Bibliography

Source	Use
https://opendata.transport.nsw.gov.au/documentation	Used to figure out how to use the transport APIs and how to read their data.
https://developers.google.com/maps/web/	Instrumental for helping me understand the Google maps API and how to add features like styling and makers.
https://stackoverflow.com/questions/11106671/google-maps-api-multiple-markers-with-info windows	The top answer from this stackoverflow question was appropriated by me to attribute different data to different Google maps markers.
https://stackoverflow.com/questions/3451111/unzipping-files-in-python	For the update tool I needed to unzip many ZIP files; I learnt how to unzip files in Python from the top answer in this post.
https://www.w3schools.com/nodejs/	I had never used Node before this point and I discovered that I would need it to call the APIs and to read text files as plain JS cannot do these things. These tutorials taught me how to use node.
https://stackoverflow.com/questions/9168737/read-a-text-file-using-node-js	The top answer from this post helped shape my file reading function.
https://snazzymaps.com/	Is where I got my copyright free map styles for my application.
https://opendata.transport.nsw.gov.au/status	Is where my application checks to ensure all APIs are online.
https://transportnsw.info/about-us/developers-site-owners	Is where I got the circular vehicle icons for the favicon.
http://data.livetraffic.com/cameras/traffic-cam.json	Is where I get the traffic camera images from.

Logbook

16 December 2016

Today I worked on getting a mostly finished UI done. Because this is an agile project, it doesn't make sense to work on a basic UI, finish the backend then redo the front end. If I mostly finish a front end, it will make the whole project go much faster. This involved creating an HTML and CSS file. The HTML file had buttons that needed to change panels on the page. To do this, I created a small bit of JS that would change panels depending on which button I clicked. This JS file was then linked in the HTML file so it knew to read it. The same was done with the CSS.

16 December 2016 (Evening)

Found a bug where if you clicked on a button then brought the mouse cursor away, the depressed button shadow would be stuck with no way to remove it without refreshing the page. This was a simple fix, but an ugly one. It involved creating an event for each button, that when the cursor was removed, the shadow would too.

17 December 2016

With the UI more or less done for the time being, I decided to implement the biggest element of this project, the map. To do this I researched into different map providers and found that Google had the most polished one with the best documentation and features. I set up an account and got an API key. I then used the code from their website (see reference) and put it into my HTML. Finally, I replaced the APIKEY placeholder in the code with my own and the map finally appeared on the page but I ran into a problem. By default the map takes up a small subset of the page and is positioned in the top left corner. This was dealt with swiftly with a little CSS.

18 December 2016

It was now time to start working on the backend, I wanted to start with showing the vehicles. To do this, I first looked up the tutorial on how to add markers to a Google map in the Google map API documentation. With this newfound knowledge, I set to work on getting information that my markers can display. To do this I needed to access the open data transport API and to do that, I needed an account. After creating one, I needed to create a project in my account's dashboard and specify which APIs I would use. After setting all that up and receiving an API key I read the documentation (see reference) to decipher how I was to get the vehicle positions in real time. All I needed to do was send a HTTP GET request from my application with my API key. I figured my client and server would be communicating often, with different sets of data, so I established

a function in my client that called and send data to the server and a function in my server that would pull data from an API and send it back to the client (see reference). The code seemed to be valid so I went to try my new communication protocol to find it didn't work. I figured out that I needed to install node and to run my server code with node via the command line and it worked!

19 December 2016

Now that I had all this protocol in place, I set to work on getting it to pull real data, the real time vehicle position data. First I called the server with my protocol and gave it the URL for the vehicle positions, I then asked it to print out it's result. I came to find out that it worked and the data I was getting was consistent with the APIs documentation. Great! I then tacked on a loop that took each vehicle from the response, pulled its latitude and longitude (see reference) and gave it to the Google maps API to create a marker. I reloaded the HTML page in Chrome and saw all the buses in Sydney. After changing the markers from Google maps pins to an arrow, I was much happier. Finally, as it was right there, I pulled the bearing of each bus and set the rotation of the marker to that bearing so you could see which way the buses were pointing.

20 December 2016

Right, time to get the shapes and stops of a certain vehicle, easy right? Just call a different API URL? Wrong. I've reread this documentation 3 times now and I've tried many combinations of URLs and decoding methods to no avail. I even tried to convert byte code to ascii to find it was gibberish. The response was encrypted in some way and no where in the documentation did it say how.

4 January 2017

Tried to give a crack at decrypting it. Tried a few different encryption methods but after 20 minutes, I gave up again. I'm dreading having to revisit it.

19 January 2017

Christmas came and went as did a camp. I didn't want to work on my project until I knew I could sit down for a few hours and figure this API out. Well today was the day. After rereading the documentation again and again and finding nothing, I turned to the forums that pointed out the fact that the gibberish I was receiving was actually a ZIP file. After manually crafting a URL with my API key and the API URL, I entered it into chrome to find a 80MB ZIP file downloading. When it finished downloading I was eager to see what was inside. It was a relief to see that the contents matched what the documentation said would be inside (not that I've memorized it at this point). I remembered reading that the data only updated daily and I figured I'd only use ⅓ of the data and there were too many files that could be put into one and it would be tough to unzip

and process large text files with a web based language so I turned to Python. I created a script that would download 5 ZIPs, one for each vehicle type, wait for the download to finish and then unzip them all (see reference) into the python project's directory. I was eager to continue but had done enough already.

20 January 2017

I sat down at a whiteboard with all 9 text files found inside a ZIP open deciphering what data I would need and what data I wouldn't. I had crafted a format that would only need 3 text files per vehicle type. Now that I knew which files I could delete and which columns of the remaining files I could delete, I added this trimming process to the python update tool. I then looped over each vehicle, and reduced each stops.txt file by condensing the information from one stop per line per shape to one line per shape. From DH101,0,20006 \n DH101,1,20414 \n DH101,2,20601 to: DH101,20006,20414,20601. This file is then sorted in alphabetical order for faster searching in the next step: generating the info file. This info file is going to be the heart of all the data. It will have the trip ID, the shape ID, the stop sequence, whether or not it was wheelchair compatible whether it ran on Tuesdays etc… The info file was generated by a script I made that went into each text file and got the relevant data that it needed, conglomerating all the data into one text file.

22 January 2017

I then continued this update tool. I created a script that did the same thing as the stops.txt file but with the shapes.txt file. Expand the latitudes and longitudes horizontally rather than vertically. This reduced the lines ~100 times and the file size by ~30%. I then wrote a script that reduced the stop locations x and y coordinates to 6 decimal places to save some space. Then came a script that would pull all the currently available traffic cameras' URL and location/bearing information. I decided to add this is the update tool since cameras aren't added too often and it makes sense to check only daily for new cameras. Finally came a script that removed used text files and original ZIPs. The conglomerated CSV's were then moved to the client's scripts folder to be accessed later. The tool was done, or so I hoped… The data formats for the CSVs change often and so the update tool was extremely temperamental. To resolve this issue I included thorough error reporting.

23 January 2017

I wrote a loop that would update vehicle positions every 20 seconds. If a vehicle is no longer identified by the API, it is removed from the map and if a new one is identified by the API, a new marker is made for it. I wrote a bit of debug code that when clicked on a vehicle, it's trip ID was printed. I set this up so that I could implement the next bit, getting that vehicles data.

26 January 2017

Happy Australia Day! I set up a function in my server that would find a line in a text file (see reference). This function was used to find the relevant data for the trip ID of the vehicle that I clicked on. This information is then sent back to the client. All the information besides stops and shape is processed and displayed on the side panel of the application so the user can see everything about that vehicle. While the user is looking at that, I wrote a function that would process the stops, and create Google map pins at each point. When a pin is clicked on, it calls another function I wrote that opens streetview for that point in a new tab. After the stops are loaded, all the points of the shape are loaded and sent to the Google maps API to draw a line.

10 February 2017

Coming back to look at this project, I wished that the vehicles would glide from their last position to their new one, that they would interpolate between position updates. This was a simple function that took the old and new position, divided it into 10 steps and applied each step's position slightly apart from each other to give the illusion of movement. Great! Works great, looks great. But… interpolating buses slowed down the program too much for my liking, so I disabled interpolation for buses only as buses always has the most vehicles.

2 March 2017

My vehicles weren't showing and I hadn't changed the code. After an hour of debugging I discovered that the real time vehicle position API was down for maintenance. Seeing the bright side of the situation I decided to add a panel to my program that would show which APIs that I was using were down (see reference). This involved scraping the web page on the transport APIs statuses and seeing which APIs were up and which were down. This is shown in a list in the API status panel. If an API is down, the button is bright red and the user is alerted that an API might be down.

3 March 2017

Didn't have much time today so I made a quick fun little function that would randomly pick a tab icon (see reference) each time the page was reloaded.

17 April 2017

I made a new panel today where the user could pick a map style to apply. This was easy as all I had to do was pass the Google maps API some JSON for the styling of the map. I found some styles online (see reference) and added them as presets. I also added the option to clear the

styling and have the map go back to its original style and just for fun, I made the map pick a random style each reload. Also in this customization panel was the option to change markers scale, opacity and color. This would loop through each affected marker and change it's styling. Again, for fun, I added a randomize scale/opacity/color button/function and a reset button/function.

29 April 2017

I had forgotten something and it was eating me away to figure out what it was. Well today I remembered. I had forgotten to implement the traffic cameras on the map. I made quick work of that however. I just read the file I created in the update tool about the cameras, interpreted the information and added markers to the map with this information. Then came a bit of code to show the image (see reference) for the camera that was clicked on. Viola. Done.

30 April 2017

On showing my dad the program, he asked if there was anyway to see which vehicle you clicked on after you've scrolled out to see its route, so I added a bit of code that shows a button after you've panned the map that, when clicked, will zoom and pan you back to your selected vehicle.

17 May 2017

The program was working well. A little too well. Well I put my suspicions to rest as today I tested all my code on different systems and what not. I found some minor bugs and patched them up quickly. Pretty proud of how stable my code is.

18 May 2017

When I fixed some bugs in my code yesterday I noticed it was a little messy so I further modularize my code, added some more comments to clarify some areas and adjusted some variable names just to be a little clearer.

19 May 2017

With my code finished, I made my user manual and installation guide today as well as writing an evaluation.