Full Stack IV – Lab

- React Setup and first React Application

Developer Note:

- Take time to read the Extension description and documentation.
- Submit the completed application after all exercises are completed

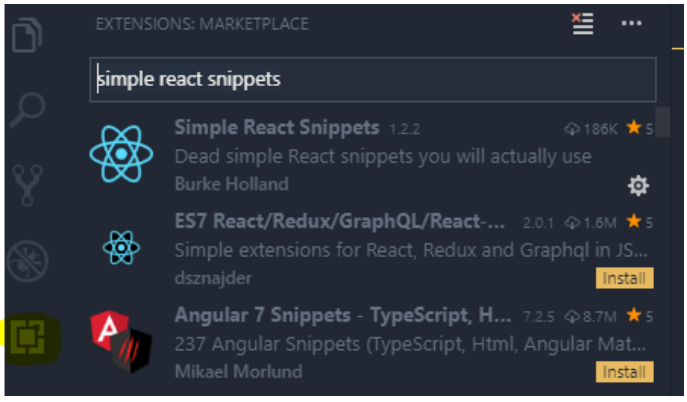
References:

https://facebook.github.io/create-react-app/
https://marketplace.visualstudio.com/items?itemName=burkeholland.simple-react-snippets
https://chrome.google.com/webstore/detail/react-developer-tools

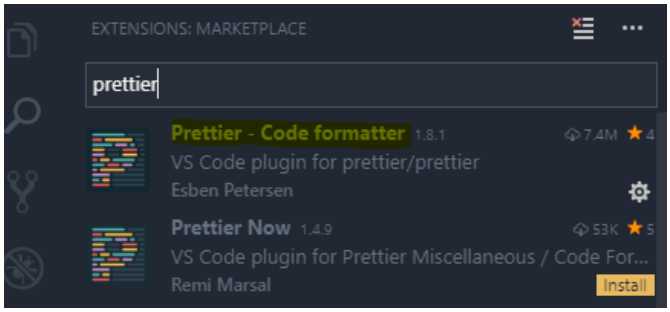
Exercise 1 – React Environment Setup

1. In Visual Studio Code editor and click the Extension tab on the left menu

2. Search and install Simple React Snippets. (An extension for collection of React Snippets and commands.)



3. Search and install Prettier code formatter



4. To install ***React Devtools***, visit the extension page for your browser of choice — Chrome or Firefox — and find the install button.

https://chrome.google.com/webstore/detail/react-developer-tools

Exercise 2 – React First Application

1. Install Create React App build tool using **npm install**

```
npm install -g create-react-app
```

2. After it's installed, you can just run the **init react-app** on the command line, followed by the name of the app you want to create. This creates the react app, with all the necessary functionality you need, already built into the app.

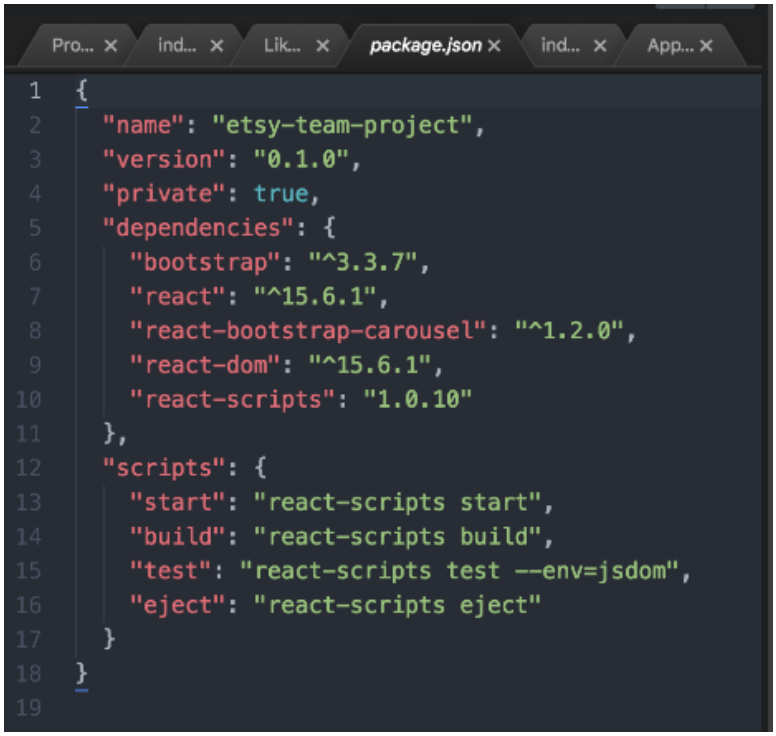
```
npm init react-app my-app
```

3. Then you can just change directory **cd my-app** and start the application with **npm start**.

4. After you've started the app, go ahead and open up the application in your text editor. The first thing you want to look at is the file structure. **Create-React-App** has already installed several files for you, and organized several of them into folders. Your main folders are a src folder and a public folder.

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

5. Go to the go to the **package.json** file, you can see that the **create-react-app** has installed several react scripts for you as well.



6. Next is the **index.html** page, located in the public folder. You will also notice the div with the id of "root". This is where all of the content will be output. You don't need to change that, but just know

that it's there.

```
    <title>React App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this
      app.
    </noscript>
    <div id="root"></div>
```

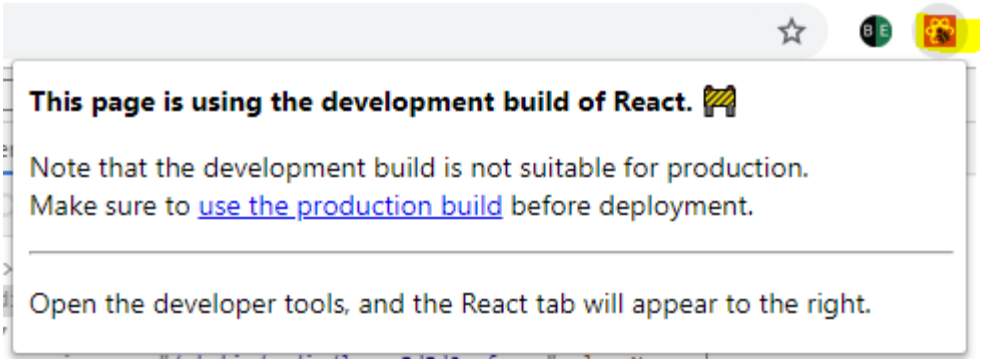
7. The next file you may want to look at is your **index.js** file in the src folder. Here, we're importing **React** and the **ReactDOM**. We are also rendering everything here to the 'root' element. You won't need to touch or alter this file either, but it's just good to know that this is how it's structured.

Pro... × ind... × Lik... × ind... × *index.js* × App... ×

```
import React from 'react';
import ReactDOM from 'react-dom';
import './styles/index.css';
import App from './components/App';
import registerServiceWorker from
'./registerServiceWorker';

ReactDOM.render(<App />,
document.getElementById('root'));
registerServiceWorker();
```

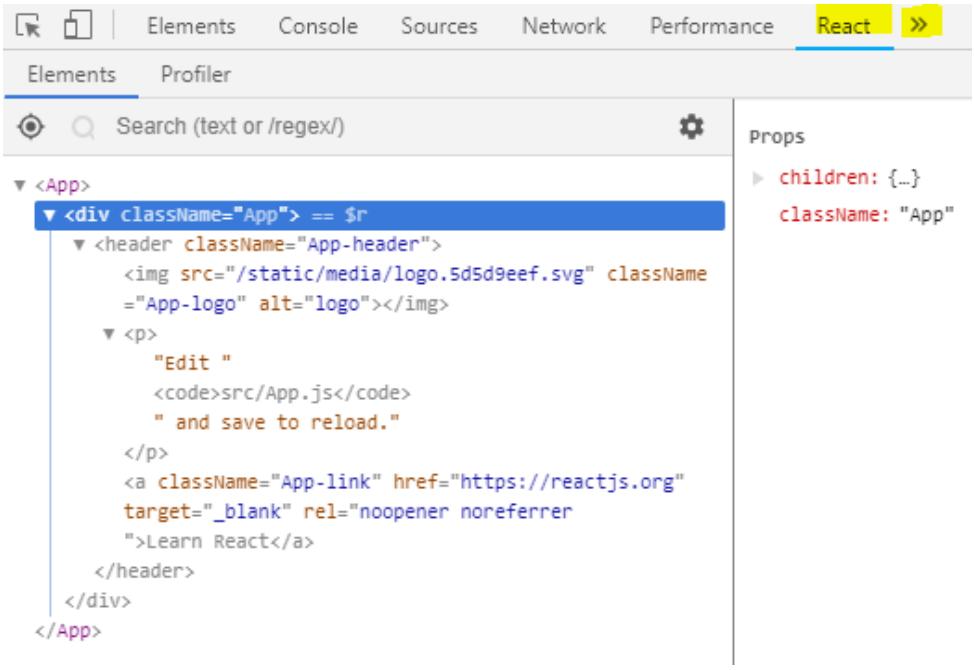
8. Click the React icon in the browser toolbar. This will become active when React code is detected in the source page.



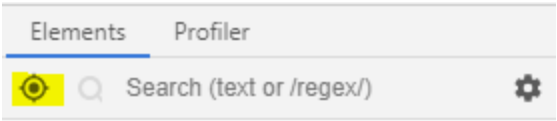
9. Press F12 and launch the **React Developer** tools. It will allow you to inspect the view of the React component tree, as well as the current state and props of the component you select.

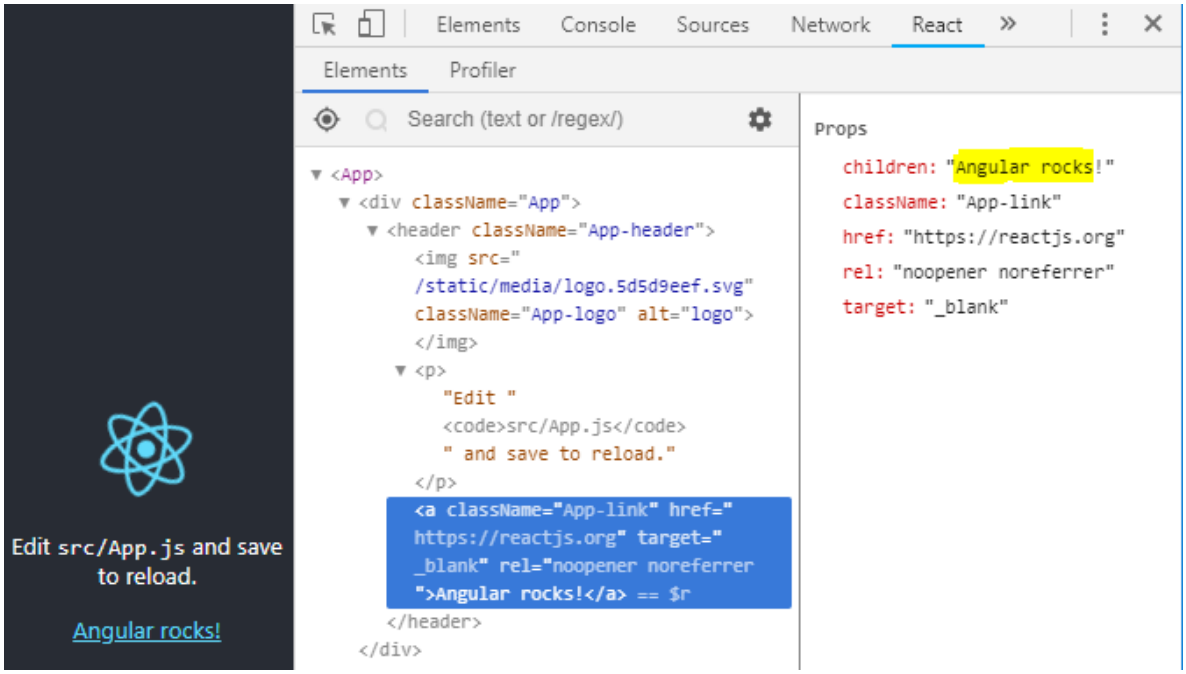
   The view of our app's component tree is similar to the plain HTML tree you'd see on the Elements tab, but there are some improvements:

   Our React elements are displayed next to plain HTML elements and are selectable.

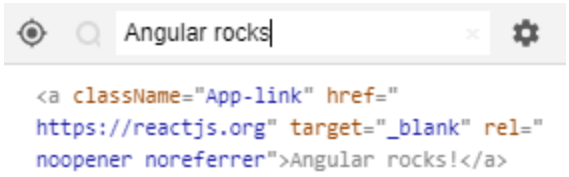


10. Click the bullseye button, then select an element on the page. Change the children element properties text to the following.



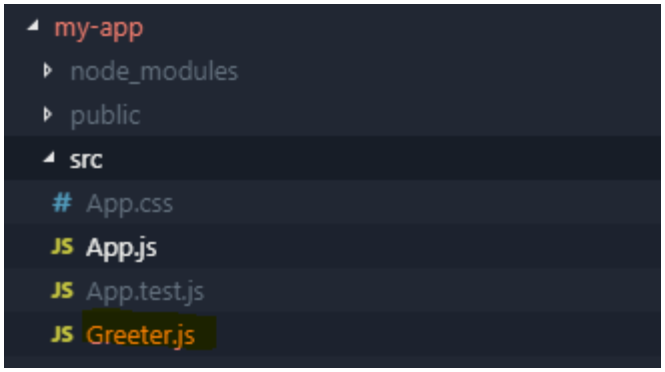


11. Use the search feature to find elements in the component tree.



Exercise 3 – Modifying the Application

1. Create a new file named ***Greeter.js*** in the ***src*** folder.



2. In the new ***Greeter.js*** file, use the React Snippet extension and commands. Type the following, **sfc** and press the tab key. The following snippet will be generated in the file.

```
const  = () => {
    return (  );
}

export default ;
```

3. Make the following changes to ***Greeter.js*** code file. This is your first React Component!

```
const Greeter = () => {
  return <h1>Hello World!</h1>;
};

export default Greeter;
```

4. Navigate to the ***index.js*** file and add the following import statement, to import the ***Greeter.js*** file

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import Greeter from './Greeter'
import * as serviceWorker from './serviceWorker';
```

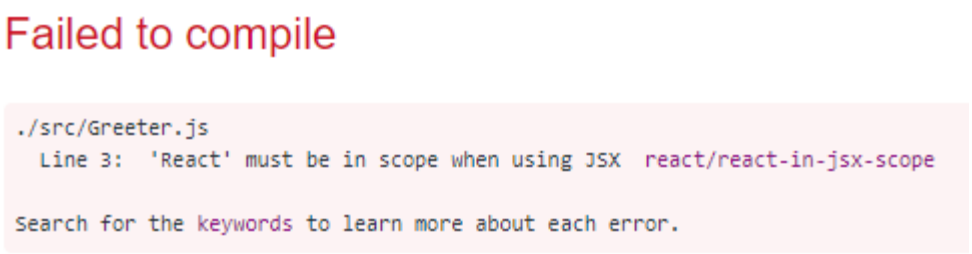
5. Find the ***ReactDOM*** render code that loads the main App component in the root element of the DOM tree.

```
ReactDOM.render(<App />, document.getElementById('root'));
```

Update the render component to be the new Greeter component we created in ***Greeter.js***

```
ReactDOM.render(<Greeter />, document.getElementById('root'));
```

The following compilation error will display in the browser.



6. Navigate back to the ***Greeter.js*** and the following import statement. The compilation process needs React to render to the HTML in the file.

```
import React from "react";
```

7. Verify the browser is rendering the following result.

Exercise 4 – Rendering Multiple Components

1. Create a new file named LikeButton.js with the following code.

```
const LikeButton = () => {
  return <button>Like!</button>;
};

export default LikeButton;
```

2. Navigate to the ***index.js*** file and add the following import statement, to import the LikeButton***.js*** file
3. Modify the ReactDom.render function to add the second component and compile.

```
ReactDOM.render(
    <Greeter />
    <LikeButton />,
  document.getElementById("root")
);
```

When you compile you will get the following parse error. This is because ReactDOM.render requires only one parent element with nested sibling elements.

```
Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...
```

4. To correct this we need to wrap these two component elements in a parent element. We will wrap these in a tag and recompile to correct the parse error

```
ReactDOM.render(
  <div>
    <Greeter />
    <LikeButton />
  </div>,
  document.getElementById("root")
);
```

This will render the following output.

Hello World!

Like!

Exercise 5 - React Fragments

1. When we inspect this code using the Developer Tools we will see the following markup.

```
▲ <div id="root">
  ▲ <div>
      <h1>Hello World!</h1>
      <button>Like!</button>
    </div>
  </div>
```

There is a nested tag under the root tag. In most cases the wrapper div is “irrelevant” and is only added because React components require you to return only one element. This kind of behaviour results in useless markup and sometimes even invalid HTML to be rendered, which is bad.

2. We can change the render to replace wrapper with React.Fragment

```
ReactDOM.render(
  <React.Fragment>
    <Greeter />
    <LikeButton />
  </React.Fragment>,
  document.getElementById("root")
);
```

React fragments let you group a list of children without adding extra nodes to the DOM because fragments are not rendered to the DOM.

When we inspect the HTML using the Developer Tool this is the new markup using React Fragment

```
<div id="root">
  <h1>Hello World!</h1>
  <button>Like!</button>
</div>
```

8. React has a short syntax for Fragments that look like an empty tag.. Change the render code to use the following Fragment short syntax.

```
ReactDOM.render(
  <>
    <Greeter />
    <LikeButton />
  </>,
  document.getElementById("root")
);
```

Exercise 6 - Inline Component

1. In the Index.js create a Display Component that will contain the React Fragment and sibling components. We don't need to create a javascript file this time, as the component will be inline JavaScript function in the Index.js. The final render method will be as follows.

```
ReactDOM.render(<Display />, document.getElementById("root"));
```

Challenge

1. In the Display Component add an iterator to display the LikeButton component 10 times, so that the UI displays the following.