**Introduction**

This is the opening post to a tutorial series on using [VueJS](https://vuejs.org/) and [Flask](http://flask.pocoo.org/) for full stack web development. In this series I am going to demonstrate how to build a survey web app where the application architecture consists of a frontend comprised of a VueJS Single Page Application (SPA) and a backend REST API using Flask.

This first post will cover basic project setup and structure, using the VueJS v-for directive, and component lifecycle stages.

**Tutorial Contents:**

1. Seup and Getting to Know VueJS
2. Navigating Vue Router
3. State Management with Vuex
4. RESTful API with Flask
5. AJAX Integration with REST API
6. Authentication
7. Deployment to a Virtual Private Server

**Frontend setup with vue-cli and webpack**

I will be using two very important tools for a VueJS project which are the official [VueJS command line interface (CLI)](https://vuejs.org/v2/guide/installation.html#CLI) and the very powerful module bundler and build tool [webpack](https://webpack.js.org/). Both of these tools rely on the [nodeJS](https://nodejs.org/en/) runtime and its package manager, [npm](https://www.npmjs.com/). If you have not already installed node then please consult the nodeJS installation [docs](https://nodejs.org/en/) for your system which will also include an installation of npm.

Install the VueJS CL (vue-cli).

$ npm install vue-cli -g

Now with the CLI installed I will use it to initialize a VueJS SPA application. The CLI takes does the following:

1. install and configure webpack to bundle my code
2. install a development server with hot-reload (server auto-restart when a file is saved)
3. Add a dependency for vue-router
4. scaffold out a basic VueJS SPA file structure

I first create a high-level folder which will contain all the code for this tutorial named survey. Next I make two more directories called frontend (for the VueJS SPA) and backend (for developing the REST API) then cd into the frontend directory.

|  |
| --- |
| $ mkdir survey $ cd survey $ mkdir frontend $ mkdir backend $ cd frontend |

Now for the real magic. The Vue CLI initialization command is entered which then prompts me to answer a series of questions.

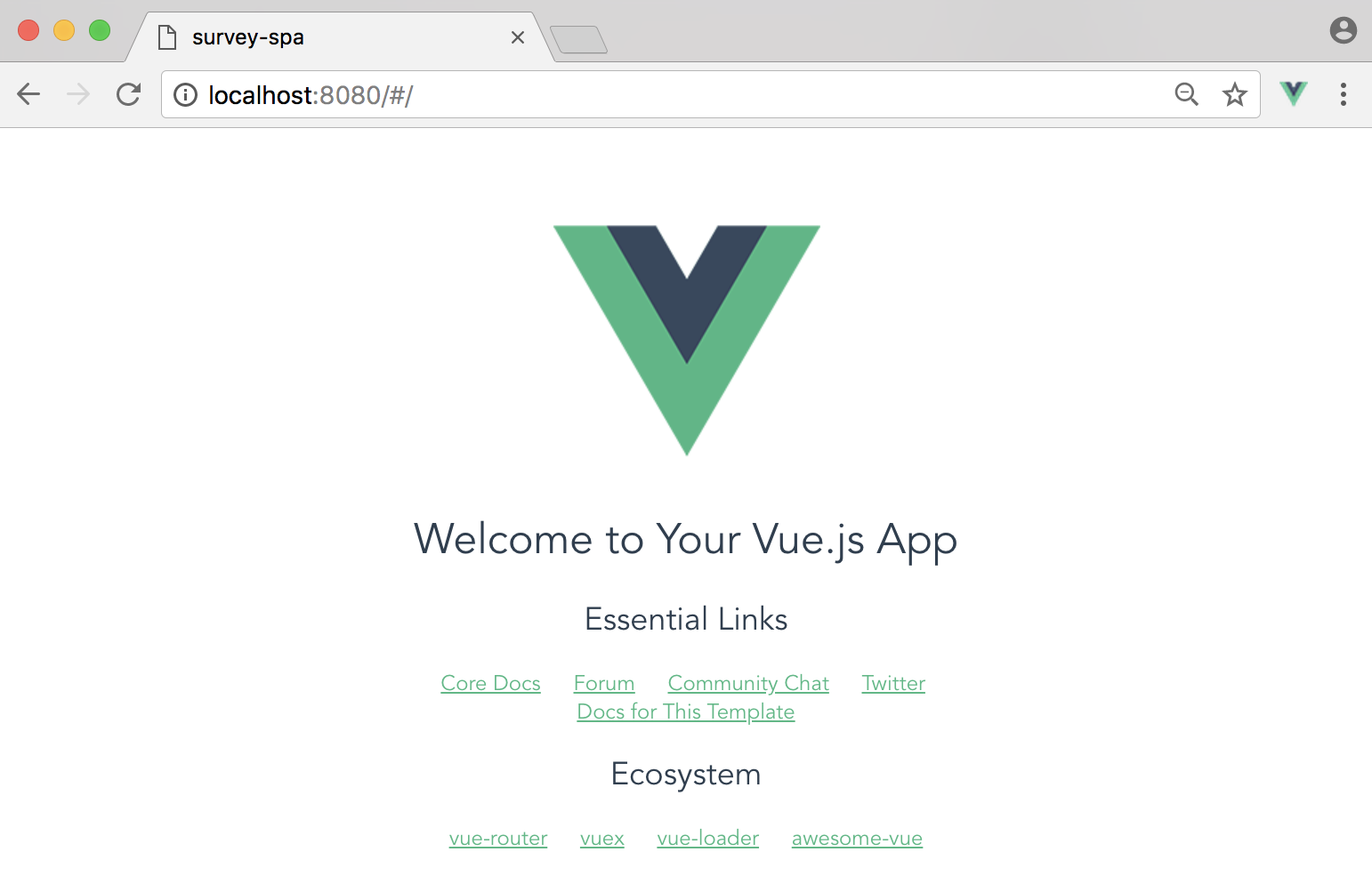
Press enter accepting the defaults for the questions (i) Project name, (ii) Project description, (iii) Project author, (iv) Build standalone. When prompted to install vue-router enter Y for yes. Enter n for the remaining entries and accept the defaults.

|  |
| --- |
| $ vue init webpack survey-spa  ? Project name survey-spa ? Project description A Vue.js project ? Author Adam McQuistan <sci.guy.mcq@gmail.com> ? Vue build standalone ? Install vue-router? No ? Set up unit tests No ? Setup e2e tests with Nightwatch? No ? Should we run `npm install` for you after the project has been created? (recommended) npm   vue-cli · Generated "survey-spa".   # Installing project dependencies ... ... |

There should now be a new directory called survey-spa. Change directories into this directory and issue an npm command to compile the project and launch the dev server.

|  |
| --- |
| $ cd survey-spa $ npm run dev |

Now I can enter <http://localhost:8080> into my browser window and I should see the boilerplate VueJS template page similar to what is shown below.



The file structure that was created looks like what is shown below. I’ve purposefully omitted the quagmire of nonsense within the node\_modules folder.

survey-spa/

├── README.md

├── build

│ ├── build.js

│ ├── check-versions.js

│ ├── logo.png

│ ├── utils.js

│ ├── vue-loader.conf.js

│ ├── webpack.base.conf.js

│ ├── webpack.dev.conf.js

│ └── webpack.prod.conf.js

├── config

│ ├── dev.env.js

│ ├── index.js

│ └── prod.env.js

├── index.html

├── package-lock.json

├── package.json

├── src

│ ├── App.vue

│ ├── assets

│ │ └── logo.png

│ ├── components

│ │ └── HelloWorld.vue

│ ├── main.js

│ └── router

│ └── index.js

└── static

This likely seems a bit overwhelming the first few times you look at it but, fear not we only really need to concern ourselves with the files under the src/ directory plus the index.html file. Of course the other files are important and perhaps some day I will get into what they are used for but, for now just ignore them.

Files under the src/ directory are where I will be writing the code to drive the functionality of the application. Let us open these files up and get an idea for what is going on.

Index.html

|  |
| --- |
| <!DOCTYPE html> <html>  <head>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width,initial-scale=1.0">  <title>survey-spa</title>  </head>  <body>  <div id="app"></div>  *<!-- built files will be auto injected -->*  </body> </html> |

This is the only html file that a Vue SPA utilizes and it rarely contains much more than what is shown above with the exception that sometimes you will link to CSS frameworks and other JavaScript libraries. The lone div element that is produced with a default id of app is what the main Vue instance will attach to. That Vue object injects the HTML and CSS that are in the components, to be discussed later, into the div to produce the UI.

main.js

|  |
| --- |
| *// The Vue build version to load with the `import` command* *// (runtime-only or standalone) has been set in webpack.base.conf with an alias.* import Vue from 'vue' import App from './App' import router from './router'  Vue.config.productionTip = false  */\* eslint-disable no-new \*/* new Vue({  el: '#app',  router,  components: { App },  template: '<App/>' }) |

The main.js file is the primary entry point for the application and is where you will register the Vue instance and extensions such as vue-router and vuex. As you can see this is where the Vue instance resides. The instance is registered to the app div discussed previously plus it is fed the router object and the high level App component.

App.vue

|  |
| --- |
| <template>  <div id="app">  <img src="./assets/logo.png">  <router-view/>  </div> </template>  <script> export default {  name: 'App' } </script>  <style> #app {  font-family: 'Avenir', Helvetica, Arial, sans-serif;  -webkit-font-smoothing: antialiased;  -moz-osx-font-smoothing: grayscale;  text-align: center;  color: #2c3e50;  margin-top: 60px; } </style> |

The App.vue file serves as the top level application components and often contains the general layout of the application. Vue components have a specific structure containing a template section for component specific HTML, a script section to define the Vue object and that components behaviors implemented in JavaScript and, a styles section for CSS / SCSS rules. That last bit can be a bit confusing though because by default the style rules you define in a component do not just pertain to that component. They actually affect all the elements in the entire project unless you add a scoped attribute to the style element.

router/index.js

|  |
| --- |
| import Vue from 'vue' import Router from 'vue-router' import HelloWorld from '@/components/HelloWorld'  Vue.use(Router)  export default new Router({  routes: [  {  path: '/',  name: 'HelloWorld',  component: HelloWorld  }  ] }) |

The index.js script in the router directory is where the urls for the application are defined and mapped to components. The first two lines import the Vue and Router objects which are then linked by the use method on the Vue object.

The default route that is provided from the vue-cli webpack template is simply the root or index route for the application which servers up the HelloWorld component. To map a route path to a component it first has to be imported then you need to define a route object in the routes array giving it a path, a name and, the component to be displayed.

components/HelloWorld.vue

|  |
| --- |
| <template>  <div class="hello">  <h1>{{ msg }}</h1>  <h2>Essential Links</h2>  <ul>  <li>  <a href="https://vuejs.org" target="\_blank">Core Docs</a>  </li>  <li>  <a href="https://forum.vuejs.org" target="\_blank">Forum</a>  </li>  <li>  <a href="https://chat.vuejs.org" target="\_blank">Community Chat</a>  </li>  <li>  <a href="https://twitter.com/vuejs" target="\_blank">Twitter</a>  </li>  <br>  <li>  <a href="http://vuejs-templates.github.io/webpack/" target="\_blank">  Docs for This Template  </a>  </li>  </ul>  <h2>Ecosystem</h2>  <ul>  <li>  <a href="http://router.vuejs.org/" target="\_blank">vue-router</a>  </li>  <li>  <a href="http://vuex.vuejs.org/" target="\_blank">vuex</a>  </li>  <li>  <a href="http://vue-loader.vuejs.org/" target="\_blank">vue-loader</a>  </li>  <li>  <a href="https://github.com/vuejs/awesome-vue" target="\_blank">  awesome-vue  </a>  </li>  </ul>  </div> </template>  <script> export default {  name: 'HelloWorld',  data () {  return {  msg: 'Welcome to Your Vue.js App'  }  } } </script>  <!-- Add "scoped" attribute to limit CSS to this component only --> <style scoped> h1, h2 {  font-weight: normal; } ul {  list-style-type: none;  padding: 0; } li {  display: inline-block;  margin: 0 10px; } a {  color: #42b983; } </style> |

The components directory is where UI components reside. The HelloWorld component above again contains the three basic sections template, script, and style which all have some example content from the vue-cli initialization template.

Take a look at the contents of the script section. Here you will see that an object is being exported. This object will get injected into the Vue instance that was initialized in the main.js file. Inside this javascript object is a data method which returns an object. This object is where you can place component level state (data) and in this example its a single property called msg.

You can access and display your component state by its property name with in the HTML in the template section. In the example provided you see this as {{ msg }}. The double curly brackets are the default template syntax for doing text interpolation and is inspired by the [Mustache](https://mustache.github.io/) template system. Any time you want to display data in the HTML of your component wrap it in double curly brackets.

**Bringing in Some Style**

To give this app a little better curb appeal I will be using the [Bulma](https://bulma.io/) CSS framework. The aim of this tutorial will not be on how to use Bulma but, I still want to include it to stave off the drab look of unstyled html. Back in the terminal in the same directory as the package.json file install and save bulma to the project with the following command.

|  |
| --- |
| $ npm install --save bulma |

In addition, I will need to install some dev tools to properly load the styles for the application so the components know how to work with them. To do this, npm install these additional packages.

|  |
| --- |
| $ npm install --save-dev vue-style-loader $ npm install --save-dev css-loader $ npm install --save-dev sass-loader $ npm install --save-dev node-sass |

Now open up App.vue and replace the style section with what is below which will import the bulma framework using scss import syntax.

|  |
| --- |
| <style lang="scss"> @import '~bulma/bulma' </style> |

**Give it a Home (page)**

Ok, now that we have a basic understanding of the structure and main parts of a VueJS SPA application I can start writing code. Start by renaming the HelloWorld.vue file to Home.vue then clear out the contents of the template, script and, style sections. Also in App.vue remove the line for the Vue image logo <img src="./assets/logo.png"> and clear out the contents of the style element while your in there. Lastly, open up router/index.js and import the component Home.vue instead of HelloWorld.vue and update the route object to use the Home component.

|  |
| --- |
| import Vue from 'vue' import Router from 'vue-router' import Home from '@/components/Home'  Vue.use(Router)  export default new Router({  routes: [  {  path: '/',  name: 'Home',  component: Home  }  ] }) |

Now open up the Home.vue file and fill the script section with what is below.

|  |
| --- |
| export default {  data () {  return {  surveys: [{  id: 1,  name: 'Dogs',  created\_at: new Date(2017, 12, 1),  questions: [  {  id: 1,  text: 'What is your favorite dog?',  choices: [  { id: 1, text: 'Beagle', selected: 0 },  { id: 2, text: 'Labrador', selected: 0 },  { id: 3, text: 'Rottweiler', selected: 0 }]  }, {  id: 2,  text: 'What is your second favorite dog?',  choices: [  { id: 5, text: 'Beagle', selected: 0 },  { id: 6, text: 'Labrador', selected: 0 },  { id: 7, text: 'Rottweiler', selected: 0 }]  }]  }, {  id: 2,  name: 'Cars',  created\_at: new Date(2017, 12, 3),  questions: [  {  id: 5,  text: 'What is your favorite car?',  choices: [  { id: 17, text: 'Corvette', selected: 0 },  { id: 18, text: 'Mustang', selected: 0 },  { id: 19, text: 'Camaro', selected: 0 }]  }, {  id: 6,  text: 'What is your second favorite car?',  choices: [  { id: 21, text: 'Corvette', selected: 0 },  { id: 22, text: 'Mustang', selected: 0 },  { id: 23, text: 'Camaro', selected: 0 }]  }]  }]  }  } } |

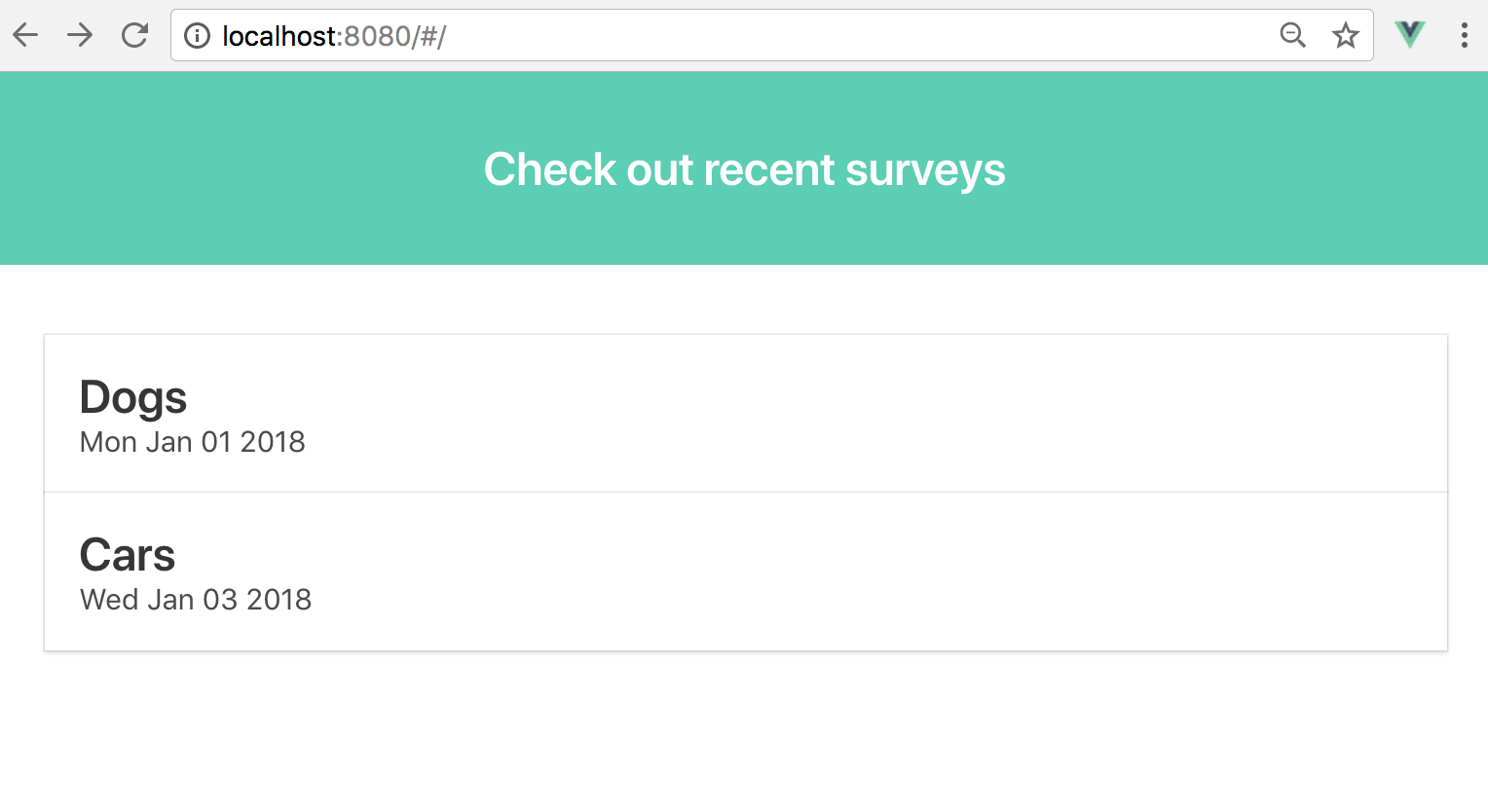
This gives us some dummy data to work with. As you can probably tell it represents two surveys, one about dogs and another about cars. The data will provide the drive the HTML we are about to write providing content to display.

Now in the template section I will add a div which will wrap all my other elements. Every Vue component must have a single parent element, there cannot be top-level sibling elements or the component will not compile. Inside this div I will add a section for a Bulma [hero](https://bulma.io/documentation/layout/hero/) header.

Below the header will be another section for displaying the name of each survey and the date it was created. It is in this block of html where we will start to see some of the awesomeness that is provided by VueJS.

|  |
| --- |
| <template> <div>  <section class="hero is-primary">  <div class="hero-body">  <div class="container has-text-centered">  <h2 class="title">Check out recent surveys</h2>  </div>  </div>  </section>  <section class="section">  <div class="container">  <div class="card" v-for="survey in surveys" v-bind:key="survey.id">  <div class="card-content">  <p class="title">{{ survey.name}}</p>  <p class='subtitle'>{{ survey.created\_at.toDateString() }}</p>  </div>  </div>  </div>  </section> </div> </template> |

Save any unsaved files in the project and refresh your browser which should now appear as shown below.



As you can see in the screenshot there are two survey Bulma cards being displayed. These two surveys map to the array of survey objects that were in my data function of my Home component which I fed into my HTML using the v-for directive.

Take a look again at the below subsection of the original template code which represents a survey. This whole element and its children are being repeated once for each survey in the surveys array. Basically, the data is driving the generation of HTML. The v-bind:key part is adding an attribute called key equal to the survey’s id to each div with class card. Vue uses these keys to keep explicit track of each node being created in the DOM which aids in bookkeeping and performance. It is recommended to always set a unique key to the outermost element being used in conjunction with a v-for directive.

|  |
| --- |
| <div class="card" v-for="survey in surveys" v-bind:key="survey.id">  <div class="card-content">  <p class="title">{{ survey.name}}</p>  <p class='subtitle'>{{survey.created\_at.toDateString()}}</p>  </div>  </div> |

The v-for directive uses a syntax of item in items where items is an iterable such as an array or an object’s properties and item is an alias for each element in the iterable. However, there are variations of this syntax which allows for greater flexibility and control.

For example, say I had an array of letters var letters = ['a', 'd', 'a', 'm'] that I wanted to use to drive the creation of a regular ole HTML unordered list that displays each letter and its corresponding index. Well that could be done with this variation of v-for.

|  |
| --- |
| <ul>  <li v-for="(letter, i) in letters" v-bind:key="i">  Index position {{ i }} has letter {{ letter }}  </li>  </ul> |

Resulting in the following output.

* Index position 0 has letter a
* Index position 1 has letter d
* Index position 2 has letter a
* Index position 3 has letter m

To iterate over the properties of an object the syntax is quite similar. Given an object such as var person = { name: 'adam', occupation: 'software developer', residence: 'lincoln, nebraska' }, iterating with a v-for directive would look like this.

|  |
| --- |
| <ul>  <li v-for="(value, key) in person" v-bind:key=”key”>  key -> {{ key }}, value -> {{ value }}  </li>  </ul> |

* key -> name, value -> adam
* key -> occupation, value -> software developer
* key -> residence, value -> lincoln, nebraska

**Mocking an AJAX Request for Surveys**

I have my first functional UI component displaying a collection of surveys but, in the end the actual application will need to fetch survey data from our REST API. To make this a bit more realistic I would like to mock out the functions for an AJAX request to feed the Home component surveys.

In the src directory create a new folder called api then add a script called index.js within it. This is where I will mock out my AJAX functions. In this new file cut and paste the surveys array from the Home.vue component as a global constant.

|  |
| --- |
| *// api/index.js* const surveys = [{  id: 1,  name: 'Dogs',  created\_at: new Date(2018, 1, 1),  questions: [{  id: 1,  text: 'What is your favorite dog?',  choices: [  { id: 1, text: 'Beagle', selected: 0 },  { id: 2, text: 'Labrador', selected: 0 },  { id: 3, text: 'Rottweiler', selected: 0 }]  }, {  id: 2,  text: 'What is your second favorite dog?',  choices: [  { id: 5, text: 'Beagle', selected: 0 },  { id: 6, text: 'Labrador', selected: 0 },  { id: 7, text: 'Rottweiler', selected: 0 }]  }] }, {  id: 2,  name: 'Cars',  created\_at: new Date(2018, 1, 3),  questions: [{  id: 5,  text: 'What is your favorite car?',  choices: [  { id: 17, text: 'Corvette', selected: 0 },  { id: 18, text: 'Mustang', selected: 0 },  { id: 19, text: 'Camaro', selected: 0 }]  }, {  id: 6,  text: 'What is your second favorite car?',  choices: [  { id: 21, text: 'Corvette', selected: 0 },  { id: 22, text: 'Mustang', selected: 0 },  { id: 23, text: 'Camaro', selected: 0 }]  }] }] |

Below this array of surveys create a function called fetchSurveys which returns a promise with the array of surveys after waiting 300 milliseconds. The function will need to be exported so that it can be accessed from the Home component.

|  |
| --- |
| export function fetchSurveys () {  return new Promise((resolve, reject) => {  setTimeout(() => {  resolve(surveys)  }, 300)  }) } |

Back in the data function of the Home component initialize the surveys property to an empty array and import the fetchSurveys function within the script section.

|  |
| --- |
| <script> import { fetchSurvey } from '@/api' export default {  data () {  return {  surveys: []  }  } } </script> |

Vue components have a series of defined lifecycle stages that are meaningful to the developer when doing various things such as pulling in data in an AJAX request. In order to feed survey data into the Home component I will need to hook into one of the Vue lifecycle stages, specifically the beforeMount stage. There are several other stages that are useful for many more things than just AJAX requests but, I defer to the official VueJS [docs](https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram) for a details explanation.

To take advantage of the beforeMount stage of the Home component all I need to do is add it as a new method to the components Vue object. Inside this function I will make a call to my fetchSurveys function and assign the returned surveys to the surveys data property.

|  |
| --- |
| <script> import { fetchSurveys } from '@/api' export default {  data () {  return {  surveys: []  }  },  beforeMount () {  fetchSurveys().then(response => {  this.surveys = response  })  } } </script> |

Saving all the files in the project and refreshing the browser now gives me the same Home page we saw previously but now with a mocked out AJAX call.

**Conclusion**

This post has covered the basics of setting up a VueJS SPA application with vue-cli using the webpack template. In addition to project setup I covered how to use data in the form of an iterable to generate UI content using the powerful v-for VueJS directive. For the last major topic I hit on component lifecycle stages and how the beforeMounted stage can be useful in loading data into a component with a mock AJAX request.

Next post I will be covering how to use the vue-router extension to flow from one page to another giving our application a workflow concept. The code for this tutorial can be found at my [GitHub](https://github.com/amcquistan/flask-vuejs-survey/tree/FirstPost) account which contains a branch for each post. Thanks for reading and please feel free to comment or critique below.