Enterprise Computing

Report on the web app 'Bus Stop Bonanza'

Team Chekov

Description of the Web App

"Bus Stop Bonanza" is an an online web app which uses the Edinburgh Transport API. It features a homepage or "feed" which lists all the available buses nearby the user with their destination times. If one of these buses is selected, it is assumed the user is now onboard the bus and the app shows the "tracker" page which displays the current bus in question with it's next stop. The tracker page then reads aloud the next stop of the bus. After a short delay, the app checks the user's location and reads aloud the next stop. This then continues until the user exits the page presumably finished on their journey, or the bus service ends. The app is live on a cloud server and can be viewed here -

https://bus-stop-bonanza.herokuapp.com

Development Information

The code was written predominantly in Python on the Django framework. Page templates were written in HTML 5 and any dynamic scripts such as the location request were written in the requested Typescript. The bulk of the logic within the app is preformed in the views.py within the app folder. The HTML can be seen in the app/templates folder and the Typescript in the /static folder. It should be noted also that there is a folder of unit tests in app/tests because we are strong advocates of test driven development. Much of the visual elements of the page were created using bootstrap which allows for dynamic rescaling of the webpage, giving us a consistent cross platform experience.

Main Functionality

Feed Page

The feed page uses the live bus locations and compares the distance each respective bus to the user's location. With this information, it then displays a grid of each bus along with their respective upcoming stops with times and destination. These can be easily filtered out if the user knows the exact service they require using the service numbers at the top.

Once the valid bus has been determined, the user clicks on the bus they require and are taken to the tracker page for that bus.

Tracker Page

Due to limitations with the bus API, each individual bus on the live tracker isn't tied to it's own journey. Pairing each individual bus to it's own journey proved to be one of the most challenging elements of this project. While each bus on the live tracker, does have a unique ID, it is for use with the "MyBusTracker" API, a different API, which only got back to us with an API key a day before the deadline. We then had to work out which respective bus was which based on location, time and stops. "The Algorithm" (we will explain the name below) was derived by taking the service number and listing it's journeys, finding the next stop time for every active journey and finding the closest one to the user. This was a very efficient way of doing it (in terms of storing and calculating data) and was far better than what we like to call the "Ayrton Massey" where a system stores 700MB of data in ram using some ridiculous algorithm and has horrible load times. We felt sorry for this unnamed individual and his group that implemented this, so we let them know what we had done, so they best be giving creds to Chekov.

Once the correct journey has been figured out, we can then display and read out aloud the next stops. The typescript automatically rechecks the user's location after a short delay and re-reads out the next stop, updating it based on the time.

Functional Requirements and Deficiencies

Functional Requirements

1. Front end must use typescript
   Done. So much typescript
2. Can't be same as bus app
   It isn't the same.
3. Must be mainly yellow
   See 7.
4. Cross platform
   Works on mobile and desktop!
5. Big picture of Edinburgh
   See 9.
6. No clichés in your app
   Does anyone else offer a bonanza with bus stops?
7. Not mainly yellow
   Red and white. Like strawberries and cream.
8. Resist temptation to add lots of features
   We went for the whole "less is more" approach.
9. You can ignore any one requirement
   We ignored number 5.

Deficiencies

Aside from the aforementioned troubles in lacking data for the API, we currently have a problem which is viewable on our feed page. The API provides duplicates of the same bus. This isn't a huge problem functionality wise, but would be something to be considered clearing up.

Another problem with the API is how it handles different times. If a bus starts before 24:00, the time simply continues upwards (e.g. 24:35). This is fine, but a lot of night buses start their times at 0:00. This is horribly inconvenient for "The Algorithm" as it disregards all times which have apparently already happened. If it's looking for 24:35 for example, it will ignore the 00:00. This is partially how Python deals with Daylight savings time. There is no easy way of properly implementing it other than adding 1 hour to the current time (which puts us on 25:00 before flipping to 01:01). This therefore means that the early bus morning buses which start after midnight won't appear on the list. Hopefully people will be too drunk that early in the morning to mind where they get off the bus.

A final comment on "The Algorithm" - which is basically perfect in every way other than this - is that it's completely reliant on buses being exactly on time. This works in theory, but in practice we know buses aren't 100% reliable time wise.

Team Contributions

Sam- Set up Django app and Github Repository. Started off the project by implementing the feed page. Set up bootstrap on both the feed and tracker. Also linked all the pages together. Did all the Typescript and tests as well as documentation. Refactored some of mingles' sloppy code, then did everything else which hasn't been mentioned.

Michael - Wrote "The Algorithm" while acting out this scene from the social network with Sam. To better understand what it was actually like, replace the names said with "mingles" and "sam" and replace the word "girls" with "buses". Then implemented "The Algorithm", linked it in with the text to speech API and presented all to the tracker page. Attempted location tracking using Typescript, but didn't really like it. Apparently wrote most of this report too.

Cammy - Wrote some code which gave us the start and stop locations and times of each bus on its journey, also helped with ideas.

Chelsea - Despite lack of technical experience, help greatly with ideas for the team and worked with Michael on the report.

Rikki - Integrated the leaflet map onto the tracker page which give live locations of the bus and the stops on a map.