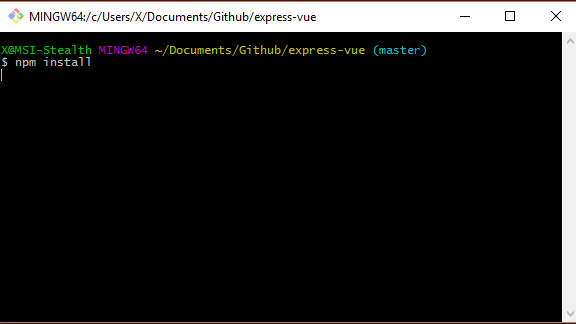
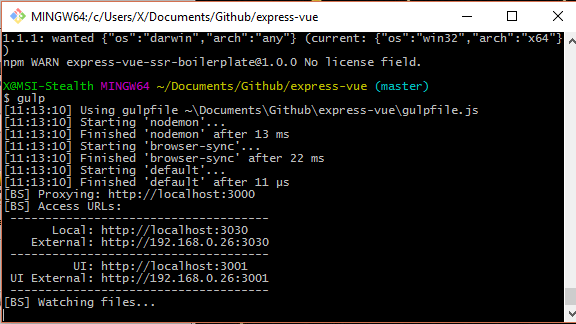
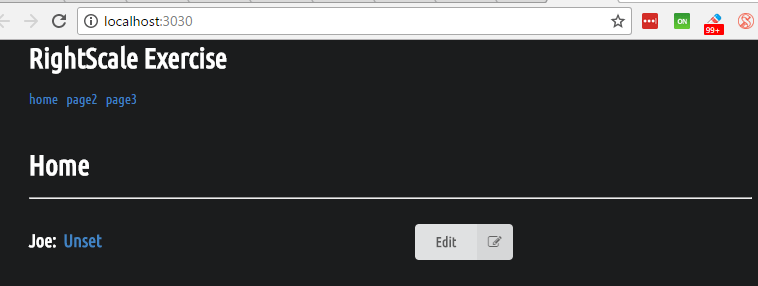
**REQUIRED:**  
- Node Package Manager (npm)  
- Gulp (optional)

**HOW TO INSTALL AND RUN:**  
$ npm install  
This will grab all the required files needed to start and run a server.   
  
Next:  
$ gulp (should automatically open a browser and set the address to localhost:3030)  
OR:  
$ npm start (if you don’t have Gulp installed)



If your browser doesn’t automatically open, open a browser of your choice and enter into the url:

localhost:3030



If node is running, then your browser should look like the one above.

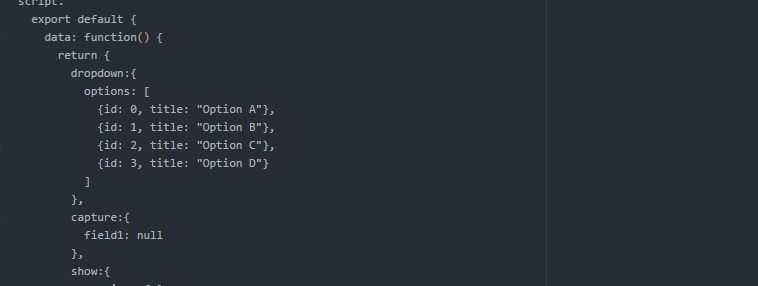
**INTRODUCTION:**

Okay, now for the fun part. Let me explain my reasoning and methodology. But first, hello! My name is Allen and I’m applying for a senior web development position at your company RightScale. I got your instructions for this exercise and to complete it, I choose a MVC framework called VueJS (<https://vuejs.org/>). It’s not as well recognized as some of the other MVC frameworks (namely Angular 1& 2, Ember, etc) or libraries such as React, but in terms of functionality and conception, it’s actually if Angular and React had a baby, except without all the hassle of Angular 2’s setup and bloat and the somewhat odd syntax for React.   
  
The initial seed project I used to bootstrap this project can be found here: <https://github.com/express-vue/express-vue> (I’m the original and sole author so it shouldn’t count as cheating or anything).  
  
Anyways, let me breakdown what you’re seeing when you start.   
  
So right off the back I’m using Express to start and serve the website; pretty basic stuff which you can find in the **root -> app.js** file. I coded it for SEO and other functional stuff, but I’ll skip the fanciful thing and get right to the exercise. Also I’m using Semantic UI for styling (<https://semantic-ui.com/>).

**MISSION STATEMENT:**  
  
So the goal here was to build a component and make it reusable for other parts of the site. The component itself can be found in the development files **pug -> components -> datacapture.pug.** Once a component is constructed, it can be used anywhere on the site by simply referring to its HTML element, in this case <datacapture>. You can see that I added the component to all three test pages (home/page2/page3) located in **pug -> pages -> [file.pug].**   
  
Looking at the component itself (**dist -> components -> datacapture.vue**) you can see that the .vue file is somewhat similar to a React .jsx file, namely that it’s broken down into their HTML, JS and CSS elements respectively.

**THE DATACAPTURE COMPONENT:**

The most vital part to web development, at least to me, is to get the correct behaviour down first and do it in the most concise way possible while at the same time making it easy to read and easy to edit. In that spirit, nothing is hardcoded into the HTML; all the important data can be altered by changing a simple property. In this example, the dropdown content can be altered to include any additional information, but for sake of simplicity, I left it as simply ID and title.



When a user selects something from the dropdown, those properties are copied onto the **capture.field1** property. When the field is empty (null), the HTML is coded to behave a certain way – in this instance, to hide or show certain buttons. This is a simple example of *conditional data-binding*, which is a fantastic way to reduce complexity while increasing functionality. (It’s incredibly   
similar to Angular 1.x ng-if/show except here it’s v-if/show).   
  
If a user has selected something from the dropdown and clicks Save, then it calls the saveData function located in the Methods section. I added a very simple validation check to ensure the user must select something to move forward. Next, it will gather all the required data into a package before stringifying it and sending out to be processed, typically via a RESTful API. (Right now, I’ve replaced the AJAX request with a simple mock timeout since, after all, there’s no endpoint to hit). If you open up the browser’s console log, you can see a quick preview of the information that’s being sent. Once it’s done that, it changes the value that controls whether you see a dropdown or not back to its default state.

**CONCLUSION:**

So for this exercise, I believe I hit all the main highlights:

* The component is recyclable and can be added to other pages/sections of the website by simply inserting <datacapture> into the HTML where it’s needed.
* The component itself behaves the way the instructions detail.
* It’s using a library that functions similar to React while syntactically similar to Angular. If I needed to code this using other frameworks, I would still employ the same methodology.

Please feel free to ask me if you have any additional questions or you want me to explain something more in-depth. Thanks for this opportunity and I hope to speak/code for you again!  
  
-Allen Royston