Joseph Powell

Dr. Beaty

CS 390P – Web Application Development

Fall 2018

**Abstract:**

The following document will cover the steps necessary to add a single-page-application (SPA) in Javascript to a Ruby on Rails (RoR) project pulling from the JSON API. It will also include some basic HTML and incorporate the Vue/Axios framework and tools to carry out this task.

**Motivation:**

While it is nice being spoiled by RoR, it’s not the way that all web applications are built, and we should be able to build outside of this. Using Javascript/HTML elements are extremely common across the web, and anyone wanting to build web applications/pages should know a little (probably a lot) bit about both.

**Introduction:**

We are using an older, and simpler RoR project to build on top of so we don’t have to worry about also building the foundation and associated dependencies for this project. Our RoR project knows about Courses/Sections/Students (at least in how it applies to our SPA). The task is to create a single page application that pulls from the JSON API of each of these entities and show the relevant data on a single page that does not change URLs.

**Method/Measurement:**

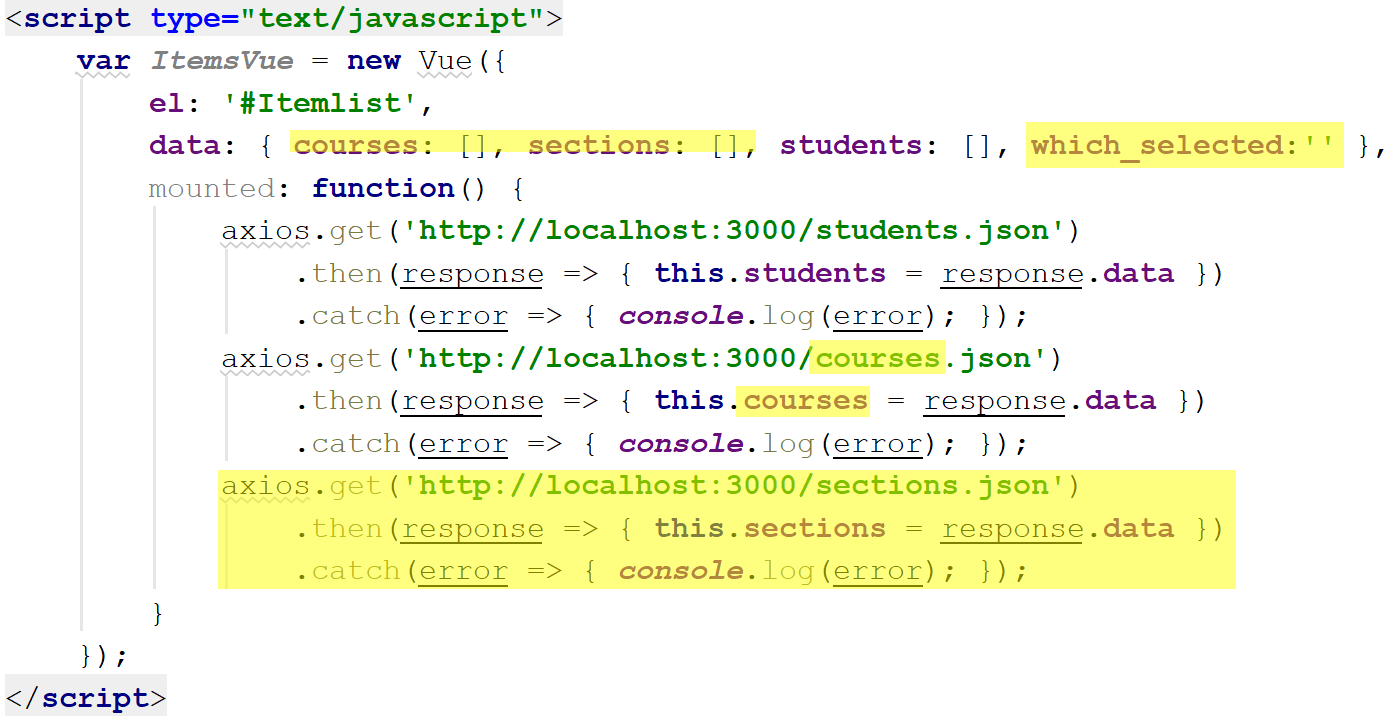
Link to my SPA: <https://github.com/jpowell47/AssignmentFiveSPA/blob/master/coursesectionstudent/public/vueSPA.html>

My starting point for this project was pulling example code from Dr.Beaty’s Github account found here: <https://github.com/drsjb80/SPAs/blob/master/vueaxiosspa1toM.html>. This is a simple SPA that uses an RoR application viewing only students/advisors, so with a little bit of tweaking, we can get this to work with our own. Create an HTML file in the “Public” folder in your RoR application, and put the code here. The first interesting thing to note, is how we are retrieving the information.



The important part here, that differs from other SPA examples, is the “mounted: function()..”. What this does is grab all of the JSON data specified upon page load, rather than grab it dynamically as we’re doing stuff on the page. The dynamic part lies in the “students\_selected”, as depending on the value in this, the page will render different data (we will go over the code that does this shortly.

Update your code to reflect the snippet below, where highlights indicate changes.



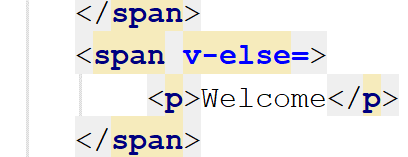
Starting from the top, we need arrays to hold the JSON data for each entity, so will create and update to reflect our application. The next piece is the which\_selected variable. Rather than hold a boolean value like the previous placeholder, it will hold a string that will be initialized as empty. Now we need to add another Axios call to pull the JSON data for our third entity, as well as update the advisors to sections or courses. This should take care of this section, we will now move up to the HTML and other vue calls.



This section will look a little bit different than the previous SPA. Starting from the top, our v-on:click lines will create a button, as well as associate it with an action (thanks, Vue). The first thing it does, is change that variable we just created from an empty string to courses, sections, or students depending on which button they press. This will make way for the following section, where we actually render the appropriate data as well as format it.

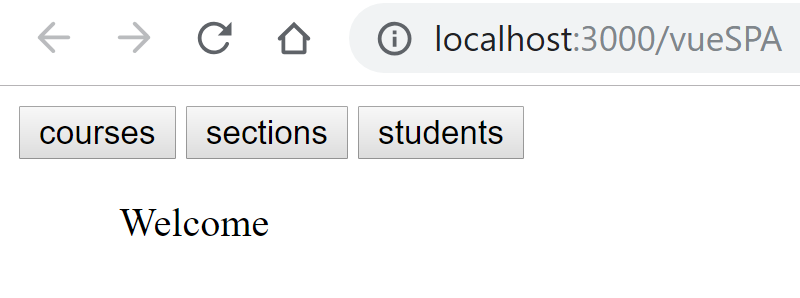
We use v-if/else statements to conditionally format our data, based on what string is in the “which\_selected” variable. Use the code within the <ul></ul> tags (the rest will be provided shortly) to create a basic table with the appropriate headings/data. The v-for is just a for Vue for loop to iterate through our list of JSON data looking for the items we specify.



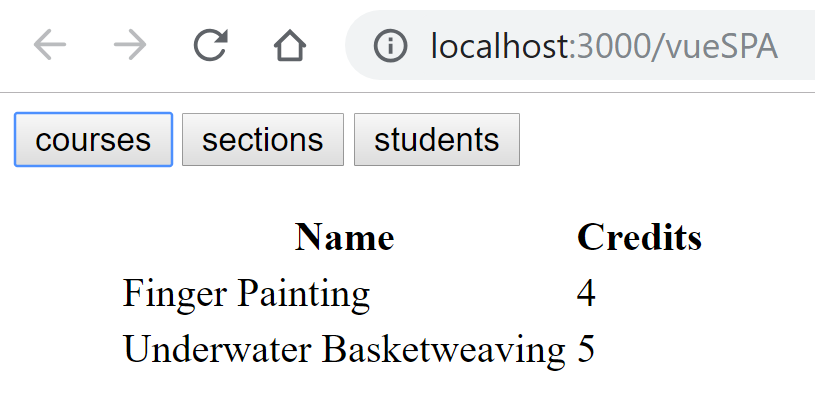


The you’ll notice that we continue along with the same idea in mind, using the appropriate else-if statements and for loops. The only special thing we do here is getting the name for a section of a course – since only an ID is stored in the section’s JSON data. Lastly, we just have our generic else that shows a message only the first time you go to the page.

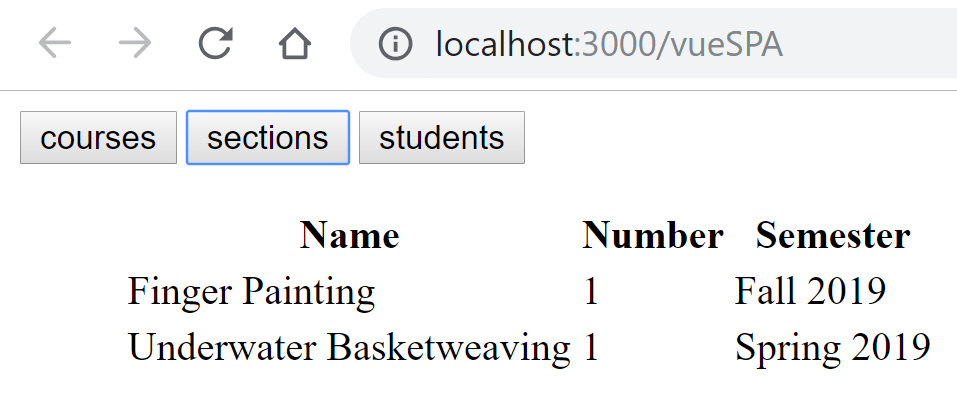
**Results:**



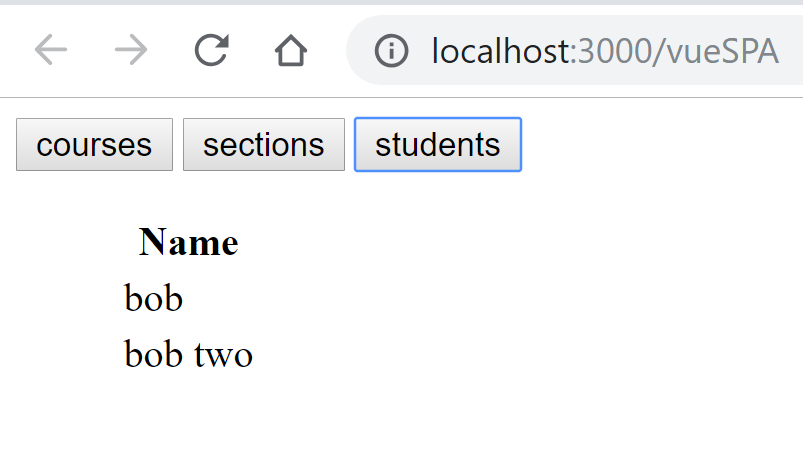
The initial page when you first visit.



The page when you press courses (URL is still the same, yay).



When you press sections.



And lastly, when you press names!

**Conclusion:**

While still not very much code, there is a great deal happening in this file. Hopefully after using this guide, and tinkering around a bit, you should be able to understand a few fundamental concepts of Javascript, Vue, and Axios. Learning these tools is crucial in understanding and moving forward in our journeys as the concepts used will help translate to other web languages and frameworks.

**Bibliography:**

<https://github.com/drsjb80/SPAs/blob/master/vueaxiosspa1toM.html>

<https://github.com/jpowell47/AssignmentFiveSPA/blob/master/coursesectionstudent/public/vueSPA.html>

<https://forum.vuejs.org/t/generating-table-from-json-array-of-objects/28434/2>

<https://medium.freecodecamp.org/an-introduction-to-dynamic-list-rendering-in-vue-js-a70eea3e321>

**Reflection:**

This assignment was probably the most challenging (since it wasn’t RoR) one we’ve had so far. The concepts seemed easy to implement, but it seems that the Vue framework works differently than what is obvious. I struggled for quite a while on just conditional formatting, it wasn’t until I found your example using Axios could I get it to work the way I wanted to. I was running into issues when trying to make ajax calls at the same time as changing a variable in data to dynamically render the tables. Having the option to pull all of the JSON data upon page load seemed much more straight forward. Thankfully, the table part of the code was just a quick google search, and once I understood how to make the conditional formatting work, so was everything else. This project while being the most difficult to understand, forced me to understand the language/frameworks in the way they were meant to be used (I think), as opposed to the way I am used to building things.