**ENTERPRISE COMPUTING COURSEWORK - Team Transporter**

        The goal of our project was to provide users with the ability to explore Edinburgh city using just a £3.50 day ticket. Users would be able to choose different attractions that Edinburgh had to offer and would be able to find the most suitable route for them to take in order to reach their destination. This project is therefore aptly named “See Edinburgh” and has been constructed using Python as its main programming language and Django as its framework. This report will attempt to describe thought processes behind decisions as well as gradual development of the project in chronological order.

Our team would meet every week on Friday in order to discuss potential future plans as well as the progress that we were making. This was useful as it helped us keep up to date on what each member was doing as well as trying to understand their thought process and how to follow each other’s examples from a coding perspective. This proved to be more difficult than originally anticipated given that everyone had their own involvements elsewhere, however we saw improvements in communication as the weeks went by.

The early stages of development revolved primarily around coming up with a design that would satisfy all provided functional and non-functional requirements. As a team, we decided it would be best to create a webapp, which would allow it to be “cross-platform” hence the decision to use Django. Django has highly descriptive documentation and as such was ideal for our team as not everyone was familiar with it and it was necessary for us all to have good working knowledge of the language and framework we would decide to work with.

We decided to use a design template built on the Bootstrap CSS framework, as this gave us a strong starting point and built in responsive design.

Originally, our sights were set quite high and were needed to be toned down to give us a realistic goal that could be accomplished. We would have to refute early plans to have things like “descriptions of places displayed while on route” and focus on what would be the core functionality of the webapp which would be getting routes to desired attractions.

The first thing that needed to be determined was how to extract the data from the given database as well as find out the format in which the output would be displayed. Using the ‘requests’ python library, the api was queried using the online guide and methods were implemented that matched the api’s functionality so that they could be called internally in the app with ease. The data retrieved was inspected to understand how to store the information. It made sense to store as much static information as possible such as the bus stop information and retrieve information like departures and bus times on the fly as this information was in constant change.

Once the data was able to be seen via the api.py file, it was important to be able to store this information using Django in an organized manner so that we could begin to use it to get some functionality for our website. The BusStops and Stops models were then written in order to determine which information we found necessary to store and database.py would then iterate through the data returned from the API and store it in our database. The data returned was in the form of dictionaries within dictionaries. This made accessibility easier as it was reliant on keys making it quick to store.

The most important attributes to store were “stop\_id”, “latitude”, and “longitude” as these would allow interactions with our theorized journey planner which would be responsible for determining the best route for users to take in order to get from on bus stop to their attraction of choice. The Stops model would therefore become the staple of the webapp and was most vital to keeping the identity of the website intact. Another model that was created was the Attraction model which would have a corresponding bus stop that would give it a latitude and longitude.

Between these two models, it seemed that there would be enough to work with that a template could begin to be implemented (base.html, app.html). Following standard practice, html files were stored in the templates folder.

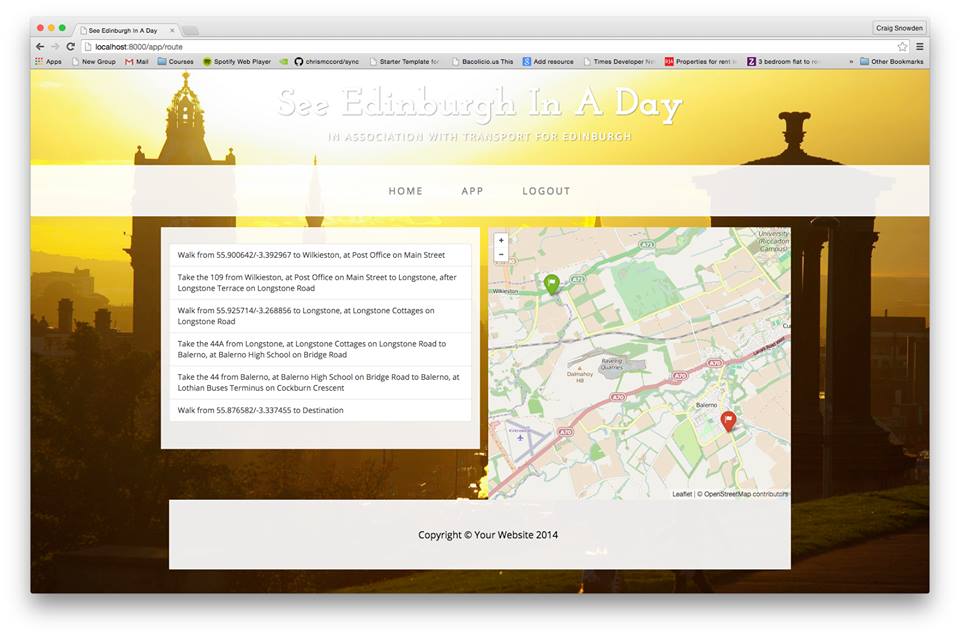
“Base.html” acts as the homepage of the website and all other html files extend it. The title of the webapp is displayed at the top of the page with a description of our team filling the rest. “App.html” would then be where the user can select his current location and his desired destination. This page would have two drop down bars for each of these options as well as an obvious “Find A Journey” button that when clicked would then retrieve a list of the available routes.

The next step was to then create a profile for each user which would allow them to login and logout giving access to “member-only” parts of the website. The built-in User model would be utilized featuring attributes “username” and “password” and an added “picture” attribute that was optional. In order to give this model its desired functionality, methods register(), user\_login(), and user\_logout() had to be introduced as well as login.html and register.html. These would then give the individual the capability to sign up to the website, and login and logout if preferred. Signing up would grant the user access to the app whereas non-members would be redirected away. This sign up process could be used to allow users to store routes in their profile for future reference.

Finally, the journey planner method needed to be introduced. The API offered a directions() method that would retrieve a set of journey “legs” from one point to another. The input required was in the form of latitude and longitudes hence the necessity to store both those data points in our Stops model. The method can be found in the api.py file. Sometimes multiple journey possibilities are offered, ordered by duration. For this application we choose the first (shortest duration), however a viable extension would be to determine the “easiest” route - a route with the fewest connections or walking required. Perhaps this could be provided by Transport for Edinburgh.

The route display itself simply lists out the “legs” provided by the API. A user can select each leg to see a map with the start and end points marked. This map is provided by Open Street Maps, using the Leaflet.js library. The volunteer community behind OSM have provided comprehensive mapping of the city, right down to the position of bollards and fencing. This will therefore serve as a useful point of reference to a tourist.

The route() method provided the greatest amount of difficulty. Available stops/attractions were selected using a drop down list provided in the app.html page. In order to retrieve the corresponding Stop objects a Stops.objects.get() method was used which would query the database for an object that had the same stop\_id (provided stop\_id is unique). The longitude and latitudes could therefore be deduced and used as input.

****

The difficulty lay in the fact that the method directions() provided by the API (written as journey\_plan() in api.py) was not working as anticipated. The image above was taken when using dummy values as input. In short, the API has stopped returning results on the journey endpoint. The dummy data used is from a previous successful attempt with the API, so we anticipate that should the endpoint start working again the route display should work equally well on live data.

As a result, our application is not able to work as intended however we have displayed the reasoning why it does not. We can successfully obtain the location (longitude and latitude) of any bus stop the user desires however that input is rejected by the directions method(). This was an unforeseen issue and we should have looked to test it more thoroughly before attempting to implement it.

**Iain** was primarily working on setting up the django project, retrieving the data from the API (api.py, transport.py) and finding out how to properly work with the given methods. **Lucy** was assigned the task of working with typescript but did not complete it in time. **Nicolas** worked with the data trying to store it in the most optimal manner so it could be easily accessed in Django (database.py) as well as constructing the models (models.py), admin (admin.py), views (views.py) and the urls (urls.py). He was also responsible for the authentication methods (templates/app/authentication and app.html). He also took over typescript responsibilities. **Craig** worked primarily with the API’s directions() method which would become the basis of our route() method and gave the desired base functionality of the website in route.html. The project’s commit history has been attached for further details on participation.