05. REACT State and Props.

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.

5.1. State

State is the place where the data comes from. We should always try to make our state as simple as possible and minimize the number of stateful components. If we have, for example, ten components that need data from the state, we should create one container component that will keep the state for all of them.

5.2. Using Props

The following sample code shows how to create a stateful component using *EcmaScript2016* syntax.

Example: *App.jsx*

```
import React from 'react';

class App extends React.Component {

constructor(props) {
      super(props);

      this.state = {
         header: "Header from state...",
         content: "Content from state..."
      }
   }

   render() {
      return (
         <div>
```

```
                <h1>{this.state.header}</h1>
                <h2>{this.state.content}</h2>
            </div>
        );
    }
}

export default App;
```

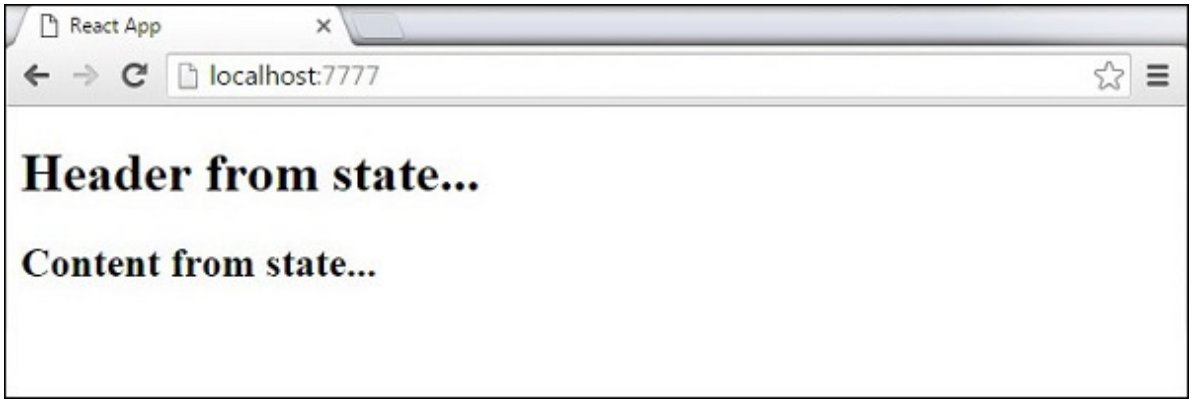
Example: *main.js*

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

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

The above code will generate the following result.

Ouput:



5.3. Props

The main difference between state and props is that **props are immutable**. This is why the container component should define the state that can be updated and changed, while the child components should only pass data from the state using props.

When we need immutable data in our component, we can just add props to ***reactDOM.render()*** function in *main.js* and use it inside our component.

Example: *App.jsx*

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>{this.props.headerProp}</h1>
            <h2>{this.props.contentProp}</h2>
         </div>
      );
   }
}
export default App;
```

Example: *main.js*

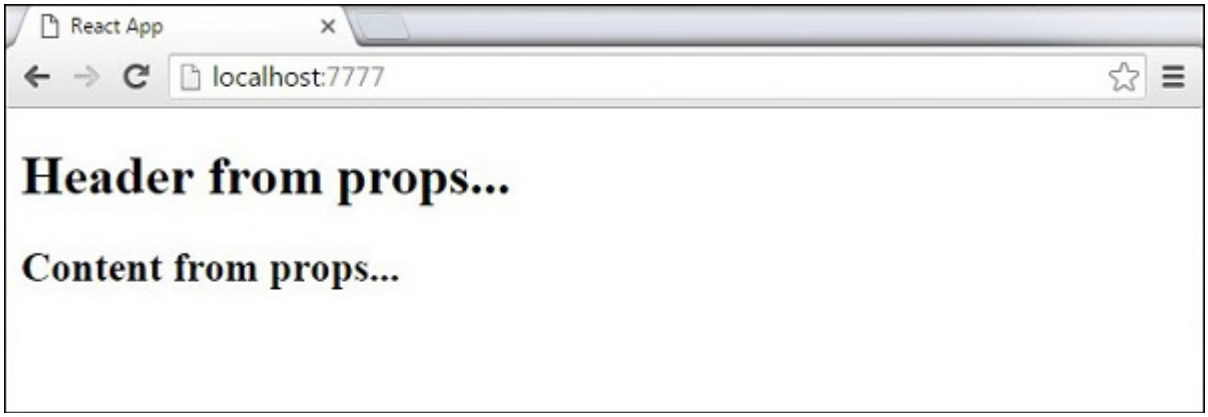
```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App headerProp = "Header from props..." contentProp
= "Content
   from props..."/>, document.getElementById('app'));

export default App;
```

This will produce the following result.

Output:



5.4. Default Props

You can also set default property values directly on the component constructor instead of adding it to the ***reactDom.render()*** element.

Example: *App.jsx*

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>{this.props.headerProp}</h1>
            <h2>{this.props.contentProp}</h2>
         </div>
      );
   }
}

App.defaultProps = {
   headerProp: "Header from props...",
   contentProp:"Content from props..."
}

export default App;
```

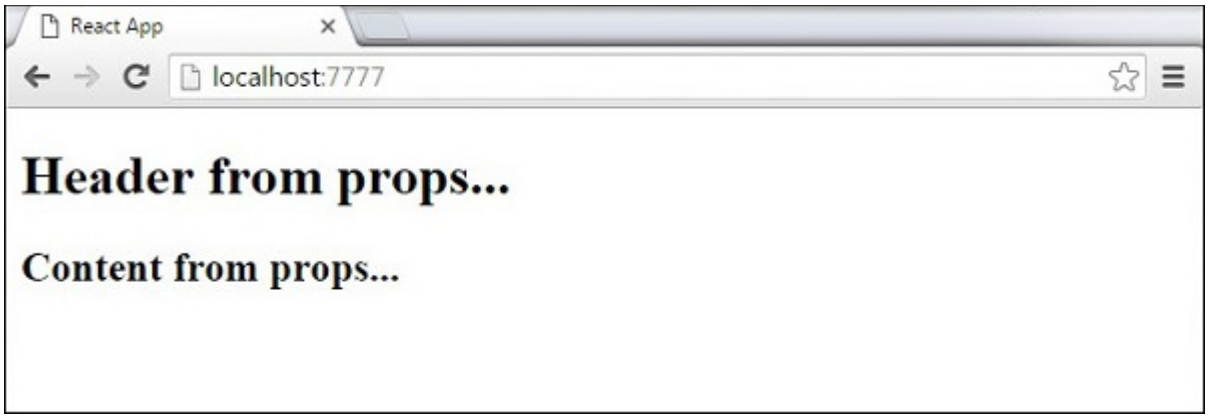
Example: *main.js*

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

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

Output is the same as before.

Output:



5.5. State and Props

The following example shows how to combine state and props in your app. We are setting the state in our parent component and passing it down the component tree using props. Inside the render function, we are setting headerProp and contentProp used in child components.

Example: *App.jsx*

```
import React from 'react';


class App extends React.Component {

   constructor(props) {
      super(props);
      this.state = {
         header: "Header from props...",
         content: "Content from props..."
      }
   }

   render() {
      return (
         <div>
            <Header headerProp = {this.state.header}/>
            <Content contentProp = {this.state.content}/>
         </div>
      );
   }
}


class Header extends React.Component {
   render() {
      return (
         <div>
            <h1>{this.props.headerProp}</h1>
         </div>
      );
   }
}
```

```
class Content extends React.Component {
   render() {
      return (
         <div>
            <h2>{this.props.contentProp}</h2>
         </div>
      );
   }
}

export default App;
```

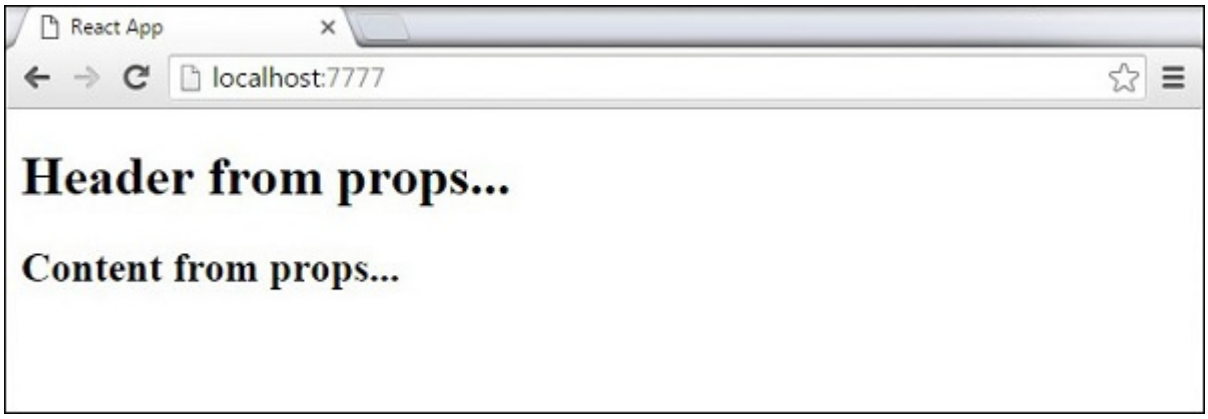
Example: *main.js*

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

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

The result will again be the same as in the previous two examples, the only thing that is different is the source of our data, which is now originally coming from the state. When we want to update it, we just need to update the state, and all child components will be updated. More on this in the ***Events*** chapter.

Output:



5.6. Validating Props

Properties validation is a useful way to force the correct usage of the components. This will help during development to avoid future bugs and problems, once the app becomes larger. It also makes the code more readable, since we can see how each component should be used.

In this example, we are creating *App* component with all the props that we need. ***App.propTypes*** is used for props validation. If some of the props aren't using the correct type that we assigned, we will get a console warning. After we specify validation patterns, we will set ***App.defaultProps.***

Example: *App.jsx*

```
import React from 'react';

class App extends React.Component {

   render() {
      return (
```

```
            <div>
                <h3>Array: {this.props.propArray}</h3>
               <h3>Bool: {this.props.propBool ? "True..." : "False..."}</h3>
                <h3>Func: {this.props.propFunc(3)}</h3>
                <h3>Number: {this.props.propNumber}</h3>
                <h3>String: {this.props.propString}</h3>
                <h3>Object: {this.props.propObject.objectName1}</h3>
                <h3>Object: {this.props.propObject.objectName2}</h3>
                <h3>Object: {this.props.propObject.objectName3}</h3>
            </div>
        );
    }
}

App.propTypes = {
    propArray: React.PropTypes.array.isRequired,
    propBool: React.PropTypes.bool.isRequired,
    propFunc: React.PropTypes.func,
    propNumber: React.PropTypes.number,
    propString: React.PropTypes.string,
    propObject: React.PropTypes.object
}

App.defaultProps = {
    propArray: [1,2,3,4,5],
    propBool: true,
    propFunc: function(e){return e},
    propNumber: 1,
    propString: "String value...",

    propObject: {
        objectName1:"objectValue1",
        objectName2: "objectValue2",
        objectName3: "objectValue3"
    }
```

```
}
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'));
```

Output:

```
Array: 12345
Bool: True…
Func: 3
Number: 1
String: String value ...
```