Contents

[REACT JS 1](#_Toc72082138)

[WHAT IS REACT? 1](#_Toc72082139)

[REACT CONCEPTS AND CREATING REACT COMPONENTS 1](#_Toc72082140)

[BASIC STEPS TO CREATE REACT COMPONENT 2](#_Toc72082141)

[OVERVIEW : CREATING REACT COMPONENT 2](#_Toc72082142)

[UNDERSTANDING JSX 2](#_Toc72082143)

[CREATING A DEMO REACT APP – BLOG POST APP 4](#_Toc72082144)

[COMMUNICATION BETWEEN COMPONENTS 5](#_Toc72082145)

[CODE IMPLEMENTATION 5](#_Toc72082146)

[CLASS BASED COMPONENTS 7](#_Toc72082147)

[BENEFITS OF CLASS BASED COMPONENT 7](#_Toc72082148)

[BUILD APP USING CLASS BASED COMPONENT 7](#_Toc72082149)

[CREATING CLASS BASED COMPONENTS 9](#_Toc72082150)

[LIFE CYCLE METHOD OF A CLASS BASED COMPONENT 10](#_Toc72082151)

[ADDING CSS TO COMPONENT 12](#_Toc72082152)

[OPERATORS 13](#_Toc72082153)

[FINAL APP DESIGN AND CODE 14](#_Toc72082154)

[HANDLING EVENTS AND USER INPUTS 14](#_Toc72082155)

[BINDING EVENTS 15](#_Toc72082156)

[HTTP REQUEST IN REACT 15](#_Toc72082157)

# REACT JS

## WHAT IS REACT?

* React is JS library which is used to create user interfaces. It has component-based architecture.
* As the name suggest – Reacts to the state change.

|  |  |  |
| --- | --- | --- |
| **INSTALLING REACT** | npm install -g create-react-app | |
| **CREATING REACT PROJECT** | create-react-app <*project\_name>*  create-react-app react-complete-app |  |
| **STARTING NPM SERVER** | cd react-complete-app  npm start |
| **CHANGING THE DEFAULT PORT** | $env:PORT=5000  npm start | |
|  | We can run the command to intstall react globally and create the react app - in one go.  “npx” only works with npm version > 5.2 | |

## REACT CONCEPTS AND CREATING REACT COMPONENTS

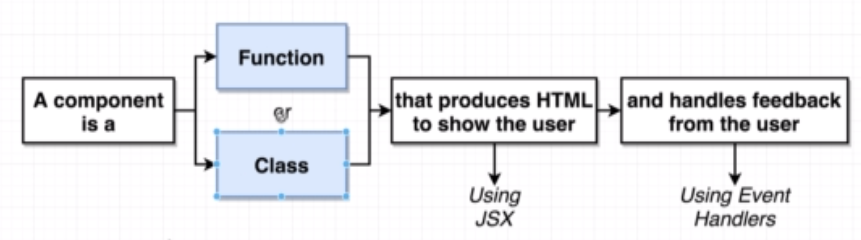
|  |  |  |  |
| --- | --- | --- | --- |
|  | | * A typical React app could be depicted as a component tree - having one root component ("App") and then a potentially infinite amount of nested child components. * When we create a react application, we build bunch of isolated, independent, and reusable components, which then composed together to build complex component. * Every React Application has once parent component called root component called “App” component. | |
|  | | * Every component in react is technically a JS class, which has   + State 🡪 The describes the state of the UI component when it will load on the page   + render() 🡪 This describes how the UI of the component will look like | |
|  | | * The react element/component is a JS object which is directly mapped with the DOM in the browser * React keep the lightweight representation of the DOM Elements in memory called Virtual DOM * When we update any react component - to keep the DOM in sync (to match the state) , it will only update that part of DOM which is mapped to that component   **THAT’S WHY IT CALLED REACT** | |
|  |  | | * The node\_modules for has all the dependencies, which we add in **package.json** file as a dependency. * In public folder has one **index.html** file which is served by the webserver. This is html file where all our react script will be injected (App root component js).   1. It has one root div which will become the parent of the react app   2. For SPA we don’t edit this html, because the root div is the place holder where we will mount our react app   3. We still can add CSS files to this file   <div id="**root**"></div> |

|  |
| --- |
| **src** - This folder has an App component which will be the parent component of all custom component we will create a in react manifest.json file give the basic functionality of PWA to react app. |

## BASIC STEPS TO CREATE REACT COMPONENT

|  |  |
| --- | --- |
| **IMPORT REACT AND REACT DOM LIBRARY** | import React from 'react';  import ReactDOM from 'react-dom'; |
| **CREATE A REACT COMPONENT (FUNCTIONAL or CLASS BASED)** | class App extends Component {  render() {  return (  <h1>Returning JSX</h1> 🡨 JSX expression  );  } }  export default App; |
| **RENDER THE COMPONENT IN THE BROWSER** | ReactDOM.render(<App /> ,document.getElementById('root')); |
|  | **OVERVIEW**   * Each component needs to return/ render some JSX code - it defines which HTML code React should render to the real DOM in the end. * JSX is NOT HTML but it looks a lot like it. Differences can be seen when looking closely though (for example className in JSX vs class in "normal HTML"). JSX is just syntactic sugar for JavaScript, allowing you to write HTMLish code instead of nested React.createElement(...) calls. * The components we create are injected in App component (as a child component) and finally the App component script is injected in index.html file. * Hence the components will be child of App component. * The class based react component can be created by extending Component * It has render method which has to return valid JSX expression (JSX looks very similar to HTML, but it’s not) * The JSX expression then gets compiled into JS (its equivalent createElement() method) |

## OVERVIEW : CREATING REACT COMPONENT



|  |  |
| --- | --- |
| 1. **FUNCTIONAL COMPONENTS** : referred to as "presentational", "dumb" or "stateless" components | const cmp = () => { return some JSX } |
| 1. **CLASS-BASED COMPONENTS**: referred to as "containers", "smart" or "state-full" components   It has render method which has to return valid JSX expression (JSX looks very similar to HTML , but it’s not).The JSX expression gets compiled into JS | class Cmp extends Component { render () {  return some JSX  } } |

## UNDERSTANDING JSX

* The JSX expression looks very much like HTML. The JSX expression is finally compiled to JS [<https://babeljs.io/repl>]

JSX COMPILATION

|  |  |  |  |
| --- | --- | --- | --- |
| **JSX** | | **COMPILED JSX** | |
| <h1 className="heading">Hello</h1> | | React.createElement("h1", { className: "heading"}, "Hello"); | |
| <p className="heading"><span className="spanStyle">Hello</span></p> | | "use strict";  React.createElement("p", { className: "heading" }, React.createElement("span", { className: "spanStyle"}, "Hello")); | |
| * The JSX expression compiled as a JS code. The compiled JSX has createElement method which created the DOM elements in the browser. So we can use React.createElement method too to render the HTML element. * ***We use “className” instead class, because the class is a reserved word in JS*** | | | |
| **UNDERSTANDING React.createElement METHOD** | | | |
| EXAMPLE : - **React.createElement('h1', {}, 'My First React Code');** | | | |
| * The first one is the type of element we're creating, in this case an <h1> tag. This could also be another React component. If we're creating an HTML element, we pass in the name as a string, just like we did above. If we're creating a React component, we pass in the variable that the component is assigned to. * The second argument is an object containing properties ('props' in React terms) that get passed to the component.. * Finally, the last argument is the children of that component. This can be a quoted string like shown above, in which case the content will be interpreted as text. However, we can also pass in a reference to another component, allowing us to nest elements and components within each other (we'll get to that in a bit). * To render the component. We do this using ReactDOM.render(). This takes two arguments: the first one being the thing we want to render (our title element), and the second one is a target DOM node to render things into. | | | |
| **APP JS** | **INDEX.JS** | | **INDEX.HTML** |
| import React, { Component } from "react";  class App extends Component {  **render**() {  **return** React.createElement("div", { className: "heading" }, "Test Data");  }  }  export default App; | import React from 'react';  import ReactDOM from 'react-dom';  import App from './App';  ReactDOM.render(<App />, document.getElementById('root')); | | <div id="root">  </div> |
|  | | |

JSX REFERING JS VARIABLES

|  |  |
| --- | --- |
| **JSX CAN ALSO REFER THE JAVASCRIPT VARIABLE** | const App = () => {  const message = "Hello World";  return <div><h1 >{message}</h1> </div>;  }; |
| **JSX CAN ALSO REFER THE JAVASCRIPT FUNCTIONS** | function getMessage() {  return "Hello World!";  }  const App = () => {  return <div><h1>{getMessage()}</h1> </div>;  }; |

JSX RESTICTIONS

* We should not use Javascript reserved word. For example we use “className” instead class , because the class is a reserved word in JS.
* There should be one parent element in a JSX expression. But there is a way to overcome this limitation

|  |  |
| --- | --- |
| **VALID** | **INVALID** |
| <div>  <p>Hello</p>  <p>World</p>  </div> | <div>  <p>Hello</p>  </div>  <p>World</p> |

## CREATING A DEMO REACT APP – BLOG POST APP

|  |  |  |
| --- | --- | --- |
|  | | Note : We will be using a third party for   * **CSS**: <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/semantic.min.css"> [TO BE ADDED IN index.html] * **AVATAR IMAGES**:   + <img src="https://source.unsplash.com/random" />   + OR BY USING FAKER NODE MODULE : <https://www.npmjs.com/package/faker>     - To install the node module : npm i faker * HTML : <https://semantic-ui.com/views/comment.html>   To create the blogpost component   1. We will one Parent Componet (Comments) . 2. The Parent componet has an **approval form component** 3. The Approval Component has child componenst “CommentDetails” . |
|  |  |
| **CONNECTING THE COMPONENTS**  **EXPORT**  export default CommentDetails;  **IMPORT**  import CommentDetails from './CommentDetails';  (This is the relative path of the JS file without having .js extension) |  |

### COMMUNICATION BETWEEN COMPONENTS

|  |  |
| --- | --- |
|  | * React uses a “props” system to communicate between a Parent component to child/ nested components. |

### CODE IMPLEMENTATION

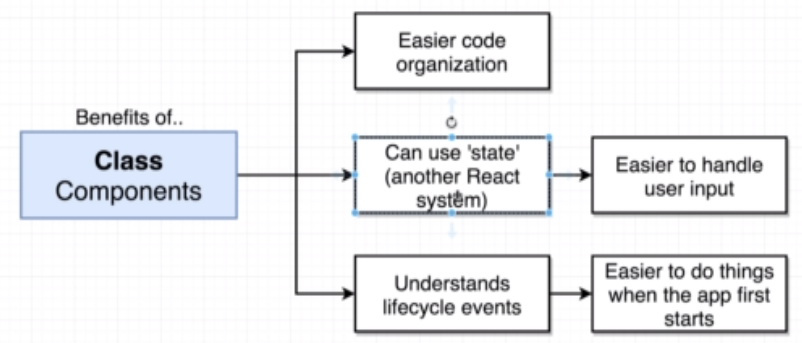
|  |  |
| --- | --- |
| **Index.js**   * The App Component is a functional component which as Child Component “ApprovalCard” component * The ApprovalCard has in turn one more child component “CommentDetails “. | import React from 'react';  import ReactDOM from 'react-dom';  import faker from 'faker';  import CommentDetails from './CommentDetail';  import ApprovalCard from './ApprovalCard';  const App = () => {  return <div className="ui container comments">  <h3 className="ui dividing header">Comments</h3>  <div class="ui cards">  <ApprovalCard>  <CommentDetails name={faker.name.firstName()} />  </ApprovalCard>  <ApprovalCard>  <CommentDetails name={faker.name.firstName()} />  </ApprovalCard>  </div>  </div>;  };  ReactDOM.render(<App />, document.getElementById("root")); |
| **ApprovalCard.js**   * The ApprovalCard component can access its child component using “{props.children}” | import React from 'react';  const ApprovalCard = (props) => {  return <div className="card">  <div className="content">  {props.children}  </div>  <div className="extra content">  <div className="ui two buttons">  <div className="ui basic green button">Approve</div>  <div className="ui basic red button">Decline</div>  </div>  </div>  </div>;  };  export default ApprovalCard; |
| **CommentDetail.js**   * The value pass as an attribute from the parent can be accessed using “{props.*attributeName*}” | import React from 'react';  const CommentDetail = props => {  return <div className="comment">  <a href="/" className="avatar">  <img alt="avatar" src="https://source.unsplash.com/random" />  </a>  <div className="content">  <a className="author">{props.name}</a>  <div className="metadata">  <span className="date">Today at 5:42PM</span>  </div>  <div className="text">  How artistic!  </div>  <div className="actions">  <a className="reply">Reply</a>  </div>  </div>  </div>;  }  export default CommentDetail; |

## CLASS BASED COMPONENTS

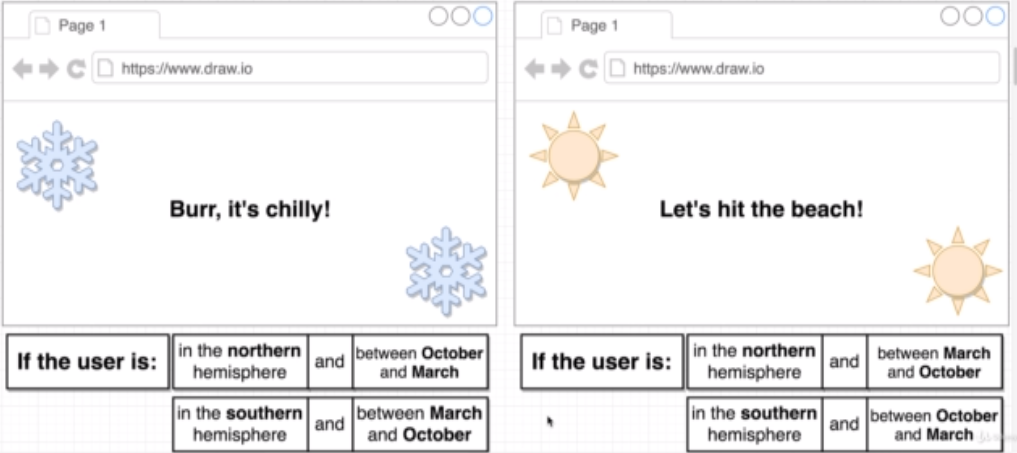
**HOW FUNCTIONAL COMPONENT AND CLASS COMPONENT ARE DIFFERENT?**

|  |  |
| --- | --- |
|  | * In early phase of react – functional componenst was only responsible for rendering the JSX. * It didn’t had access to life cycle methods and state system * Earlier - The Class based component can able to access life cycle hooks and had capability to do state management * With recent version of React - Functional components has become powerful due to ***Hook System which enabled it to manage the component’s state as well.*** |
|  |

### BENEFITS OF CLASS BASED COMPONENT



### BUILD APP USING CLASS BASED COMPONENT



***We will be building a demo app to see the usability of Class based component***

**APP OVERVIEW -** The UI will show up a season information based on the User’s Geolocation and the Month.

|  |  |
| --- | --- |
| **STEP NEED TO CREATE THE DEMO APP** | **SOLUTION** |
|  | **Step 1: USER PHYSICAL LOCATION**  To fetch the Geolocation we will be using the Geo location API (<https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API> )  **EXAMPLE**  window.navigator.geolocation.getCurrentPosition(  position => console.log(position), 🡨 SUCCESS CALLBACK  error => console.log(error.message) 🡨 ERROR CALLBACK  ); |
| In the browser we will be getting a pop-up which ask for a permission to allow the location. If permission granted – The code will end-up with success call back – other wise error callback |

**GOTCHAS IN GETTING THE USER LOCATION**

|  |  |
| --- | --- |
|  | **FUNCTIONAL COMPONENT**  import React from 'react';  import ReactDOM from 'react-dom';  const App = () => {  window.navigator.geolocation.getCurrentPosition(  position => console.log(position),  error => console.log(error.message)  );  return <h1>Latitude:</h1>;  };  ReactDOM.render(<App />, document.getElementById('root')); |
|  | **WHAT EXTRA WE GET WITH CLASS BASED COMPONENT?**  ***There is an issue with this implementation – Getting the geolocation is an asych operation. So, by the time the API gives the response, the functional component will return the JSX. There is no way for a functional component to re-render the component again with the updated Latitude value. Due to incapability of State Management the CLASS BASED COMPONENT is the right choice*** |

### CREATING CLASS BASED COMPONENTS

|  |  |
| --- | --- |
|  | **EXAMPLE (Person.js)**  import React from 'react';  import ReactDOM from 'react-dom';  class Person extends React.Component {  render() {  return (  <div>  <p>My name is {this.props.firstName} {this.props.lastName}</p>  </div>  );  }  } |

#### STATE MANAGEMENT IN CLASS BASED COMPONENT

RULES OF STATE

1. The state is a JavaScript object contains the data relevant to a component
2. State is usable only with class-based components.
   1. Note –With the recent React development state can be updated too using functional component using “hooks” system.
3. **Updating the state causes the component to re- render (render method is called as many times we change the state)**
4. **The State must be initiazed when the component is created (May be in constructor)**
5. **STATE CAN ONLY BE UPDATED USING THE FUNCTION setState()**

#### STATE MANAGEMENT IN SEASONS APP

|  |  |
| --- | --- |
| import React from 'react';  import ReactDOM from 'react-dom';  class App extends React.Component {  constructor(props) {#1  super(props);  this.state = { lat: null, errorMessage: '' };#2  window.navigator.geolocation.getCurrentPosition(  position => {  this.setState({ lat: position.coords.latitude }); #3  },  error => {  this.setState({ errorMessage: error.message }); #4  }  );  }  render() {  if (this.state.errorMessage !== '' && this.state.lat === null)  return <h1>Error: {this.state.errorMessage}</h1>; #6  else if (this.state.lat !== null && this.state.errorMessage === '')  return <h1>Latitude: {this.state.lat}</h1>; #5  return <h1>Loading!</h1>;  }  }  ReactDOM.render(<App />, document.getElementById('root')); | * #1 🡪 The class-based component has constructor which has “props” value which in turn calls the super(props) * #2 🡪 Intiallization of state object (in constructor)   #3 🡪   * The setState() function is called when response is received * It updates the state of the component and re- renders the component as well. * #4 – Set State in error scenario * #5 - Updates the value when the state of the “lat” property is updated. * In render method we are doing the condition rendering |

### LIFE CYCLE METHOD OF A CLASS BASED COMPONENT

|  |  |
| --- | --- |
|  | 1. The class based component go through a series of life cycle methods during the rendering 2. **componentDidMount**() 🡪 It gets called once when the component is visible on the screen 3. **componentDidUpdate() 🡪 It get called everytime the state of the component is updated i.e when the component’s setState() method is called.** 4. **As the render() method is called also as number of times we update the state of the component.** |
|  |

#### RESPONSIBILITIES OF LIFE CYCLE METHODS

|  |  |
| --- | --- |
|  | |
| **componentDidMount** | * This is the right spot to get the initial data loading like initiating the XHR call |
| **componentDidUpdate** | * This method is called whenever we update the state of the component * Usability can be   + making a XHR call with every user input or auto complete feature.   + Components getting a new set of **props** from the parent component |
| **Render** | This method should have responsibility to return a JSX |
| **componentWillUnMount** | To some clean up |

**CODE REFACTORING – LEVARAGING LIFE CYCLE METHODS**

|  |  |
| --- | --- |
| class App extends React.Component {  constructor(props) {  super(props);  this.state = { lat: null, errorMessage: '' };  window.navigator.geolocation.getCurrentPosition(  position => {  this.setState({ lat: position.coords.latitude });  },  error => {  this.setState({ errorMessage: error.message });  }  );  }  render() {  if (this.state.errorMessage !== '' && this.state.lat === null)  return <h1>Error: {this.state.errorMessage}</h1>;  else if (this.state.lat !== null && this.state.errorMessage === '')  return <h1>Latitude: {this.state.lat}</h1>;  return <h1>Loading!</h1>;  }  } | class App extends React.Component {  state = { lat: null, errorMessage: '' };  componentDidMount() {  window.navigator.geolocation.getCurrentPosition(  position => this.setState({ lat: position.coords.latitude }),  error => this.setState({ errorMessage: error.message })  );  }  render() {  if (this.state.errorMessage !== '' && this.state.lat === null)  return <h1>Error: {this.state.errorMessage}</h1>;  else if (this.state.lat !== null && this.state.errorMessage === '')  return <h1>Latitude: {this.state.lat}</h1>;  return <h1>Loading!</h1>;  }  } |
| * In the code refactoring the XHR call in the constructor has neen moved to componentDidMount() method * The initialization of state object can be done without constructor without using “**this**”. They are identical | |

#### DEFAULT PROPERTY

|  |  |
| --- | --- |
|  | * The userName props can be passed from the Parent component . * But If it is not passed – it will take the value passed in the “defaultProps” |

### ADDING CSS TO COMPONENT

#### EXTERNAL CSS

|  |  |  |
| --- | --- | --- |
| **FOLDER STRUCTURE** | **IMPORT IN COMPONENT** | **HTML** |
|  | import React, { useState } from "react";  import "./Person.css";  const person = () => {  const [personState, setPersonState] = useState({  firstName: "Max",  lastName: "Doe"  });  return (  <div>  <p className="personTitle">  My name is {personState.firstName} {personState.lastName}  </p>  </div>  );  };  export default person; | * When we add any css in the component level it is automatically added in head on the HTML(globally) |
| **Person.css**  **.personTitle {**  **color: red;**  **}** |

#### INLINE CSS

|  |  |
| --- | --- |
| import React, { Component } from "react";  class Dog extends Component {  state = {  color: "brown",  breed: "Labra"  };  render() {  const inlineStyle = {  color: "red",  backgroundColor: "green"  };  return (  <div>  <h1 style={inlineStyle}>  Dog is {this.state.color} in color and of {this.state.breed} Breed  </h1>  </div>  );  }  }  export default Dog; | * The inline styling can be given using a JS object and finally using the object as style={inlineStyle} , on HTML element * Note the CSS in the JS must be given in camel case e.g.   background-color attribute in CSS is written as backgroundColor in JS. |

### OPERATORS

#### CONDITIONAL OPERATOR

|  |  |
| --- | --- |
| **USING TERNARY OPERATOR**  import React, { Component } from "react";  class Dog extends Component {  state = {  color: "brown",  breed: "Labra",  showContent: true  };  toggleContent = () => {  const currentState = this.state.showContent;  this.setState({  showContent: !currentState  });  };  render() {  return (  <div>  { this.state.showContent ? (  <h1>  Dog is {this.state.color} in color and of {this.state.breed} Breed  </h1>  ) : null}  <button onClick={this.toggleContent}>Click</button>  </div>  );  }  }  export default Dog; | * We are |

### FINAL APP DESIGN AND CODE

##### VIEW

GITHUB REPO : <https://github.com/avishekhsinhaRepo/React-Applications.git>

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

## HANDLING EVENTS AND USER INPUTS

Application Name : Image Search

|  |  |
| --- | --- |
|  |  |

### BINDING EVENTS

|  |  |
| --- | --- |
| class SearchBar extends React.Component {  onInputChange(event) {  console.log(event.target.value);  }  render() {  return (<div className="ui segment"><form className="ui form" >  <div className="field">  <label>Image Search</label>  <input type="text" name="first-name" placeholder="Search" onChange={this.onInputChange} />  </div>  </form></div>);  }  }  export default SearchBar; | **Note – We do not put “()” while calling the event handler – otherwise it will be invoked when the component is loaded.** |

#### CONTROLLED VERSUS UNCONTROLLED INPUTS

## HTTP REQUEST IN REACT

* In React paradigm the HTTP request can be accomplished using
  + XMLHTTPRequest – Native JS Code
  + Third party library called **Axios**

|  |  |
| --- | --- |
| INSTALLING AXIOS | npm install axios --save |

**WHEN TO IMPLEMENT HTTP REQUEST**

