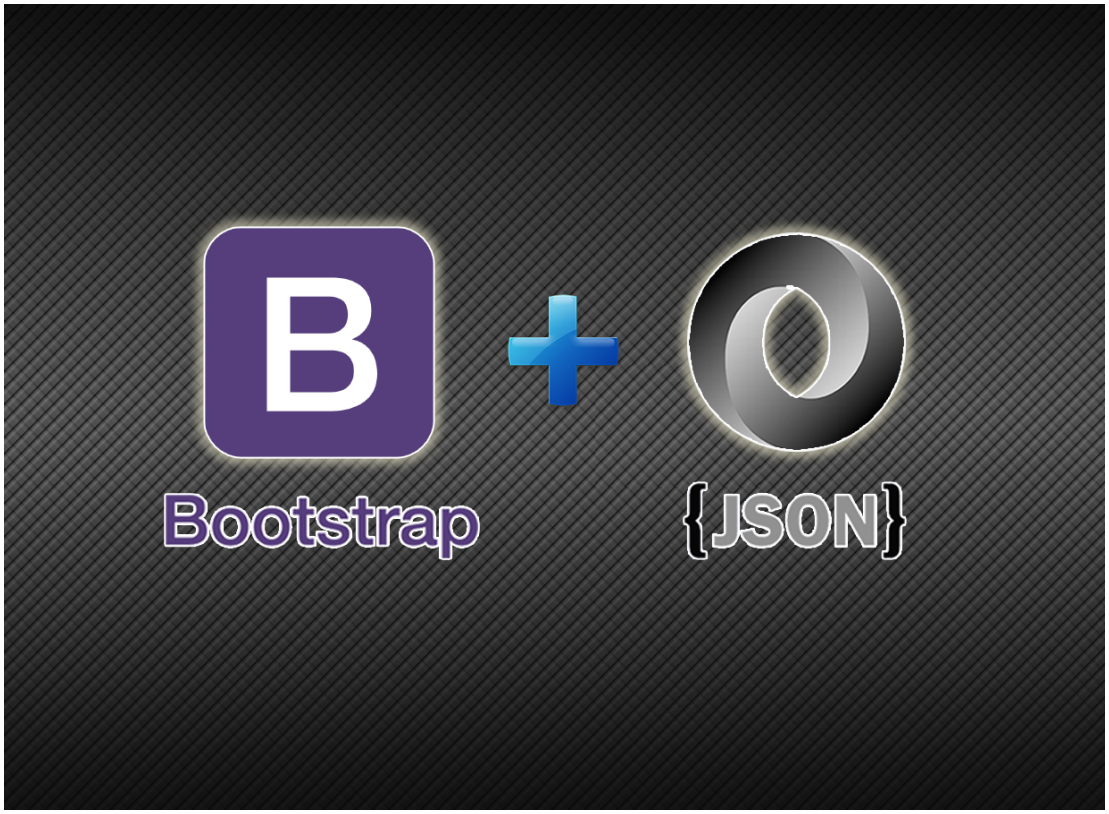


Dynamically Loading JSON Data into A Bootstrap Web Page

By Chad Jordan – July 9th, 2015

Introduction

In this guide you will learn:

1. The fundamental definitions of Bootstrap, and JSON
2. The key features of using the Bootstrap 3.3.5 front-end framework
3. How to create a responsive HTML web page to comply with mobile devices
4. The process of dynamically loading external JSON *(JavaScript Object Notation)* data into a Bootstrap-built Chevrolet web page

In August 2011 an API *(Application Programming Interface)* known as bootstrap was created as a free, open-source JavaScript framework for responsive, mobile-first, front-end web development. Bootstrap is a powerful toolkit that works hand-in-hand with HTML, CSS, and JavaScript to build web pages and applications. The technology was originally created for Twitter by Mark Otto *(Design Architect)*, and Jacob Thornton *(Engineer)*. The framework excelled very quickly among communities of web designers and developers due to its responsive nature, diverse flexibility, ease of use, and consistent design with reusable components. It offers great extensibility with JavaScript, coming with built-in support for jQuery plugins and a programmatic JavaScript API. Bootstrap can be used with any IDE or editor, and any server-side technology and language, from ASP.NET to PHP or Ruby on Rails. Bootstrap allows developers to focus their attention strictly on development without the burden of design, providing a sleek website up and running quickly. It also allows web designers to hone their skills and create Bootstrap themes. Bootstrap is available in two forms; **1)** as a precompiled version, and **2)** as a source code version. The source code version uses the **Less CSS preprocessor**, but if you're more into **Sass** *(Syntactically Awesome Stylesheet)*, there is an official Sass port of Bootstrap also available.

In my earlier guide about dynamically loading XML data into a Flash website, I provide details from an external **XML** *(Extensible Markup Language)* data file, create the internal content using Flash, and wrote all of the functionality using ActionScript 3.0. This guide will obviously be very different for the purposes of the content being loaded and the programming being used for said content and functionality, but the concept of dynamically loading external data into a website is still very similar. For this guide, I'll be using **JSON** *(JavaScript Object Notation)* to load my external data. JSON is simpler than XML, but XML is more powerful.

JSON Example:

For common applications, JSON's concise semantics results in easier to read code.

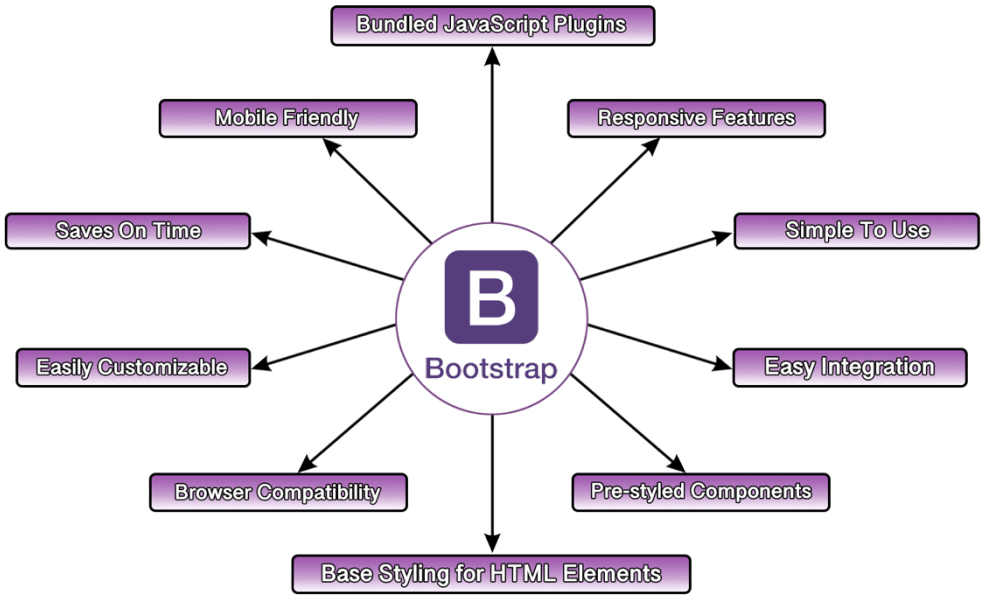
XML Example:

For applications with complex requirements surrounding data enterprise exchanging, XML is more powerful with verbose features which significantly reduces software vulnerabilities.

The purpose of this small task is to demonstrate the use of dynamically loading JSON data into a Bootstrap webpage to provide the user with basic information. The overall purpose is to simply load the external data, and verify that the JSON data is read in, properly displays on the screen, and filter the data by type. In the case of a Chevrolet webpage, the user should be able to filter their search by the type of vehicle that they are wanting information for. These tasks are fairly straightforward thanks to the features of Bootstrap.

Features of Bootstrap

Plugins can be included individually *(using Bootstrap's individual .js files)*, or all at once *(using bootstrap.js or the minified bootstrap.min.js)*. Bootstrap includes a **responsive**, mobile-first grid system that appropriately scales up to 12 columns as the device or viewport size increases. It includes predefined classes for easy layout options, as well as powerful amalgamations for generating more semantic layouts. Bootstraps **simplicity** allows for much faster implementation due to its huge assortment of JavaScript plugins. This also allows for **easier integration** of custom-created content from other platforms. With the pre-compiled version of Bootstrap, developers can already use pre-defined/**pre-styled components** at their fingertips. Over a dozen reusable components are built to provide iconography, dropdowns, input groups, navigation, alerts, and much more. Bootstrap includes over 250 glyphs in font format from the *Glyphicon Halflings set*. There are also **elements of base styling** with abbreviated stylizes of implementation with HTML's **<abbr>** tag for elements using abbreviations and acronyms to show the expanded version on hover. Abbreviations with a *title* attribute have a light dotted border at the base which helps the cursor on *hover*, providing additional context on hover events. **Example:**




```
<abbr title="HyperText Markup Language" class="initialism">HTML</abbr>
```

We can do the same thing with *Addresses* by using the **<address>** tag. This presents contact information for the nearest ancestor or the entire body of work. There's also a lot more baseline styling available for blockquotes, lists *(Ordered and Unordered)* inline, description, alignment, and emphasis classes. Bootstrap has **multi-browser compatibility** for Chrome, Firefox, Opera, Safari, and Microsoft Edge across Windows and macOS. *Unofficially*, Bootstrap should look and behave well enough in Chromium and Chrome for Linux, Firefox for Linux, and Internet Explorer 9. **Customization** for Bootstrap is truly unique because for maintainability you can leave the initial Bootstrap source code as-is and merely add custom code in your external stylesheet. The code in this external stylesheet will override the existing styles and is good for basic customizations however, for more extensive alterations, SASS is the recommended method. Doing more with less code has always been JQuery's tagline, but with even more features, and plugins built into Bootstrap, not only can we do more with less code, but we can **do even more in less time** as well. When it comes to responsive design, grid layout

plays a significant role, and creating a mobile-friendly website is easy and smooth with the help of Bootstrap. It has ready-made classes which help to recognize the number of spots in the grid system. Using Bootstrap, responsive designs are made easy which means that it's far more **mobile-friendly** than other frameworks.

Building A Bootstrap Webpage

Writing code for Bootstrap isn't much different from HTML since it's a framework of JavaScript plugins rather than a language. In this guide, I'm not installing any additional dependencies or software, but rather referencing the current online files that are provided by the Bootstrap website. Otherwise, the process of building this webpage is going to be very similar to my previous guides for front-end web development. Unlike my previous guides when I declared XHTML strict, ever since HTML 5 came out last year, we no longer declare XHTML strict in our main document. In modern HTML we simply add **<!DOCTYPE html>** at the top which in itself is far easier than the extensive info that we used to declare. The document mode meta tag on **line 5** with the ***X-UA-Compatible*** declaration allows the developer to choose what version of Internet Explorer the page should be rendered as. When optimizing the web for mobile devices, it's important to understand how these responsive elements are implemented, and part of this implementation starts on **line 6**.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7          <title>Chevrolet Information Explorer</title>
8      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
9      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap-theme.min.css">
10     <link rel="stylesheet" href="animate.min.css">
11     <link rel="stylesheet" href="style.css">
12         <script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
13 </head>
```

Responsive control

Typically, mobile-optimized sites will have a meta viewport tag like this. The viewport is the area of the window where the web content can be seen. This is generally not the same size as the rendered page, in which case the browser provides scrollbars for the user to scroll around and access all the content. The responsive control is what will allow for the optimization of mobile devices. The remaining parts that control the responsive content are written on **lines 8 and 9**. As far as changes to your CSS element priorities, changes in your style.css file will override the existing stylesheets that are being referenced on lines 8 and 9. With this in mind, I leave my style.css file blank unless I decide to make changes later on, but for the sake of this guide, Bootstrap will handle all of the styling and responsive controls that I need for this guide. For my **<script>** tag on **line 12**, this source URL will reference the file location and allow the animation effects that I want to create. Beginning on **line 15** I create a div for the main

```
14 <body>
15     <div class="container">
16         <div class="row">
17             <div class="col-xs-12 text-center" id="header">
18                 <div class="jumbotron">
19                     <h1>Chevrolet Explorer</h1>
20                     <p>Find the vehicle that fits your style &amp;amp; budget.</p>
21                 </div>
22             </div>
23         </div>
```

container to hold the title within the header **<h1>** tags and then print out the tagline underneath. After closing out with my ** ** tags, this is the output result in the browser so far:

Per my earlier mention of allowing the user to filter the vehicles by type, this is how I provide a basic layout of controls for the user to perform these actions. Basic checkboxes are not difficult to make, even in traditional HTML code with no CSS required. The class, *checkbox* is made as

Creating Checkboxes in HTML

```
24          <div class="row">
25              <div class="col-md-4" id="filter">
26                  <p> </p>
27                  <div class="well">
28                      <h5>Filter</h5>
29                      <div class="checkbox">
30                          <label><input type="checkbox" class="filter-type" value="cars"> Cars</label>
31                      </div>
32                      <div class="checkbox">
33                          <label><input type="checkbox" class="filter-type" value="performance"> Performance</label>
34                      </div>
35                      <div class="checkbox">
36                          <label><input type="checkbox" class="filter-type" value="suvs"> SUVs</label>
37                      </div>
38                      <div class="checkbox">
39                          <label><input type="checkbox" class="filter-type" value="trucks"> Trucks</label>
40                      </div>
41                  </div>
42              </div>
```

an **<input>** tag just as I wrote in my previous regex validation guide. In HTML, plain checkboxes can be made with little to no effort. These checkboxes are made from **lines 29 to 40**. Each **<input>** tag is given an input type, in this case, *"checkbox"* and they can be passed a class, and id, a name, or a value attribute. I've passed a class, and a value for mine, and simply provided a name outside of the input tag, but leaving the name inside the <label> tag will provide a name right next to each corresponding checkbox. This next example will be positioned above the container that displays the vehicles. Each time the user changes a checkbox within the search

```
43              <div class="col-md-8">
44                  <p class="text-muted text-center"><span id="vehicleMatchCount"></span> Vehicles Matching Your Filters:
    </p>
45                  <div id="vehicleMatches"></div>
46              </div>
47          </div>
48      </div>
```

filter, the numbered results will change based on the data that is included in the JSON file. **Example:** If there are 4 SUVs, and the only checkbox selected is SUVs, then only SUVs will display with the number **4** beside the *"Vehicles Matching Your Filters:"* text. By default, no checkboxes will be selected at runtime which means all vehicles will display in the container with the total number of cars beside the *"Vehicle Matching"* text. This next block controls how we will set up and reference the files needed for the animation effects in Bootstrap. Just as I referenced bootstrap data at *maxcdn.boostrapcdn.com* for styling my page with the most current CSS files, I do the same thing again except this time with JavaScript. The divs contain the individual sections for each piece of content that will be displaying our search results for

```
49      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
50      <script type="text/template" id="templateVehicle">
51          <div class="panel panel-default animated fadeInRight">
52              <div class="panel-body">
53                  <div class="row">
54                      <div class="col-sm-4">
55                          <a href="#zoom" class="thumbnail"><img id="vehicleImage" src="http://placehold.it/350x250"
    class="img-responsive" title=""></a>
56                      </div>
57                      <div class="col-sm-8">
58                          <h3 id="vehicleTitle">2014 Chevy Malibu (White)</h3>
59                          <p id="vehiclePrice" class="lead">$18,500.00</p>
60                      </div>
61                  </div>
62              </div>
63          </div>
64      </script>
65      <script src="app.js"></script>
66  </body>
67  </html>
```

the vehicles and references back to the CSS styling that I call earlier in the head of my page. You'll notice on **line 55** that I set the anchor tag reference to ***#zoom*** because by putting the hashtag symbol before the variable this prefix is declaring a private variable. Setting it this way will nullify any click on the thumbnail image that the user may attempt to make. The main purpose of this exercise is to merely demonstrate how to load JSON data into a Bootstrap page so there is no need for any additional functionality here. Finally, near the bottom of the page, I write a script tag to reference the JavaScript file *(app.js)* that I will use to write all of the functionality and other behavior for the page. However, before I start writing that, I'm going to reference the JSON file. As I mentioned earlier, when it comes to JSON versus XML syntactically, JSON is easier to read and write. The first list in the section of vehicles falls under the categories of *cars, performance, SUVs, and trucks.* Under the *cars* category, I have the 2015 Spark (Green) car, the Sonic (Red), the Cruze (Cobalt) *etc. etc.* You can see that I specify a model, a color, a year, and price for each car. This is the information contained in the JSON file that will be fed into the webpage at runtime. Just like XML, we can

```
1  {
2          "cars" : [
3                  {
4                          "model" : "Spark",
5                          "color" : "Green",
6                          "year"  : "2015",
7                          "price" : 12270
8                  },{
9                          "model" : "Sonic",
10                         "color" : "Red",
11                         "year"  : "2015",
12                         "price" : 14245
13                 },{
14                         "model" : "Cruze",
15                         "color" : "Cobalt",
16                         "year"  : "2015",
17                         "price" : 16170
18                 },{
```

store however much data we want within however many different categories that we wish. Once I've finished categorizing the normal vehicles into the *'car'* category, I store the more high-performance vehicles into the 'performance' category, and so forth until I have placed each car into its own category. The remaining data isn't required to post in this guide since it all follows the same format for every category. These images among several others are stored in a folder called *'img'* and they will also be loaded into the webpage via my *app.js* file.

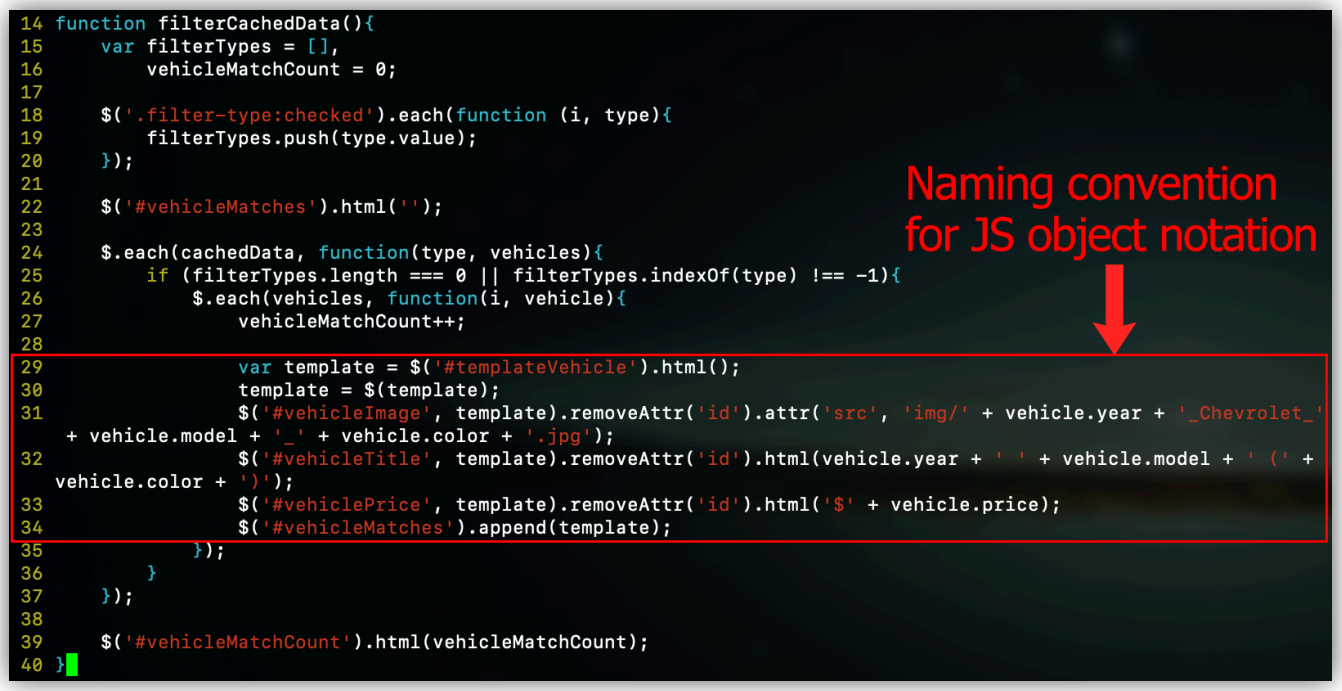
```
19                          "model" : "Malibu",
20                          "color" : "White",
21                          "year"  : "2015",
22                          "price" : 22465
23                  },{
24                          "model" : "Impala",
25                          "color" : "Blue",
26                          "year"  : "2015",
27                          "price" : 27060
28                  },{
29                          "model" : "Volt",
30                          "color" : "White",
31                          "year"  : "2015",
32                          "price" : 34345
33                  }
34          ],
35
36          "performance" : [
37                  {
38                          "model" : "SS Sedan",
39                          "color" : "Black",
40                          "year"  : "2015",
41                          "price" : 45745
42                  },{
43                          "model" : "Camaro",
44                          "color" : "Red",
45                          "year"  : "2015",
46                          "price" : 23705
```

Creating the Functionality

Just like everything else in this program, the JavaScript portion won't be that bad to write. Beginning on the first line we write a variable to cache the JSON data. Next on **line 3** when the document is ready, this function will set up the page and the variable *getJSON* is executed to retrieve the data from the JSON file *chevrolet.json*, takes that data, and then assigns it back to the variable *cachedData*. Starting on **line 9** we create a function

```
1  var cachedData;
2
3  $(document).ready(function(){
4      $.getJSON('chevrolet.json', function(data){
5          cachedData = data;
6          filterCachedData();
7      });
8
9      $('.filter-type').on('change', function(){
10         filterCachedData();
11     });
12 });
```

call to 'filter-type' and execute it against any change made to the cached data. Next, we have to write a function for *filterCachedData* so this function call is made on **line 14** and then on **lines 15 and 16** we get the types of vehicles selected. The variable *filterTypes* is set to an array to store the types of vehicles, but since the page loads with no options selected, we ensure that *vehicleMatchCount* is set to 0. Once the user makes choices, the information data in the array changes. The function on **line 18** is simply checking each time the user pushes a value for each checkbox. **Line 22** clears the match list so there isn't any unwanted data clogging up the cache. **Line 24** sets up an iterator that checks the length of the filters in the vehicle list and if there is a match, it increments *vehicleMatchCount* allowing to render the matches of the vehicles found.




```
14 function filterCachedData(){
15     var filterTypes = [],
16         vehicleMatchCount = 0;
17
18     $('.filter-type:checked').each(function (i, type){
19         filterTypes.push(type.value);
20     });
21
22     $('#vehicleMatches').html('');
23
24     $.each(cachedData, function(type, vehicles){
25         if (filterTypes.length === 0 || filterTypes.indexOf(type) !== -1){
26             $.each(vehicles, function(i, vehicle){
27                 vehicleMatchCount++;
28
29                 var template = $('#templateVehicle').html();
30                 template = $(template);
31                 $('#vehicleImage', template).removeAttr('id').attr('src', 'img/' + vehicle.year + '_Chevrolet_'
    + vehicle.model + '_' + vehicle.color + '.jpg');
32                 $('#vehicleTitle', template).removeAttr('id').html(vehicle.year + ' ' + vehicle.model + ' (' +
   vehicle.color + ')');
33                 $('#vehiclePrice', template).removeAttr('id').html('$' + vehicle.price);
34                 $('#vehicleMatches').append(template);
35             });
36         }
37     });
38
39     $('#vehicleMatchCount').html(vehicleMatchCount);
40 }
```

Starting on **line 29** we create a variable called *templateVehicle* which will be searching for a very specific file format for each private variable called *vehicleImage*. JSON searches for a specific naming convention for object notation, and this is not just the data within the file, but in order to load the images, we declare how we want the image files to be named in a format that JSON will search for. In this case, the attributes of the format are ***vehicle year*,** followed by ***model*, *color*,** and then the file extension which in this case is **.jpg**
Example: *'2015_Chevrolet_Camaro_Red.jpg'* Now with this code in place, after testing it, I can confirm this program runs flawlessly, is fully responsive and performs the necessary functions that I wanted to demonstrate. Would you like to test the demo? Click here.

Conclusion

My hope is that this guide has helped teach how with just a little effort, a lot more can be done using Bootstrap than traditional markup and JavaScript. I only recently started learning Bootstrap but since I've previously been exposed to numerous other programming languages & technologies over the years, the foundations of computer science have taught me to be more adaptable in diverse development environments. Essentially Bootstrap to JavaScript is what

Laravel is to PHP. As of now, it's the best framework I can think to use as an API with JavaScript. All diagrams and code presented in this guide were created and written by Chad Jordan for learning purposes only. This Bootstrap page was made with the latest version 3.3.5 and written using the Vim code editor. For any possible inquiries such as general questions regarding this guide or other professional inquiries please feel free to email me at cjordan@wondercreationstudios.com

Resources Used:

- Getbootstrap.com
- Bootstrapdocs.com
- Freecodecamp.org
- W3schools.com