Utilizing ReactJS for Page Generation

Setup

It is expected that you have previously downloaded and installed Node.js as well as the npm pac kage manager on your computer.

Make a directory called project4 and then extract the contents of the project4 file into it. The nec essary beginning files for this project are included in the zipped download.

This assignment calls for the use of a number of different node modules that come equipped with the instruments (such as Webpack, Babel, and ESLint) required to construct a ReactJS web appli cation. Additionally, the use of a straightforward Node.js web server (ExpressJS) is required in o rder to deliver the application to your web browser. In order to get these modules, you will need t o execute the following command inside the project4 directory:

```
npm install
```

This command will retrieve around 600 node modules into the subfolder node modules, which w ill take up approximately 100 MB of space.

It is possible for us to utilise npm to execute the numerous tools that we had it retrieve for us. Th e following execute commands are accessible, as is shown by the "scripts" attribute of the packag e.json file:

• npm run lint runs ESLint on each and every JavaScript file associated with the project. It is exp ected that the code you provide will pass ESLint without any warnings.

• npm run build packages all of the project's JSX files into a single JavaScript bundle and places i t in the directory named compiled. This command is executed by Webpack using the configurati on file webpack.config.js as its source.

• npm run build:w - Runs Webpack in the same manner as the "run build" command, except that it calls webpack with the —watch flag, which causes it to monitor the React components and reb uild the bundle if any of them are modified. This option is helpful for development since it enabl es changes made to components to be picked up by simply refreshing the browser to load the fres hly updated bundle. Changes made to components can then be used in the application. In that cas e, you will need to make a mental note to always remember to "npm run build" after making any changes. There is a possibility that you may get a deprecation warning labelled DEP WEBPACK WATCH WITHOUT CALLBACK. You are free to disregard this message.

The project4 directory is where you should put into action the solutions you came up with for all of the issues listed below.

This project makes use of the widely used framework for developing web apps known as ReactJ S. The purpose of this project is to get you up to speed with ReactJS and the coding principles co vered in CS142 to the point that you will be able to use it to construct a web application for the s ubsequent project.

For the purpose of retrieving our online application over the HTTP protocol, we make use of a ba sic Node.js web server, which can be activated by issuing the following command from inside th e project4 directory:`node webServer.js`

An URL that begins with http://localhost:3000 may be used to get any and all of the files include d in the project4. To determine whether or not your web server is active, go to http://localhost:30 00. Your web browser should get the index.html file from the server.

When you are testing and creating your project, it is recommended that you arrange your develop ment environment to run webpack in watch mode. This will need you to run the node webserver i n addition to webpack. It is possible to do this by running the apps in separate command line win dows. Babel is responsible for finding and reporting syntax mistakes, which enables webpack's o utput to be helpful. If you are using an operating system like MacOS that has a shell that is simila r to unix. The instruction:

```
node webServer.js & npm run build:w
```

operates the web server in the background while simultaneously running the webpack in the foref ront, all inside the same window.

Using these two Windows commands, you may start the web server in the background on Windo ws, while also starting webpack in the foreground.

```
start /B node webServer.js
npm run build:w
```

You can stop the background webserver with the command:

```
taskkill /IM node.exe /F
```

Getting Started

We demand that you apply the model, view, and controller pattern that was discussed in class for this particular project. Because there are many different ways to organise code using this pattern, we will present an example that not only displays some fundamental ReactJS capabilities but also demonstrates the file system structure and the module pattern that we would want you to use in the projects that you work on.

The first thing you need to do is access the example in your browser by going to the URL http://localhost:3000/getting-started.html. This is where you should start. This website provides illustrative examples on how to use ReactJS. A div is provided in the getting-started.html file so that ReactJS may draw the application into it, and a script tag is used to include the app's JavaScript bundle, which is located in the directory named compiled/gettingStarted.bundle.js. This bundle is directed to be generated from the ReactJS code entitled gettingStarted.jsx by the webpack configuration file named webpack.config.js. gettingStarted.jsx is a JSX programme that renders the ReactJS component named Example into the div located in the getting-started.html file.

We use a file structure that co-locates the ReactJS component and its related CSS stylesheet in a subfolder of a directory that is called components in order to facilitate reusable components. This directory is named components. The Example component may be found in the components/example folder, which contains the files Example.jsx and Example.css.

You should take a look at the files that are invoked in the getting-started.html view (getting-started.html, gettingStarted.jsx, components/example/Example.jsx) because they display the JavaScript and JSX statements that are required to run a ReactJS web application along with comments that explain their purpose. You need to make sure that the additional components that you construct for the class follow this pattern and naming convention for the files.

In most cases, the data for the model is obtained from the webserver, which in turn obtains the data from a database. We will provide you with an HTML script element that will allow you to load the model data straight into the DOM of the browser from the local file system so that you may avoid having to set up a database for this project. Under the property name cs142models, the models will be shown in the Document Object Model (DOM). It will be accessible to you in a ReactJS component under the name window.cs142models when you use that component.

> ## The first challenge is to comprehend and modify the sample view
>
> It is recommended that you go through the getting-started.html view and the Example component in order to get an understanding of both. Do the following to show that you comprehend what is being asked of you:
>
> 1.
>
> Replace "Unknown name" with your own name in the model data for the Example component, and be sure to save the changes. You need to look for "Unknown name" and then change it with anything else.
>
> 2.
>
> Replace the content of the div region in the Example component that has the class motto-update with some JSX statements that show your name and a brief motto (up to 20 characters in length), and save the changes. In the same way that the user's name should come in with the model data, the initial value for motto should as well. You are going to need to add some style to Example.css in order for this display to work properly.
>
> 3.

> Extend the display that you created in the previous step so that it gives the user the ability to change the slogan that is being shown. It is recommended that the default value be obtained from the model data in the future.

Create a brand new component called the states view (Problem 2)

Make a new component view that, when loaded, will show the names of all the states that have a certain substring that you provide. In order for your view to function properly, it has to provide an input field that can take substrings. A list of all of the states whose names include the specified substring will be shown in alphabetical order when the view is activated (ignoring differences in case). As an instance, the view for the substring "al" ought should list the states of Alabama, Alaska, and California. Additionally, the website need to indicate the substring that was used in the state filtering process. In the event that there are no states that meet your criteria, the website need to provide a notification to reflect this reality (rather than just showing nothing). When the substring is empty, it is expected that all states will be presented.

In the same vein as Problem #1, we will provide you with the model data along with the states. After it has been included with: window.cs142models.states is the address you may use to access it.

```
<script src="modelData/states.js"></script>
```

> For a discussion of the structure of the states data, please refer to the states.js file.
>
> You may use the file p2.html that we supplied for you as a starting point and as a reference to the file name conventions that we want you to follow. This file will load and show the bundle compiled/p2.bundle.js that was built by webpack from p2.jsx, and it will display the React component States. This document may be accessed in your web browser by entering the URL http://localhost:3000/p2.html.
>
> The following is a list of the files that you will need to implement:
>
> • components/states/States.jsx is the file that contains your ReactJS component for the states component.
>
> • components/states/States.css - This file contains any CSS styles that are required by your component. You are need to incorporate some kind of formatting for the list of states here.

Personalization of the Layout (Problem No. 3)

> Develop a ReactJS component that you'll call Header and assign it the task of displaying a customised header at the very top of a view. Include this header in all of the ReactJS web applications that are part of your assignment (gettingStarted.jsx, p2.jsx, p4.jsx, p5.jsx). It is important to keep in mind that you should not change the portion from the first segment (your name and motto). That section should not be combined with your header in any way. Create a

header that is "uniquely you" by drawing on your imagination and other creative abilities. This may be more photographs, graphics, or whatever else you like to include. You are free to enhance the JSX and JavaScript that are already included in the components, but you are not allowed to add any other ReactJS Components or JavaScript libraries like JQuery. Use your imagination!

The following is a list of the files that you will need to implement:

The ReactJS Component of your header component may be found in the components/header/Header.jsx file. This is a class Header that is a React.Component in its defining definition.

• components/header/Header.css - This file contains any CSS styles that are required by your component. You are need to specify some formatting for your header in this section.

Note that the customised header from Problem 3 should be placed at the top of the gettingStarted.jsx file, and that the section containing the motto from Problem 1.2 should be placed just below it. Your unique header from Problem 3 should be included on all of the other page views, including p2.html, p4.html, and p5.html.

The fourth challenge is to provide dynamic switching of the views

Make a p4.html file as well as a related p4.jsx file that contains both view components (the Example and States components). A capability to toggle between the displays of the two components has to be included into the p4.jsx file. A button that, when clicked, would convert the currently shown view to the alternative view and vice versa. This button would appear above the currently seen view. For instance, while the States view is active, the button just above it should say "Switch to Example," and when the button is pressed, the States view should vanish and be replaced with the Example view.

For this particular issue, you will need to generate the aforementioned files in addition to modifying the webpack configuration file webpack.config.js in order to generate the file compiled/p4.bundle.js, which you will then be able to utilise in the p4.html file. After making changes to code>webpack.config.js, you will need to restart Webpack if you are running it with the —watch option (that is, if you are using npm run build:w).