Alexander Lew
CS50 Final Project: T Tracker

Design
This document is to walk through many of the design decisions while writing the site and its code. The MBTA real time data feed is not perfect, and still has many data errors including misclassification of routes, predictions in the wrong order, and weird station nomenclature. Ways to mitigate such data issues is addressed in this document.

Homepage and Database
The homepage `index.php` gives several options for the user to choose from. Let's start with the **predictions by stop**, the core part of this site. An SQL database of the stops was created. The database includes the name of the stop, the line, and the stop code, a 3-5 letter station code (e.g. bbsta is the Back Bay station) that the MBTA uses for data requests. The database is sorted in alphabetical order. For stations with two lines, there is one entry for each line despite having the same station code. The reason for choosing this is to have stations listed by line in an easy manner.

For each of the lines, a SQL query (using `CS50::query` in PHP) searches for all the stations by each line, and then outputs a form from which the user can select a station. Upon selecting the line, the station code is passed as `$GET` to the line's prediction page. Because `$GET` is passed through the URL, this allows the user to bookmark stations.

Predictions by Stop
For each line, there is a separate page: `blue.php`, `green.php, orange.php`, and `red.php`. The reason for separating out the lines is to style the page is to better cater the style and content to the line, including the color and the direction labels.

Using SQL Query again, PHP determines what the station code refers to the full name of the station. This station is printed at the top. Some stylistic HTML arranges the text into grey boxes and white text. A script marks the time of last refresh.

A javascript function is called to predict the arrivals. This function predict is defined in `scripts.js` and is discussed below. In order to obtain the `$GET` variable is passed to a javascript variable, which is then used in the predict function. HTML is written via this function and put into the `<p id=predictiontext0>` and `<p id=predictiontext1>` .

Predictions by Line Function
In the scripts.js file, the function `predict(Stop_ID, Line, Direction)`takes in the 3-5 letter stop code, line, and direction, queries the MBTA real time predictions, and then outputs HTML that is then returned to the prediction page.

The JSON response of the MBTA real time information feed is a nested array whereby each array has several sub-arrays, which also include additional sub-arrays within these sub-arrays. The JSON response contains a lot of extraneous information including *all modes* (bus, commuter rail) in addition to the subway that operate in a nearby vicinity.

Because of the way the data is returned, it is necessary to several nested loops in order to obtain the information I actually needed: (1) the number of minutes before the train arrives

converted from seconds, (2) the destination of the train, and (3) unique trip id number (more on this later).

As I developed the loops, I noticed two problems with the JSON response from the MBTA:

- The data is not sorted by arrival time at the requested station, rather it is by the time the train departed the terminal station (e.g. Braintree/Ashmont or Alewife for the Red Line). Because the Braintree branch is much longer than the Ashmont branch, an earlier departure from Braintree may arrive later than a later departure from Ashmont. Thus, the data was out of order. This required building a multi-dimensional array in which to put data, and then sort it after all data is inside it.
- In addition, the MBTA misfiled some lines in a different mode array. Despite being a subway, the Park Street Red Line and Green Line were stored as two separate modes. At Downtown Crossing, which also hosts two lines, both lines are filed in the same mode. Weird, right? This required a loop through the modes array, and was added after realizing the problem. Thus, you may notice that I start my first loop with variable with variable h instead of i.

After looping through and storing the data into an array, I read up on sort functions on Stackoverflow to sort the aforementioned array in order by which the train arrives.

Finally, I built a string that formatted the data stored in the previous array. This HTML was then sent back to the tags with `predictiontext0` or `predictiontext1` depending on requested direction by using the `documentgetElementByID().innerHTML`.

A number of predicted failure events, such as no data, no more trips from that particular line/mode/direction, or failure to get the JSON request all will return back "No Prediction."

Predictions by trip
The previous function received the trip id number from the JSON response, imbedding it into HTML, and producing a link to `trip.php?trip_id=(some number).` Each trip in the MBTA system is given a unique trip id. Using this id number, a subsequent request can find the predictions by trip, which shows what time an individual train will arrive at subsequent stops on the route.

Like the last function, this requires looping through a JSON request, however there is less number of arrays and sub-arrays as there is only one trip returned for any given trip_id. Similarly, a string was created by concatenating HTML and the needed information, and then passed via `documentgetElementByID().innerHTML` as well.

Map
The live map shows the location of all vehicles, using the Google API as in PSET 8. Although there is some overlap with the code from PSET 8, a further consideration to presentation and design was made in this use of the Google Maps API. `Mapscripts.js` is where the scripts are stored for the map.

Let's first start with getting the actual vehicles onto the map. The function `addtrain(Line)` will add markers for all trains of the requested subway line. The MBTA real time information

feed has an option to find vehicles by route, which returns the latitude, longitude and vehicle bearing of all trains for a certain route (along with many other information items such as the vehicle's stop sequence). Because vehicles are stored by each individual route, it requires using this function multiple times to get all of system-wide vehicles.

Like with the other predictions, several nested loops were required to find the needed information. Latitude and longitude will be subsequently be used for the location of the marker. Bearing, which is the compass direction that the train is facing, is going to affect the marker's graphic design.

For each marker, the marker must be uniquely tailored to both the line and the bearing. I custom designed a symbol, shaped like an arrow, but rectangular like a vehicle. Google Maps API can take symbols from svg format, which is an export option for Adobe Illustrator. The bearing would affect the rotation of the symbol so that vehicles are "follow" the alignment of the track. The requested route/line affects the color. This uses the official MBTA colors.

The marker is then placed on the map, and a global array stores the marker.

The function `configure()` is what calls the `addTrain` function and configures the map as a whole. In order to have the location of the trains updated every 10 seconds, the function will first add all lines to the map, then using the `setInterval` function, will first remove the markers and then call the `addTrain` function again to get the new location of the trains 10 seconds later. Thus, it is not "moving" the markers per se, but instead removing the markers, receiving the new location, and then placing the markers again.

Several graphic design considerations were also considered. First, the transit layer was turned on. Second, all other layers were de-saturized and given a slight hue in order to fade out the background. This allows the transit map and the vehicles to have more emphasis on the page, while fading less important geographic layers.

References:

MBTA Real Time API Documentation:
http://realtime.mbta.com/Portal/Content/Documents/MBTA-realtime_APIDocumentation_v2_0_1_2014-09-08.pdf

Sort Functions:
http://stackoverflow.com/questions/3423394/algorithm-of-javascript-sort-function