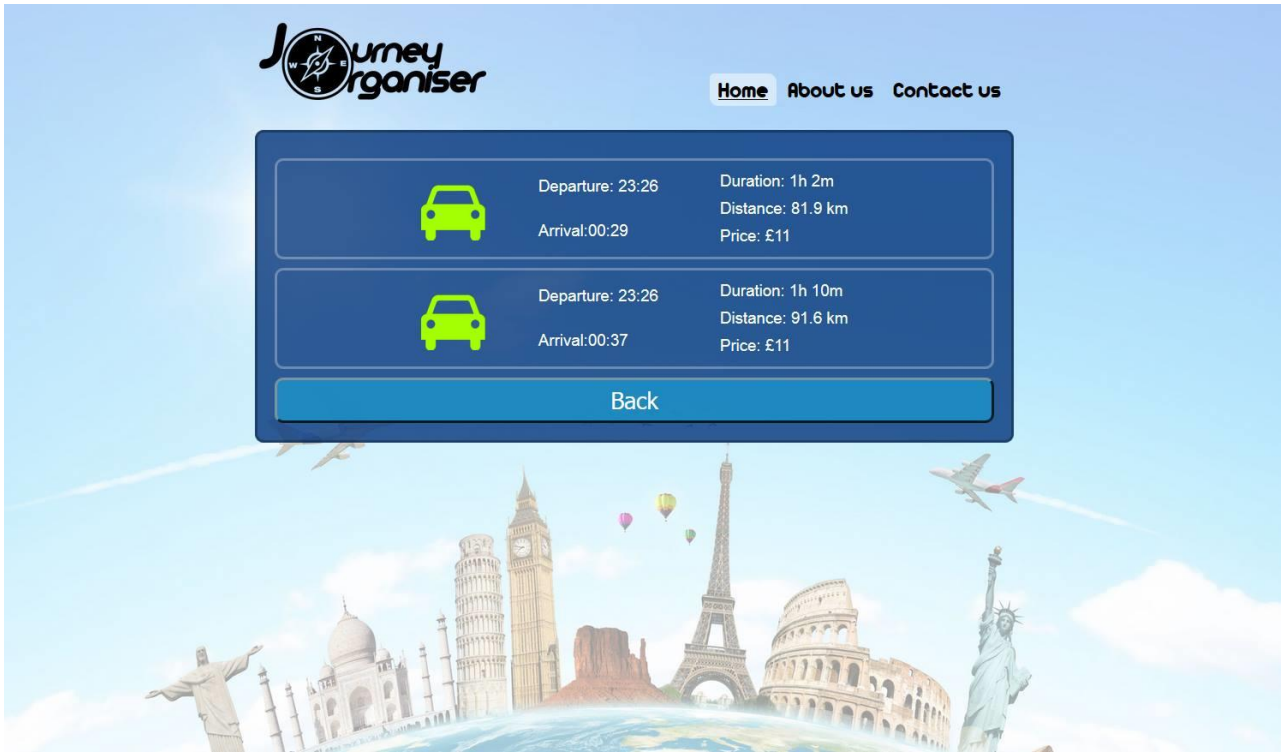

Figure 7 Results Page