How to Run Our Code

Group Number:101

Team: Jacob Feldman and Andrew Mumm

To run our code, you'll need to set up a Laravel Homestead environment. Once you've done that, you could unzip our project and put it into your Homestead projects directory; or, what we will probably do is demo it to you. Once you've set your environment up and started up the Vagrant server, go to homestead.app in a web browser. You'll see a blank page with a toolbar and tabs at the top. Click the plus button on the top right to add a 'Course.' Once you've added a Course or two, you can add tasks to each one. Each task has a weight (for the example of Courses and tasks, the weight could be in hours), a name, and a checkbox next to it to indicate if that task has been completed or not. Try creating a few Courses with a few tasks each, and marking a few as complete or not. Once you've done that, you can look at the Graphs tab to see a graphical representation of how close you are to completing a Course; a progress bar, if you will. Also, if you click the button in the very top left, a 'drawer' will pop out from the left that shows all the courses you've added. This isn't that useful, we just thought it was cool.

The project backend is a Laravel PHP server. We built the front end using Material-UI, a Library of ReactJS components that follow Google's material design principles. The graphs are from a JavaScript library called Chart.js.

Check List

The first place to look in our code is our index.blade.php. file inside of the laravel/resources/views folder. This file isn't complex, it's just the place where all the content is eventually displayed and then returned to the user. The .blade just means that Laravel will run the Blade templating engine on it, which adds a bit of functionality to minimalize duplication of code. We don't use much of that functionality, since it is more useful for websites with an assortment of webpages and ours is a single page application, by design.

The next place to look, which actually contains complexity, is the App.jsx file in the laravel/resources/assets/js folder. This is a ReactJS file in which we create and render our 'App' class. The App class contains everything in our app; the Appbar, the Tabs and all their contents, the Drawer, and all the functions for interacting with those elements. The App class also dynamically renders a new Course and a new canvas (for the graph to be drawn onto) every time you add one. The Course component is defined in a different file inside of Course.jsx in the laravel/resources/assets/js/components folder though.

Course.jsx creates and exports the Course component. Each course component has an array of task objects (a simple inner class defined in the same file) that stores the name, weight, is_completed, and color fields of each task in the course. Course.jsx also handles writing(and rewriting) the graphs onto the canvases each time the Course's states are updated (this is all contained in the renderChart function of the Course class). The render method of this class dynamically renders the tasks in a table based on the contents of the arrays contained in the state field of the class.

Since both files are jsx files, we had to use a combination of gulp and laravel-elixir to compile them down to regular, old JavaScript files that are renderable by modern browsers. We struggled to find and configure the correct tools to accomplish this and place the compiled files into the right folders, but once set up, compiling itself was as simple running gulp on the command line. Meshing ReactJS and Material-UI with Laravel was one of our biggest issues. Other major issues we had include making the charts from Chart.js render dynamically and setting up our Laravel environment.

Title Page

COMS 319

Group 101

Andrew Mumm

Jacob Feldman

We created a Laravel web project with Material-UI and Chart-JS as the front end providing the user with a simple tool for keeping track of progress on tasks of their creation.

Table of Contents

Overview

For this portfolio, we decided to learn a new framework for the frontend and backend. For our frontend we decided to learn the Material-UI library, which is a library of components built on ReactJS. It was made by Google in accordance with their material design standards. To use ReactJS we had to use jsx files, then compile them to plain JavaScript using a mixture of Gulp, Laravel-elixir, and Browserify. Since we were already using jsx files, we decided to use some of the new functionality in ES6, such as arrow function notation. For our backend, we used the Laravel PHP framework; we also used Laravel Homestead to boot up a server on a virtual machine to test our application.

We had to use a back-end to host our web application and install a development environment. We learned about front-end frameworks like React-JS and Material-UI. We decided to use these frameworks among many others because of the dynamic features that they offered. We wanted to make a single page web app because we learned that this is the current trend of websites. We applied what we have learned about these frameworks into making a dynamic single page application that is user friendly. Overall if we had to make the website again, or one that uses similar tools, we would be able to do so very quickly. Once we got passed the learning curve for these tools, development became easy. In conclusion, these tools fit the purpose of our project very well and provided us with a great new skillset to use in the future.

Creation: We created a dynamic single page web application with modern front-end frameworks that we have never used before.

Evaluation: There was a pretty steep learning curve with the frameworks that we used, but after learning how to use them we believe they are very quick and effective ways of building a modern website.

Analysis: The current trend of web apps is to make them as dynamic as possible and to try to include all of the content on a single. To do so a developer may have to learn to use new tools to make sure that all of the pieces of the project, front-end and back-end, work together properly.

New and Complex Section

For this project we decided to learn and use a new framework for both the backend and frontend. For the backend, we used the Laravel PHP framework. We learned a lot about Laravel in this project, and we may use it again in the next portfolio since we liked it so much. We used Laravel to create a web server on a virtual machine (and, later, the server that the class provides). While Laravel is a framework that is designed to simplify almost every aspect of building a it still requires a good amount of set up to use it. To simplify this setup, we used Laravel Homestead, which helps you create a Vagrant server on a virtual machine on your computer. Again, while Homestead strives to make development easier, it still requires setup to use.

For the frontend, we decided to learn Material-UI, a recently released library of components built by Google to implement their material design principles. To learn and use Material-UI, we had to first learn how to use ReactJS, the JavaScript library that Material-UI is built on. Learning how to use ReactJS and the syntax it uses to do things like import components from node modules or export custom components to be used in other files was another thing we struggled with when building this project. Once we got the hang of it, though, it became easy to see that React is a great way to build single-page, interactive web-applications.

One major problem we had several times with Material-UI is that, since it is such a new library, it still has bugs in it. Like, for instance, the OnClick method of some components doesn't work correctly or, sometimes, at all. There also isn't a huge community of developers working in it currently, so finding other instances of your specific problem can often be difficult.

To use ReactJS, we had to learn to compile jsx files into plain JavaScript files, since modern web browsers haven't been updated to understand jsx files yet. To do this compilation, or more specifically transcompilation, we used a combination of gulp, laravel-elixir, and browserify; this was one area we struggled to set up and use. Since we were already using jsx files, we also learned a little bit about ES6, and used some of its functionality, like arrow function notation (which is great, by the way).

Lastly, we also learned how to use the Chart.js library; this library is another cool JavaScript library that lets you render a variety of sharp-looking graphs onto html canvas elements. I'd like to use this library again in the future as well. From this library, we used the donut chart. One major problem we had when using these charts is that they are not "interactive" by default. To re-render a graph, you first must delete the canvas it's currently on, create a new one, and then render the updated data onto the new canvas. And when, when you're dynamically creating graphs, this leads to increasing levels of complexity. Luckily, ReactJS helps a little with that in that it is designed to 'force' you to modularize your code by making components. It also 'forces' you to separate your JavaScript into an MVC pattern, with the state variables of each component as the Model, the render method as the view, and the functions inside each class as the Controllers.

Bloom's Taxonomy

Analyzing:

We created a Laravel project to host our website. This was the first time we have used Laravel to house a project. Laravel is a php framework used to help tie the frontend and backend of a web project together. Laravel can supply you with a database, a local server, and many packages that you can use within your project for compiling and adding features to your code. When we were developing the project, we used Laravel development environment called Homestead. Homestead provides you with a virtual Linux environment that maps the location of your project to the Linux environment where it hosts the project. You are then required to configure a file called homestead.yaml to let the Linux vm know where to look for the project and what the hostname for the project will be when you launch the browser. When we have the virtual machine running all we have to do to test the project was to go to the browser and type homestead.app. This was the name of the project in the Laravel directory. When this is typed, the website is loaded and ready to use.

For the front-end part of the project we decided to learn something new as well. We used a framework that is made by Google that is called Materials-UI. Materials-UI is built off of another framework called React-JS. React is a way of writing code so that elements of a website are kept up to date with each other by constantly comparing its current state to its previous state. If the state changes then the website will update the element and the user will see the current state displayed. Materials-UI contains a bunch of elements and styles for writing code in React. React uses a newer version of JavaScript and in order to use a lot of the nice features of react you need to use newer aspects of Javascript that not all browsers are capable of compiling yet. One such feature is the use of import statements within a JavaScript file. This is a syntax that is introduced in ES6. ES6 is the new version of Javascript. In order to take on this project we had to learn a bit about ES6 and some new syntax. When writing React code the file extension needs to be changed from .js to .jsx.

Using the newer syntax and newer version of JavaScript presented us with an obstacle. Due to the fact that ES6 is not currently supported by all browsers, we needed to find a way to compile the code before it entered the browser. To do this we installed a package into our Laravel project called gulp and browserify. These tools act as a transpiler that converts jsx code into js code that the browser can handle. After installing these tools, we just needed to tell our gulp file what files it need to look for and then when in the project folder we have to run "gulp" from the command line. After this the transpiled code would be added to the appropriate folders so that the project can be opened from within the browser.

We also used another Javascript framework called Chart-JS. This was used to help display some of the data on our website in a visually pleasing format for the user.

In summary, we used a php framework called Laravel as the backend. We use a developing environment called Laravel Homestead that provides a Linux vm for running the project. We use Materials-UI for the front end which is a framework built on the React-JS framework. React-JS uses the newest version of Javascript which is ES6. In order to compile this code, we use the transpiling tools, gulp and browserify. Additionally we npm installed the Chart-JS framework to use in our project.

With all of these pieces working together we created a dynamic single page web project that we called Progress Bar. With this web app users are able to create tasks and add a weight to them. As tasks are completed the user can check them off and view their progress in the form of a circular chart that compares the weights of completed tasks to the weights of incomplete tasks.

Evaluating:

We learned a lot about newer styles of web apps that are starting to appear. We looked at several different front-ends before we decided to settle on Material-UI and Chart-JS. The current trend for web applications is having it be a single page. This is because if done right it can look much better for the user and you don't have to worry about load times for content. Once the page has been loaded it's finished. You don't have to load a new page when a link is clicked or anything like that because everything the user wants to see is already somewhere on the page. Some ways of keeping the website organized is by creating tabs that the user can click in order to hide and show different content. This is something that we did with our project. The user is able to create "Tasks" on one tab and on another tab the user can view the completion of their tasks. A challenge that comes along with having a single page web app is that the size of a file can become too large to manage if all of the content is contained within a single file. With newer frameworks and transpiling tools, this problem is solved. With the newest version of JavaScript, you can include any dependencies for libraries or other files that a particular file may need within the same file. For example, if file A needs to use a library as well as files B and C then you can simply import B and C and the library at the top of the page. When you do this for every file you will take care of the problem of dependencies but you still will have a bunch of files to work with. Now you can create another file say, app.js, that imports all of the files you have written for your project. When you use tools like gulp and browserify all you will need to do is compile one file, app.js, and it will build a new file for you that contains every single dependency from all of the files that app.js imports. This way you can use the compiled version of app.js in your project. This allows you to use a single file for your web project.

Once you get the hang of React and some of the tools necessary to use it, it is very efficient and it can help you make some cool web apps. Laravel was a great way to host this particular web app and I will probably use it again in the future.

Creating:

The first art of the project that we worked on was the back-end. We wanted to make sure that we could effectively host our web app before we worked on any off the main functionality. We installed Laravel and worked on getting all of the configurations set up for our use. To make sure that the back-end was ready we needed to get our Homestead development environment up and running as well. Once we got the virtual machine running and all of the directories mapped to the correct location on our machine we were able to go to the browser and load a basic Laravel welcome page.

When we got to this point we knew that we would be able to successfully host our project with Laravel.

The next thing that we did was start researching Material-UI and React-JS. There was a pretty steep learning curve at first with each of these tools. Fortunately there is plenty of well written documentation on the subjects. React takes dynamic web projects to a new level. We learned that it bases what it shows the user entirely off of states. If the state has changed internally from the last time that it checked then it will immediately update what is shown to the user. We found out that these tools use a newer form a Javascript and using this would open up a lot of neat features that we could use such as, import statements in Javascript. While using new tools allowed us to do some pretty cool things with our project, it also presented more work for us.

When using newer syntax and features browsers no longer know how to compile the code. As a result, we discovered that you have to use a transpiler such as gulp in order to change your code into a form that the browser can handle. We learned how to import gulp and browserify into our Laravel project. From there we were able to compile our code right into the project to that it could be used by the browser.

The last thing that we learned about is Chart-JS. Chart-JS is a Javascript framework that is used for displaying data in a dynamic and visually appealing way. This was the perfect framework to use with our project because we wanted the user to be able to see their progress completion with various activities they may be working on. Chart-JS was easily imported into Laravel where it could then be used seamlessly in our project.

From start to finish we created the back-end portion of our website, then we got a development environment ready so we could test, next we learned how to use React and Material-UI for the front-end. After that we used gulp to bridge the gap between our code and the browser. Lastly, we added Chart-JS to give our website something extra special.