04. REACT Components.

In this chapter, 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.

4.1. 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.

Example: *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.

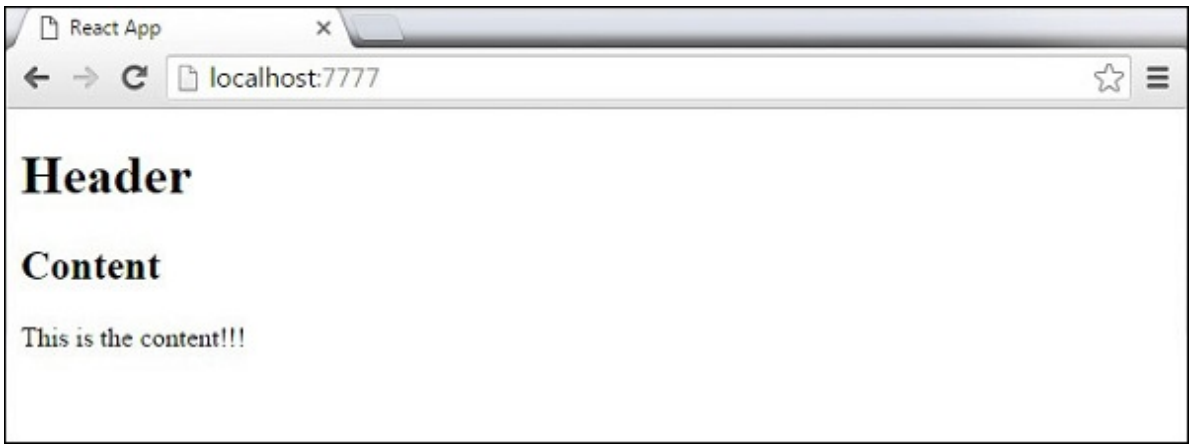
Example: *main.js*

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

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

The above code will generate the following result.

Ouput:



4.2. Stateful Example

In this example, 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.

Example: *App.jsx*

```
import React from 'react';

class App extends React.Component {
   constructor() {
      super();
      this.state = {
         data:
         [
            {
```

```
                "id":1,
                "name":"Foo",
                "age":"20"
            },
            {
                "id":2,
                "name":"Bar",
                "age":"30"
            },
            {
                "id":3,
                "name":"Baz",
                "age":"40"
            }
        ]
    }
}
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;
```

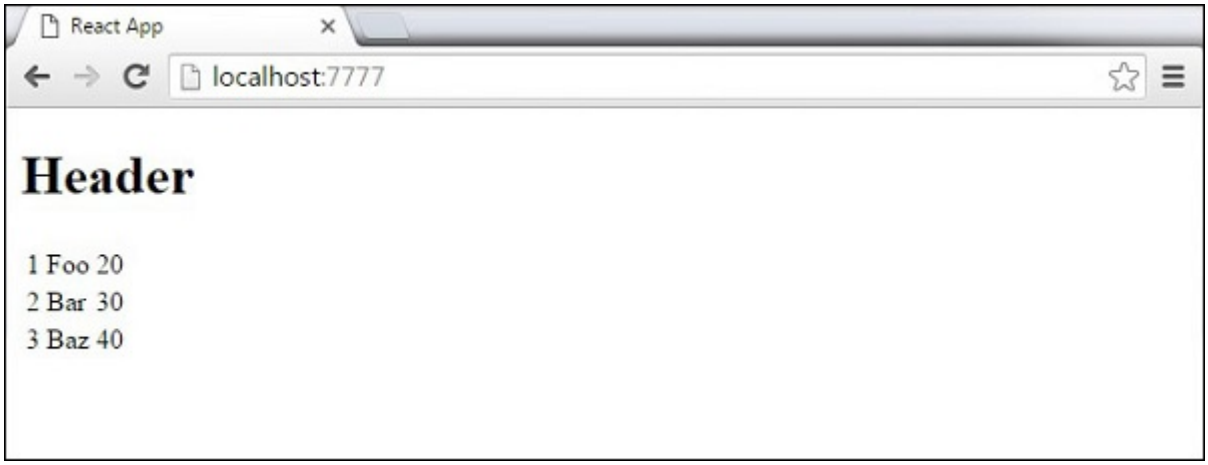
Example: *main.js*

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

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

Note − Notice that we are using key = {i} inside map() function. This will help React to update only the necessary elements instead of re-rendering the entire list when something changes. It is a huge performance boost for larger number of dynamically created elements.

Output:


React App
localhost:7777
Header
1 Foo 20
2 Bar 30
3 Baz 40