Camosun College

ICS 211 - Web Applications

Lab 5 - JSX

Due Date: Lab Demo Due by Oct. 26 @ 2:20 PM

Lab Quiz (5 Questions) Due by Oct. 26 @ 11:59 PM

Background and Theory

JSX

JSX is "syntactic sugar" for JavaScript. It allows you to write HTML-like syntax right in JavaScript. This makes your React code easier to read, more concise, and hopefully, easier to code! The one large issue with JSX is that browsers can't execute it. Thus, you need to use a *transpiler* that converts JSX into JavaScript that the browser can understand. One transpiler that can do this is called Babel. There are two different ways you can use Babel. You can setup your dev environment so that Babel runs before you deploy your application. The other option is to add the Babel library as a CDN link directly in your web application. This causes the browser to convert your JSX to JavaScript "on-the-fly". This seriously degrades performance and should *never* be used in production. However, for our purposes, it is the easiest way to get started.

JSX essentially gets rid of `React.createElement()` calls. For example, this:

```
ReactDOM.render(
  React.createElement('h1', null, 'ICS 211 Web Apps'),
  document.getElementById('container')
);
```

In JSX:

```
ReactDOM.render(<h1>ICS 211 Web Apps</h1>,
    document.getElementById('container')
);
```

JSX also makes it easier to nest elements AND apply HTML attributes. For example, this:

```
let p = React.createElement('p', null,
  React.createElement ( 'a', { href: 'https://reactjs.org/docs/react-without-jsx.html' }, 'Doin
```

In JSX:

```
<p>
  <a href="https://reactjs.org/tutorial/tutorial.html">React Tutorial using JSX</a>
</p>
```

Creating a React Element from a React Component is also more concise. This:

```
ReactDOM.render(
  React.createElement(ICS211),
  document.getElementById('container')
);
```

IN JSX:

```
ReactDOM.render(
  <ICS211/>,
  document.getElementById('container')
);
```

Props in JSX use curly braces `{ }`. This:

```
let h2 = React.createElement('h2', this.props, 'You did a lab on ' +
      this.props.labTopic + ' this semester.');
```

In JSX (rendering only the id attribute):

```
<h2 id={this.props.id}>
  You did a lab on {this.props.labTopic} this semester.
</h2>
```

Values for Props on Element creation is also more concise. This:

```
React.createElement(ICS211, {
  id : 'lab 1',
  labTopic : 'Hexo'
})
```

In JSX:

```
<ICS211 id='lab 1' labTopic='Hexo'/>
```

See the associated lecture notes for more examples of JSX.

Partitioning a UI and Composite Components

Components in React are *composable*. They should be easy to reuse, move around, and be part of other components. When used inside another Component, it is said to be a *child* Component and have a *parent-child* relationship. A Component that contains child components is said to be a *Composite or Complex Component*.

Partitioning a UI into Components takes practice. You want relationships (whether parent-child or sibling) between UI Elements to make sense. You also have to take into account how the UI Elements are used. For example, a UI Element that is used repeatedly on a page is a good candidate to be a Component. A good ROT (Rule of Thumb) is your Component should do *just one thing* (this doesn't apply to Composite Components).

Styling Components in JSX

To adhere to React's composable, reusable Component model, you want to encapsulate any CSS for your Component *within* your Component. To do so, you must use inline CSS and it must be a JavaScript Object. CSS properties, with two or more words, become Camel-Cased. For example:

```
<p style={{textIndent: '3em'}}>This paragraph is indented.</p>
```

The first set of curly braces is for rendering in JSX while the second is for the JavaScript Object. Another option is to assign styles to a variable:

```
const pStyle = { textIndent: '3em' };

<p style = {pStyle}>This paragraph is indented.</p>
```

You can also use props with CSS. **A full example of this is in the associated lecture notes on JSX.**

Tasks

Task 1 - Refactor Your Lab 4 Solution to Use JSX

In Lab 4, you made a very simple web application using React. You used two custom React Components: *Title* and *Site*. The Site Component took props. Your task for this lab is to refactor what you did in Lab 4 to use JSX.

1. Use your solution for Lab 4 as a starting point. First, in the `<head>` section, add the CDN for Babel:

   ```
   <script crossorigin src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
   ```

2. Add the `type` attribute to the `<script>` tag that's in your HTML Document's `<body>` tag:

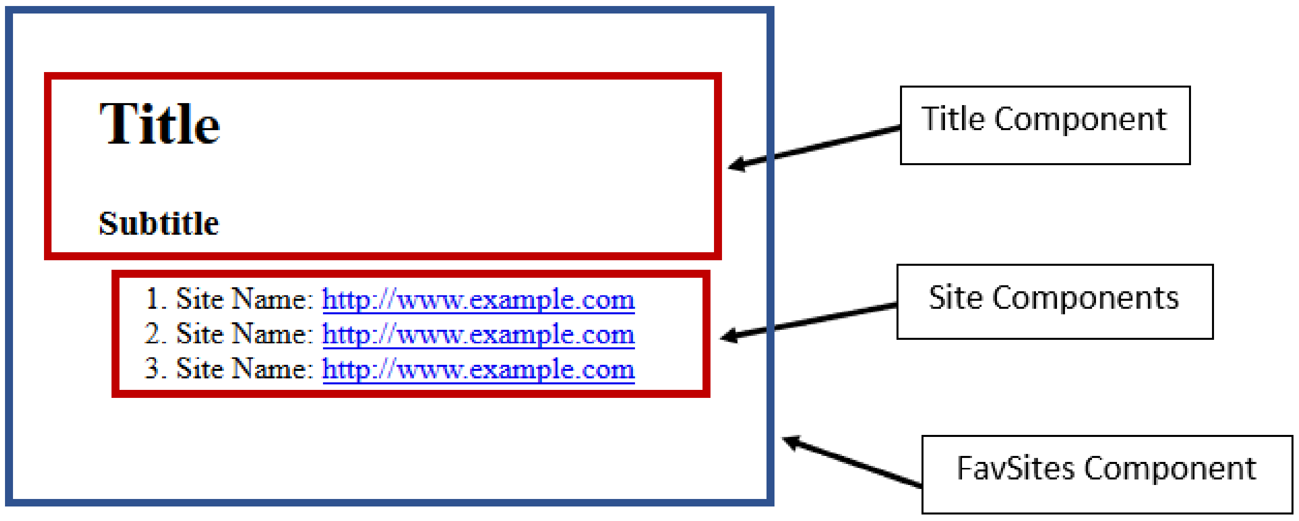
   ```
   <script type="text/babel">
   ```

3. Refactor both the Title and Site Components to use JSX.

4. Refactor `ReactDOM.render()` to use JSX.

Task 2 - Make a Composite Component

1. Now suppose you want to package your UI as a single Unit. Perhaps you want to re-use your Favorite Sites UI in a larger application. You are going to create a new **FavSites Composite Component** that will act as a UI Container:


Title
Subtitle
1. Site Name: http://www.example.com
2. Site Name: http://www.example.com
3. Site Name: http://www.example.com
Title Component
Site Components
FavSites Component

Add the following Class to your Application:

```
class FavSites extends React.Component {
  render() {
    const favSitesStyle = {
      height: 200,
      width: 400,
      padding: 0,
      backgroundColor: '#FFF2CC',
      boxShadow: '0px 0px 5px #666'
    }

    return (
      <div style={favSitesStyle}>
        {/* Put your render code here */}
      </div>
    )
  }
}
```

This Component has no functionality besides acting as a UI Container with some CSS Styling added.

2. Move what is currently between the `<div>` tags of your page's `ReactDOM.render()` method call and put it in the `<div>` tags of the `return()` method in `FavSites`. Now your Title and Site Components are nicely encapsulated in the FavSites Composite Component.

3. Now in your page's `ReactDOM.render()` call, you need only a single Element for the first argument:

```
<FavSites />
```

Task 3 - Styling the Rest of Your UI

1. Style your Title Component with at least **four** CSS Rules. If you don't remember CSS, a PDF of your Basic CSS slides from ICS 111 Web Fundamentals has been posted alongside this Lab.

2. Add at least **two** CSS Rules to your Site Component. One of the rules **must** use props.

3. You may need to adjust the `FavSites` styles to accommodate the styles you added for your Components.

Lab Submission (see top of lab for due date)

1. Demo to the Lab Instructor (10 marks):
    - Your functioning website using `http-server`
    - Your React Code that shows:
        - Both Title and Site Component Classes Re-factored using JSX
        - The `ReactDOM.render()` method call re-factored using JSX
        - Your Components are encapsulated in the *FavSites* Component
        - The *Title* Component has at least four CSS Rules
        - The *Site* Component has at least two CSS Rules
            - one **must** use props
2. Do the Lab 5 Quiz in D2L (5 marks).