Setting Up the UI Toolkit

In **"Project Planning and Setup"**, we sketched out a feature list and some UI mockups for ***Project Mini-Mek***, created a new project using Create-React-App, added a basic Redux configuration, and enabled the use of Hot Module Replacement so that we could see changes in our UI without having to reload the entire page. In this part, **we'll set up the initial application layout to match our UI mockups, talk about folder structures, and look at an example of managing UI state with Redux**.

Choosing a UI Toolkit

There's a seemingly infinite number of web UI toolkits and CSS frameworks out there, of which Bootstrap and its variations are probably the most popular. I've been using **Semantic-UI** in another project, and been very happy with it. It has a nice appearance out of the box, uses very readable markup, and allows considerable customization for theming, including several built-in theme options for various parts such as buttons.

Semantic-UI consists of two main aspects: CSS-only content for styling, and logic for smarter widgets (such as AJAX fetching capabilities for the Dropdown component). The original framework uses jQuery to implement its smart widgets.

You *can* use Semantic-UI in a React app without any additional tools or libraries. You only need to include its CSS file in your page, and render markup like `<div className="ui label">I'm a Semantic-UI label!</div>` to have the styles applied correctly. However, as React developers, it's nice to be able to think of things in terms of encapsulated components rather than lower-level markup. In addition, most React apps don't have a need for jQuery at all, and so dragging it in just to provide interactivity for some of SUI's widgets is something that we'd like to avoid.

Happily, in addition to the core Semantic-UI project, there's also the official **Semantic-UI-React** library. It provides React components that correspond to all of the different UI pieces provided by Semantic-UI. SUI-React's components accept props like any other React component, and take care of rendering the correct markup and classnames as needed. In addition, all of the interactive widgets have been rewritten to use just React and plain JS, eliminating the need for jQuery completely. So, **we'll be using Semantic-UI-React for the UI layout and styling**.

It's important to note that Semantic-UI-React is only about generating the correct markup and adding additional logic on top. In order for that markup to change any appearances, we need to include Semantic-UI's CSS. Semantic-UI has its own build system that can be used to generate a customized theme, but to simplify things, we're just going to use the **semantic-ui-css** package, which provides a prebuilt version of the full Semantic-UI CSS output.

Setting Up Semantic-UI

Our first step is to add `semantic-ui-react` and `semantic-ui-css` to our project;

```
yarn add semantic-ui-react semantic-ui-css
```

Normally, adding new packages will result in changes to our `package.json` (which lists requested versions of direct dependencies) and the `yarn.lock` file (which lists resolved versions for all recursive dependencies). Since we set up the "offline mirror" feature, we should also see a few new files located in the `./offline-mirror` folder. These are the package tarballs that Yarn downloaded from NPM, and copied into the offline mirror cache we configured. We'll add all those files to Git so that they're available whenever anyone tries to clone and set up this project.

```
git add package.json yarn.lock offline-mirror
```

> **Commit 30a0820: Add Semantic-UI-React and Semantic-UI-CSS**

Then, we need to import the main Semantic-UI CSS file into our code so it gets included in the bundle. We should also render a single Semantic-UI `<Header>` component to make sure that's working as intended.

> **Commit 73fd477: Render an initial SUI component**

index.js

```
import {Provider} from "react-redux";

-import './index.css';
+import "semantic-ui-css/semantic.css";
```

App.js

```
import React, { Component } from 'react';
import {Header} from "semantic-ui-react";

import './App.css';

import SampleComponent from "./SampleComponent";

class App extends Component {
    render() {
        return (
            <div className="App">
                <div className="App-header">
                    <Header inverted as="h1">Project Mini-Mek</Header>
                </div>
                <SampleComponent />
            </div>
        );
    }
}

export default App;
```

While we're at it, let's yank out the sample component we set up and the associated test reducer, since we won't be needing it any more, and also do some cleanup on the original CSS and HTML that generated when the project was created (such as removing the spinning logo, and shrinking the size of the header bar).

Commit 7180805: Remove unused code and files

With the sample code cleaned up, our initial empty UI now looks like this:

```
import {combineReducers} from "redux";

const rootReducer = combineReducers({
});

export default rootReducer;
```

Yay - a black bar with some white text! Isn’t this exciting? :) Don’t worry, we’ll start adding a lot more from here.