ReactJS Learning – Notes (Module -2)

We will cover following topics in this session:

- What is JSX?
- JSX Syntax
- Nested Element
- Attribute
- Expression & Ternary Expression
- Styling
- Comments
- Naming Convention
- Components (Stateless & Stateful)

What is JSX?

React uses JSX for templating instead of regular JavaScript.

- It is faster because it performs optimization while compiling code to JavaScript.
- It is also type-safe and most of the errors can be caught during compilation.
- It makes it easier and faster to write templates, if you are familiar with HTML.

Syntax:

App.jsx

```
import React from 'react';


class App extends React.Component {
  render() {
    return (
      <div>
        Hello World!!!
      </div>
```

```
    );
  }
}
export default App;
```

Nested Elements

If we want to return more elements, we need to wrap it with one container element. Notice how we are using **div** as a wrapper for **h1**, **h2** and **p**elements.

App.jsx

```
import React from 'react';


class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
        <h2>Content</h2>
        <p>This is the content!!!</p>
      </div>
    );
  }
}
export default App;
```

Attributes

We can use our own custom attributes in addition to regular HTML properties and attributes. When we want to add custom attribute, we need to use **data-**prefix. In the following example, we added **data-myattribute** as an attribute of **p** element.

```
import React from 'react';
```

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
        <h2>Content</h2>
        <p data-myattribute = "somevalue">This is the content!!!</p>
      </div>
    );
  }
}
export default App;
```

JavaScript Expressions

JavaScript expressions can be used inside of JSX. We just need to wrap it with curly brackets **{}**. The following example will render **2**.

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{1+1}</h1>
      </div>
    );
  }
}
export default App;
```

Ternary expressions

We cannot use **if else** statements inside JSX, instead we can use **conditional (ternary)** expressions. In the following example, variable **i** equals to **1** so the browser will render **true**, If we change it to some other value, it will render **false**.

```
import React from 'react';


class App extends React.Component {
  render() {
    var i = 1;
    return (
      <div>
        <h1>{i == 1 ? 'True!' : 'False'}</h1>
      </div>
    );
  }
}
export default App;
```

Styling

React recommends using inline styles. When we want to set inline styles, we need to use **camelCase** syntax. React will also automatically append **px** after the number value on specific elements. The following example shows how to add **myStyle** inline to **h1** element.

```
import React from 'react';


class App extends React.Component {
  render() {
    var myStyle = {
      fontSize: 100,
```

```
      color: '#FF0000'
    }
    return (
      <div>
        <h1 style = {myStyle}>Header</h1>
      </div>
    );
  }
}
export default App;
```

Comments

When writing comments, we need to put curly brackets **{}** when we want to write comment within children section of a tag. It is a good practice to always use **{}** when writing comments, since we want to be consistent when writing the app.

```
import React from 'react';


class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
        {//End of the line Comment...}
        {/*Multi line comment...*/}
      </div>
    );
  }
}
export default App;
```

Naming Convention

HTML tags always use **lowercase** tag names, while React components start with **Uppercase**.

Note – You should use **className** and **htmlFor** as XML attribute names instead of **class** and **for**.

Components

We will learn how to combine components to make the app easier to maintain. This approach allows to update and change your components without affecting the rest of the page.

Stateless Example

Our first component in the following example is **App**. This component is owner of **Header** and **Content**. We are creating **Header** and **Content** separately and just adding it inside JSX tree in our **App** component. Only **App** component needs to be exported.

App.jsx

```
import React from 'react';


class App extends React.Component {
  render() {
    return (
      <div>
        <Header/>
        <Content/>
      </div>
    );
  }
}
class Header extends React.Component {
  render() {
    return (
```

```
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}
class Content extends React.Component {
  render() {
    return (
      <div>
        <h2>Content</h2>
        <p>The content text!!!</p>
      </div>
    );
  }
}
export default App;
```

To be able to render this on the page, we need to import it in **main.js** file and call **reactDOM.render()**. We already did this while setting the environment.

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

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

we will set the state for owner component (**App**). The **Header** component is just added like in the last example since it doesn't need any state. Instead of content tag, we are creating **table** and **tbody** elements, where we will dynamically insert **TableRow** for every object from the **data**array.

It can be seen that we are using EcmaScript 2015 arrow syntax (⇒) which looks much cleaner than the old JavaScript syntax. This will help us create our elements with fewer lines of code. It is especially useful when we need to create a list with a lot of items.

App.jsx

```
import React from 'react';


class App extends React.Component {
  constructor() {
    super();
    this.state = {
      data:
      [
        {
          "id":100,
          "name":"Raman",
          "age":"20"
        },
        {
          "id":200,
          "name":"Chahat",
          "age":"22"
        }
      ]
```

```
    }
  }
  render() {
    return (
      <div>
        <Header/>
        <table>
          <tbody>
            {this.state.data.map((person, i) => <TableRow key = {i}
              data = {person} />)}
          </tbody>
        </table>
      </div>
    );
  }
}
class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}
class TableRow extends React.Component {
  render() {
    return (
```

```
      <tr>
        <td>{this.props.data.id}</td>
        <td>{this.props.data.name}</td>
        <td>{this.props.data.age}</td>
      </tr>
    );
  }
}
export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

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