









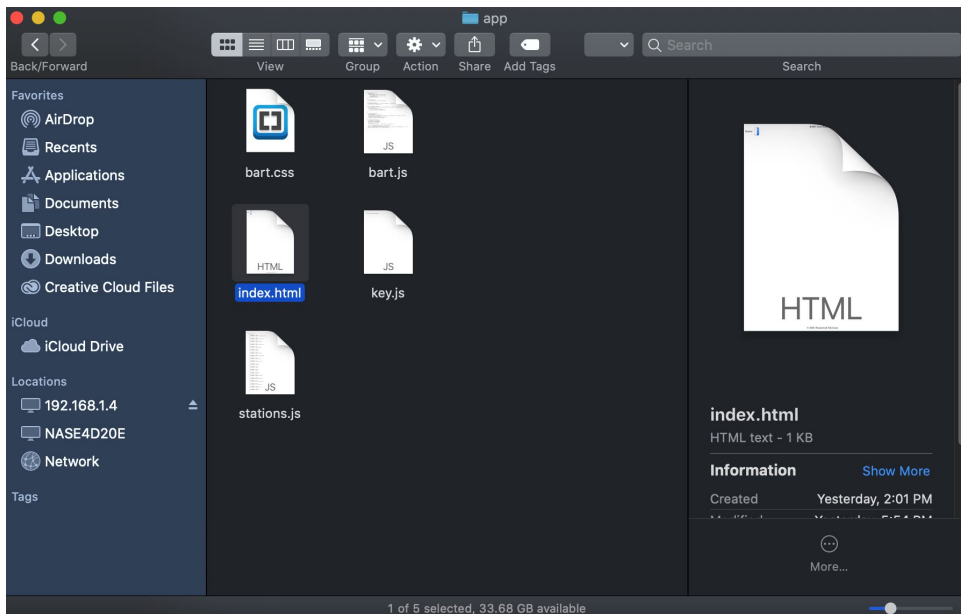
BUILD A BART TRANSIT APP WITH JAVASCRIPT & JQUERY

BART Train Departures	
Station:	
16th St. Mission (SF) ▼	
Antioch	
	Time: 1 minutes away. Platform: 2 Length: 10
	Time: 16 minutes away. Platform: 2 Length: 10
	Time: 31 minutes away. Platform: 2 Length: 10
Daly City	
	Time: Leaving minutes away. Platform: 1 Length: 10
	Time: 9 minutes away. Platform: 1 Length: 9
	Time: 10 minutes away.
© 2020 Framework Television	

Framework Project Guide

Welcome. This guide is for the BART Transit App project. This application will show trains arriving at any BART station in real time.

This web-based application runs on any contemporary web browser and uses the jQuery Mobile library so that it is easily adaptable to a mobile application environment.



The Code

The code for this application is packaged with this guide. There are five files that come with the application code.

stations.js: This file contains the data about stations in the Bart system in JSON format

bart.css: Most of the styling for this app is done using jQuery Mobile, however, this file contains a few custom styles that were used.

key.js: This file contains the API key used for the application. (Get your own from <https://www.bart.gov/schedules/developers/api>)

bart.js: This file contains the core JavaScript functions that drive the application.

index.html: The application scaffolding and links to libraries.










```
BART — Python -m SimpleHTTPServer 8080 — 101x33

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208850.
[Marks-MBP:~ marklassoff$ cd Desktop/BART/
[Marks-MBP:BART marklassoff$ python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
127.0.0.1 - - [14/Feb/2020 10:48:11] "code 404, message File not found
127.0.0.1 - - [14/Feb/2020 10:48:11] "GET /moment.js HTTP/1.1" 404 -
127.0.0.1 - - [14/Feb/2020 10:48:11] "code 404, message File not found
127.0.0.1 - - [14/Feb/2020 10:48:11] "GET /moment-timezone.js HTTP/1.1" 404 -
127.0.0.1 - - [14/Feb/2020 10:48:11] "code 404, message File not found
127.0.0.1 - - [14/Feb/2020 10:48:11] "GET /moment.js HTTP/1.1" 404 -
127.0.0.1 - - [14/Feb/2020 10:48:11] "code 404, message File not found
127.0.0.1 - - [14/Feb/2020 10:48:11] "GET /moment-timezone.js HTTP/1.1" 404 -
127.0.0.1 - - [14/Feb/2020 10:48:11] "code 404, message File not found
127.0.0.1 - - [14/Feb/2020 10:48:11] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [14/Feb/2020 10:48:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:48:23] "GET /bart.css HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:48:23] "GET /key.js HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:48:23] "GET /bart.js HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:49:13] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:49:35] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:49:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:50:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:50:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:50:12] "GET /bart.css HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:50:12] "GET /key.js HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:50:12] "GET /stations.js HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:50:12] "GET /bart.js HTTP/1.1" 200 -
127.0.0.1 - - [14/Feb/2020 10:50:13] "code 404, message File not found
127.0.0.1 - - [14/Feb/2020 10:50:13] "GET /favicon.ico HTTP/1.1" 404 -
```

Running the Application

It is advisable to run this application in a web server. The easiest way to do this is on your command line. The Mac has a built in HTTP Server available.

- 1) Navigate to the folder that contains your code using the command line. (Remember **cd** changes the director. **cd myApp** will open a directory in your current directory called **myApp**).
- 2) Start the server with the following command:
`python -m SimpleHTTPServer 8080`

BART Train Departures	
Station:	Embarcadero (SF) ▼
Antioch	
	Time: 2 minutes away. Platform: 2 Length: 10
	Time: 18 minutes away. Platform: 2 Length: 10
	Time: 32 minutes away. Platform: 2 Length: 10
Daly City	
	Time: Leaving minutes away. Platform: 1 Length: 9
	Time: 9 minutes away. Platform: 1 Length: 9
	Time: 11 minutes away. Platform: 1 Length: 10
Dublin/Pleasanton	
	Time: 14 minutes away. Platform: 2 Length: 10
	Time: 29 minutes away. Platform: 2 Length: 10
	Time: 44 minutes away. Platform: 2 Length: 9
Millbrae	
	Time: 4 minutes away. Platform: 1 Length: 9
© 2020 Framework Television	

Running the Application

- 3) Open your web browser and type the following address in the address bar:

localhost:8080

Your application should load into the browser at this point.

Once you've got the application running, test it by creating and completing a few goals. To complete a goal, click on it and respond to the prompt.

The process for windows machines is similar by you may have to install [Python and SimpleHTTPServer first](#).

```
<!DOCTYPE html>
<html>
  <head>
    <title>BART APP</title>
  </head>
  <body>
  </body>
</html>
```

Building the App: Index.html

Like all web-based apps (or mobile apps based on the web languages HTML, CSS and JavaScript) we start with a basic document structure.

This structure underlies every web site and many mobile apps that you use daily.

```
<head>
  <title>BART</title>
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
  <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="bart.css" type="text/css" rel="stylesheet" />
  <script src="key.js"></script>
  <script src="stations.js"></script>
</head>
```

Next we'll add the necessary content to the document head element. The **meta** tag correctly sizes the viewport for the screen the app is being displayed on. It also ensures that magnification is at 100% with "initial-scale=1".

The balance of the code links to JavaScript code or style sheets. You'll notice that after the link to the Goal.js document there are three lines linking to external document. These three lines of code are loading the jQuery Mobile libraries. These were taken directly from the [jQuery Mobile documentation](#).

The next **link** tag in the head of the document loads the little bit of custom CSS that the application uses to anchor the footer displayed in the interface and create the colored boxes indicating the train lines. Links to two additional JavaScript documents complete the head. **key.js** contains the key used to unlock the BART API. **stations.js** contains the necessary JSON data about each station in the BART system.

```
<div data-role="page">
  <div data-role="header">
    <h1>Page Title</h1>
  </div>
  <!-- /header -->
  <div role="main" class="ui-content">
    <p>Page content goes here.</p>
  </div>
  <!-- /content -->
  <div data-role="footer">
    <h4>Page Footer</h4>
  </div>
  <!-- /footer -->
</div>
<!-- /page -->
```

A jQuery Mobile Page template actually has three distinct areas. The header section the main section and the footer section.


```
<div data-role="page">  
<div data-role="header" id="header">BART Train  
Departures</div>
```

In the header for our document we only placed the name of the application in a heading one (h1) tag.

```
<div role="main" class="ui-content">
  <form>
    <div class="ui-field-contain">
      <label for="select-native-2">Station:</label>
      <select name="select-native-2"
id="select-native-2" ></select>
    </div>
  </form>
  <ul data-role="list-view" id="trains"></ul>
</div>
```

Most of what is statically displayed on screen is in the **main** section.

There are two primary UI components that are used in the BART app. The first is a drop down created with the **select** tag. It has a label associated with it with the word "Station". You'll notice there is no data (created with **option** tags) in the **select** element. The data (station names) will be created dynamically by JavaScript.

The second item in the UI is an **unordered list** (ul) element. This is where the **listview** containing the data about each train will be placed.

```
<div data-role="footer" id="footer">&copy; 2020 |  
Framework Television</div>  
<script src="bart.js"></script>  
</div> <!-- closes page -->
```

The HTML file is completed by the footer section from the jQuery Mobile page template and a link to a final script which is the core JavaScript for the page.

By convention, this link is at the bottom of the page to ensure that all the HTML components are loaded before the JavaScript executes.

```
#header, #footer{
    text-align: center;
    height: 30px;
    width: 100%;
}
#footer{
    bottom: 0;
    position: fixed;
}
#trains{
    margin-top: 30px;
}
.trainBox{
    width: 35px;
    height: 35px;
}
.trainInfo{
    margin-left: 10px;
}
```

Building the App: bart.css

The bart.css file is short because most of the styling is done using jQuery Mobile's default styles.

The style here is added because (for reasons no one understands) the footer in the jQuery page is not anchored to the bottom of the page and instead floats by default.

Additionally, CSS in this file styles the colored boxes in the app referencing the individual train lines.

```
const stations = [{
  "stationAbbr": "12th",
  "stationName": "12th St. Oakland City Center"
},
{
  "stationAbbr": "16th",
  "stationName": "16th St. Mission (SF)"
},
{
  "stationAbbr": "19th",
  "stationName": "19th St. Oakland"
},
{
  "stationAbbr": "24th",
  "stationName": "24th St. Mission (SF)"
},
{
  "stationAbbr": "ashb",
  "stationName": "Ashby (Berkley)"
},
{
  "stationAbbr": "antc",
  "stationName": "Antioch"
},
{
  "stationAbbr": "balb",
  "stationName": "Balboa Park (SF)"
},
}
```

Building the App: stations.js

This file is loaded in the **head** section of the index.html document.

The data for each station is stored in a constant named **stations** in this file. Each station has an abbreviation which is used as an internal identifier by the API . The full name of each station is also stored in the JSON. (Partial file displayed).

```
$(document).ready(function () {
    loadStations();
    $("#select-native-2").on("change", function (event)
    {
        //console.log(event);
        let stationAbbr = event.currentTarget.value;
        $("#trains").html("");
        callBartAPI(stationAbbr);
    });
});
```

Building the App: bart.js

This file contains the primary JavaScript that drives the application.

The initial function runs when the document ready event occurs— after all components and libraries have been loaded by the browser.

The function then calls the **loadStations()** function which loads the stations into the dropdown in the UI. Next, the change event is associated with the the drop down so that when the value of the drop down changes, the station abbreviation is sent to a function called **callBartAPI()**.

```
function loadStations() {
    stations.forEach(station => buildSelect(station));
}

function buildSelect(station) {
    let option = "<option value = '" +
station.stationAbbr + "'>" + station.stationName +
"</option>";
    $("#select-native-2").append(option);
}
```

The **loadStations()** function is called during initialization. This function starts the process of populating the drop down (**select** tag) with the station names.

The **loadStations()** function iterates through all the stations stored in the JSON referenced by the **stations** constant. As it loops through it sends each station entity to the **buildSelect()** function which builds the **select** element.

The **buildSelect()** function builds the **option** tag for insertion into the **select** element. The **option** tag uses the station abbreviation as its value and the station name as its content. The **option** tag is then appended to the **select** element.

```
function callBartAPI(stationAbbr) {  
    let url =  
    "http://api.bart.gov/api/etd.aspx?cmd=etd&orig=" +  
    stationAbbr + "&json=y&key=" + key;  
    console.log(url);  
    $.get(url, function (data) {  
        let trains = data.root.station[0].etd;  
        trains.forEach(train =>  
            displayTrain(train));  
    });  
}
```

The **callBartAPI()** function is called next. This function calls the BART API and provides the necessary parameters to obtain train schedule data. The function first creates the URL and parameters, including the station abbreviation and then contacts the URL with the jQuery **\$.get** method. The data is returned in JSON format and the individual destinations (with the departures embedded in them) are stored in a variable called trains.

For each train element **displayTrain()** is called with the train passed to it.

The screenshot shows a web browser window with the URL `localhost:8080/app/`. The page displays the "BART Train Departures" app. A dropdown menu for "Station:" has "Embarcadero (SF)" selected. Below it, the "Antioch" section is expanded, showing two train entries: one with a yellow background and another with a light blue background. The Chrome DevTools Console is open, showing the network log for the API call `http://api.bart.gov/api/etd.aspx?cmd=etd&orig=embarcadero&json=y&key=ZPRSS-5K6K-9R7T-DWE9`. The console displays a JSON array of train objects. The last object, for "Warm Springs", is expanded, showing its nested properties: `destination`, `abbreviation`, `limited`, and `estimate` (an array of 3 objects). The first object in the `estimate` array is shown with properties: `minutes` (14), `platform` (2), `direction` (North), `length` (10), `color` (GREEN), and `hexcolor` (#339933).

Using the “Developer Tools” Console tab in Chrome you can actually see the individual train objects returned from the BART API.

In this case with the Embarcadero station selected, there are multiple destinations with multiple departures.

```
function displayTrain(train) {  
    console.log(train);  
    const destination = train.destination;  
    $("#trains").append("<li  
data-role='list-divider' style='font-size: 1.5em;'>" +  
destination + "</li>");  
    const arrivals = train.estimate;  
    arrivals.forEach(arrival =>  
displayArrival(arrival));  
    $('#trains').listview().listview('refresh');  
}
```

Each individual train (destination) is passed to **displayTrain()**. The **destination** mode is extracted from the JSON. That information is used to create a list-divider according to the jQuery **listview** styling instructions.

The dividers contain the names of the various destinations available from an individual station.

For each destination, the individual departures are then looped through and passed to the **displayArrival()** function which extracts the information about each individual train run.

When this function completes all data will be displayed, so the listview is refreshed to apply jQuery styling.

```
function displayArrival(arrival) {
  const minutes = arrival.minutes;
  const platform = arrival.platform;
  const length = arrival.length;
  const color = arrival.hexcolor;
  let out = "";
  out += "<strong>Time:</strong> " +
arrival.minutes + " minutes away.";

  out += "<br/><strong>Platform:</strong> " +
arrival.platform;
  out += " | <strong>Length:</strong> " +
arrival.length;
  $("#trains").append("<li><div style='display:
flex;'><div class='trainBox' style='background-color:
" + color + "'></div><div class='trainInfo'>" + out +
"</div></div></li>");
}
```

The **displayArrival()** function extracts the information about each individual train out of the JSON returned by the BART API. The information about each departure is passed in to the function in a variable called **arrival**. From the arrival object the minutes, platform, length of the train and color of the line (in hex) are stored in local variables.

In the variable **out** the output is constructed using the information passed in to the function. The train information along with a logical division for the blank colored box representing the line is appended to the **listview**.