



ReactJS Overview

A JS Library for building interfaces

What is React?

- React (ReactJS / React.js) –A JavaScript library for building user interfaces (UI)
- Open-source, declarative, component-based, by Facebook.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello React</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <!-- Don't use this in production: -->
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="app">A React component will be rendered here</div>
    <script type="text/babel">
      ReactDOM.render(<div>Hello, React!</div>,
        document.getElementById('app'));
    </script>
  </body>
</html>
```

Setup React Server-Side Build

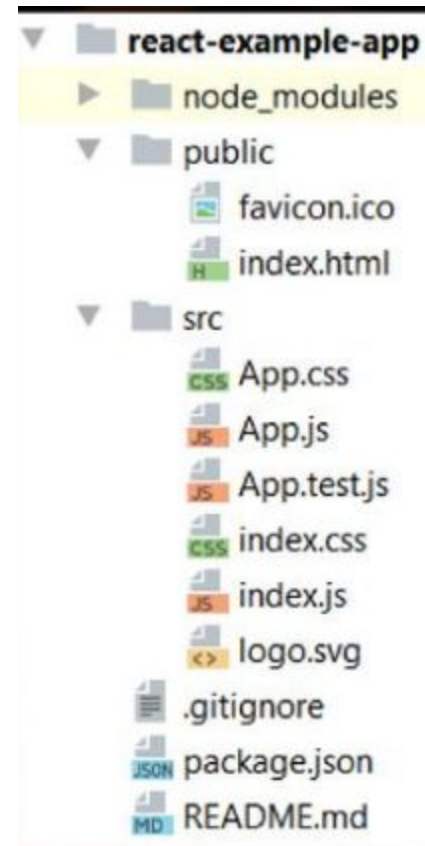
Configuring the Development Environment

- install the React app creator (on-time global install)
`npm -g install create-react-app`
- Run the React app creator (this is very slow: ~5-10 minutes!)
`create-react-app react-example`
- Starts your React app from the command line
`npm start`
- Browse you app from
<http://localhost:3000>



React App Structure

- **package.json** – project configuration
 - Module name, dependencies, build actions
- **index.html**
 - App main HTML file
- **index.js**
 - App main JS file (startup script)
- **App.js, App.css, App.test.js**
 - React component "**App**"



What is JavaScript XML Syntax(JSX)

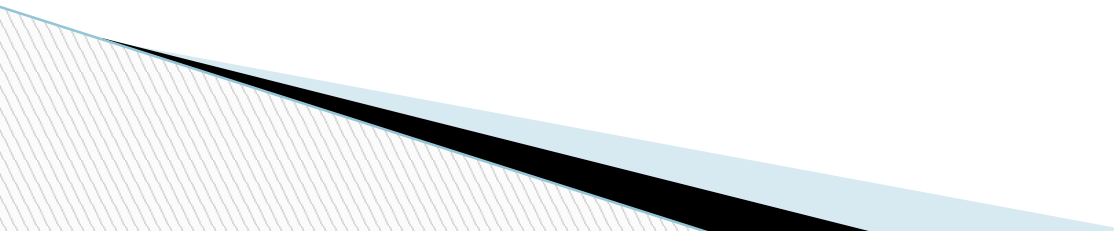
- JSX is an extension to javascript. It is a template script where you will have the power of using HTML and Javascript together.
- We need JSX in React because it allows us to make use of Html and javascript in the same file and take care of the state changes in the DOM in an efficient manner.
- JSX is an extension created by Facebook that adds XML syntax to JavaScript. In order to use JSX you need to include the Babel library and change `<script type="text/javascript">` to `<script type="text/babel">` in order to translate JSX to Javascript code.

Running react in the browser

- In-browser React transpilation with "babel"

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello React</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <!-- Don't use this in production: -->
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="app">A React component will be rendered here</div>
    <script type="text/babel">
      class HelloMessage extends React.Component {
        render() {
          return <div>Hello {this.props.name}</div>;
        }
      }
      ReactDOM.render(<HelloMessage name="CSE" />,
        document.getElementById('app'));
    </script>
  </body>
</html>
```

What are Components in ReactJS?

- ❑ Components are like pure javascript functions that help make the code easy by splitting the logic into reusable independent code.
 - ❑ Components need data to work with. There are two different ways that you can combine components and data: either as **props** or **state**. props and state determine what a component renders and how it behaves.
 - ❑ A React component can be of two types: either a class component or a functional component. The difference between the two is evident from their names.
- 

Props

- If components were plain JavaScript functions, then props would be the function input. Going by that analogy, a component accepts an input (what we call props), processes it, and then renders some JSX code.
- Although the data in props is accessible to a component, React philosophy is that props should be immutable and top-down.
- The parent component can pass on whatever data it wants to its children as props, but the child component cannot modify its props.
- props are used to pass data and methods from a parent component to a child component.
- They are immutable.¹
- They allow us to create reusable components

states

- State, on the other hand, is an object that is owned by the component where it is declared. Its scope is limited to the current component. A component can initialize its state and update it whenever necessary.
- When the state is passed out of the current scope, we refer to it as a prop.
- A state is a javascript object similar to props that have data to be used with the reactjs render. The state data is a private object and is used within components inside a class.
- It allows you to create components that are interactive and reusable. It is mutable and can only be changed by using `setState()` method.

props: example

component_ex.html

class

function

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React Components</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      class Car1 extends React.Component {
        render() {
          return (
            <div>
              <h2>Brand: {this.props.brand}!</h2>
              <h2>Make: {this.props.make}</h2>
            </div>
          )
        }
      }
      ReactDOM.render(<Car1 brand="Ford" make="2000"/>, document.getElementById('root'));
    </script>
  </body>
</html>
```

```
<script type="text/babel">
  function Car2(props) {
    return (
      <div>
        <h2>Brand: {props.brand}!</h2>
        <h2>Make: {props.make}</h2>
      </div>
    )
  }
  let mycar = <Car2 brand="Maruthi" make="2014"/>;
  ReactDOM.render(mycar, document.getElementById('root'));
</script>
```

Brand: Maruthi!

Make: 2014

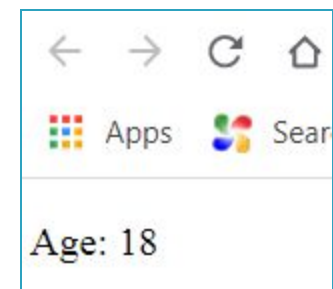
Brand: Ford!

Make: 2000

Creating JSX components

```
jsx_ex.html x
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello React</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <!-- Don't use this in production: -->
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="app"></div>
    <script type="text/babel">
      let element = <h1>Hello, <b>React</b>!</h1>;
      console.log(element);
      let age = 18;
      let ageElement = <p>Age: {age}</p>;
      ReactDOM.render(ageElement, document.getElementById('app'));
      /*ReactDOM.render(element, document.getElementById('app')) ;*/
    </script>
  </body>
</html>
```

```
▼ Object
  $$typeof: Symbol(react.element)
  _owner: null
  _self: null
  _source: null
  ▶ _store: Object
    key: null
  ▼ props: Object
    ▼ children: Array[3]
      0: "Hello, "
      ▶ 1: Object
      2: "!"
      length: 3
      ▶ __proto__: Array[0]
    ▶ __proto__: Object
  ref: null
  type: "h1"
  ▶ __proto__: Object
```



Functional Components

- Functional components are just JavaScript functions. They take in an optional input that we call as props.

```
const Hello = ({ name }) => (<div>Hello, {name}</div>);
```

By using an arrow function, we can skip the use of two keywords, function and return, and a pair of curly brackets.

Creating JSX component (with function)

```
jsx_function.html x
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="UTF-8" />
5    <title>Hello React</title>
6    <script src="react/react.development.js"></script>
7    <script src="react/react-dom.development.js"></script>
8    <script src="react/prop-types.js"></script>
9    <script src="react/babel.min.js"></script>
10 </head>
11 <body>
12   <div id="root"></div>
13   <script type="text/babel">
14     /* Function Component*/
15     function Test1() {
16       return <h2>Hi, I am also Hello World!</h2>;
17     }
18     ReactDOM.render(
19       <div><Test1 /></div>, document.getElementById('root'))
20   </script>
21 </body>
22 </html>
```

← → ↻ 🏠 ⓘ File | Y:/hks/WEE

📦 Apps 🔍 Search 🔄 Help Desk 🖨️

Hi, I am also Hello World!

Class Components

- **Class components** offer more features. The primary reason to choose class components over functional components is that they can have **state**.
- There are two ways that you can create a class component. The traditional way is to use `React.createClass()`. ES6 introduced a syntax that allows you to write classes that extend `React.Component`. However, both the methods are meant to do the same thing.

Creating JSX component (with class)

```
jsx_class.html x
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello React</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      //Class Component
      class Test extends React.Component {
        render() {
          return <h1>Hello World!</h1>;
        }
      }
      ReactDOM.render(
        <div><Test /></div>, document.getElementById('root'));
    </script>
  </body>
</html>
```

← → ↻ 🏠 ⓘ File | Y:
📱 Apps 🔍 Search 🌐 Help De

Hello World!

Stateful Components

- ❑ Stateful components are those components which have a state. The state gets initialized in the constructor. It stores information about the component's state change in memory. It may get changed depending upon the action of the component or child components.
- ❑ Stateful components are always class components. stateful components have a state that gets initialized in the constructor.
- ❑ The `this` keyword here refers to the instance of the current component.
- ❑ React components are equipped with a method called `setState` for updating the state. `setState` accepts an object that contains the new state.

Example-1 Stateful

```
stateful1_class.html x
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React Components</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      class StateExample extends React.Component {
        constructor() {
          super();
          this.state={
            first_name: 'Ram',
            last_name: 'Mohan'
          }
        }
        render() {
          return(
            <div>
              <p> Class Component (stateful) </p>
              <p>{this.state.first_name}</p>
              <p>{this.state.last_name}</p>
            </div>
          )
        }
      }
      ReactDOM.render(<StateExample />, document.getElementById('root'));
    </script>
  </body>
</html>
```

Example-2: Stateful Component

```
stateful_class.html
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React Components</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      class Car extends React.Component {
        constructor(props) {
          super(props);
          this.state = {
            brand: "Ford",
            model: "Mustang",
            color: "red",
            year: 1964
          };
        }
        changeColor = () => {
          this.setState({color: "blue"});
        }
        render() {
          return (
            <div>
              <h1>My {this.state.brand}</h1>
              <p>
                It is a {this.state.color}
                {this.state.model}
                from {this.state.year}.
              </p>
              <button
                type="button"
                onClick={this.changeColor}
              >Change color</button>
            </div>
          );
        }
      }
      ReactDOM.render(<Car />, document.getElementById('root'));
    </script> </body></html>
```

My Ford

It is a redMustangfrom 1964.

Change color

My Ford

It is a blueMustangfrom 1964.

Change color

Stateless Components

- We use either a function or a class for creating stateless components. But unless we need to use a lifecycle hook in our components, we should go for stateless functional components.
- They are easy to write, understand, and test, and you can avoid the `this` keyword altogether. Stateless components are those components which don't have any state at all, which means you can't use *`this.setState`* inside these components. It is like a normal function with no render method. It has no lifecycle, so it is not possible to use lifecycle methods such as *`componentDidMount`* and other hooks. When react renders our stateless component, all that it needs to do is just call the stateless component and pass down the props.

Examples: Stateless

```
stateless1_func.html x
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React Components</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      function Example(props) {
        return(
          <div>
            <p> Functional Component (stateless) </p>
            <p>{props.first_name}</p>
            <p>{props.last_name}</p>
          </div>
        )
      }
      ReactDOM.render(<Example first_name="Ram" last_name="Mohan"/>,
        document.getElementById('root'));
    </script></body>
</html>
```

```
stateless_func.html x
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React Components</title>
    <script src="react/react.development.js"></script>
    <script src="react/react-dom.development.js"></script>
    <script src="react/prop-types.js"></script>
    <script src="react/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      const Hello1={()=>{
        const sayHi=(event)=>{
          alert('Hi, This is stateless(functional) component');
        };
        return(
          <div>
            <button type="button"
              onClick={sayHi}>Say Hi</button>
          </div>
        );
      }
      ReactDOM.render(<Hello1 />, document.getElementById('root'));
    </script> </body>
</html>
```

Stateful vs. Stateless

- A stateless component can render props, whereas a stateful component can render both props and state. A significant thing to note here is to comprehend the syntax distinction. In stateless components, the props are displayed like `{props.name}` but in stateful components, the props and state are rendered like `{this.props.name}` and `{this.state.name}` respectively. A stateless component renders output which depends upon props value, but a stateful component render depends upon the value of the state. A functional component is always a stateless component, but the class component can be stateless or stateful.