# Project 4: Page Generation with React

Due: Thursday, Oct 17th 11:59 PM

## Setup

Although this project has you run code in your browser, you need to have Node.js installed on your system to run the code quality checker. If you haven't already installed Node.js and the npm package manager, follow the installation instructions (https://nodejs.org/en/learn/getting-started/how-to-install-nodejs) now.

Once you have Node.js installed, Create a directory `project4` and extract the contents of this zip file (./p4_code.zip) into the directory. The zip file contains the starter files for this assignment.

This assignment requires many node modules that contain the tools (e.g. Webpack (https://webpack.js.org/), Babel (https://babeljs.io/), ESLint (https://eslint.org/)) needed to build a React (https://react.dev/) web application as well as a simple Node.js web server (ExpressJS (http://expressjs.com/)) to serve it to your browser. These modules can be fetched by running the following command in the `project4` directory:

```
npm install
```

That command will fetch around 600 node modules using around 100 megabytes of space into the subdirectory `node_modules`.

We can use npm to run the various tools we had it fetch. As can be seen in the `"scripts"` property of the `package.json` file, the following run commands are available:

- `npm run build` - Runs Webpack (https://webpack.js.org/) using the configuration file `webpack.config.js` to package all of the project's JSX files into a single JavaScript bundle in the directory `compiled`.
- `npm run build:w` - Runs Webpack (https://webpack.js.org/) like the `npm run build` command except it invokes webpack with --watch (https://webpack.js.org/api/cli/#watch), so it will monitor the React components and regenerate the bundle if any of them change. This option is useful for development so changes made to components can be picked up by simply refreshing the browser to load the newly updated bundle. Otherwise, you would need to remember to run `npm run build` after every change. You might get a deprecation warning `[DEP_WEBPACK_WATCH_WITHOUT_CALLBACK]` that you can safely ignore.
- `npm run lint` - Runs ESLint on all the project's JavaScript files. The code you submit should run ESLint without warnings.

Your solutions for all of the problems below should be implemented in the `project4` directory.

This project uses React (https://react.dev/), a popular framework for building web applications. The project's goal is to get you enough up to speed with React so that you will be able to build a web application with it in the next project.

In order to fetch our web app via the HTTP protocol, we use a simple Node.js web server that can be started with this command from the `project4` directory:

```
node webServer.js
```

All the files in the `project4` can be fetched using an URL starting with http://localhost:3000 (http://localhost:3000). Click on http://localhost:3000 (http://localhost:3000) to verify your web server is running. It should serve the file `index.html` to your browser.

We recommend you configure your development environment to run webpack in watch mode, which means you will need to run the node webserver ( `node webServer.js` ) and webpack ( `npm run build:w` ) when building and testing your project. You could do this by running the programs in different command line tabs or windows. Syntax errors get detected and reported by Babel, so the output of webpack is useful.

## Getting Started

In this project we require that you use the model, view, controller pattern described in class. There are many ways of organizing code under this pattern so we provide an example that both demonstrates some basic React features as well as showing the file system layout and module pattern we would like you to follow in your projects.

You should start by opening the example in your browser by navigating to the URL http://localhost:3000/getting-started.html (http://localhost:3000/getting-started.html). The page displays examples of React in action. The HTML in `getting-started.html` provides a `<div>` for React to draw the app into and a script tag include the app's JavaScript bundle `compiled/gettingStarted.bundle.js` . The webpack config file `webpack.config.js (http://localhost:3000/webpack.config.js)` directs that this bundle be created from the React file `gettingStarted.jsx` , a JSX program that renders the React component named `Example` into the `<div>` in `getting-started.html` .

To support reusable components, we adopt a file organization that co-locates the React component and its associated CSS stylesheet in a subdirectory of a directory named `components` . The `Example` component is located in the files `components/Example/{index.jsx,styles.css}` .

You should look through the files invoked in the `getting-started.html` view ( `getting-started.html` , `gettingStarted.jsx` , `components/Example/{index.jsx}` ) since it shows the JavaScript and JSX statements needed to run an React web application along with explanatory comments. You should use this pattern and file naming convention for the other components you build for the class.

Model data is typically fetched from the webserver which retrieves the data from a database. To avoid having to set up a database for this project we will give you an HTML script tag to load the model data directly into the browser's DOM from the local file system. The models will appear in the DOM under the property name `models` . You will be able to access it under the name `window.models` in a React component.

## Problem 1: Understand and update the example view (5 points)

You should look through and understand the `getting-started.html` view and the `Example` component. To demonstrate your understanding do the following:

1. Update the model data for the Example component to use your name rather than "Unknown name". You should find where "Unknown name" is and replace it.
2. Replace the contents of the `div` region with the class `motto-update` in the Example component with some JSX statements that display your name and a short (up to 20 characters) motto. Like the user's name, the initial value for motto should come in with the model data. You must include some styling for this display in `styles.css` .
3. Extend the display you did in the previous step so it allows the user to update the motto being displayed. The default value should continue to be retrieved from the model data.

# Problem 2: Create a new component - states view (10 points)

Create a new component view that will display the names of all states containing a given substring. Your view must implement an input field that accepts a substring. The view will display in alphabetical order an HTML **list** of all states whose names contain the given substring (ignoring differences in case). For example, the view for the substring of "al" should list the states Alabama, Alaska, and California. The list should automatically update after each character that the user types in. The page should also display the substring that was used to filter the states in a new element separate from the input field. If there are no matching states then the web page should display a message indicating that fact (rather than just showing nothing). All states should be displayed when the substring is empty.

As in Problem 1 we provide you the model data with states. It can be accessed via `window.models.states` after it is included with:

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

See `states.js` for a description of the format of the states data.

To help you get started and guide you to the file naming conventions we want you to use we provided a file `problem2.html` that will load and display the bundle `compiled/problem2.bundle.js` which is generated by webpack from `problem2.jsx` which displays the React component `States`. You can open this file in your browser via the URL http://localhost:3000/problem2.html (http://localhost:3000/problem2.html).

The files you will need to implement are:

- `components/States/index.jsx` - The React Component of your states component.
- `components/States/styles.css` - Any CSS styles your component needs. **You must include some styling for your state list here.**

# Problem 3: Personalizing the Layout (5 points)

Create a React component named `Header` that will display a personalized header at the top of a view. Add this header to all React web apps in your assignment ( `gettingStarted.jsx` , `problem2.jsx` , `problem4.jsx` , `problem5.jsx` ). Note that you **should not** replace the section from part 1 (your name and motto). That section should be separate from your header. Use your imagination and creativity to create a header that is "uniquely you". This can include additional images, graphics, whatever you like. You can extend the JSX/JavaScript in the components but you may not use external React Components or JavaScript libraries such as JQuery. Be creative! A colored rectangle with plain text is not sufficient.

The files you will need to implement are:

- `components/Header/index.jsx` - The React Component of your header component.
- `components/Header/styles.css` - Any CSS styles your component needs. **You must include some styling for your header here.**

# Problem 4: Add dynamic switching of the views (10 points)

Create a `problem4.html` and a corresponding JSX file `problem4.jsx` that includes both view components (the `Example` and `States` components). The `problem4.jsx` needs to implement an ability to switch between the display of the two components. When a view is displayed there should be a button above it that switches to display the other view. For example, when the `States` view is displayed the button above it should read "Switch to Example," and when pushed the `States` should disappear and the `Example` view should be displayed.

For this problem you will need to create the files above as well as modify the webpack configuration file `webpack.config.js` to build a file `compiled/problem4.bundle.js` that you can uses in `problem4.html` file. Note that if you are using Webpack with the `--watch` flag (i.e. `npm run build:w`), you will need to restart it after changing `webpack.config.js`.

## Problem 5: Single page app (5 points)

Although the approach taken in Problem 4 allows you to switch between the two views, it does not allow you to bookmark or share a URL pointing at a particular view. Even doing a browser refresh event causes the app to lose track of which view was being displayed.

We can address this deficiency by storing the view information into the URL. React Router (https://v5.reactrouter.com/) provides this functionality for React. For this problem make a copy of your `problem4.html` solution into a file named `problem5.html` and copy your `problem4.jsx` into a file named `problem5.jsx`. Convert the code to use React Router (https://v5.reactrouter.com/) to switch between the two component views. You should have a **styled toolbar-like control** (simple plain text links are not sufficient) that will allow the user to switch between the example and states component views.

Since this is the first extension from the core React we import, we're providing you with step-by-step instructions.

1. The project's `package.json` specifies react-router so the npm install command already fetched it for us. We do need to explicitly import it into our `problem5.jsx` file. Add the following import line:

   ```
   import { HashRouter, Route, Link } from "react-router-dom";
   ```

   The line uses the JavaScript import (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import) statement to bring in the React components from React Router: `HashRouter (https://v5.reactrouter.com/web/api/HashRouter)`, `Route (https://v5.reactrouter.com/web/api/Route)`, and `Link (https://v5.reactrouter.com/web/api/Link)`. The `HashRouter` module of React Router uses the fragment portion of the URL for storing information. So we can make `problem5.html#/states` mark showing the States view while `problem5.html#/example` specifies the Example component view.
2. The most common way of using React Router is to conditionally render the view we want based on the current URL. The `Route (https://v5.reactrouter.com/web/api/Route)` component implements this conditional rendering when placed inside a `HashRouter` element like:

   ```
   <HashRouter>
     ...
     <Route path="/states" component={States} />
     <Route path="/example" component={Example} />
     ...
   </HashRouter>
   ```

   which would render the States component if the URL had `#/states` and the Example component if the URL had `#/example`.
3. Although we could use hyperlinks (i.e. `<a>` tags) to switch views, react-router recommends using the `Link` component to generate the hyperlinks. For example:

   ```
   <Link to="/states">States</Link>
   ```

   generates a hyperlink with `href="#/states"` and the text "states".

For this problem again you will need to create the files above as well as modify the webpack configuration file `webpack.config.js` to build a file `compiled/problem5.bundle.js` that you can uses in `problem5.html` file. Note that if you are using Webpack with the `--watch` flag (i.e. `npm run build:w`), you will need to restart it after changing `webpack.config.js`.

## Style (5 points)

These points will be awarded if your solutions have proper MVC decomposition. In addition, your code and templates must be clean and readable. Remember to run ESLint before submitting. ESLint should raise no errors.

## Submission Procedure

Create a folder with your netid as the name. This directory should include the starter files we gave you along with code files you created or modified for the problems including `problem2.jsx`, `problem2.html` `problem4.jsx`, `problem4.html`, `problem5.html` and `problem5.jsx`. Make sure your submission runs `npm run lint` with no errors. Do NOT include the `node_modules` and `compiled` folders inside your submission folder. Zip the folder and submit it on elearning. Note the folder should not contain any unrelated files, and name of the folder should be exactly as should as specified. Failing to follow the submission procedure will result in loss of points.