**INFO263**

**Web Design and Development**

**Auckland Transport API Project**

Riley Aitken (rai29)

Bailey Horne (bho55)

Sam Green (sgr97)

Alex Johnston (ajo100)

**How the Website Works Under the Hood**

**The Google Maps API**

Importing a Google Maps widget was as simple as communicating with the Google API service (with a valid API key), and initialising a new Map object. We were able to specify the default location for the map by specifying the latitude and longitude values associated with ‘position’. When users interact with the routes and the vehicle positions were retrieved, adding markers on the map was done by initialising ‘Marker’ objects with the specific latitude/longitude values, which are supported by the Google Map API.

Users can get vehicle information by clicking on the markers that exist on the map; an ‘InfoWindow’ is a feature of the Google Maps API, whereby ‘content’ is associated about the specific vehicle the marker corresponds to. We added a listener to check for a click event on the marker, and when this occurs, this ‘InfoWindow’ is displayed above the marker’s position.

The map auto-resize functionality was implemented by looping over all the markers array containing all markers on display, and extending the bounds due to the positions of each of these markers. The resulting map is bounded such that all markers just fit within the screen without having to manually zoom in/out.

**Client-Side Functionality**

We added a ‘select’ element to our webpage, which displays all the available routes on the screen. It retrieves all the available routes by calling our ‘routes.php’ module, which deals with pulling all the route names from the database (explained further below) and returning them to the JavaScript code. When users click on one of the routes, this selection is realised by JavaScript and is passed back to our ‘db.php’ module (by way of a JQuery ‘post’ method call) for server-side processing. The result of this server-side processing is returned to JavaScript as an array of vehicle positions and other relative information; this bus information is then used to update the Map.

The map interactivity also fits under this umbrella of client-side functionality, as no server communication is required when the user interacts with the map in any way.

**Server-Side Functionality**

The first stages of server-side processing are undergone as soon as the user loads the webpage. JavaScript requests a list of all the route (short) names, which is retrieved from the database.

To establish this connection to the database, a ‘mysqli’ connection object is instantiated, with the correct username, password, hostname and database values passed. The ‘query’ method of this object is called, along with the corresponding query to select distinct routes from the database. The result rows of this query are then ‘printed’ back to JavaScript as an array.

The other instance of server-side processing required is when a route is selected on screen and the vehicles (if any) must be retrieved on the server-side. The selected route is realised in the ‘db.php’ module by use of a ‘$\_POST’ call that is looking for a value associated with ‘route’. This route value is then included in a following query to retrieve the ‘trip\_ids’ associated with that route. As this query is dependent on ‘route’, which is user input, this query must be sanitised to reduce the threat of SQL database injection. To further secure this querying process, we also used prepared statements. The process of executing this query consisted of preparing a statement (the query) on the ‘mysqli’ object, using ‘bind\_param’ to associate a variable with the query (which checks that the type of value passed is valid for the query) and then executing the statement on the connection object. The results were then retrieved by calling ‘fetch’ on the object.

These trip\_ids are then passed to the ‘realtime/vehiclepositions’ API service to retrieve vehicle information about the buses on these trips, and consequently, the route the user selected on the webpage.

Every 30 seconds, the positions of the vehicles on the map update. This was achieved by repeating the steps outlined above for obtaining the trip\_ids and getting the positions for each vehicle currently on the map.

**Collaboration**

**Team Communication**

We met consistently throughout the assignment period, and were able to discuss design/implementations with one another often, so each of us have a good understanding of how we came to an end result. Committing our time on Wednesday/Friday afternoons to putting in some work on the project was an effective development schedule as we were all present to progress collectively.

**How We Managed Code Collaboration**

We had our troubles with github; the push/pull method was largely unsuccessful because of commit/merge conflicts confusing the process. Because of this, we would often upload our separate files and merge them manually, which is time consuming enough as it is, even for a small-scale project.

**Separation of Tasks**

Because much of the modules are highly coupled (reliant on each other to work) it was difficult managing and assigning tasks to one another for the most part. There were a few tasks that could be worked on without the prerequisite of other work being completed such as the Google Maps API, querying and CSS. We did not plan out the assignment of tasks early on, however, and as such our development process could have been more efficient. Separation of concerns/languages (separating PHP, HTML, etc) would have aided in making development a more parallel process instead of a sequential one also.

**Reflection**

**What Went Well**

Our team communication was on the whole effective, as we were able to work collaboratively to solve problems due to our dedicated development sessions. We never were stuck on one aspect/requirement for too long as collectively our problem solving skills were effective. As such we all have a good understanding of how the various modules work together to provide a functioning end result.

**What Could Have Gone Better**

We could have more thoroughly planned out the development process (i.e assigning specific tasks and giving deadlines) from the beginning so that the process would have been more parallel as opposed to sequential. We also didn’t put much thought into overall program design (separation of concerns) initially which made the debugging process a mission when something was going wrong.