Contents

[REACT JS 2](#_Toc132960116)

[WHAT IS REACT? 2](#_Toc132960117)

[REACT CONCEPTS AND CREATING REACT COMPONENTS 2](#_Toc132960118)

[BASIC STEPS TO CREATE REACT COMPONENT 3](#_Toc132960119)

[UNDERSTANDING JSX 3](#_Toc132960120)

[JSX COMPILATION 3](#_Toc132960121)

[HTML TO JSX 4](#_Toc132960122)

[ADDING CSS LIBRARY WITH NPM 4](#_Toc132960123)

[JSX REFERING JS VARIABLES 5](#_Toc132960124)

[JSX RESTICTIONS 5](#_Toc132960125)

[REACT FRAGMENT 5](#_Toc132960126)

[CREATING A DEMO REACT APP – BLOG POST APP 5](#_Toc132960127)

[COMMUNICATION BETWEEN COMPONENTS – PARENT TO CHILD 7](#_Toc132960128)

[HANDLING EVENTS AND USER INPUTS 8](#_Toc132960129)

[BINDING EVENTS 8](#_Toc132960130)

[CHILD TO PARENT COMMUNICATION 9](#_Toc132960131)

[COMPONENTS 9](#_Toc132960132)

[STATES IN COMPONENTS 9](#_Toc132960133)

[CHILD TO PARENT COMMUNICATION 10](#_Toc132960134)

[HTTP REQUEST IN REACT 10](#_Toc132960135)

[APP OVERVIEW- SEARCH IMAGES 11](#_Toc132960136)

[useReducer 12](#_Toc132960137)

# 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 | |
|  |  | | * 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). * It has one root div which will become the parent of the react app * For SPA we don’t edit this html, because the root div is the place holder where we will mount our react app * We still can add CSS files to this file * <div id="root"></div> |

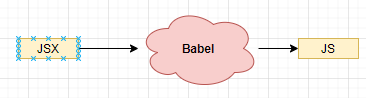
|  |
| --- |
| **src** - This folder has an App component which will be the parent component of all custom component we will create a in react manifest.json file give the basic functionality of PWA to react app. |

## 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) |

## 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> |
|  | | |

### HTML TO JSX

#### RULES

1. **All prop names should follow camelCase**
2. **Number attributes use curly braces**
3. **Boolean true can be written with just the property name. ‘False’ should be written with curly braces.**
4. **The “class” attribute is written as ‘className’**
5. **In-line style are provided as an object**

**EXAMPLE (RULE 1 & 2)**

|  |  |
| --- | --- |
| **HTML** | **EQUIVALENT JSX** |
| <input maxlength=”5”/> | <input maxLength={5}/> |
| <form autocapitialize/> | <form autoCapitialize/> |
| <form novalidate/> | <form noValidate/> |

**EXAMPLE (RULE 3)**

|  |  |
| --- | --- |
| **HTML** | **EQUIVALENT JSX** |
| <input spellcheck=”true”/> | <input spellCheck/> |
| <input spellcheck=”false”/> | <input spellCheck={false}/> |

**EXAMPLE (RULE 5)**

|  |  |
| --- | --- |
| <a style=”text-decoration:’none’; padding:’5px’;”/> | <a style**={{textDecoration:’none’,padding:’5px’}}/**> |

### ADDING CSS LIBRARY WITH NPM

|  |  |
| --- | --- |
| * Let’s say we are going to use “**bulma**” css library in our project- <https://bulma.io/> * INSTALL BULMA - **npm install bulma** | import "bulma/css/bulma.css";  function App() {  return (  <section className="hero is-primary">  ....  </section>  </div>  );  }  export default App; |

### ADDING CUSTOM CSS

|  |  |
| --- | --- |
|  | function App() {  return(  ...JSX  )  }  export default App; |

### 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

* All prop names follow camelCase
* Number attributes use curly braces.
* Boolean “true” can be written with just property name. ‘False’ should be written with curly braces
* Inline-styles are provided as objects
* 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> |

### REACT FRAGMENT

A common pattern in React is for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM.

|  |  |
| --- | --- |
|  | * <https://reactjs.org/docs/fragments.html> |

## 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 – PARENT TO CHILD

|  |  |
| --- | --- |
|  | * 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; |

## HANDLING EVENTS AND USER INPUTS

Application Name : Image Search

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

### BINDING EVENTS

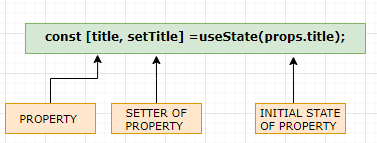
|  |  |
| --- | --- |
| function ClickHandler() {  const handleClick = () => {  console.log("Click");  };  return (  <div>  <button onClick={handleClick}>Click</button>  </div>  );  } | INLINE THE HANDLER  function ClickHandler() {  return (  <div>  <button  onClick={() => {  console.log("Click");  }}> Click </button>  </div>  );  } |
| * Note – We do not put “()” while calling the event handler – otherwise it will be invoked when the component is loaded. | |

### CHILD TO PARENT COMMUNICATION

## COMPONENTS

### STATES IN COMPONENTS

* Data that changes as user interacts with our application. When the data changes, react will update the content on the screen automatically



|  |  |
| --- | --- |
| import { useState } from 'react';  const ExpenseItem = (props) => {  **const [title, setTitle] =useState(props.title);**  const **changeTitle** =() =>{  setTitle("Updated");  }  return (  <h2>{title}</h2>  **<button onClick = {changeTitle}>Change Title</button>**  </div>  );  }  export default ExpenseItem; | * In the above example – “title” is a state variable which are getting managed using state variable. * The state of “title” property can be managed only by its setters. * ***Calling the setter cause the entire component to re-render and the updated state value.*** |

EXAMPLE 2: The primary focus of the below example is

* **LIST BUILDING IN REACT**
* **IMPORTING IMAGES**

|  |  |
| --- | --- |
| App.js | AnimalShow.js |
| import { useState } from "react";  import AnimalShow from "./AnimalShow";  function getRandomAnimal() {  const animalsList = ["bird", "cat", "cow", "dog", "gator", "horse"];  return animalsList[Math.floor(Math.random() \* animalsList.length)];  }  function App() {  const [animals, setAnimals] = useState([]);  function handleClick() {  const animal = getRandomAnimal();  setAnimals([...animals, animal]);  }  function renderAnimal() {  return animals.map((animal, index) => {  return <**AnimalShow** type={animal} key={index} />;  });  }  return (  <div>  <button onClick={handleClick}>Add Animals</button>  <p>{renderAnimal}</p>  </div>  );  }  export default App; | **import cow from "./images/cow.svg";**  **import cat from "./images/cat.svg";**  **import dog from "./images/dog.svg";**  **import gator from "./images/gator.svg";**  **import horse from "./images/horse.svg";**  **import heart from "./images/heart.svg";**  const svgIcons = {  cow,  cat,  dog,  gator,  horse,  };  function AnimalShow({ type, key }) {  return (  <div>  <img src={svgIcons[type]} alt="animal"/>  <img src={heart} alt="heart"/>  </div>  );  }  export default AnimalShow; |

#### STATE SYSTEM ELEMENT

|  |  |
| --- | --- |
| ADDING ELEMENT | const [colors, setColors] = useState([]);  const updatedColors = [... colors,newColor]  setColor(updatedColors) |
| ADDING ELEMENT ON SPECIFIC LOCATION | const [colors, setColors] = useState([]);  const addColorAtIndex = (newColor, index) => {  const updatedColors = [ ... colors.slice(0, index),  newColor, ... colors.slice(index)  ]  setColors(updatedColors);  }; |
| REMOVING ELEMENT WITH PARTICULAR VALUE |  |
| REMOVING ELEMENT WITH PARTICULAR INDEX | const [colors, setColors] = useState(['red', 'green', 'blue']);  const removeColorAtIndex = (indexToRemove) => {  const updatedColors = colors.filter((color, index)=>{  return index != indexToRemove  });  setColors(updatedColors)  }; |
| MODIFYING ELEMENT WITH PARTICULAR INDEX |  |
|  |  |

### CHILD TO PARENT COMMUNICATION

|  |  |
| --- | --- |
| **PARENT** | **CHILD** |
| function App() {  const handleClick = (term)=>{  // Some code  }  return (  <div className="App">  <SearchBar onSearch={handleClick}/>  </div>  );  }  export default App; | function SearchBar({onSearch}) {  const [term, setTerm] = useState("");  const handleOnSubmit = (event) => {  event.preventDefault();  onSearch(term)  };  const handleOnChange = (event) => {  setTerm(event.target.value);  };  return (  <form onSubmit={handleOnSubmit}>  <input value={term} onChange={handleOnChange} />  </form>  );  }  export default SearchBar; |

## HTTP REQUEST IN REACT

In React paradigm the HTTP request can be accomplished using

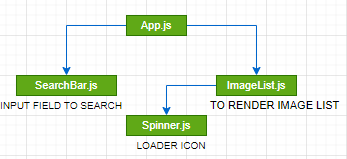
* XMLHTTPRequest or fetch API– Native JS Code
* Third party library called **Axios**

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

### APP OVERVIEW- SEARCH IMAGES

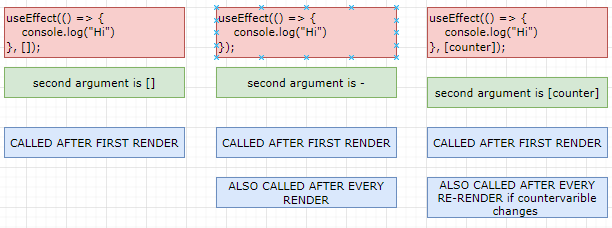
* GitHub repo : <https://github.com/avishekhsinhaRepo/React-Applications/tree/master/pics>

#### COMPONENT HIERARCHY



|  |  |
| --- | --- |
| **APP.JS**  import React from "react";  import SearchBar from "./SearchBar";  **import unsplash from "./api/unsplash";**  import ImageList from "./ImageList";  class App extends React.Component {  state = { images: [], isLoading: false };  onSearchEvent = async (term) => {  this.setState({ isLoading: true });  const response = **await unsplash**.get("/search/photos", {  params: {  query: term,  } });  this.setState({ images: response.data.results,isLoading: false });  };  render() {  return (  <div className="ui container" style={{ marginTop: "10px" }}>  <SearchBar onSearch={this.onSearchEvent} />  <ImageList imageList={this.state.images} isLoading={this.state.isLoading} />  </div>  )  }  }  export default App; | **SearchBar.js**  import React from 'react';  class SearchBar extends React.Component {  state= {term:''}  onFormSubmit=(event)=>{  event.preventDefault();  this.props.onSearch(this.state.term);  }  render() {  return (  <div className="ui segment">< form className="ui form" onSubmit={this.onFormSubmit}>  <div className="field">  <label>Image Search</label>  <input type="text" name="term" placeholder="Search" value={this.state.term} onChange={(e) => {this.setState({term: e.target.value})}} />  </div>  </form></div>);  }  }  export default SearchBar; |
| **Spinner.js**  const Spinner = (props) => {  return (  <div>  <div className="ui active dimmer">  <div className="ui text loader">{props.message}</div>  </div>  <p></p>  </div>  );  };  export default Spinner; |
| **ImageList.js**  import Spinner from "./Spinner";  const ImageList = (props) => {  const searchedImagesList = props.imageList.map((image) => {  return (  <img key={image.id} src={image.urls.thumb} alt={image.alt\_description} />  );  });  if (!props.isLoading) {  return <div>{searchedImagesList}</div>;  } else {  return <Spinner message="Fetching Images For You..." />;  }  };  export default ImageList; |
| **unsplash.js**  **This is a custom http client – getting called in App.js** |
| import axios from "axios";  export default axios.create({  baseURL: 'https://api.unsplash.com/',  headers: {  Authorization: "Client-ID  6S0qY-SVrLYgCbys4NiqHBXaiY0bKwywf7Ydl6FcmbU"  }  }); |

#### useEffect HOOKS



## COMMUNICATION WITH CONTEXT SYSTEM

## useReducer

* Used to manage the state in a react component.