React JS

Prerequisites:

- HTML
- CSS
- JavaScript ES6

Topics

- What is React?
- Why React?
- Difference between React, Angular, Vue etc.
- Features and Limitations
- Understanding React Application Architecture
- Components, Props and State
- Styling with Material UI
- Debug React Apps
- Life Cycle
- React Hooks
- Ajax Calls / Http Requests
- Integration with Server Side [MERN]
- Routing
- Forms and Validations
- Deploying
- Redux
- Error Handling

- Unit Testing
- Web Pack
- React Native
- End to End Integration

Which one I have to choose? [Angular or React]

Projects – Which are completely integrated with lot of server-side interactions and every concept is handled at server level only they are looking for good UI to make faster interactions then “React JS”.

Projects – Which need to control their application flow both client side and server side – Angular.

React JS is library

Angular is Framework

Mill of Users are accessing some content using content. Only few interactions like message, chat. Not complete EPR. [Not Querying – Transactions, complex operations] – React is Great

Twitter, Instagram

Java - Spring

PHP - Cake PHP, Code Igniter

Python - Django, Flask

Ruby - Ruby on Rails

.NET - ASP.NET MVC

JavaScript - SPINE, Angular

MVC – Model View Controller

React is only with “View” UI – Interactions

Web Development 3+ Years: Angular, React

Java, .NET, PHP, Python 3+ years [Angular, React]

What is React and How it Works?

- React JS is a library.
- Client Side we have languages, libraries and frameworks.
- **Client-Side Languages**
    - JavaScript
    - TypeScript
- **Client-Side Libraries**
    - jQuery
    - RxJS

    - jQlite
    - React JS
- **Client-Side Frameworks**
    - Angular JS
    - Knockout JS
    - Backbone JS
    - Angular

What is difference between Language and Library?

- Language requires functions defined explicitly to handle various interactions.
- Language requires lot of references.
- Library is a set of factories.
- Factory is a set of pre-defined functions.
- You can implement the existing function and define the functionality.
- Library reduces the compatibility issues.
- React JS is a library.

What is difference between Library and Framework?

- Library can build application.
- Library is passive [not-active]
- Library can’t control the application flow.
- It requires browser events to make library active.

- Framework is a software architectural pattern that provides set of libraries to build application and can also control the application flow.

What is React JS?

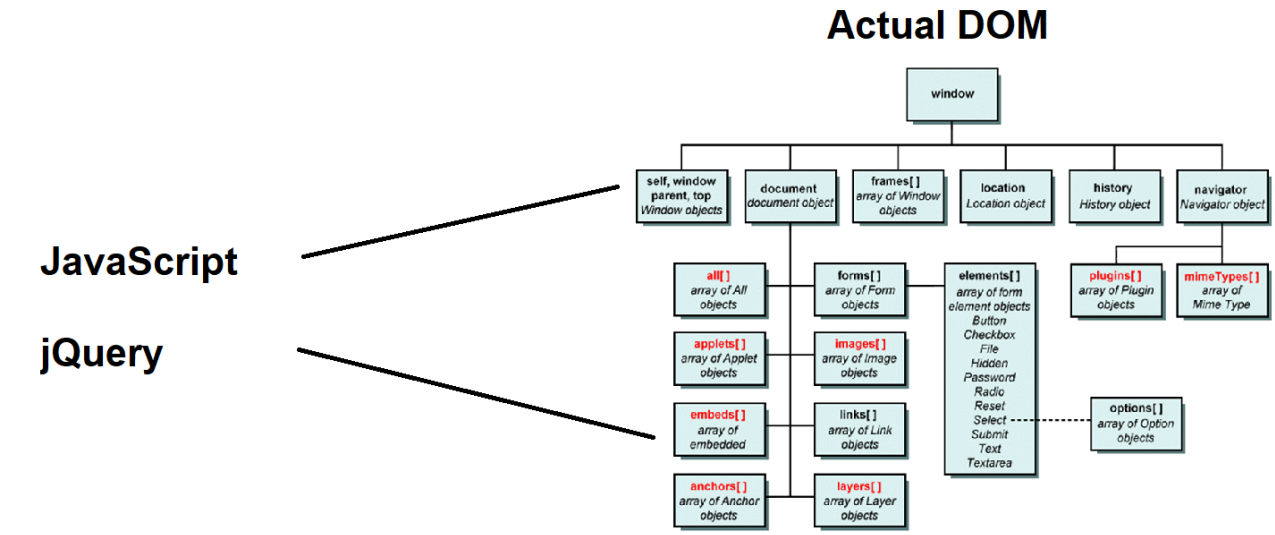
- React JS is a JavaScript library.
- Building User interfaces and handling User interactions client side.
- Facebook – Jordan Walke – for building SPA.
- 2013 Initial release.
- 2020 – 17.0.1 latest stable version.

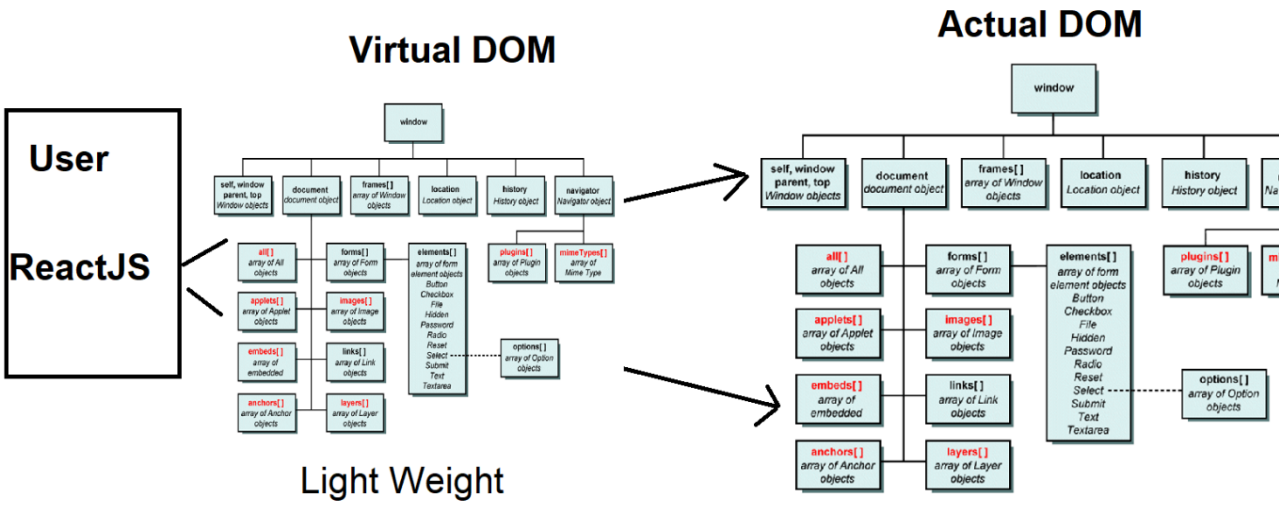
It is not designed for what you are using. Hence limited library.

React JS library requires several external libraries to integrate and handle functionality.

Features of ReactJS

- ReactJS uses virtual DOM that makes the user experience better.





- Faster Interactions.
- React uses JSX, which is easy to handle the manipulation on DOM elements.
    - JSX is faster that JavaScript DOM manipulations.
    - Logic and markup can be defined in single file.
    - It is easy to create template.
    - Template comprises of presentation and logic.
- React uses “One Way Data Binding”

    - Data binding is a technique used to update the component data to View [UI]
    - Any change in component data will update to UI. [HTML]
    - Changes made in UI are not directly update back to component, they are handled on virtual DOM.
- Component based architecture.
    - It is an alternative for legacy type library.
    - It is more asynchronous
    - It loads only what is required for the situation.
    - It improves the performance of application.
    - It makes your library modular.

Concerns with ReactJS

- Very high pace of development.
- Problem for developers to catch and update the pace of developing features.
- Poor documentation.
- Poor SEO
- JSX [Presentation with Logic] – Insecure content and ignore
  XSS [Cross Site Scripting Attacks]
- Need lot of external libraries to handle various interactions.

Setup Environment for ReactJS

Install Node JS

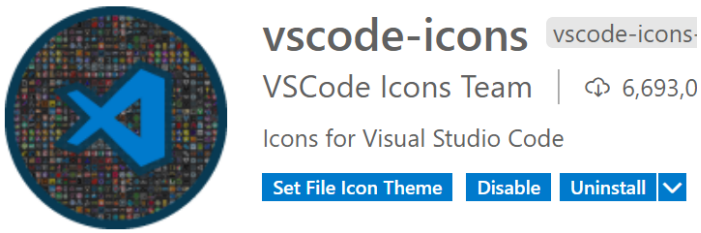
- Node JS provides a package manager NPM.
- Package manager is a tool used to install and configure library required for our application.
- There are several package managers like: Yarn, NPM, Bower, Ruby Gems, NuGet etc.
- Installing Node JS on your PC will get you NPM.
  https://nodejs.org/en/download/ [Make sure that Node JS version 10+]
- After installing Node JS check the version from command prompt
  > node -v
  > npm -v

Install Visual Studio Code [IDE – Integrated Development Environment]

- IDE provides environment to build, debug, test and deploy application.
- You can use various editors: “editorconfig.org”
- Download and install VS Code
  https://code.visualstudio.com/

- Go to “Customize” and “Install Support for JavaScript”
- Go to “Extensions” and Install following
    - **Live Server:** It required to preview your application.

    - **Vscode-icons:** It gives suitable icons for specific file extensions and names, so that developer can easily identify the file type.



    - **IntelliSense for CSS Class Name:** It provides help on CSS

Creating a new Project

- Create a new folder for project by name “ReactApp” on your PC
  “C:\ReactApp”

- Open your folder in Visual Studio Code: File Menu -> Open Folder -> C:\ReactApp
- Open Terminal: Go to Terminal Menu and select "New Terminal" or Ctrl + ` [backtick]
- Run the command "npm init"
  C:\ReactApp> npm init
- It will generate "package.json" that comprises of project meta data.

  package name: (reactapp) reactapp

  version: (1.0.0)

  description: This is our first react application

  entry point: (index.js)

  test command:

  git repository:

  keywords: react shopping, smart shopping

  author: Naresh I Technologies

  license: (ISC) MIT

- **Install React Modules into Project**
  C:\ReactApp>npm install --save react
  C:\ReactApp>npm install --save react-dom
- Installed library will be maintained in "**node_modules**" folder

Create a React Application

> npm install -g create-react-app

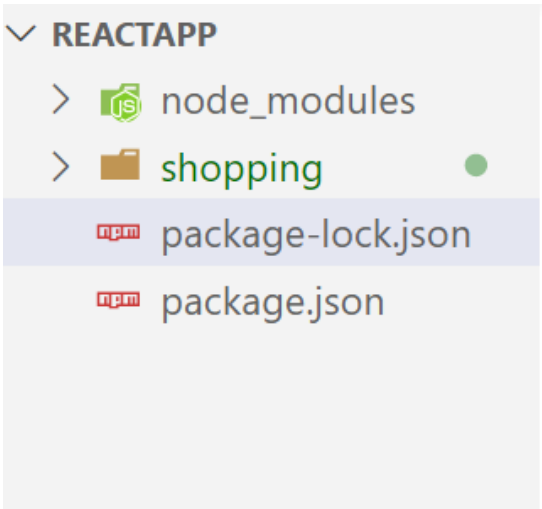
> create-react-app shopping

> Change to shopping
C:\reactapp\shopping>

> **npm start** [Start the react application]

React Workspace

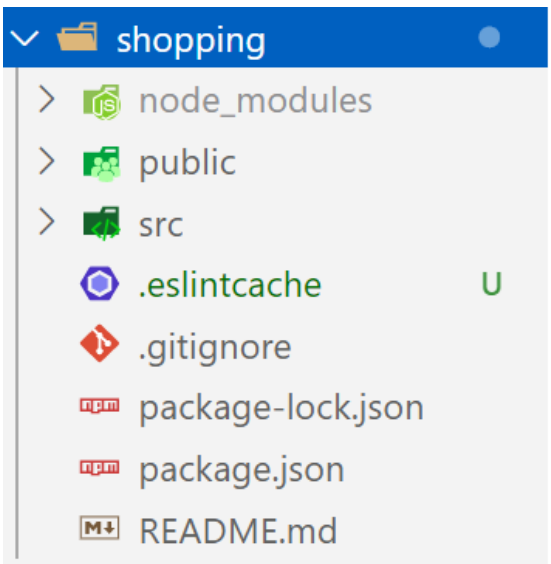
- It comprises of shared libraries for all your ReactJS projects.
- In workspace you can maintain multiple projects
- Workspace comprises of



File / Folder	Description
node_modules	- It comprises of project “dependencies” and

	"devDependencies". - The library required for your react application is in "node_modules". - These dependencies can be shared to all projects in workspace.
package.json	- It comprises of information about the packages installed in your workspace.
packagelock.json	- It comprises of packages meta data.
Shopping	- It is your react application in the workspace.

The files and folders of React Application



File / Folder	Description

node_modules	It contains local dependencies.
public	It comprises of all our project static files, like Images, html, text etc.
src	It comprises of all our project dynamic files like, CSS, JavaScript etc. [LESS, SASS]
.eslintcache	It is used to track and identify the issues in your JavaScript code. It can configure the rules for JavaScript.
.gitignore	It specifies the dependence to ignore while deploying of GIT.
Packagelock.json	Local dependencies meta data
Package.json	Local dependencies information.
README.md	Help document.

Files in Public folder

favicon.ico	Shortcut icon used for bookmarking.
Index.html	It is the start up page.
manifest.json	It comprises of metadata. Information about react application. Browsers can

	understand your application by accessing details from manifest.
Robots.txt	It comprises of User Agent details, which is information about the browsers supported.

Files in SRC folder

App.css	It comprises of global styles used for app component.
App.js	It comprises the logic required for default component that is “app” component. Every project contains set of several components. React application by default comes with “app” component. Application starts with “app” component. Every component comprises of 3 files - .css : Styles - .js : Logic (JSX – HTML and JS) - .spec/test.js : It is test file

App.test.js	It is the test file for “app” component.
Index.css	Styles for Index Page
Index.js	JSX for Index Page
Setuptests.js	It configures the testing environment for application testing.
reportWebVitals.js	Error reporting

What is Toolchain for React?

- Toolchain is a set of programming tools that are used in performing a complex software development task or to create a software product.
- In general, “toolchain” is a set of tools used for building/developing an application.
  Ex:
  Language : used for writing the logic
  Compiler : for translating in machine format.
  Debugger : to identify and fix the bugs. [error/issues]
  Testing : a framework for testing
  Deployment : tool for deploying application on production.

What is Recommended toolchain for ReactJS Application?

- Package Manager [NPM, Yarn]
- Bundler [Webpack, Parcel]
- Compiler [Babel – differential loading]

Package Manager	Installing and Maintaining Dependencies EX: NPM, Yarn
Bundler	Writing modular code and bundle it together for application. Ex: Webpack, Parcel
Compiler	It allows to handle differential loading. Library will be classified according to browser. [Legacy , Modern]

What are the popular toolchains used for ReactJS?

- Neutrino
- Nx
- Parcel
- Razzle

When we need the individual toolchains?

- When we are trying to integrate ReactJS into existing application.

What is NPX?

- It is a tool that comes with NPM.
- It can directly create react application.
- We don't require "create-react-app" to install if we are using "npx".

> npx create-react-app yourAppName

How to add "ReactJS" into existing application?

- Create a new folder for project "C:\HtmlReactApp"
- Open your folder in "Visual Studio Code"
- Open Terminal [Ctrl + `] or Open folder location in command prompt
- Run the command
  > npm init
  {

"name": "htmlreact",

"version": "1.0.0",

"description": "HTML Website with React",

"main": "index.js",

```
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "HTML",
    "Application",
    "React",
    "Application"
  ],
  "author": "Naresh-I-Technologies",
  "license": "MIT"
}
```

- “package.json” file is created
- Add a new folder into project by name “public”
- Add a new file into public folder “index.html”
- You can install dependencies for project using “npm”
  Ex:
  > npm install bootstrap
- It will generate “node_modules” and “package-lock.json”
- To integrate “ReactJS” into your page you need

    - react.development.js
    - react-dom.development.js
- You have get these file from official ReactJS website CDN links https://reactjs.org/docs/cdn-links.html
- Add following scripts into head section

<script crossorigin src="https://unpkg.com/react@17/umd/react.development.js">
</script>
<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js">
</script>

- Add Babel CDN to your page https://babeljs.io/docs/en/babel-standalone

<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

Create React JS Component

- Components are building blocks for creating applications.
- Technically component is a template.
- It comprises of design and functionality, which you can inject and use in any page.

- Components will provide re-usable templates for React JS application.
- Components are classified into 2 major types
    - Function Component
    - Class Component
- Component is designed by using JSX [JavaScript Extension]
- React JS component can be embedded into Page or can be defined in module.
    - Embedded Technique
      Component is designed and used in the same page.
    - Module Technique
      Component is designed in a separate JavaScript [“.js”]
      Component is linked to any page.

Ex: React JS components embedded into Page

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML React App</title>
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
```

```
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx">
      ReactDOM.render(
        <p>Welcome to React JS</p>,
        document.getElementById('root')
      );
    </script>
    <script type="text/javascript">
      function bodyload(){
        alert("React Integrated");
      }
    </script>
  </head>
  <body onload="bodyload()">
    <h2>HTML React App</h2>

    <div id="root">
```

```
    </div>
  </body>
</html>
```

Note: The MIME type of React JS “JSX” script must be “text/jsx” or “text/babel”

Syntax:

<script type=”text/jsx”> </script>

<script type=”text/babel”></script>

Embedded Component:

- In embedded technique the component is designed in the page.
- It is faster in access.
- It reduces the load time.
- It reduces the number of requests made to page.
- Code reusability issues
- Extensibility issues
- Maintainability and testability are the issues.

Module Technique:

- Component is designed in a separate JavaScript file ".js"
- JavaScript file is considered as a Module.
- Module comprises of classes, functions etc.
- Easy to extend
- Easy to re-use
- Easy to test
- Using an external file will always increase the number of requests made to page and also the page load time.

Ex:

- Go to "src" folder and add a new file "hello.js"
- Add the following code into "hello.js"

```
ReactDOM.render(
  <p>Welcome to React JS - External
Component</p>,
   document.getElementById('root')
 );
```

- Link to your HTML page.

```
<!DOCTYPE html>
<html>
```

```
  <head>
    <title>HTML React App</title>
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin
src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx"
src="../src/hello.js"></script>
  </head>
  <body>
    <h2>HTML React App</h2>

    <div id="root">

    </div>
  </body>
</html>
```

JSX

- JSX is a statically-typed, object-oriented programming language.
- It compiles to standalone JavaScript.
- It can use all feature of JavaScript.
- It finally compiles to JavaScript.
- It is a combination of presentation and logic.
- It makes inline technique more effective.
- It makes JavaScript more expressive and effective.

What is statically typed language?

- Generally, if the type of variable is known at the compile time then it is known as statically typed.
- Dynamically typed is determined during run time.

Install JSX on your PC:

> npm install -g jsx

Creating JSX program

- JSX programs must have the extension ".jsx"
- You can compile and run JSX by using
  > jsx --run program.jsx

JSX Program Structure

- Program comprises of a class with main function.

- JSX uses all statically typed content.
- JavaScript is not statically typed.
- JSX requires the data type to be defined for every variable and parameter you define.
- JSX can use the JavaScript data types
    - Primitive Types
    - Non-Primitive Types / Object Types

JSX Primitive:

- number
- string
- boolean
- null
- undefined

JSX Variables

- var
- let
- const

Syntax:

var variableName : DataType;

let variableName: DataType;

const variableName: DataType = value;

JSX also supports Type Inference?

- It is a technique where JSX identifies the data type according to the value and assigned the type to variable.

Ex:

var variableName = value;

var username = “John”; //username : string

var salary = 45000.55; // salary : number

JSX Non-Primitive Type:

- They are object types
- Typically JSX Object Types are
  - Date()
  - Array()
  - Map()
  - Object

Syntax:

Var d = new Date();

Var a = new Array();

Var o = { }

JSX Operators

JSX Expression

JSX Statements

JSX functions

JSX Classes

JSX Modules

JSX Program Structure

- Every JSX program must contain "_Main" class
- "_Main" is the start-up class in JSX.
- In JSX program there can be any number of classes but it starts with only "_Main" class.
- Class can contain any type of functions.
- JSX "_Main" class must contain "main()" function.
- "main()" function is the entry point for program.
- The "main()" function must be configure with string array type args.

Note: Module level entry point is "_Main" class and function level entry point is "main()" function.

Create a new JSX Program

Demo.jsx

```
class _Main
{
```

```
    static function main() : void{
        log "Welcome to JSX";
    }
}
```

Compile and run JSX Program

- Open Terminal
  > jsx --run demo.jsx

JSX OOPS

Contracts in OOP:

- A contract defines rules for designing any component.
- Contracts are specified by using “interface”.
- Interface can contain only rules.
- It must have only declaration not implementation.
  Syntax:

```
interface InterfaceName
{
   Name : string; // valid
   Price  : number = 45600.55; // invalid – rules can’t have implementation
}
```

- Interface can also define function declaration, which is implemented later.
- Interface functions can't have definition.
  Ex:
  ```
  interface IProduct
  {
      Name:string;
      Price:number;
      Stock:boolean;
      Total():number;
      Print():void;
  }
  ```
- Every rule defined in contract is mandatory to implement.
- You can't define any new implementation if it is not in contract.

JSX Class

- Class is a program template.
- Class is used as entity, model or template.
- Class comprises of data and logic.
- JSX class have following characteristics.
    - All JSX classes are derived from a base class "Object".
    - Object is a built-in class of JSX.

- Class can extend another class by using “extends” keyword.
- JSX class can be defined with following attributes

abstract	It is used when any method in class is abstract.
final	It declares that the calls may not be extended.

- JSX class can contain
  - Member Variables and
  - Member Functions

Ex:

```
class Product
{
   var _name : String;                          → Variable
   function constructor(name: string){          → Constructor Member Function
      this._name = name;
   }
   function Print(): void {                     → Member Function
      log "Product Name : " + this._name;
   }
```

}

- Class variable in JSX can be defined with var, const, and let.
    - Member variable in a JSX class can be defined with
        - abstract : If a variable is using expression, which needs implementation.
        - static : It uses continuous memory.
- Class member function in JSX can be defined with following attributes

Function Attribute	Description
abstract	It declares that the function should be overridden by a class that extends the current class.
final	It declares that function may not be overridden.
static	It declares that function is static. It uses continuous memory.
override	It declares that the definition of the function is overriding and definition is extending.

Ex:

```
interface ProductName {
   abstract function PrintName():void;
```

```
}
abstract class ProductPrice
{
   function PrintPrice():void {
      log "Price = 45000";
   }
}
class Product extends ProductPrice implements
ProductName {
   override function PrintName():void {
      log "Name = Samsung TV";
   }
}
class _Main {
   static function main(args: string[]): void {
      var tv = new Product();
      tv.PrintName();
      tv.PrintPrice();
   }
}
```

Functions and Closures in JSX

- JSX supports all JavaScript functions and their manipulations.
- Functions with parameters.
- Function with rest parameters.

Ex:

```
function PrintList(...list){
    for(var item of list) {
        document.write(item + "<br>");
    }
}
PrintList("Samsung TV","Mobile","Shoe");
```

- Every function can have only one rest parameter.
- Rest parameter must be the last parameter in formal list.

- Function recursion.
- Anonymous function.
- Function closures.
- Arrow functions [LAMBDA] () parameters => return value

  Ex:

```
var hello = (name)=> `Hello ! ${name}`;
document.write(hello("john"));
```

- **JSX** functions are “first-class objects” and they have static types.
- You can define closures using “function” expression or “function” statement.
- Typically function closures are used to implement callbacks.

Ex:

```
class _Main
{
   var userName = "Hello ! John";
   function constructor(){
     var hello = function():void {
         log this.userName;
      };
      hello();
   }
   static function main(args: string[]):void {
      var obj = new _Main();
   }
}
```

JSX Modules

- JavaScript module systems are “Common JS, AMD [Asynchronous Module Distribution]” etc.
- Module comprises of set of classed which are exported and imported into another library.
- JSX provide built-in modules to handle various interactions.
    - timer.jsx
    - js/web.jsx etc.

- JSX also allows to create and configure custom modules to build library for react application.

Ex:

```
import "timer.jsx";
class _Main
{
  static function main(args: string[]):void {
      Timer.setTimeout(function():void {
         log "Hello ! JSX";
      }, 2000);
      Timer.setTimeout(function():void{
         log "Welcome to JSX";
      }, 5000);
  }
}
```

Summary

- Variables
- Data Types
- Operators
- Statements
- Interface
- Class

- Member Variables
- Member Functions
- Modules

JSX with React JS

- JSX will allow to configure any expression and store in a variable reference.
- The variables that are handling JSX expression must be initialized.
- If you are referring to any variable that is not defined with value the it returns “undefined”.
- Hence JSX variables must be initialized.
- Initialization of variables is done by using “const”.
- JSX expressions are directly defined without enclosing in quotes.
- The dynamic values are embedded into expression by using “{ }”

Ex:

- Go to “src” folder and add a new file “app.js”
- Add following code into “app.js”

```
const title = 'Amazon Shopping';
const element = <h1>{title}</h1>;

ReactDOM.render(
   element,
```

```
    document.getElementById('root')
);
```

- Link app.js to index.html page

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML React App</title>
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin
src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
  </head>
  <body>
    <div id="root">
```

```
        </div>
    </body>
</html>
```

Configuring React Elements

Note: JSX expression allows only single line in any variable reference.

Ex:

```
const  element = <div>
                    <h2> Welcome </h2>
                    </div>;       //Invalid
```

You can define expression in multiple lines by enclosing in “( )”

Ex:

```
const  element = (
                    <div>
                    <h2> Welcome </h2>
```

```
</div>
);
```

Note: You can define multiple lines in JSX expression but all lines must be defined under one element.

Ex: **Invalid**

```
const element = (
                <h1> Amazon </h1>
                <p> Online Shopping </p>
                );
```

You can’t have individual elements in “( )” all must be enclosed in one element.

Ex: **Valid**

```
const element = (
                <div>
                <h1> Amazon </h1>
                <p> Online Shopping </p>
                </div>
                );
```

- Adding an additional container for multi lines my effect the hierarchy of your DOM.
- You can use an empty element representation.

  “

  <> starting

  </> ending

  “

Ex: **Best Technique**

```
const element = (
                    <>
                    <h1> Amazon </h1>
                    <p> Online Shopping </p>
                    </>
                    );
```

Note: JSX will not allow void element syntax.

Ex:

```
const element = <img src={pic}>;        // Invalid
const element = <img src={pic}></img>  // valid
const element = <img src={pic} />          // valid
```

Ex: app.js

```
const pic = 'shoe.jpg'
const element = (
        <>
        <h2>Amazon Shopping</h2>
        <p>Online Shopping Site</p>
        <img src={pic} />
        </>
        );
ReactDOM.render(
   element,
   document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>HTML React App</title>
      <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
```

```
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx" src="../src/app.js"></script>
  </head>
  <body>
    <div id="root">

    </div>
  </body>
</html>
```

Note: JSX dynamic values can be bound only to the properties of element, you can't bind to attributes.

FAQ: What is difference between "attribute" and "property"?

Attribute	**Property**
Immutable	Mutable
Can't change its state.	Can changes its state.
Statically defined	Dynamically defined
Used for Tags	Used for Elements

<img src> :src is attribute

Var pic = new Image();

pic.src= “” : src is property

- HTML tags have attribute and HTML Elements have properties.
- **Every HTML tag attribute is not available as property.**
  Ex: table have height as attribute but height is not available as property.

- Every HTML tag attribute is not available with the same name as property.
  Ex: table have “class” as attribute but it is used as “className” dynamically.
- React JS will not be able bind any dynamic data to attribute while creating element dynamically, you have to always bind to property.

Ex:

App.js

```
const pic = 'shoe.jpg';

const effect = 'container';
```

```
const element = (
            <div className={effect}>
            <h2 align="center">Amazon Shopping</h2>
            <p align="center">Online Shopping Site</p>
            <img src={pic} width="100" height="100" />
            </div>
            );
ReactDOM.render(
   element,
   document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>HTML React App</title>
      <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
```

```
    <script crossorigin
src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.
min.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
    <style>
        .container {
            width: 400px;
            height: 200px;
            border:2px solid darkcyan;
            box-shadow: 3px 4px 5px darkcyan;
            border-radius: 20px;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div id="root">
```

```
        </div>
    </body>
</html>
```

JSX allows to create element and configure the properties dynamically:

- Elements are created dynamically and added to UI.
- User can add the components in UI according to state and situation.
- Dynamically elements can be created by using the react method **“React.createElement()”**

Syntax:

```
React.createElement(
  “elementName”,
  { property : value, … },
  “default text”
)
```

Ex:

App.js

```
const pic = 'shoe.jpg';
const element = React.createElement(
   'div',
    {className: 'container', align: 'center'},
   'Welcome to React JS'
)
ReactDOM.render(
   element,
   document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>HTML React App</title>
      <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
      <script crossorigin
src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
```

```
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.
min.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
    <style>
        .container {
            width: 400px;
            height: 200px;
            border:2px solid darkcyan;
            box-shadow: 3px 4px 5px darkcyan;
            border-radius: 20px;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div id="root">

    </div>
</body>
```

```
</html>
```

Creating Nested Elements and rendering

App.js

```
const pic = 'shoe.jpg';
const element = React.createElement(
    'div',
     {className: 'container', align: 'center'},
    'Welcome to React JS',
     React.createElement(
         'h2',
         {align: 'center'},
         'JSX Introduction'
     ),
     React.createElement(
        'p',
         null,
        'React Presentation'
     ),
     React.createElement(
```

```
        'img',
        {src: 'shoe.jpg', width: 100, height: 100}
    )
)
ReactDOM.render(
   element,
   document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>HTML React App</title>
      <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
      <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
      <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

```
        <script type="text/jsx"
src="../src/app.js"></script>
        <style>
            .container {
                width: 400px;
                height: 220px;
                border:2px solid darkcyan;
                box-shadow: 3px 4px 5px darkcyan;
                border-radius: 20px;
                padding: 20px;
            }
        </style>
    </head>
    <body>
        <div id="root">

        </div>
    </body>
</html>
```

You can configure elements in Object Style

- You can use JavaScript object type to configure elements and render into React Virtual DOM.
- Object comprises of Key and value.
- The keys required for designing element
    - type : It defines the element type
    - props : It defines the properties [object]. Properties will allow all JS properties.

Ex:

```
const element = {
   type: 'div',
   props : {
      className: 'container',
      children: 'Welcome to React'
   }
};
```

Rendering the Elements

- The elements configured for React are rendered into “Virtual DOM”.
- The “ReactDOM” method “render()” is used to create a virtual DOM and render your element into the DOM.
  Syntax:

ReactDOM.render(What [element], Where[Target]) : It is rendering element into virtual DOM.

- **ElementName**: Specifies the element to render.
- **Target Container**: Specifies the id of element where the resulting markup must be rendered.

Ex:

```
const element = (
   <div className='container'>
     <h2 align='center'>React JS Elements</h2>
   </div>
);
ReactDOM.render(
   element,
   document.getElementById('root')
);
```

- **You can use a Query Selector of JavaScript to access the element by id and use for rendering the React element into virtual DOM.**

Ex:

App.js

```
const element = (
   <div className='container'>
     <h2 align='center'>Welcome to React JS</h2>
   </div>
);
const target = document.querySelector('#root');
ReactDOM.render(element, target);
```

Ex:

App.js

```
const element = (
   <div className='container'>
     <h2 align='center'>Hello ! React JS</h2>
   </div>
);
const target = document.getElementById('root');
ReactDOM.render(element, target);
```

You also use “React.Component” class to render any content:

Syntax:

```
class  Demo  extends React.Component {
    render() {
        return “<h2> You Content </h2>”
    }
}
```

React JS Components

- Components are the building blocks of any React application.
- Typically, a react application is built with components.
- Component is built with UI and logic as a template.
- You can reuse the components in application.
- You can customize according the requirements.
- React JS application can use pre-defined components and also allows to create custom components.
- Pre-defined components like
    - Material-UI

    - Telerik
- Custom components are designed by using JavaScript functions and classes.
- Technically a component is React JS can be
    - JavaScript function
    - JavaScript Class

Functional Components

- The components designed by using JavaScript functions are known as Functional Components.
- Every functional component is by default anonymous type in memory.

  Syntax:

  ```
  function()
  {
    // content to render.
  }
  ```
- Anonymous function requires a reference name to access.

  Syntax:

  const f1 = function() { };
- You can render the functional component into Virtual DOM by using the reference as “element”

  Syntax:

  <f1 />

- DOM allows only an element, hence the functional component is configured as element.

Note:

- The custom components must not collide with HTML DOM elements.
- The component name must not be the same as HTML DOM element.
- Every component must follow “PascalCase or lowercase” in naming.
- Lowercase is suggested only for elements in browser.
- PascalCase is suggested for component in React to render in virtual DOM.
- Pascal Case: Every word first letter must be a capital letter.

Ex:

App.js

```
const Titletext = () => <h2>Welcome to React JS</h2>;


ReactDOM.render(
  <Titletext />,
  document.getElementById('root')
```

);

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML React App</title>
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
  </head>
  <body>
    <div id="root">


    </div>
  </body>
```

```
</html>
```

Ex:

App.js

```
const titleText = "User Login";
const Login = () => (
  <>
    <h3 align="center">{titleText}</h3>
    <dl>
      <dt>User Name</dt>
      <dd><input type="text" className="form-control" /></dd>
      <dt>Password</dt>
      <dd><input type="password" className="form-control" /></dd>
    </dl>
    <button className="btn btn-primary btn-block">Login</button>
  </>
);
ReactDOM.render(
```

```
  <Login />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>HTML React App</title>
      <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
      <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
      <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
      <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

```
        <script type="text/jsx"
src="../src/app.js"></script>
        <style>
            div {
                margin-top: 40px;
                margin-left: 300px;
                justify-content: center;
                align-items: center;
                padding: 10px;
                border:2px solid darkcyan;
                border-radius: 20px;
                width: 300px;
            }
        </style>
    </head>
    <body>
        <div id="root">

        </div>
    </body>
```

</html>

Class Components

- Components can be designed by using ES6 Classes instead of Functions.
- Components designed with classes are known as Class Components.
- Component is defined as a JavaScript Class that extends "**React.Component**" base class.

  Ex:

  class Login → Not a component class

  {

  }

  class Login extends React.Component → Component class

  {

  }
- React.Component base class provides "render()", which you have to implement in your component class.

  Syntax:

  class Login extends React.Component

  {

  render() {

  return "JSX";

  }

}

- Class component is accessed and rendered as element into Virtual DOM.
  Syntax:
  ReactDOM.render(
  <Login />,
  document.getElementById('root')
  );

Ex:

App.js

```
class Login extends React.Component
{
  render(){
    return (
    <>
    <h3 align="center">User Login</h3>
    <dl>
      <dt>User Name</dt>
      <dd><input type="text" className="form-control" />></dd>
      <dt>Password</dt>
```

```
      <dd><input type="password" className="form-control" /></dd>
    </dl>
    <button className="btn btn-primary btn-block">Login</button>
    </>
    );
  }
}
ReactDOM.render(
  <Login />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML React App</title>
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-
```

```
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M
0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.develop
ment.js"></script>
    <script crossorigin src="https://unpkg.com/react-
dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.mi
n.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
    <style>
        div {
            margin-top: 40px;
            margin-left: 300px;
            justify-content: center;
            align-items: center;
            padding: 10px;
            border:2px solid darkcyan;
            border-radius: 20px;
            width: 300px;
```

```
        }
    </style>
  </head>
  <body>
    <div id="root">

    </div>
  </body>
</html>
```

Note: The data for class component can come from Variable or Property

- Variable is immutable
- It is defined in module
- It is accessible directly by reference name.

- Property is mutable
- It is defined in class.
- It is accessible within the class by using “this” keyword.

Ex:

App.js

```
var copyright = "copyright 2021";  // variable
class Login extends React.Component
{
  heading = "User Login";  // property
  render(){
    return (
    <>
    <h3 align="center"> {this.heading} </h3>
    <dl>
      <dt>User Name</dt>
      <dd><input type="text" className="form-control" /></dd>
      <dt>Password</dt>
      <dd><input type="password" className="form-control" /></dd>
    </dl>
    <button className="btn btn-primary btn-block">Login</button>
    <p align="center"><i>{copyright}</i></p>
    </>
```

```
    );
  }
}
ReactDOM.render(
  <Login />,
  document.getElementById('root')
);
```

FAQ: Can a class component have function?

A.No. But it can have a function call.

Ex:

App.js

```
function template(){
  return "React JS Class Component";
}
class Login extends React.Component
{
    render(){
      return <h2 align="center">{template()}</h2>;
```

```
    }
}
ReactDOM.render(
  <Login />,
  document.getElementById('root')
);
```

Functional Component vs Class Component

Functional Component

- Functional Components are referred as “Stateless” [dumb, presentational]
- Functional because they are designed using a JavaScript function.
- Stateless because they don’t hold or manage state.
- It is difficult for transporting data from one component to another.
- It uses lot of round trips.
- It will be slow in access.
- It is more secured.
- It manages the memory.
- Tightly coupled.
- No easy to extend.

Class Component

- It is referred as “Stateful”

- Class of ES6
- It comprises of logic and data.
- It can hold and manage memory
- It is good for transporting data across components.
- It can have local state.
- It is not just a class but a container with constructor, properties, methods, accessors.
- Loosely coupled.
- Easy to extend.
- Maintainability and Testability.

Properties in Component

[React Props]

- A component can accept properties.
- So that component can customize the content that it is going to return.
- It allows the component to change the behaviour.
- You need properties to define behaviour for component, which can change according to situation and state.
- Props allows to customize the functionality of component.
- All properties of react components are read-only.

- Property can’t be modified [its name, structure] but its value can change.

Properties in Functional Components:

- Functional components are defined with properties as parameters.
  Syntax:
  const hello = function(props) {

  };
  props – can be any type: string, number, boolean, object, array etc.
- The props of functional component are accessed by using attribute reference.
  Syntax:
  <hello propertyName=”{propertyValue}” />
- Every property passed into a component have to handle “Key and Value”
- Key refers to the property name
- Value refers to the value to store under specific property name.
- The props that you define can handle any type of data but in a **“object”** form.

Syntax:

```
const hello = obj => <h2> {obj.PropertyName} </h2>;
```

<hello PropertyName={‘Value’} />

- Every property in react component is “**Pure”** type.
- Pure prop can’t change its structure but can change its value.

FAQ: What are Pure properties in React?

- The properties of component are often known as Pure properties.
- They can’t change according to state and situation dynamically.

Ex:

App.js

```
const Product = obj => (
  <div className="card">
    <div className="card-header">
      <h3>{obj.Name}</h3>
    </div>
    <div className="card-body">
      <img src={obj.Photo} width="50%" height="300"></img>
```

```
      </div>
      <div className="card-footer">
         <h4>{obj.Price}</h4>
      </div>
   </div>
);
ReactDOM.render(
  <Product Name={'Nike Casuals'} Photo={'shoe.jpg'}
Price={4500.55} />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML React App</title>
        <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/d
ist/css/bootstrap.min.css" integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M
0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
```

```
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
    <style>


    </style>
  </head>
  <body>
    <div id="root">


    </div>
  </body>
</html>
```

Properties in class components:

- Class will not access props.
- Props are not passed as an argument into the class.
- Properties are configured and accessed by using “this” keyword.
- You have to create a dynamic object using “this” keyword with properties.
  Syntax:
  this.PropertyName
- You can create a local property in class and access by using “this” keyword.
- Properties can be defined with default values.

Ex:

App.js

```
class Product extends React.Component
{
  Name = "Nike Casuals";
  Price = 4500.55;
  Photo = "shoe.jpg";
  render() {
    return (
      <div className="card">
        <div className="card-header">
```

```
            <h3>{this.Name}</h3>
          </div>
          <div className="card-body">
            <img src={this.Photo} width="50%"
height="300"></img>
          </div>
          <div className="card-footer">
            <h4>{this.Price}</h4>
          </div>
        </div>
      )
    }
  }
ReactDOM.render(
  <Product/>,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>HTML React App</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
    <style>


    </style>
</head>
<body>
```

```
        <div id="root">


        </div>
    </body>
</html>
```

- **In order to modify the property value of class you need to create a “props” object for class.**

Ex:

App.js

```
class Product extends React.Component
{
  render() {
    return (
        <div className="card">
          <div className="card-header">
             <h3>{this.props.Name}</h3>
          </div>
          <div className="card-body">
             <img src={this.props.Photo} width="50%"
height="300"></img>
          </div>
```

```
        <div className="card-footer">
          <h4>{this.props.Price}</h4>
        </div>
      </div>
    )
  }
}
ReactDOM.render(
  <Product Name={'Nike'} Photo={'shoe.jpg'} Price={4500.56}/>,
  document.getElementById('root')
);
```

Reusing the components with different values passed into Props

- You can compose any component with another component.
- This allows to reuse the existing components and render in page.

Ex:

App.js

```
class Header extends React.Component
{
  render(){
    return (
      <header className="bg-danger text-white text-
center">
        <h1>Amazon Shopping</h1>
      </header>
    )
  }
}
class MenuContent extends React.Component
{
  render() {
    return (
      <nav>
        <ul>
          <li><span className="fa fa-home"></span>
Home</li>
          <li><span className="fa fa-tv"></span>
Electronics</li>
```

```
        <li><span className="fa fa-bell"></span>
Fashion</li>
       </ul>
     </nav>
   )
  }
}
class Product extends React.Component
{
  render() {
    return (
       <div className="card">
         <div className="card-header text-center">
            <h3>{this.props.Name}</h3>
         </div>
         <div className="card-body text-center">
            <img src={this.props.Photo} width="200"
height="200"></img>
         </div>
         <div className="card-footer text-center">
            <h4>{this.props.Price}</h4>
```

```
        </div>
      </div>
    )
  }
}
class Products extends React.Component
{
   render(){
     return (
       <div className="card-deck">
         <Product Name={'Nike Casuals'} Price={5600.66}
Photo={'shoe.jpg'} />
         <Product Name={'Lee Boot'} Price={4000.55}
Photo={'shoe1.jpg'} />
         <Product Name={'JBL Speaker'} Price={5000.55}
Photo={'speaker.jpg'} />
       </div>
     )
   }
}
class MainContent extends React.Component
```

```
{
  render(){
    return (
      <>
       <Header />
       <section>
        <div className="row">
         <div className="col-3">
           <MenuContent />
         </div>
         <div className="col-9">
           <Products />
         </div>
         </div>
       </section>
       <footer className="text-center">
         &copy; Copyright 2021;
       </footer>
      </>
    )
```

```
  }
 }
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>HTML React App</title>
      <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
      <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
crossorigin="anonymous" >
```

```
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/jsx"
src="../src/app.js"></script>
    <style>
      section {
        height: 600px;
      }
      footer {
        background-color: maroon;
        color:white;
      }
      ul{
        list-style: none;
      }
      li {
```

```
            border:1px solid maroon;
            background-color: maroon;
            padding: 4px;
            margin-top: 10px;
            border-radius: 10px;
            width: 200px;
            color: white;
            font-size: 30px;
          }
        </style>
    </head>
    <body>
        <div id="root">

        </div>
    </body>
</html>
```

Task: Design the above example with functional components

Ex:

App.js

```
const Nav = () =>
 <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
    <button className="navbar-toggler"
          type="button"
          data-toggle="collapse"
          data-target="#NavMenu">
      <span className="navbar-toggler-icon"></span>
    </button>
    <a href="#" className="navbar-brand">
     Amazon Shopping
    </a>
    <div className="collapse navbar-collapse" id="NavMenu">
          <ul className="navbar-nav">
           <li className="nav-item">
             <a className="nav-link" href="#">Home</a>
           </li>
           <li className="nav-item">
```

```
              <a className="nav-link"
href="#">Electronics</a>
            </li>
            <li className="nav-item">
              <a className="nav-link"
href="#">Footwear</a>
            </li>

            <li className="nav-item">
              <a className="nav-link" data-
toggle="modal" href="#login">Login</a>
            </li>
            <li className="nav-item">
               <SearchBox />
            </li>
          </ul>
     </div>
 </nav>

const SearchBox = () =>
<div className="input-group">
```

```
   <input type="text" className="form-control" />
   <div className="input-group-append">
    <button className="btn btn-warning">
     <span className="fa fa-search"></span>
    </button>
   </div>
</div>

const Login = () =>
<div className="modal fade" id="login">
  <div className="modal-dialog">
    <div className="modal-content">
      <div className="modal-header">
        <h3>User Login</h3>
      </div>
      <div className="modal-body">
        <dl>
          <dt>User Name</dt>
          <dd><input type="text" className="form-
control" /></dd>
```

```
          <dt>Password</dt>
          <dd><input type="password" className="form-control" /></dd>
        </dl>
      </div>
      <div className="modal-footer">
        <button className="btn btn-primary">Login</button>
        <button data-dismiss="modal" className="btn btn-danger">Cancel</button>
      </div>
    </div>
  </div>
</div>

class MainContent extends React.Component
{
  render(){
    return (
      <>
       <Nav />
```

```
      <Login />
     </>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>HTML React App</title>
      <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
```

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css" crossorigin="anonymous" >

<script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

<script crossorigin src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx" crossorigin="anonymous"></script>

```
    <script type="text/jsx"
src="../src/app.js"></script>


  </head>
  <body>
    <div id="root">


    </div>
  </body>
</html>
```

Data in React Component

[Data-Driven Components]

- Data for component comes from a service.
- JavaScript based libraries or frameworks usually handle data by using
    - Object
    - Array
    - Map
- **Object:** Represents a pseudo class with set of properties and methods.
- **Map:** Represents a collection with key and value references.

- **Array:** Represents a collection of values stored in sequential order.

FAQ: What is difference between Map and Object?

Object	Map
- Contains default keys	- No default keys
- Can’t know the count of keys	- Can know the length of count.
- Object requires an iterator to read keys and value.	- Map is an iterator.

FAQ: What is Array.map()?

- It uses a call back function that creates a new array with the result of any query.

Ex:

```
<script>
    var products = [
        {Name: "TV", Price: 45000.55},
        {Name: "Mobile", Price: 33000.44},
        {Name: "Shirt", Price: 3100.33}
    ];
    function GetDetails(product) {
```

```
        var details = [product.Name, product.Price].join("-->") + "<br>";
        return details;
    }
    function PrintDetails(){
        document.write(products.map(GetDetails));
    }
    PrintDetails();
</script>
```

Ex:

```
<script>
    var products = [
        {Name: "TV", Price: 45000.55},
        {Name: "Mobile", Price: 33000.44},
        {Name: "Shirt", Price: 3100.33}
    ];
    function GetDetails(product) {
        var details = `${product.Name} - ${product.Price}<br>`;
        return details;
```

```
    }
    function PrintDetails(){
        document.write(products.map(GetDetails));
    }
    PrintDetails();
</script>
```

Find()

Filter()

Ex:

```
<script>
    var products = [
        {Name: "TV", Price: 45000.55},
        {Name: "Mobile", Price: 33000.44},
        {Name: "Shirt", Price: 3100.33}
    ];
    function GetDetails(product) {
      return product.Price >= 30000;
    }
    function PrintDetails(){
        var result = products.filter(GetDetails);
```

```
    for(var item of result) {
        document.write(`${item.Name} - ${item.Price} <br>`);
    }
  }
  PrintDetails();
</script>
```

React Component with Data

[Data Driven Component**]**

- Data for component is provided by properties. [props]
- In order to define data for any component you have to configure properties.
- You can’t directly use the local values of module.
- You can access data from local object of module or from the property of another class by using “aggregation” [Object-to-Object] communication. [Has-A-Relation]

Ex:

App.js

```
var speaker =  {
  name: 'JBL Speaker',
```

```
  price: 6000.55,
  photo: 'speaker.jpg',
  mdf: new Date()
};
class Category
{
   CategoryName = 'Electronics'
}
class Product extends React.Component
{
  product = speaker;
  render(){
     const category = new Category();
     return (
      <div className="card">
        <div className="card-header">
          <h3>{this.product.name}</h3>
        </div>
        <div className="card-body">
```

```
        <img src={this.product.photo} width="200"
height="200" />
      </div>
      <div className="card-footer">
        <h3>{this.product.price}</h3>
        <p>{this.product.mdf.toLocaleDateString()}</p>
        <p>{category.CategoryName}</p>
      </div>
    </div>
    )
  }
}
class MainContent extends React.Component
{
  render(){
    return (
      <>
      <Product />
      </>
    )
```

```
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML React App</title>
        <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
crossorigin="anonymous" >
```

```
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx"
crossorigin="anonymous"></script>
    <script type="text/jsx"
src="../src/app.js"></script>

  </head>
  <body>
```

```
        <div id="root">


        </div>
    </body>
</html>
```

Ex: Multi Level Hierarchy

App.js

```
class ProductDetails
{
  name = 'JBL Speaker';
  price = 5600.55;
  photo = 'speaker.jpg'
}
class CategoryDetails extends ProductDetails
{
  mdf = new Date("2020-02-10");
  CategoryName = 'Electronics';
}
```

```
class Product extends React.Component
{
  render(){
    const prod = new CategoryDetails();
    return (
     <div className="card">
       <div className="card-header">
         <h3>{prod.name}</h3>
       </div>
       <div className="card-body">
         <img src={prod.photo} width="200" height="200"
/>
       </div>
       <div className="card-footer">
         <h3>{prod.price}</h3>
         <p>{prod.mdf.toLocaleDateString()}</p>
         <p>{prod.CategoryName}</p>
       </div>
     </div>
    )
```

```
  }
}
class MainContent extends React.Component
{
  render(){
    return (
      <>
      <Product />
      </>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>HTML React App</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
crossorigin="anonymous" >
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
```

```
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dis
t/js/bootstrap.bundle.min.js" integrity="sha384-
ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7Rs
weNrtN/tE3MoK7ZeZDyx"
crossorigin="anonymous"></script>
    <script type="text/jsx"
src="../src/app.js"></script>


  </head>
  <body>
    <div id="root">


    </div>
  </body>
</html>
```

Presenting Collection

- Data collection is defined in JavaScript Array.
- It is an Array of Objects. [JSON]
- To return elements from an array, JavaScript provides array functions
    - toString()
    - join()

- filter()
- find()
- map()

Ex:

App.js

```
const products = [
  {
    name: "JBL Speaker",
    price: 4500.55,
    photo: 'speaker.jpg'
  },
  {
    name: 'Nike Casuals',
    price: 6000.55,
    photo: 'shoe.jpg'
  },
  {
    name: 'Shirt',
    price: 2000.33,
    photo: 'shirt.jpg'
  },
```

```
  {
    name: 'Earpods',
    price: 4500.55,
    photo: 'earpods.jpg'
  }
];

class Product extends React.Component
{
   productsData = products;
   render(){
     return (
       <div className="card-deck">
         {
           this.productsData.map((product)=> {
             return (
               <div className="card">
                 <div className="card-header text-center">
                   <h3>{product.name}</h3>
                 </div>
```

```
            <div className="card-body text-center">
              <img src={product.photo} width="200"
height="200" />
            </div>
            <div className="card-footer text-center">
              <h3>{product.price}</h3>
            </div>
          </div>
        )
      })
    }
  </div>
 )
 }
}


class MainContent extends React.Component
{
  render(){
```

```
    return (
      <div className="container-fluid">
       <h1 className="text-center">Products
Catalog</h1>
       <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML React App</title>
        <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/d
```

```
ist/css/bootstrap.min.css" integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M
0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">

    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css"
crossorigin="anonymous" >

    <script crossorigin
src="https://unpkg.com/react@17/umd/react.develop
ment.js"></script>

    <script crossorigin src="https://unpkg.com/react-
dom@17/umd/react-dom.development.js"></script>

    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.mi
n.js"></script>

    <script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE
+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dis
t/js/bootstrap.bundle.min.js" integrity="sha384-
ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7Rs
```

```
weNrtN/tE3MoK7ZeZDyx"
crossorigin="anonymous"></script>
        <script type="text/jsx"
src="../src/app.js"></script>

    </head>
    <body>
        <div id="root">

        </div>
    </body>
</html>
```

Ex:

App.js

```
const products = [
  {
    name: "JBL Speaker",
    price: 4500.55,
    photo: 'speaker.jpg'
  },
```

```
  {
    name: 'Nike Casuals',
    price: 6000.55,
    photo: 'shoe.jpg'
  },
  {
    name: 'Shirt',
    price: 2000.33,
    photo: 'shirt.jpg'
  },
  {
    name: 'Earpods',
    price: 4500.55,
    photo: 'earpods.jpg'
  }
];

class Product extends React.Component
{
   productsData = products;
```

```
  render(){
    return(
      <table className="table table-hover">
        <thead>
          <tr>
            <th>Name</th>
            <th>Price</th>
            <th>Preview</th>
          </tr>
        </thead>
        <tbody>
          {
            this.productsData.map((product)=>{
              return(
                <tr>
                  <td>{product.name}</td>
                  <td>{product.price}</td>
                  <td>
                    <img src={product.photo} width="100"
height="100" />
```

```
                </td>
              </tr>
            )
          })
        }
      </tbody>
    </table>
  )
 }
}


class MainContent extends React.Component
{
 render(){
  return (
   <div className="container-fluid">
    <h1 className="text-center">Products
Catalog</h1>
    <Product />
```

```
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Task:

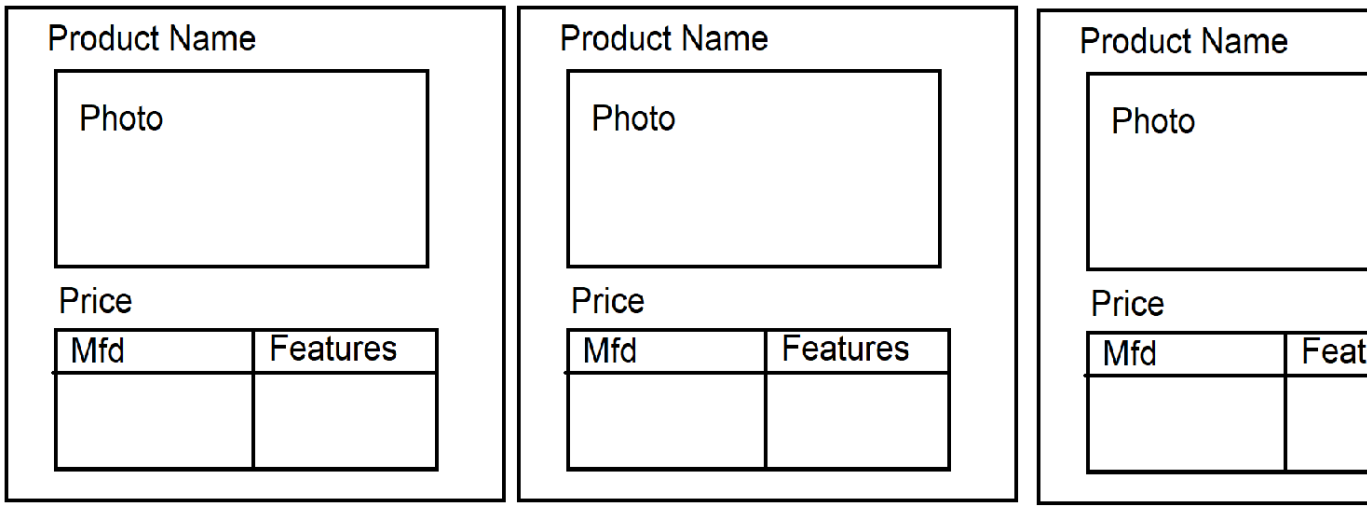
```
const products = [
  {
    name: "JBL Speaker",
    price: 4500.55,
    photo: 'speaker.jpg',
    description: [
            {
                Mfd : Date(“2020-10-20”),
                Features: “some description”
```

```
        }
    ]
},
{
  name: 'Nike Casuals',
  price: 6000.55,
  photo: 'shoe.jpg',
     description: [
        {
           Mfd : Date(“2020-10-20”),
           Features: “some description”
        }
    ]

},
{
  name: 'Shirt',
  price: 2000.33,
  photo: 'shirt.jpg',
     description: [
```

```
            {
                Mfd : Date(“2020-10-20”),
                Features: “some description”
            }
        ]

    },
    {
      name: 'Earpods',
      price: 4500.55,
      photo: 'earpods.jpg',
      description: [
            {
                Mfd : Date(“2020-10-20”),
                Features: “some description”
            }
        ]
    }
];
```



State in React

- Web application uses “HTTP”
- HTTP is a state less protocol.
- It uses the mechanism “Go-Get-Forget”
  GO : It establishes connection with server.
  GET : It gets response for the request.
  FORGET : It cleans up everything. [Removes all traces of current request]
- Stateless nature of HTTP is an advantage for web application, as it manages the memory.
- Stateless nature will be an issue for web application that need continuous connection.
- Sharing information between components will be difficult during runtime.
- State is an Object that holds some information that is provided across multiple requests with information changing according to situation.

- State uses
    - Single Call
    - Single Ton
- **Single Ton:** Object is created for the first request and same object is used across multiple requests.
- **Single Call:** Object is created for every request individually.
- **State maintains object that contains information provided to component across requests.**
- In ReactJS functional components handle data in properties they are state less.
- Class components are stateful
- React JS “Component” base class [**React.Component**] provides a state object that handle data for application.

What is difference between Props and State?

- Props are immutable
- Once you define a prop it can’t be changed.
- State is mutable.
- It can change according to situation.

Rules for Configuring state in React JS application

- State object must be defined with an initial value.

- State must be initialized.
- It must be defined in initialization phase of component.
- Initialization in a class component is defined using “Constructor”
- Hence state is possible only in “Class Component” not in functional component.
- State can’t be changed explicitly.
- State can change its structure implicitly but can’t be modified explicitly.

Where the state object is maintained for React JS application?

- Persistent State Object [Stored Permanently - Cookie]
- In Memory state object [It is in browser memory – Removed when browser is closed]

Configuring State for React JS component

- State in React JS is an object derived from “React.Component” base class.
- You can create a component that implements “React.Component” and allows access to state by using “this” keyword.

Syntax:

```
this.state = { }

class ComponentClass extends React.Component
{
        constructor() {
                //define the state object.
        }
        render(){}
}
```

- As per the rules of OOP in JavaScript the constructor of derived class must have a super call.
- You can set state by using “this.setState()”

Syntax:

```
class ComponentClass extends React.Component
{
        constructor() {
                super();
                //define the state object.
        }
        render(){}
}
```

Ex:

App.js

```
class Product extends React.Component
{
  product = {
    name: 'LG Mobile',
    price: 44000.44
  };
  constructor() {
    super();
    this.state = {
      name: 'JBL Speaker',
      price: 4500.55
    }
  }
  render(){
    return (
      <dl>
        <dt>Name</dt>
        <dd>{this.state.name}</dd>
        <dt>Price</dt>
        <dd>{this.state.price}</dd>
```

```
        <dt>Mobile</dt>
        <dd>{this.product.name}</dd>
        <dt>Price</dt>
        <dd>{this.product.price}</dd>
      </dl>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
       <Product />
      </div>
    )
  }
}
```

```
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Add State to the component

Syntax:

```
class Product extends React.Component
{
   constructor(props){
      super(props);
      this.state = {
        data : [
          {
            name: 'JBL Speaker',
            price: 4500.44,
            photo: 'speaker.jpg'
          }
        ]
      }
```

```
  }
}
```

- Constructor is required to initialize the state.
- Constructor requires props as arguments to pass into super class constructors.
- “props” are used as component properties.

Render Data from State

Ex

App.js

```
class Product extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      data : [
        {
          name: 'JBL Speaker',
          price: 4500.44,
          photo: 'speaker.jpg'
        }
      ]
```

```
    }
  }
  render(){
    return (
      <div className="card">
        <div className="card-header">
          {this.state.data[0].name}
        </div>
        <div className="card-body">
          <img src={this.state.data[0].photo}
width="200" height="200" />
        </div>
        <div className="card-footer">
          {this.state.data[0].price}
        </div>
      </div>
    )
  }
}
```

```
class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h2>Products Catalog</h2>
       <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: Multiple records in State

App.js

```
class Product extends React.Component
```

```
{
  constructor(props){
    super(props);
    this.state = {
      data : [
        {
          name: 'JBL Speaker',
          price: 4500.44,
          photo: 'speaker.jpg'
        },
        {
          name: 'Nike Casuals',
          price: 5600.55,
          photo: 'shoe.jpg'
        }
      ]
    }
  }
  render(){
   return (
```

```
<div className="card-deck">
  {
    this.state.data.map((product)=> {
      return (
        <div className="card">
          <div className="card-header">
            {product.name}
          </div>
          <div className="card-body">
            <img src={product.photo}  width="200" height="200" />
          </div>
          <div className="card-footer">
            {product.price}
          </div>
        </div>
      )
    })
  }
</div>
```

```
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h2>Products Catalog</h2>
       <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: Add and remove Array elements

```
class Product extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      data : [
        {
          name: 'JBL Speaker',
          price: 4500.44,
          photo: 'speaker.jpg'
        },
        {
          name: 'Nike Casuals',
          price: 5600.55,
          photo: 'shoe.jpg'
        }
      ]
    }
```

```
    this.state.data.push({name: 'Shirt', price:
1200.33, photo: 'shirt.jpg'});
    this.state.data.splice(1,1);
  }
  render(){
   return (
    <div className="card-deck">
      {
       this.state.data.map((product)=> {
          return (
           <div className="card">
            <div className="card-header">
             {product.name}
            </div>
            <div className="card-body">
             <img src={product.photo}  width="200"
height="200" />
            </div>
            <div className="card-footer">
             {product.price}
            </div>
```

```
            </div>
          )
        })
      }
    </div>
  )
 }
}

class MainContent extends React.Component
{
 render(){
  return (
   <div className="container-fluid">
    <h2>Products Catalog</h2>
    <Product />
   </div>
  )
 }
}
```

```
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

React JS PropTypes

- In real-time project we need to share our component across applications and teams.
- We can use “props, state, life-cycle hooks, JSX, PropTypes etc.”
- If data for component is coming from API [Services] then the component must understand and restrict the type of data.
- React JS uses “prop-types” object that handles different types of data and also can restrict the data.
- Alternative for “prop-types” is use a language that can restrict the type. [**Type Script]**
- The prop-types are import into React JS application to restrict the data type.
- React JS “PropTypes” are categorized into following groups

Basic Types

Type	Example	Class
String	“John”, ‘John’ , `John`	PropTypes.string
Number	-10, 10, 3.4, 45.55	PropTypes.number
Boolean	true/false	PropTypes.bool
Symbol	Symbol(“literal”)	PropTypes.symbol
Object	{Name: “TV”}	PropTypes.object
Anything	‘John’, true, 10, { }	PropTypes.node

Syntax:

```
Product.propTypes = {
    Id: PropTypes.number,
    Name: PropTypes.string,
    Stock: PropTypes.bool
    Description: PropTypes.node                    // any
}
```

Collection Types

Type	Example	Class
Array	[ ]	PropTypes.array
Array of Numbers	[10, 20, 30]	PropTypes.arrayOf([PropTypes.number])

Enum	[‘NotFound’, ‘Bad Request’]	PropTypes.oneOfType([PropTypes.number])

Syntax:

```
Product.propTypes = {
    Cities: PropTypes.array,
    Products: PropTypes.arrayOf(PropTypes.string)
}
```

Object Types

Type	**Example**	**Class**
Object	{name: ‘TV’}	PropTypes.object
Number Object	{price: 4500}	PropTypes.objectOf(PropTypes.number)
String Object	{name: ‘TV’}	PropTypes.objectOf(PropType.string)

Syntax:

```
Product.propTypes = {
```

```
    Details : PropTypes.Object,
    Price : PropTypes.objectOf(PropTypes.number)
}
```

React Types

- It is used to define JSX type
- It is for configuring element type

Element	<Product />	PropTypes.element

Requiring Types

- Requiring types are used for restricting the values by handling various validations.

Syntax:

```
Product.propTypes = {
        Name: PropTypes.string.isRequired
 }
```

Custom Types

- You can create a function or class to configure a custom type
- You can use the custom type to restrict the value.

Syntax:

```
 Product = {
     Name: PropTypes.string
     Price: PropTypes.number
 }
Products.propTypes = {
       TV : Product(Name, Price)
}
```

What is the Default Type?

- Object

How to define Union of Types?

- A property must allow multiple types but not all types.
- You define union of types by using "PropTypes.oneOfType([Type1, Type2])"

Ex:

App.js

```
const Product = (props) => {
 return(
```

```
<dl>
  <dt>Product Code</dt>
  <dd>{props.code}</dd>
  <dt>Name</dt>
  <dd>{props.name}</dd>
  <dt>Price</dt>
  <dd>{props.price}</dd>
  <dt>Shipped To</dt>
  <dd>
    <ul>
      {
        props.shippedTo.map((val)=>{
          return(
            <li key={val}>{val}</li>
          )
        })
      }
    </ul>
  </dd>
</dl>
```

```
  )
}

Product.propTypes = {
  name: PropTypes.string,
  price: PropTypes.number,
  code: PropTypes.oneOfType([PropTypes.number,
PropTypes.string]),
  shippedTo: PropTypes.arrayOf(PropTypes.string)
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h2>Product Details</h2>
        <Product code={'B101'} name={'Samsung TV'}
price={45000.55} shippedTo={['Delhi','Hyd']} />
      </div>
    )
```

```
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML React App</title>
        <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
crossorigin="anonymous" >
```

<script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/prop-types/15.7.2/prop-types.js" integrity="sha512-Ttnt6nUh/1oV+4T/KWkG/ORvlIEOZohMCT/GIzelxTceoIreR2A1r6zmgFnEvi7GgstoDblmpMO/Gi9CXKTqGQ==" crossorigin="anonymous"></script>

<script crossorigin src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx" crossorigin="anonymous"></script>

```
    <script type="text/jsx"
src="../src/app.js"></script>


  </head>
  <body>
    <div id="root">


    </div>
  </body>
</html>
```

Styles in React Application

- Styles provide attributes to make HTML more interactive and responsive.
- Styles can be configured using
    - Inline Styles
    - Embedded Styles
    - Cascading Style Sheets (CSS)
    - Styling Libraries [Radium, React Materials]

Inline Styles

- Styles are defined for every element in component individually by using “style” prop.

- You can pass any dynamic value into “style” attribute so that it can change dynamically according to state and situation.
  Ex: JavaScript

  document.getElementById(“container”).style.backgroundColor = “red”;
- React JS also uses “camel case” for configuring the styles dynamically.
- Dynamic styles are defined by using a style object.

Syntax: Applying styles using Style object

```
const styleObject = {
   styleAttribute: ‘value’,
   styleAttribute: ‘value
}
<div  style={styleObject}> </div>
```

Syntax: Apply attributes directly to element

```
<div style={ {attribute:value, attribute:value } }> </div>
```

Ex:

App.js

```
const headingStyle = {
  backgroundColor: 'darkcyan',
  color: 'white',
  textAlign: 'center',
  padding: '5px',
  marginTop: '10px'
}
const Product = (props) => {
  return(
    <dl>
      <dt style={{backgroundColor: 'gray', color:
'white'}}>Product Code</dt>
      <dd>{props.code}</dd>
      <dt>Name</dt>
      <dd>{props.name}</dd>
      <dt>Price</dt>
      <dd>{props.price}</dd>
      <dt>Shipped To</dt>
      <dd>
```

```
        <ul>
          {
            props.shippedTo.map((val)=>{
              return(
                <li key={val}>{val}</li>
              )
            })
          }
        </ul>
      </dd>
    </dl>
  )
}

Product.propTypes = {
  name: PropTypes.string,
  price: PropTypes.number,
  code: PropTypes.oneOfType([PropTypes.number,
PropTypes.string]),
  shippedTo: PropTypes.arrayOf(PropTypes.string)
```

```
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h2 style={headingStyle}>Product Details</h2>
        <Product code={'B101'} name={'Samsung TV'}
price={45000.55} shippedTo={['Delhi','Hyd']} />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Embedded Styles

- Embedded styles are defined in “HTML” page by using <style> element.
- It can embed a set of styles, which are allowed to reuse across elements.

Index.html

```
<style>
        dt {
           background-color: gray;
           color:white;
           padding:5px;
        }
     </style>
```

CSS

- Styles are maintained in a separate CSS file and are linked to your document.
- You can also import CSS files into component and use.
- If you are using a modular system for react application then you have to import the CSS files into component.

  Syntax:

  app.css

  selector {

attribute: value
}

app.js

import “./app.css”

- If you are not using a modular system then you have to link all your CSS files into index.html
  <link rel=”stylesheet” href=”../src/app.css”>

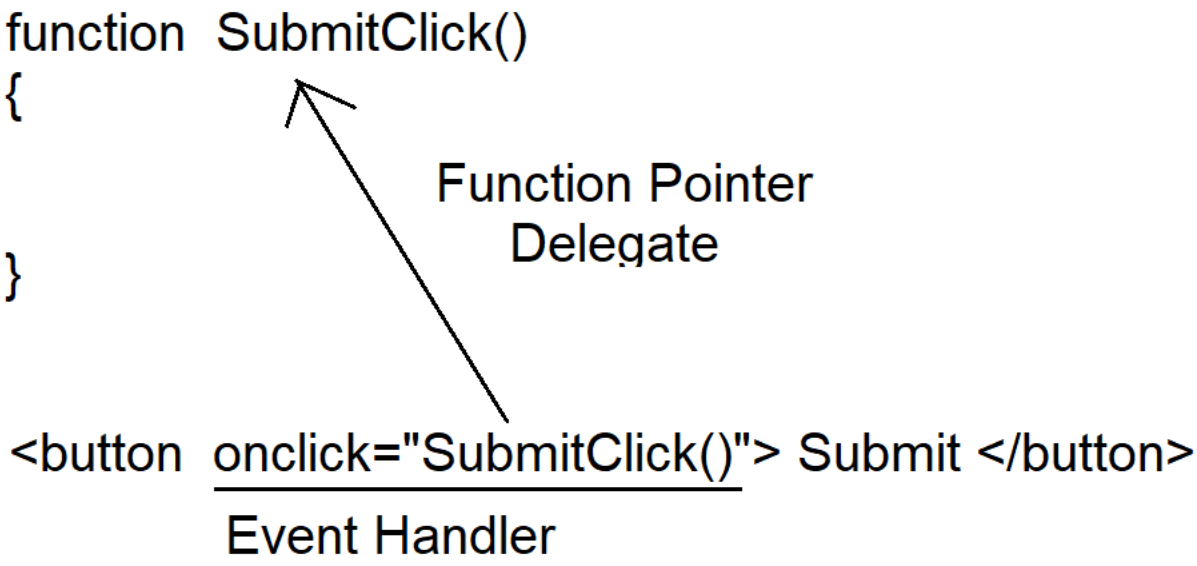
Note: You can also use “less and Sass” for CSS.

Events and Interactions

What is Event?

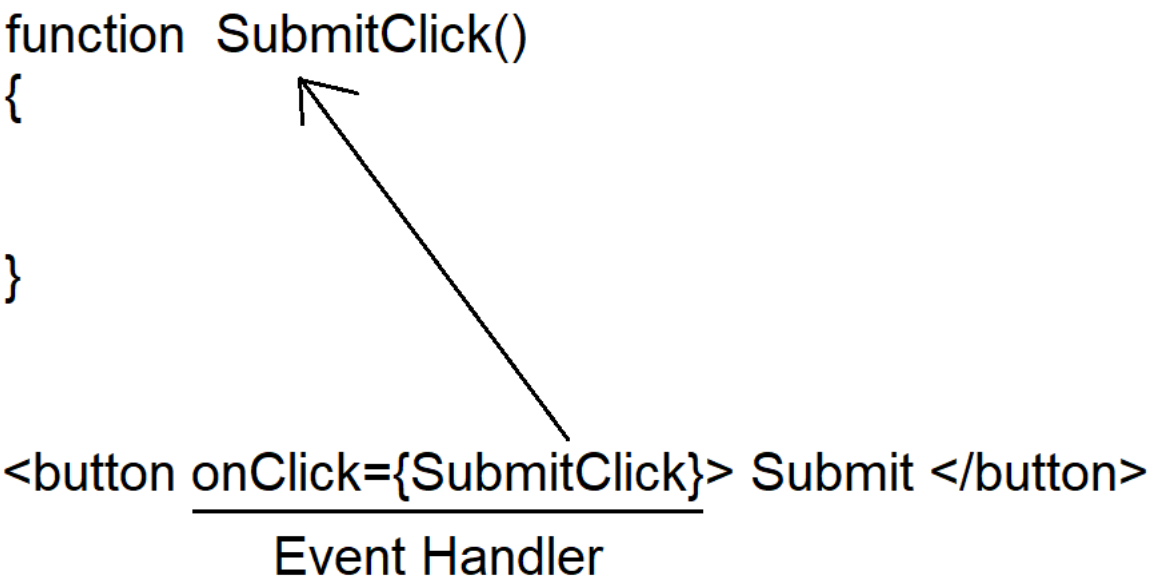
- Event is a message sent by sender to its subscriber in order to notify the change.
- Event follows “Observer”, which is a communication design pattern.
- Event uses “Delegate mechanism” – Function Pointer.

JavaScript

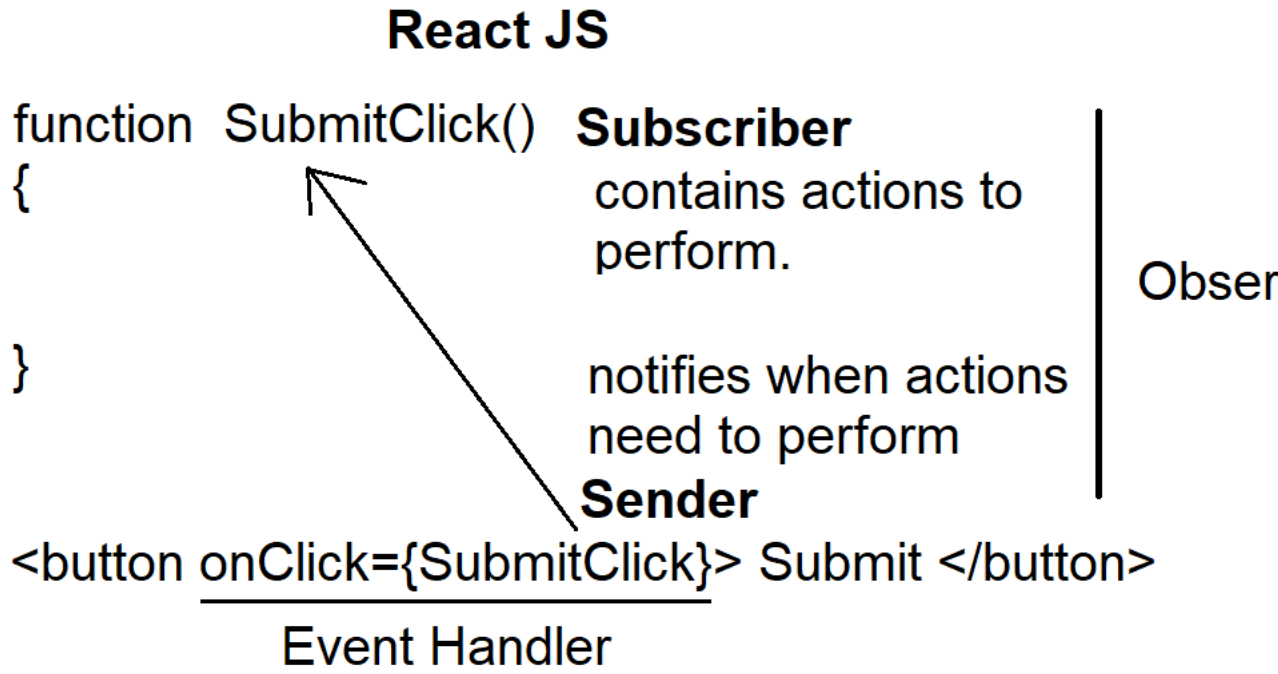


- React JS uses all JavaScript browser events in the same way.
- Events are defined in “Camel Case”.
- Event handler uses function pointer as “JSX expression”

React JS



- Sender notified the changes.
- Subscriber comprises of actions to perform.



- **React JS can use all JavaScript Browser events, which includes**
  - Clipboard Events
  - Composition Events
  - Keyboard Events
  - Focus Events
  - Generic Events
  - Mouse Events
  - Pointer Events
  - Selection Events
  - Touch Events
  - UI Events
  - Wheel Events
  - Media Events

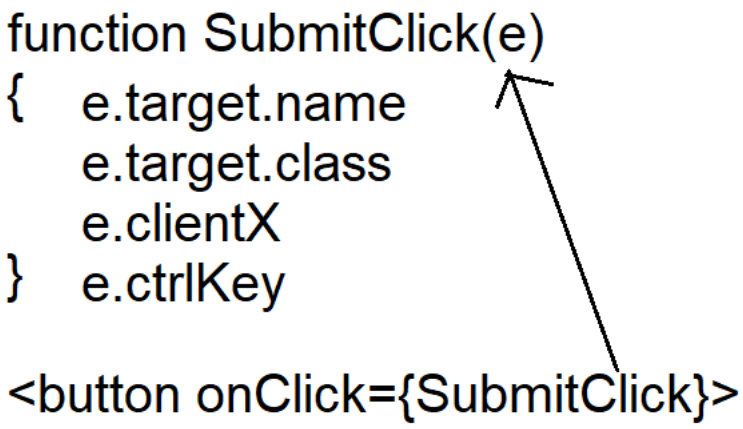
- Image Events
- Animation Events
- Transition Events
- Timer Events etc.

- Java Script events have 2 arguments
  - this : Sends information about the object [related properties and methods]
  - event : Sends information about the event [related properties and methods]

```
function SubmitClick(obj, e)
{
    obj.name        e.clientX
}   obj.class       e.ctrlKey
```

```
<button name="btn" class="btn-primary"  onclick="SubmitCl
```

- React JS events have only “event” argument. [You can’t use this as argument].
- Object related properties can be accessed by using “event.target” object.



Ex: Event in Functional Component

App.js

```
function Product(){
  function InsertClick(){
    alert("Record Inserted");
  }
  return (
    <div>
      <button onClick={InsertClick} name="btnInsert"
className="btn btn-primary">Insert</button>
    </div>
  )
}
```

```
class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: **Event Argument** [event]

App.js

```
function Product(){
  function InsertClick(e){
    alert(`You Clicked At X Position ${e.clientX}\n Button Class Name=${e.target.className}`);
  }
  return (
    <div>
      <button onClick={InsertClick} name="btnInsert" className="btn btn-primary">Insert</button>
    </div>
  )
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
```

```
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Events in Class Component

- In Class component events actions are defined as Methods.
- Event handler points towards specific method.

Ex:

App.js

```
class Product extends React.Component
{
  InsertClick(e){
    alert(`X Position: ${e.clientX}\nName :
${e.target.name}`);
  }
```

```
  render(){
    return(
      <div>
        <button name="btnInsert"
onClick={this.InsertClick} className="btn btn-
primary">Insert</button>
      </div>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
    )
  }
```

```
}


ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML React App</title>
        <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
crossorigin="anonymous" >
        <link rel="stylesheet" href="../src/app.css">
```

```
    <script crossorigin
src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/prop-types/15.7.2/prop-types.js" integrity="sha512-Ttnt6nUh/1oV+4T/KWkG/ORvlIEOZohMCT/GIzelxTceoIreR2A1r6zmgFnEvi7GgstoDblmpMO/Gi9CXKTqGQ==" crossorigin="anonymous"></script>
    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx"
crossorigin="anonymous"></script>
```

```
        <script type="text/jsx"
src="../src/app.js"></script>
    </head>
    <body>
        <div id="root">


        </div>
    </body>
</html>
```

Event Handler

- Event handler is a function pointer.
- Multiple controls can use the same function to handle various functionalities.

Ex:

App.js

```
class Product extends React.Component
{
```

```
  DataOperations(e){
    switch(e.target.value)
    {
      case "Insert":
      alert("Record Inserted");
      break;
      case "Update":
      alert("Record Updated");
      break;
      case "Delete":
      alert("Record Deleted");
      break;
    }
  }
  render(){
    return(
      <div>
        <div className="btn-group">
        <button name="btnInsert" value="Insert"
onClick={this.DataOperations} className="btn btn-
primary">Insert</button>
```

```
        <button name="btnUpdate" value="Update"
onClick={this.DataOperations} className="btn btn-
success">Update</button>
        <button name="btnDelete" value="Delete"
onClick={this.DataOperations} className="btn btn-
danger">Delete</button>
        </div>
        <h2 align="center"></h2>
      </div>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
```

```
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Note: State defined for a component is initialized into component memory.

The methods configured with event handler must bind with the component memory.

Syntax:

{this.buttonClick.bind(this)**}**

Using State in React User Interactions

[Define State and Set State on Event Handling]

- State is required to configure in a component in order to store the data and use between interactions.
- Properties are read-only and will not allow to set value.
- Hence state is required to configure value dynamically.
- State is initialized
  Syntax:
  this.state = {
  property: “value”
  }
- You can modify the state by using “setState()”
  Syntax:
  this.setState(function(state){
  property: state.property = “New Value”
  })
- Events can’t access current state. You have to bind the current context to event.
- If event has to use any of the component related properties and methods then you have to bind with current context.
  Syntax:
  this.eventHandlerName.bind(this)
- You can bind in the initialization phase.
  Syntax:

constructor(props)
{
super();
this.state = { };
this.eventHandler =
this.eventHandler.bind(this);
}

- You can bind at the event handler while configuring event for DOM element.
  Syntax:
  <button onClick={this.eventHandler.bind(this)}>
  Text </button>

Ex:

App.js

```
class Product extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      msg: "Select Database Command"
    }
    this.InsertClick = this.InsertClick.bind(this);
```

```
    this.UpdateClick = this.UpdateClick.bind(this);
    this.DeleteClick = this.DeleteClick.bind(this);
}
InsertClick(){
    this.setState(state=>({
      msg: state.msg = "Record Inserted"
    }));
}
UpdateClick(){
  this.setState(state=>({
     msg: state.msg = "Record Updated"
  }));
}
DeleteClick(){
  this.setState(state=>({
     msg: state.msg = "Record Deleted Successfully.."
  }));
}
render(){
  return(
```

```
    <>
     <div className="btn-group">
       <button onClick={this.InsertClick} className="btn
btn-primary">Insert</button>
       <button onClick={this.UpdateClick}
className="btn btn-success">Update</button>
       <button onClick={this.DeleteClick} className="btn
btn-danger">Delete</button>
     </div>
     <h2 className="text-center">{this.state.msg}</h2>
    </>
   )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
     <div className="container-fluid">
       <h3>Event Handling</h3>
```

```
        <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML React App</title>
        <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
```

```
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css"
crossorigin="anonymous" >

    <link rel="stylesheet" href="../src/app.css">

    <script crossorigin
src="https://unpkg.com/react@17/umd/react.develop
ment.js"></script>

    <script crossorigin src="https://unpkg.com/react-
dom@17/umd/react-dom.development.js"></script>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/prop-
types/15.7.2/prop-types.js" integrity="sha512-
Ttnt6nUh/1oV+4T/KWkG/ORvlIEOZohMCT/GIzelxTceoI
reR2A1r6zmgFnEvi7GgstoDblmpMO/Gi9CXKTqGQ=="
crossorigin="anonymous"></script>

    <script crossorigin
src="https://unpkg.com/@babel/standalone/babel.mi
n.js"></script>

    <script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE
+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
```

```
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dis
t/js/bootstrap.bundle.min.js" integrity="sha384-
ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7Rs
weNrtN/tE3MoK7ZeZDyx"
crossorigin="anonymous"></script>
        <script type="text/jsx"
src="../src/app.js"></script>
    </head>
    <body>
        <div id="root">


        </div>
    </body>
</html>
```

Ex: Instead on using multiple methods you can use single method to handle all interactions.

App.js

```
class Product extends React.Component
{
  constructor(props){
```

```
    super(props);
    this.state = {
      msg: "Select Database Command"
    }
    this.DatabaseOperations =
this.DatabaseOperations.bind(this);
  }
   DatabaseOperations(e)
   {
     switch(e.target.value)
     {
       case "Insert":
       this.setState(state=>({
         msg : state.msg = "Record Inserted"
       }))
       break;
       case "Update":
       this.setState(state=>({
         msg : state.msg = "Record Updated"
       }))
```

```
        break;
        case "Delete":
        this.setState(state=>({
          msg : state.msg = "Record Deleted"
        }))
        break;
      }
    }
  render(){
    return(
     <>
      <div className="btn-group">
        <button value="Insert"
onClick={this.DatabaseOperations} className="btn
btn-primary">Insert</button>
        <button value="Update"
onClick={this.DatabaseOperations} className="btn
btn-success">Update</button>
        <button value="Delete"
onClick={this.DatabaseOperations} className="btn
btn-danger">Delete</button>
      </div>
```

```
      <h2 className="text-center">{this.state.msg}</h2>
     </>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
```

```
  document.getElementById('root')
);
```

FAQ: What are the various event binding techniques used for accessing the current component members?

- *You can bind the event in initialization phase of component “constructor()”*
- *You can bind the event in “event handler” of DOM element.*
- *You can bind the event without using “bind()” method, you have to uses a call back technique to return the event and bind to element.*
- *You can use the event return value and bind to event by using anonymous technique.*

Ex: Constructor

App.js

```
class Product extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      msg : 'Select Database Operation'
```

```
  }
  this.InsertClick = this.InsertClick.bind(this);
 }

 InsertClick(){
  this.setState({
   msg: 'Record Inserted'
  })
 }
 render(){
  return(
   <>
    <div className="btn-group">
     <button onClick={this.InsertClick} className="btn
btn-primary">Insert</button>
    </div>
    <h2 className="text-center">{this.state.msg}</h2>
   </>
  )
 }
```

```
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: DOM Event Handler

App.js

```
class Product extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      msg : 'Select Database Operation'
    }

  }

  InsertClick(){
    this.setState({
      msg: 'Record Inserted'
    })
  }
  render(){
    return(
     <>
      <div className="btn-group">
```

```
        <button onClick={this.InsertClick.bind(this)}
className="btn btn-primary">Insert</button>
      </div>
      <h2 className="text-center">{this.state.msg}</h2>
     </>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
    )
  }
}
```

```
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: Without bind() method

```
class Product extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      msg : 'Select Database Operation'
    }

  }

  InsertClick(){
    this.setState({
      msg: 'Record Inserted'
    })
```

```
  }
  render(){
    return(
     <>
      <div className="btn-group">
        <button onClick={()=> this.InsertClick()}
className="btn btn-primary">Insert</button>
      </div>
      <h2 className="text-center">{this.state.msg}</h2>
     </>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
```

```
        <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: **Anonymous**

App.js

```
class Product extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      msg : 'Select Database Operation'
    }
```

```
}

 InsertClick = ()=> {
  this.setState({
    msg: 'Record Inserted'
  })
}

 render(){
  return(
   <>
    <div className="btn-group">
      <button onClick={this.InsertClick} className="btn
btn-primary">Insert</button>
    </div>
    <h2 className="text-center">{this.state.msg}</h2>
   </>
  )
}
```

```
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

FAQ: What are controlled and uncontrolled components?

- **Uncontrolled component:** It handles the events by using basic DOM manipulations, It doesn’t require any event binding.
- **Controller Component:** It handles the events by using its internal state. It requires event binding.

Ex: Uncontrolled Component [No-State defined]

App.js

```
class Product extends React.Component
{

  InsertClick(){
    document.write("Record Inserted");
  }

  render(){
    return(
     <>
      <div className="btn-group">
```

```
        <button onClick={this.InsertClick} className="btn
btn-primary">Insert</button>
      </div>
      <h2 className="text-center"></h2>
     </>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Product />
      </div>
    )
  }
}
```

```
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Note: React can use all browser events for elements, which require event binding based on the component type. [Controlled Components require – Event Binding]

Ex: Uncontrolled Component using “onChange”

App.js

```
class Product extends React.Component
{
  CityChanged(e){
    document.write(`Selected City : ${e.target.value}`);
  }
  render(){
    return(
     <>
```

```
      <div className="btn-group">
        <select onChange={this.CityChanged}
className="form-control">
          <option>Delhi</option>
          <option>Hyd</option>
          <option>Chennai</option>
        </select>
      </div>
      <h2 className="text-center"></h2>
    </>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
```

```
      <Product />
     </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

FAQ: How you can manage cross-browser compatibility while working with events in “React.js”?

A: React.js manages the cross-browser compatibility by using “**SyntheticEvent”.**

SyntheticEvent

- SyntheticEvent is a part of React.js Event Handling.
- It is responsible for making the react event compatible with “cross-browser”.
- Events can work identically across all browsers.

- The event argument “event” is managed by SyntheticEvent object of React.js
- It is responsible for making the event native for specific browser.

Ex: Get all properties of SyntheticEvent

App.js

```
class Register extends React.Component
{
  RegisterClick(SyntheticEvent){
    for(var property in SyntheticEvent)
    {
      document.write(property + "<br>");
    }
  }
  render(){
    return(
      <>
        <button value="Register"
onClick={this.RegisterClick}>Register</button>
      </>
    )
```

```
  }
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Register />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

- **Every SyntheticEvent object comprises of following attribute**

bubbles	Boolean
cancelable	Boolean
isTrusted	Boolean
target	DOMEventTarget
type	String
preventDefault()	Void
stopPropagation()	Void
persist()	Void
nativeEvent	DOMEvent

- You can access the browser native event related details by using “nativeEvent”

Keyboard Events

React uses Keyboard events to handle various interactions with regard to key code, and key actions.

- onKeyDown : Indicates the actions to perform on key down
- onKeyUp : Indicates actions to perform on key up.

- onKeyPress : Indicates actions to perform when user finish a key and uses another key.

Properties:

- keyCode
- charCode
- shiftKey
- ctrlKey
- which
- key [Return the key used]

Ex: Identifying the Key used

App.js

```
class Register extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      msg: ''
    }
    this.VerifyPassword = this.VerifyPassword.bind(this);
  }
```

```
  VerifyPassword(e){
     if(e.key == "CapsLock"){
       this.setState({
         msg: 'Caps Lock ON'
       })
     } else {
       this.setState({
         msg: ''
       })
     }
  }
   render(){
     return(
       <>
        <div>
          <lable>Password</lable>
          <div>
            <input onKeyUp={this.VerifyPassword}
type="password" className="form-control" />
            <span>{this.state.msg}</span>
```

```
        </div>
      </div>
     </>
    )
  }
}


class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Register />
      </div>
    )
  }
}
```

```
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: **Actions on specific key handling**

```
class Register extends React.Component
{
   VerifyChar(e){
       if(e.key=="y") {
         location.href="http://www.amazon.in";
       } else {
         document.write("Action Canceled..");
       }
    }
   render(){
     return(
       <>
         <input onKeyUp={this.VerifyChar} type="text"
placeholder="Y to Continue and N to Stop" />
```

```
      </>
    )
  }
}


class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Register />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
```

```
  document.getElementById('root')
);
```

Ex: **Verify Value entered in input element on Key Event**

App.js

```
class Register extends React.Component
{
   constructor(props){
     super(props);
     this.state = {
       users: [
         {UserName: 'john'},
         {UserName: 'john1'},
         {UserName: 'david'},
         {UserName: 'david_nit'}
       ],
       msg: ''
     };
     this.VerifyUser = this.VerifyUser.bind(this);
```

```
}
VerifyUser(e){
  for(var user of this.state.users)
  {
    if(user.UserName==e.target.value)
    {
      this.setState({
        msg: 'User Name Taken - Try Another'
      })
      break;
    } else {
      this.setState({
        msg: 'User Name Available'
      })
    }
  }
}
render(){
  return(
    <>
```

```
        <div>
          <label>User Name</label>
          <div>
            <input onKeyUp={this.VerifyUser} type="text"
className="form-control" />
            <span>{this.state.msg}</span>
          </div>
        </div>
      </>
    )
  }
}


class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
```

```
        <Register />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: **Verify User Name and Password**

App.js

```
class Register extends React.Component
{
   constructor(props){
     super(props);
     this.state = {
       users: [
         {UserName: 'john'},
```

```
      {UserName: 'john1'},
      {UserName: 'david'},
      {UserName: 'david_nit'}
    ],
    msg: '',
    regExp: /(?=.*[A-Z])\w{4,15}/,
    pwdMsg: ''
  };
  this.VerifyUser = this.VerifyUser.bind(this);
  this.VerifyPassword = this.VerifyPassword.bind(this);
}
VerifyUser(e){
   for(var user of this.state.users)
   {
      if(user.UserName==e.target.value)
      {
         this.setState({
           msg: 'User Name Taken - Try Another'
         })
         break;
```

```
      } else {
        this.setState({
          msg: 'User Name Available'
        })
      }
    }
}
VerifyPassword(e){
   if(e.target.value.match(this.state.regExp)){
     this.setState({
       pwdMsg: 'Strong Password'
     })
   } else {
      if(e.target.value.length<4){
        this.setState({
          pwdMsg: 'Poor Password'
        })
      } else {
        this.setState({
         pwdMsg: 'Weak Password'
```

```
        })
      }
    }
  }
  render(){
    return(
      <>
        <div className="form-group">
          <label>User Name</label>
          <div>
            <input onKeyUp={this.VerifyUser} type="text"
className="form-control" />
            <span>{this.state.msg}</span>
          </div>
        </div>
        <div className="form-group">
          <label>Password</label>
          <div>
            <input onKeyUp={this.VerifyPassword}
type="password" className="form-control" />
            <span>{this.state.pwdMsg}</span>
```

```
          </div>
        </div>
      </>
    )
  }
}


class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <Register />
      </div>
    )
  }
}
```

```
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Mouse Events

Mouse events specifies actions to perform with regard to mouse interactions on web page or app.

- onClick
- onContextMenu
- onDoubleClick
- onDrag
- onDragEnd
- onDragEnter
- onDargExit
- onMouseDown
- onMouseEnter
- onMouseMove
- onMouseLeave
- onMouseOver etc.

Properties

- altKey
- button

- clientX
- clientY etc..

Ex: onMouseOver, onMouseLeave (onmouseout)

App.js

```
class MouseDemo extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
      styleObj: {
        color: 'black'
      }
    }
    this.GetColor = this.GetColor.bind(this);
    this.SetDefaultColor =
this.SetDefaultColor.bind(this);
  }
  GetColor(e){
    switch(e.target.id){
      case "red":
      this.setState({
```

```
    styleObj: {
      color: 'red'
    }
  })
  break;
  case "green":
  this.setState({
    styleObj: {
      color: 'green'
    }
  })
  break;
  case "blue":
  this.setState({
    styleObj: {
      color: 'blue'
    }
  })
  break;
}
```

```
  }
  SetDefaultColor(){
    this.setState({
      styleObj: {
        color:'black'
      }
    })
  }
  render(){
    return(
      <>
       <div onMouseOver={this.GetColor}
onMouseLeave={this.SetDefaultColor} className="row
mr-5">
         <div className="col" id="red" className="bg-
danger">
             Red color
         </div>
         <div className="col" id="green" className="bg-
success" >
            Green color
```

```
          </div>
          <div className="col" id="blue" className="bg-
info">
             Blue color
          </div>
        </div>
        <h1 style={this.state.styleObj}>Sample Text</h1>
       </>
     )
   }
}


class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <MouseDemo />
```

```
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: Accessing the mouse pointier position on mousemove

App.js

```
class MouseDemo extends React.Component
{
   constructor(props){
     super(props);
     this.state = {
       styleObj : {
         position: '',
         left: '',
```

```
        top: ''
      }
    }
    this.GetMousePosition =
this.GetMousePosition.bind(this);
  }
  GetMousePosition(e){
    this.setState({
      styleObj: {
        position: 'fixed',
        left: e.clientX + 'px',
        top: e.clientY + 'px'
      }
    })
  }
  render(){
   return(
     <>
      <div onMouseMove={this.GetMousePosition}>
      <div style={{height:'1000px'}}>
```

```
        </div>
        <img style={this.state.styleObj} src="flag.gif"
height="50" width="50"/>
        </div>
       </>
     )
   }
}


class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <MouseDemo />
      </div>
    )
```

```
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: Controlling Animations on Mouse Event

App.js

```
class MouseDemo extends React.Component
{
   StartScrolling(e){
      e.target.start();
   }
   StopScrolling(e){
      e.target.stop();
   }
   render(){
```

```
    return(
      <>
        <div>
          <marquee onMouseLeave={this.StartScrolling}
onMouseOver={this.StopScrolling} scrollamount="10">
            <img src="speaker.jpg" width="100"
height="100" />
            <img src="shirt.jpg" width="100" height="100"
/>
            <img src="earpods.jpg" width="100"
height="100" />
            <img src="shoe.jpg" width="100" height="100"
/>
            <img src="jeans.jpg" width="100" height="100"
/>
          </marquee>
        </div>
      </>
    )
  }
}
```

```
class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <MouseDemo />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: onChange Event

App.js

```
class EventsDemo extends React.Component
{
    constructor(props){
      super(props);
      this.state = {
        msg: 'Select a City'
      }
      this.CityChanged = this.CityChanged.bind(this);
    }
    CityChanged(e){
       this.setState({
         msg: e.target.value
       })
    }
    render(){
      return(
        <>
         <div>
           <label>Select City</label>
           <div>
```

```
            <select onChange={this.CityChanged}>
              <option>Select Your City</option>
              <option>Delhi</option>
              <option>Hyd</option>
              <option>Chennai</option>
            </select>
          </div>
        </div>
        <h3 align="center">City Name :
{this.state.msg}</h3>
       </>
     )
   }
}
class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
```

```
        <EventsDemo />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

onBlur, onFocus, onSelect, onCut, onCopy, onPaste, onDoubleClick, onContextMenu

Ex: Double Click Native is “ondblclick” – SyntenticEvent “onDoubleClick”

```
class EventsDemo extends React.Component
{
    constructor(props){
      super(props);
      this.state = {
```

```
      msg: 'Select a City'
    }
    this.CityChanged = this.CityChanged.bind(this);
  }
  CityChanged(e){
    this.setState({
      msg: e.target.value
    })
  }
  OpenImage(){
    window.open('shoe.jpg','Nike','Width=400
height=500');
  }
  render(){
    return(
      <>
       <div>
         <img onDoubleClick={this.OpenImage}
src="shoe.jpg" width="40" height="40"/>
         <p>Double Click to View Large</p>
       </div>
```

```
        <div>
          <label>Select City</label>
          <div>
            <select onChange={this.CityChanged}>
              <option>Select Your City</option>
              <option>Delhi</option>
              <option>Hyd</option>
              <option>Chennai</option>
            </select>
          </div>
        </div>
        <h3 align="center">City Name :
{this.state.msg}</h3>
       </>
     )
   }
}


class MainContent extends React.Component
```

```
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <EventsDemo />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

onSubmit

- It is the event configured for “form” element.
- It fires up the functionality when form is submitted.

Ex:

App.js

```
class EventsDemo extends React.Component
{
  SubmitClick(e){
alert(`Name=${e.target[0].value}\nMobile=${e.target[1
].value}`)
  }
  render(){
    return(
      <>
       <form  onSubmit={this.SubmitClick}>
         <dl>
           <dt>User Name</dt>
           <dd><input type="text" name="username"
/></dd>
           <dt>Mobile</dt>
           <dd>
             <input type="text" name="mobile" />
           </dd>
         </dl>
         <button>Submit</button>
       </form>
```

```
      </>
    )
  }
}


class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <EventsDemo />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
```

```
  document.getElementById('root')
);

Ex: Reading all element values

class EventsDemo extends React.Component
{
   SubmitClick(e){
      for(var element of e.target) {
        document.write(element.value + "<br>");
      }
   }
   render(){
     return(
       <>
        <form  onSubmit={this.SubmitClick}>
          <dl>
            <dt>User Name</dt>
            <dd><input type="text" name="username"
/></dd>
```

```
          <dt>Mobile</dt>
          <dd>
            <input type="text" name="mobile" />
          </dd>
        </dl>
        <button>Submit</button>
      </form>
     </>
    )
  }
}
class MainContent extends React.Component
{
 render(){
   return (
     <div className="container-fluid">
       <h3>Event Handling</h3>
       <EventsDemo />
     </div>
   )
```

```
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Touch Events

Specifies interactions with devices that support "Touch". User can touch, swipe using pointing devices or even mouse.

- onTouchStart : When user starts swipe
- onTouchMove : While swiping on screen
- onTouchEnd : When user ends swipe.

Every Touch event comprises of 3 major properties related to "Touch"

- touches
- targetTouches
- changedTouches

Ex: Touch Event

App.js

```
class TouchEvent extends React.Component
{
  constructor(props) {
    super(props);
    this.state = {
      shoeName: ''
    }
    this.TouchStart = this.TouchStart.bind(this);

  }

  TouchStart(e){
     if(e.touches[0].clientX >= 24 &&
e.touches[0].clientX<=203) {
       this.setState({
         shoeName: 'Nike Casuals'
       });
     }
```

```
    if(e.touches[0].clientX >= 230 &&
e.touches[0].clientX<=401) {
      this.setState({
        shoeName: 'Lee Cooper Boot'
      });
    }


  }

  render(){
    return (
      <>
       <div onTouchStart={this.TouchStart}>
       <img src="shoe.jpg" width="200" height="200" />
       <img src="shoe1.jpg" width="200" height="200" />
       </div>
       <h2>You Touched : {this.state.shoeName} </h2>
      </>
    )
  }
```

```
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <h3>Event Handling</h3>
        <TouchEvent/>
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex:

App.js

```
class TouchEvent extends React.Component
{
  constructor(props) {
    super(props);
    this.state = {
      styleObj : {
        position: 'fixed',
        top: '',
        left: ''
      }
    }
    this.TouchStart = this.TouchStart.bind(this);
  }
  TouchStart(e){
     this.setState({
       styleObj: {
         position: 'fixed',
         left: e.touches[0].clientX + 'px',
```

```
        top: e.touches[0].clientY + 'px'
      }
    })
  }
  render(){
    return (
      <>
       <div onTouchStart={this.TouchStart}>
         <img style={this.state.styleObj} src="ball.png"
height="50" width="50"/>
       </div>
      </>
    )
  }
}

class MainContent extends React.Component
{
  render(){
    return (
```

```
      <div className="container-fluid">
        <TouchEvent/>
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

onTouchStart : It fires up when user starts the touch.

onTouchMove : It fires up while dragging on touch.

onTouchEnd : It fires up when touch stopped.

How to handle swipe on application?

- Swipe uses the 3 touch events
    - TouchStart

    - TouchMove
    - TouchEnd
- You can access the touch property called “onSwiped” and “onSwipping”

Ex:

App.js

```
class TouchEvent extends React.Component
{
  constructor(props) {
    super(props);
    this.state = {
      styleObj : {
        position: 'fixed',
        top: '',
        left: ''
      },
      ads : {
        position : 'fixed',
        top: '',
        left: ''
      }
```

```
  }
  this.TouchStart = this.TouchStart.bind(this);
  this.MoveAd = this.MoveAd.bind(this);
}
TouchStart(e){
   this.setState({
     styleObj: {
       position: 'fixed',
       left: e.touches[0].clientX + 'px',
       top: e.touches[0].clientY + 'px'
     }
   })
}
MoveAd(e){
   this.setState({
     ads: {
       position: 'fixed',
       left: e.touches[0].clientX + 'px',
       top: e.touches[0].clientX + 'px'
     }
```

```
    })
  }
  render(){
    return (
      <>
       <div onTouchMove={this.MoveAd}
style={this.state.ads} className="card">
         <div className="card-header">
           <h3>Ads - Nike</h3>
         </div>
         <div className="card-body">
           <img src="shoe.jpg" width="50" height="50" />
         </div>
       </div>
       <div onTouchMove={this.TouchStart}>
         <img style={this.state.styleObj} src="ball.png"
height="50" width="50"/>
       </div>
      </>
    )
  }
```

```
}

class MainContent extends React.Component
{
  render(){
    return (
      <div className="container-fluid">
        <TouchEvent/>
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Conditional Rendering in React

- A component can render specified content in UI.
- It is defined by using “render()” method.
- In various situations a component must able to render different types of contents.
- You can design component with conditional rendering, so that it can render content according to state and situation.
- It requires the traditional JavaScript operator like “if, switch, ternary etc.”

```
function LoginSuccess(props){
   return <h2>Login Success..</h2>
}
function LoginFailure(props){
   return <h2>Login Failed..</h2>
}

class MainContent extends React.Component
{
  constructor(props){
    super(props);
    this.state = {
```

```
    statusmsg: false,
    content: ''
   }
   this.LoginClick = this.LoginClick.bind(this);
  }
  LoginClick(e){

    if(e.target[0].value=='john' &&
e.target[1].value=='admin')
    {
      this.setState({
        content: <LoginSuccess />
      })

    } else {
      this.setState({
        content: <LoginFailure />
      })
    }
  }
```

```
  render(){
    return (
      <>
        <h3>User Login</h3>
        <form onSubmit={this.LoginClick} >
          <dl>
          <dt>User Name</dt>
          <dd><input type="text"/></dd>
          <dt>Password</dt>
          <dd><input type="password"/></dd>
           </dl>
          <button>Login</button>
        </form>
        <div>
          {this.state.content}
        </div>
      </>
    )
  }
}
```

```
ReactDOM.render(
  <MainContent />,
  document.getElementById('root')
);
```

Ex: Changing the View using condition – Views are configured as separate components.

```
function EditView(){
  return (
    <div>
      Price :
      <input type="text" />
      <button>Save</button>
    </div>
  )
}
function DisplayView(){
  return (
    <div>
```

```
      Price : <label></label>
      <button>Edit</button>
    </div>
  )
}

class ProductComponent extends React.Component
{
    constructor(props) {
      super(props);
      this.state = {
        product: {
          Name: 'JBL Speaker',
          Price: 5300.33,
          Photo: 'speaker.jpg'
        },
        viewName: 'display'
      }
    }
    render(){
```

```
        if(this.state.viewName=='display'){
          return(
            <DisplayView />
          )
        } else {
          return(
            <EditView />
          )
        }
      }
}

class MainContent extends React.Component {
  render(){
    return(
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
```

```
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

Ex: Changing view without defining separate components.

```
class ProductComponent extends React.Component
{
    constructor(props) {
      super(props);
      this.state = {
        product: {
          Name: 'JBL Speaker',
          Price: 5300.33,
          Photo: 'speaker.jpg'
        },
```

```
    viewName: 'edit'
  }
}
render(){
  if(this.state.viewName=='display'){
    return(
      <div>
      Price : <label></label>
      <button>Edit</button>
      </div>
    )
  } else {
    return(
      <div>
      Price :
      <input type="text" />
      <button>Save</button>
      </div>
    )
  }
```

```
    }
}

class MainContent extends React.Component {
  render(){
    return(
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

Ex: Editing Record

```
class ProductComponent extends React.Component
{
    constructor(props) {
      super(props);
      this.state = {
        product: {
          Name: 'JBL Speaker',
          Price: 5300.33,
        },
        newProduct: {
          Name: '',
          Price: ''
        },
        viewName: 'display'
      }
      this.EditClick = this.EditClick.bind(this);
      this.SaveClick = this.SaveClick.bind(this);
      this.ChangePrice = this.ChangePrice.bind(this);
      this.ChangeName = this.ChangeName.bind(this);
    }
```

```
EditClick(){
  this.setState({
    viewName: 'edit'
  })
}
SaveClick(){
  this.setState({
    viewName: 'display',
    product: {
      Name: this.state.newProduct.Name,
      Price: this.state.newProduct.Price
    }
  })
}
ChangePrice(e){
  this.setState({
    newProduct: {
      Name: this.state.newProduct.Name,
      Price: e.target.value
    }
```

```
  })
}
ChangeName(e){
 this.setState({
   newProduct: {
     Name: e.target.value,
     Price: this.state.newProduct.Price
   }
 })
}
render(){
  if(this.state.viewName=='display'){
    return(
      <div>
       <table width="300" border="1">
         <thead>
           <tr>
           <th>Name</th>
           <th>Price</th>
           <th>Action</th>
```

```
            </tr>
          </thead>
          <tbody>
           <tr>
            <td> {this.state.product.Name}</td>
            <td> {this.state.product.Price} </td>
            <td><button onClick={this.EditClick}
>Edit</button></td>
           </tr>
          </tbody>
         </table>
        </div>
      )
    } else {
      return(
        <div>
          <table width="300" border="0">
           <thead>
            <tr>
             <th>Name</th>
```

```
            <th>Price</th>
            <th>Action</th>
           </tr>
          </thead>
          <tbody>
          <tr>
            <td> <input type="text"
onChange={this.ChangeName}
value={this.state.newProduct.Name} /></td>
            <td> <input type="text"
onChange={this.ChangePrice}
value={this.state.newProduct.Price} /> </td>
            <td><button onClick={this.SaveClick}
>Save</button></td>
          </tr>
          </tbody>
         </table>
        </div>
      )
    }
  }
```

```
}

class MainContent extends React.Component {
  render(){
    return(
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

Ex: Editable Table Row

```
class ProductComponent extends React.Component
```

```
{
    constructor(props) {
      super(props);
      this.state = {
        product: {
          Name: 'JBL Speaker',
          Price: 5300.33,
        },
        newProduct: {
          Name: '',
          Price: ''
        },
        viewName: 'display'
      }
      this.EditClick = this.EditClick.bind(this);
      this.SaveClick = this.SaveClick.bind(this);
      this.ChangePrice = this.ChangePrice.bind(this);
      this.ChangeName = this.ChangeName.bind(this);
    }
    EditClick(){
```

```
    this.setState({
      viewName: 'edit',
      newProduct: {
        Name: this.state.product.Name,
        Price: this.state.product.Price
      }
    })
  }
  SaveClick(){
    this.setState({
      viewName: 'display',
      product: {
        Name: this.state.newProduct.Name,
        Price: this.state.newProduct.Price
      }
    })
  }
  ChangePrice(e){
    this.setState({
```

```
      newProduct: {
        Name: this.state.newProduct.Name,
        Price: e.target.value
      }
    })
}
ChangeName(e){
  this.setState({
    newProduct: {
      Name: e.target.value,
      Price: this.state.newProduct.Price
    }
  })
}
render(){
   if(this.state.viewName=='display'){
     return(
       <div>
        <table width="300" border="1">
          <thead>
```

```
            <tr>
            <th>Name</th>
            <th>Price</th>
            <th>Action</th>
            </tr>
          </thead>
          <tbody>
           <tr>
            <td> {this.state.product.Name}</td>
            <td> {this.state.product.Price} </td>
            <td><button onClick={this.EditClick}
>Edit</button></td>
           </tr>
          </tbody>
         </table>
        </div>
      )
     } else {
      return(
       <div>
```

```
<table width="300" border="0">
 <thead>
  <tr>
   <th>Name</th>
   <th>Price</th>
   <th>Action</th>
  </tr>
 </thead>
 <tbody>
 <tr>
   <td> <input type="text"
onChange={this.ChangeName}
value={this.state.newProduct.Name} /></td>
   <td> <input type="text"
onChange={this.ChangePrice}
value={this.state.newProduct.Price} /> </td>
   <td><button onClick={this.SaveClick}
>Save</button></td>
 </tr>
 </tbody>
</table>
```

```
          </div>
        )
      }
    }
}

class MainContent extends React.Component {
  render(){
    return(
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

Handling Multiple Conditions

- You can use “if..else if.. else”
- You can use “switch..case..default”
- You can use “ternary” ?:

Ex: **Rendering with “if..else if.. else”**

```
class ProductComponent extends React.Component
{
  constructor(props) {
    super(props);
    this.state = {
      viewName: ''
    }
  }

  render(){
    if(this.state.viewName=='Basic'){
      return(
        <div>
          <h3>Basic Details</h3>
```

```
        </div>
      )
    } else if (this.state.viewName=='Preview'){
      return(
        <div>
          <h3>Product Preview</h3>
        </div>
      )
    } else {
      return(
        <div>
          <h3>Summary</h3>
        </div>
      )
    }
  }
}
class MainContent extends React.Component {
  render(){
    return(
```

```
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

FAQ: How to render nothing if value is not among specified conditions?

A. You have to configure the render method to return “null” if condition is not matching.

EX: Multiple conditions using ternary operator

```
{
        this.state.viewName=='Details'?
```

```
        <h3>Details</h3>:
        this.state.viewName=='Preview'?
        <h3>Preview</h3>:
        <h3>Summary</h3>
}
```

Ex:

```
function BasicInfo(){
  return(
    <div>
      <h2>Basic Info</h2>
      <div className="card">
        <div className="card-header">
          <h2>JBL Speaker</h2>
          <p>Price : &#8377; 6700.66/-</p>
        </div>
      </div>
    </div>
  )
```

```
}

function Preview(){
  return (
    <div>
    <h2>Preview</h2>
    <div className="card">
      <div className="card-body">
        <img src="speaker.jpg" width="200" height="200"
/>
      </div>
    </div>
  </div>
  )
}
function Description(){
  return(
    <div>
    <h2>Description</h2>
    <div className="card">
```

```
      <div className="card-header">
        <h3>Features</h3>
        <ul>
          <li>Feature-1</li>
          <li>Feature-2</li>
          <li>Feature-3</li>
        </ul>
      </div>
    </div>
  </div>
  )
}

class ProductComponent extends React.Component
{
   constructor(props) {
     super(props);
     this.state = {
       viewName: 'Description',
       container: ''
```

```
  }
  this.ViewChanged = this.ViewChanged.bind(this);
}
ViewChanged(e){
   switch(e.target.name)
   {
     case "BasicInfo":
     this.setState({
       container: <BasicInfo />
     });
     break;
     case "Preview":
     this.setState({
       container: <Preview />
     });
     break;
     case "Description":
     this.setState({
       container: <Description />
     });
```

```
      break;
    }
  }
  render(){
    return (
     <div>
      <h2>Product Details</h2>
     <div className="btn-toolbar">
       <div className="btn-group">
         <button onClick={this.ViewChanged}
name="BasicInfo" className="btn btn-primary">Basic
Info</button>
         <button onClick={this.ViewChanged}
name="Preview" className="btn btn-
info">Preview</button>
         <button onClick={this.ViewChanged}
name="Description" className="btn btn-
danger">Description</button>
       </div>
     </div>
     <div>
        {this.state.container}
```

```
      </div>
    </div>
     )
   }
}
class MainContent extends React.Component {
  render(){
    return(
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

Component Lifecycle

- **Every component has various phases.**
- The component in “React” is responsible for managing various interactions in an order implicitly.
- Component comprises various lifecycle methods that executed to handle various interactions.
- You have to override the lifecycle methods to improve the performance of application.
- You can prevent memory leaks; cross page posting and can improve the load time and performance.
- Every component comprises of 3 phases
    - **Mounting**
    - **Updating**
    - **Unmounting**

Mounting: When a component is initialized

- getDefaultProps
- getInitialState
- componentWillMount
- **render**
- **componentDidMount**

Updating: When a component has state changed [Change Detection]

- shouldComponentUpdate - comprises of value before update

- componentWillUpdate
- **render**
- **componentDidUpdate** - comprises of value after update

Unmounting: When a component is unmounting. [Cleaning up / destroying]

- **componentWillUnmount**

Ex: Component Mount and Unmount

App.js

```
class ProductComponent extends React.Component
{
  constructor(props) {
    super(props);
     this.state = {
       date : new Date()
     }
  }
  timer(){
    this.setState(
      {
```

```
      date : new Date(),
      msg: ''
    }
  )
  this.componentWillUnmount =
this.componentWillUnmount.bind(this);
 }
 displaytime;
 componentDidMount() {
  this.displaytime = setInterval(()=> this.timer(),
1000);
  this.setState({
   msg: 'Component Mounted'
  })
  alert("Component Mounted");
 }
 componentWillUnmount(){
  clearInterval(this.displaytime);
  this.setState({
   msg: 'Component Unmounted | Cleaned Up'
  })
```

```
  }
  Refresh(){
    location.reload();
  }
  render(){
    return(
      <div>
        <button onClick={this.componentWillUnmount}
>Unmount</button>
        <button onClick={this.Refresh}>Refresh</button>
        <p>{this.state.date.toLocaleTimeString()}</p>
        <p>{this.state.msg}</p>
      </div>
    )
  }
}
class MainContent extends React.Component {
  render(){
    return(
```

```
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

Ex:

App.js

```
class SuccessComponent extends React.Component
{
  componentWillMount(){
    alert("Success component requested..");
  }
  componentDidMount() {
```

```
    alert("Success Component Mounted..");
  }
  componentWillUnmount(){
    alert("Success Disposed..");
  }
  render(){
    return(
      <h2>Login Success..</h2>
    )
  }
}
class ErrorComponent extends React.Component
{
  componentWillMount(){
    alert("Error Component Requested");
  }
  componentDidMount() {
    alert("Error Component Mounted..");
  }
  componentWillUnmount(){
```

```
    alert("Error Disposed..");
  }
  render(){
    return(
      <h2>Invalid Password</h2>
    )
  }
}
class ProductComponent extends React.Component
{
   constructor(props) {
     super(props);
     this.state = {
       msg: ''
     }
     this.SuccessClick = this.SuccessClick.bind(this);
     this.ErrorClick = this.ErrorClick.bind(this);
   }

   SuccessClick(){
```

```
    this.setState({
      msg: <SuccessComponent />
    })
  }
  ErrorClick(){
    this.setState({
      msg: <ErrorComponent />
    })
  }
  render(){
   return(
     <div>
       <button onClick={this.SuccessClick}
>Success</button>
       <button onClick={this.ErrorClick}>Error</button>
       <div>
         <p>
           {this.state.msg}
         </p>
       </div>
```

```
      </div>
    )
  }
}
class MainContent extends React.Component {
  render(){
    return(
      <div className="container-fluid">
        <ProductComponent />
      </div>
    )
  }
}

ReactDOM.render(
  <MainContent/>,
  document.getElementById('root')
);
```

React Application Interacting with API

Distributed Computing Architecture

- Distributed computing allows 2 different applications running on 2 different machines to share information. OR 2 different applications running of 2 different process in the same machine can share information.
- There are various distributed computing technologies
    - CORBA
    - DCOM
    - RMI
    - EJB
    - Web Service [All technologies]
    - Remoting etc.
- Web Service Specifications
    - SOAP
        - Request and Response will in XML
    - REST
        - Request will a simple query and Response will in XML or Optionally JSON.
    - JSON
        - Request and Response will be JSON.
- React requires a “Tool Chain” to configuring distribute computing architecture and build distributed computing apps.
- **Client-Side Application** is created by using “Create-React-App” [**SPA** in React]

    - A Package Manager [NPM]
    - Bundler [Webpack or Parcel]
    - A Compiler [Babel]
- The “Tool Chain” used for React Distributed Application
    - **Next.js** – Framework for static and server-rendered applications [Node.js, JSP, .NET, PHP].
    - **Gatsby –** For creating static website with React. [Improve the load time]
    - **Nx –** tool kit for “Full Stack development” using React. [React, Next.js, Express, MongoDB]
    - **Razzle –** Server-Side Rendering Framework
    - **Ionic, Native Script, Cordova –** Framework to build cross platform mobile app with React.

- Creating a Client Side React Application
    - Install Node.js 8+ version
    - Run the command
      > npm install -g create-react-app
    - Run the command
      > npx create-react-app **amazon**
    - To Start your application
      > npm start

Create a new Database in MongoDB

- Download and Install MongoDB database server on your PC.
  https://www.mongodb.com/try/download/community
- Start MongoDB Server
  Open → Services.msc → MongoDB Server [Right Click and Start]
- Connect to MongoDB server from client
  C:\Program Files\MongoDB\Server\4.4\bin>mongo.exe

MongoDB Terminology:

RDBMS	**MongoDB**
Database	Database
Table	Collection
Records / Rows	Documents
Field	Field
Joins	Embedded Documents

MongoDB Commands: [Case sensitive]

- To View the list of existing databases
  > show dbs
- To create a new database
  > use databaseName
  > use shoppingdb
- To Create a new Table [Collection] in Database
  > **db.createCollection(“tableName”)**
  > db.createCollection(“tblproducts”)
- To View the tables in database
  > show collections
- To Add records into table [Collection]
  >
  db.collectionName.insertOne({FieldName:value, FieldName: value})
  > db.collectionsName.insertMany([{ }, { }, { }])

Ex:

> db.tblproducts.insertOne({ProductId:1, Name: "JBL Speaker", Price: 4500.55, InStock:true})

> db.tblproducts.insertMany([{ProductId:2, Name:'Earpods', Price: 6000.55, InStock:true},{ProductId:3, Name: 'Nike Casuals', Price: 7000.44, InStock:true}])

- To View Documents present in Collection
  > db.collectionName.find({query})

> db.tblproducts.find().pretty()

Create Web API / Service using Server-Side Technology

[Node.js – Server-Side Scripting, Express – Middleware]

- **Go to Your "ReactFullStack" project**
- **Create a new Folder "Server"**
- **Install following packages**
  > npm install mongodb [database connection]
  > npm install express [middleware]
- **Create a new JavaScript file [Node.js] by name "api.js"**

Api.js

```
var mongoClient = require("mongodb").MongoClient;
var express = require("express");
var url = "mongodb://127.0.0.1:27017";
var app = express();
app.get("/getproducts", function(req, res){
   mongoClient.connect(url, function(err, clientObj){
      if(!err) {
         var database = clientObj.db("shoppingdb");
```

```
database.collection("tblproducts").find().toArray(functi
on (err, documents){
            if(!err) {
               res.send(documents);
            }
         })
      }
   })
});
app.listen(8080);
console.log("Server Started: http://127.0.0.1:8080");
```

Product.js

```
import React from 'react';
import  $ from 'jquery';
export default class ProductComponent extends
React.Component {
   constructor(props) {
      super(props);
      this.state = {
         ProductId: '',
```

```
        Name: '',
        Price: 0,
        InStock: false
    }
}
componentDidMount(){
    this.fetch();
}
fetch() {
    var context = this;
    $.ajax({
        url: 'http://127.0.0.1:8080/getproducts',
        method: 'GET',
        success: function(response){
            context.setState({
                ProductId: response[0].ProductId,
                Name: response[0].Name,
                Price: response[0].Price,
                InStock: response[0].InStock
            })
```

```
            }
        })
    }
    render(){
        return(
            <div>
                <h2>Get Data from API</h2>
                <dl>
                    <dt>Product Id</dt>
                    <dd>{this.state.ProductId}</dd>
                    <dt>Name</dt>
                    <dd>{this.state.Name}</dd>
                    <dt>Price</dt>
                    <dd>{this.state.Price}</dd>
                </dl>
            </div>
        )
    }
}
```

Consume REST API using jQuery Ajax in React

- Create a new Database Table in MongoDB
  Database: amazondb
  Collection: tblproducts
- Create a new folder by name “Server”.
- Install the following packages into server folder
  > npm install mongodb --save
  > npm install express --save
  > npm install cors --save

- Create a new JavaScript file “api.js”

Api.js

```
var mongoClient = require("mongodb").MongoClient;
var express = require("express");
var cors = require("cors");



var url = "mongodb://127.0.0.1:27017";
var app = express();
app.use(cors());

app.get("/getproducts", function(req, res){
   mongoClient.connect(url, function(err, clientObj){
```

```
        if(!err){
            var dbo = clientObj.db("amazondb");
dbo.collection("tblproducts").find().toArray(function(er
r, documents){
                if(!err) {
                    res.send(documents);
                }
            })
        }
    })
});
app.listen(8080);
console.log("Server Started: http://127.0.0.1:8080");
```

- **Start API**
  > node api.js
- Go to “React Application”
- Go to “src” and add the file “ProductComponent.js”
  ProductComponent.js

```
import React from 'react';
import $ from 'jquery';
```

```
export default class ProductComponent extends
React.Component
{
  constructor(props) {
    super(props);
    this.state = {
      products: []
    }
  }

  componentDidMount() {
    this.fetch();
  }

  fetch(){
    var context = this;
    $.ajax({
      url: 'http://127.0.0.1:8080/getproducts',
      method: 'GET',
      success: function(response){
        context.setState({
          products: response
        })
      }
```

```
    })
}
render(){
    return(
        <div>
          <h2>Products List </h2>
          <table width="400" border="1">
            <thead>
              <tr>
               <th>Product Id </th>
               <th>Name </th>
               <th>Price </th>
              </tr>
            </thead>
            <tbody>
              {
                  this.state.products.map(product =>
                    <tr key={product.ProductId}>
                     <td>{product.ProductId}</td>
                     <td>{product.Name}</td>
                     <td>{product.Price}</td>
                    </tr>
                  )
              }
            </tbody>
          </table>
```

```
        </div>
      )
    }
  }
```

- **Go to Index.js**

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import ProductComponent from
'./ProductComponent';

ReactDOM.render(
  <React.StrictMode>
    <ProductComponent />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in
your app, pass a function
// to log results (for example:
reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more:
https://bit.ly/CRA-vitals
```

```
reportWebVitals();
```

Consuming REST API using Fetch

- jQuery uses “Ajax” explicitly to connect with API.
- Fetch uses window.fetch, which implicitly uses XmlHttpRequest.
- Fetch will return the data, which need to be parsed into JSON.
- Fetch provides various details in response data.
- You have to convert data into JSON.

Ex:

ProductsDataComponent.js

```
import React from 'react';


export default class ProductsDataComponent extends
React.Component
{
  constructor(props) {
    super(props);
    this.state = {
      products: []
    }
```

```
}
componentDidMount(){
  fetch("http://127.0.0.1:8080/getproducts")
  .then(response => response.json())
  .then(response => {
    this.setState({
      products: response
    })
  })
}
render(){
  return(
    <div>
      <h2>Products List</h2>
      <table width="400" border="1">
        <thead>
          <tr>
            <th>Product Id</th>
            <th>Name</th>
            <th>Price</th>
```

```
          </tr>
        </thead>
        <tbody>
          {
            this.state.products.map(product =>
              <tr key={product.ProductId}>
                <td>{product.ProductId}</td>
                <td>{product.Name}</td>
                <td>{product.Price}</td>
              </tr>
            )
          }
        </tbody>
      </table>
    </div>
  )
 }
}
```

Consuming APIs with Axios

- Axios can handle data by default in JSON format.

- Fetch requires explicit JSON parsing.
- Axios converts the data into JSON and provides to application.
- It can make used of “async and await” for handling Asynchronous requests.
- It is provided with built-in Request Forgery Token. That prevents Cross Page Posting.
  [**XSRF], [XSS]**

Features of Axios:

- Default data format is JSON.
- Provides streamlined Error Handling
- Protection against XSRF.
- Response timeout.
- Ability to cancel the request.
- Uses await and async.
- Supports CORS
- Supports Legacy Browsers [Old Version]

Requests with Axios:

```
axios({
  method: ‘GET’,
  url: ‘api_url’
```

```
});

axios({
   method: 'POST',
   url: 'api_url',
   data: { }
})
```

Shorthand Requests

```
axios.get(url)
axios.post(url, data[])
axios.put()
axios.delete()
```

Axios can handle multiple Requests:

Syntax:

```
axios.all([
     axios.get(URL),     [0]
     axios.get(URL),     [1]
])
then(response=> {
```

```
    console.log('First URL', response[0].data),
    console.log('SecondURL', response[1].data)
}
```

You have to use the state Hooks of React to handle interaction with APIs

- useState()
- useEffect()
- useContext()
- useReducer()

useState():

- It allows React developers to update, handle and manipulate state inside functional components without converting into class component.

Syntax:

```
function App() {
    const [name, setName] = useState('John');
    OnButtonClick = () => setName('David');
}
return  <div> {name} </div>
```

Axios in React Application

- Install axios for your project
  > npm install axios
- Add a new file into “src”
  ProductDetailsComponent.js

```
import React from 'react';
import axios from 'axios';

export default class ProductDetailsComponent
extends React.Component
```

```
{
   constructor(props) {
      super(props);
      this.state = {
         products: []
      }
   }
   componentDidMount(){
axios.get("http://127.0.0.1:8080/getproducts")
      .then(res => {
         this.setState({
            products: res.data
         })
      })
   }
   render(){
      return(
      <div>
      <h2>Products List</h2>
      <table width="400" border="1">
       <thead>
         <tr>
           <th>Product Id</th>
           <th>Name</th>
           <th>Price</th>
         </tr>
```

```
          </thead>
          <tbody>
            {
              this.state.products.map(product =>
                <tr key={product.ProductId}>
                  <td>{product.ProductId}</td>
                  <td>{product.Name}</td>
                  <td>{product.Price}</td>
                </tr>
              )
            }
          </tbody>
        </table>
      </div>
      )
    }
}
```

Note: You can also use “whatwg-fetch” for fetching remote data.

Posting Data to Remote API

- Server-side API must be configured with POST method.
- Install “body-parser” in your server-side project

> npm install body-parser

- Create “Post” URL in your Server-Side API

Api.js

```
var mongoClient = require(“mongodb”).MongoClient;
var express = require(“express”);
var cors = require(“cors”);
var bodyParser = require(“body-parser”);


var url = “mongodb://127.0.0.1:27017”;
var app = express();
app.use(cors());
app.use(bodyParser.urlencoded({
   extended: true
}));
app.use(bodyParser.json());


app.get(“/getproducts”, function(req, res){
   mongoClient.connect(url, function(err, clientObj){
```

```
        if(!err){
            var dbo = clientObj.db("amazondb");
dbo.collection("tblproducts").find().toArray(function(er
r, documents){
                if(!err) {
                    res.send(documents);
                }
            })
        }
    })
});
app.post("/addproduct", function(req, res){
    mongoClient.connect(url, function(err, clientObj){
        if(!err) {
            var dbo = clientObj.db("amazondb");
            var data = {
                ProductId: req.body.ProductId,
                Name: req.body.Name,
                Price: req.body.Price
            };
```

```
        dbo.collection("tblproducts").insertOne(data,
function(err, result){
            if(!err) {
                console.log("Record Inserted");
            } else {
                console.log(err.message);
            }
        });
    }
  })
})
app.listen(8080);
console.log("Server Started: http://127.0.0.1:8080");
```

Create a React Component Posting Data into API

ProductDetailsComponent.js

```
import React from 'react';
import axios from 'axios';

export default class ProductDetailsComponent extends
React.Component
```

```
{
    constructor(props) {
        super(props);
        this.state = {
            products: [],
            msg: '',
            data: {
              ProductId: 0,
              Name: '',
              Price: 0
            }
        }
        this.IdChanged = this.IdChanged.bind(this);
        this.NameChanged =
this.NameChanged.bind(this);
        this.PriceChanged = this.PriceChanged.bind(this);
    }
    componentDidMount(){
        axios.get("http://127.0.0.1:8080/getproducts")
        .then(res => {
```

```
        this.setState({
            products: res.data,
            msg: res.status
        })
    })
}
IdChanged = event => {
  this.setState({
    data : {
      ProductId: event.target.value,
      Name: this.state.data.Name,
      Price: this.state.data.Price,
    }
  })
}
NameChanged = event => {
 this.setState({
   data : {
     ProductId: this.state.data.ProductId,
     Name: event.target.value,
```

```
      Price: this.state.data.Price,
    }
  })
}
PriceChanged = event => {
 this.setState({
   data : {
     ProductId: this.state.data.ProductId,
     Name: this.state.data.Name,
     Price: event.target.value,
   }
 })
}
SubmitClick = event => {

   const data = {
     ProductId: this.state.data.ProductId,
     Name: this.state.data.Name,
     Price: this.state.data.Price
   }
```

```
    axios.post(`http://127.0.0.1:8080/addproduct`, data)
    .then(res => {
      console.log('Record Inserted');
      console.log(res.data);
    })
    alert("Record Inserted");
  }
   render(){
      return(
      <div className="container">
      <h2>Add Product</h2>
      <form onSubmit={this.SubmitClick}>
        <div>
          <dl>
            <dt>Product Id</dt>
            <dd><input type="text" onChange={this.IdChanged} /></dd>
            <dt>Name</dt>
            <dd><input type="text" onChange={this.NameChanged} /></dd>
```

```
            <dt>Price</dt>
            <dd><input type="text"
onChange={this.PriceChanged} /></dd>
          </dl>
          <button>Add Product</button>
        </div>
      </form>
      <h2>Products List {this.state.msg} </h2>
      <table className="table table-hover">
        <thead>
          <tr>
            <th>Product Id</th>
            <th>Name</th>
            <th>Price</th>
          </tr>
        </thead>
        <tbody>
          {
            this.state.products.map(product =>
              <tr key={product.ProductId}>
```

```
                <td>{product.ProductId}</td>
                <td>{product.Name}</td>
                <td>{product.Price}</td>
              </tr>
             )
           }
        </tbody>
      </table>
    </div>
     )
  }
}
```

Forms in React

- Form provides an UI that allows the users to interact with application.
- React JS supports one way binding, hence handling interaction with form requires “State”.
- Class component can maintain state for form.
- Functional component is state less hence it requires “useState()” hook.

- You can configure a state hook for every element and access its value.

Ex:

```
import React from 'react';


export default function FormDemo() {
   const [productname, setName] = React.useState("");
   const [productprice, setPrice] = React.useState("");
   const [shippedto, setShippedTo] =
React.useState("");
   const [instock, setStock] = React.useState("Yes");


   const submitClick = (event) => {
      document.write(`
      Name=${productname} <br>
      Price=${productprice} <br>
      Shipped To=${shippedto} <br>
      InStock =${instock}
      `);
   }
```

```
    return(
        <form onSubmit={submitClick}>
            <h2>Register Product</h2>
            <div className="form-group">
             <label>Name</label>
             <div>
                <input onChange={e=>
setName(e.target.value)} name="productname"
value={productname} className="form-control"
type="text" />
             </div>
            </div>
            <div className="form-group">
             <label>Price</label>
             <div>
                <input
onChange={e=>setPrice(e.target.value)}
name="productprice" value={productprice}
className="form-control" type="text"/>
             </div>
            </div>
            <div className="form-group">
```

```
            <label>Shipped To</label>
            <select
onChange={e=>setShippedTo(e.target.value)}
name="shippedto" value={shippedto}
className="form-control" >
                <option>Delhi</option>
                <option>Hyd</option>
                <option>Mumbai</option>
            </select>
        </div>
        <div className="form-group">
            <label>In Stock</label>
            <div>
                <input value={instock}
onChange={e=>setStock(e.target.value)}
name="instock" type="checkbox" /> Yes
            </div>
        </div>
        <div class="form-group">
            <button className="btn btn-primary btn-
block">Register</button>
        </div>
```

```
        </form>
    );
}
```

Accessing Data from Multiple Form Elements

- Create a state for handling multiple values

  Syntax:

  ```
  this.state = {
  product  : {
        Id: ‘ ‘,
        Name: ‘ ‘
   }
  ```

- Bind the state with elements

  ```
  <input type=”text”  value={this.state.product.Id}
  name=”Id />
  ```

- Configure event binding for element

  ```
  <input type=”text”
  onChange={this.handleChange} />
  ```

- Define the event method to access the name and value of element

  ```
  handleChange(event)
  {
    name  = event.target.name;
    value =  event.target.value;
    this.setState({
        product : {
  ```

```
                ...this.state.product,
                 [name] : value
            }
        })
    }
```

Ex:

FormDemo.js

```
import React from 'react';
export default class FormClassDemo extends
React.Component
{
   constructor(props) {
      super(props);
      this.state = {
         product : {
            ProductId: '',
            Name: '',
            Price: ''
         }
      }
      this.updateFormData =
this.updateFormData.bind(this);
```

```
    }
    updateFormData(event) {
        const name = event.target.name;
        const value = event.target.value;
        this.setState({
            product : {
              ...this.state.product,
              [name] : value
            }
        })
    }
    render(){
        return(
            <div>
                <dl>
                    <dt>Product Id</dt>
                    <dd>
                        <input type="text" name="ProductId"
value={this.state.product.ProductId}
onChange={this.updateFormData} />
                    </dd>
```

```
                <dt>Name</dt>
                <dd>
                  <input type="text" name="Name"
value={this.state.product.Name}
onChange={this.updateFormData}/>
                </dd>
                <dt>Price</dt>
                <dd>
                  <input type="text" name="Price"
value={this.state.product.Price}
onChange={this.updateFormData} />
                </dd>
              </dl>
              <h3>Product Details</h3>
              <dl>
                <dt>Product Id</dt>
                <dd>{this.state.product.ProductId}</dd>
                <dt>Name</dt>
                <dd>{this.state.product.Name}</dd>
                <dt>Price</dt>
                <dd>{this.state.product.Price}</dd>
```

```
            </dl>
          </div>
        )
    }
}
```

External Library for Designing the Form

- Formik is a library used for building complex forms and handling validation.
- Formik is a group of React components and hooks for building and validating forms.
- The library of “formik" provides hooks and component, which are accessed using “formik” prefix.

  Syntax:

  <form onSubmit= {formik.handleSubmit} >
  <input type=”text” value={formik.values.Nam} />
- In order to use Formik in your React component. You have to inject “useFormik”.

Ex:

```
import React from 'react';
import {useFormik} from 'formik';
```

```
const FormLibraryDemo = ( ) => {
   const formik = useFormik({
      initialValues: {
         ProductId: '',
         Name: '',
         Price: ''
      },
      onSubmit: values => {
         console.log(JSON.stringify(values));
      }
   })
   return(
      <div>
       <form onSubmit={formik.handleSubmit} >
         <dl>
            <dt>Product Id</dt>
            <dd>
               <input type="text" name="ProductId"
value={formik.values.ProductId}
onChange={formik.handleChange} />
            </dd>
```

```
            <dt>Name</dt>
            <dd>
               <input type="text" name="Name"
value={formik.values.Name}
onChange={formik.handleChange} />
            </dd>
            <dt>Price</dt>
            <dd>
               <input type="text" name="Price"
value={formik.values.Price}
onChange={formik.handleChange} />
            </dd>
          </dl>
          <button>Register</button>
        </form>
      </div>
   )
}
export default FormLibraryDemo;
```

Validations in React Form

- Validation is the process of verifying user input.
- Validation is required to ensure that contractionary and unauthorized data is not get stored into database.
- Validation can be handled manually or by using pre-defined library functions and properties.
- Formik supports synchronous and asynchronous validations.
- Formik handles validation at 2 levels
    - Formstate validation / Form-level validation
    - Inputstate validation / Field-level validation
- “Yup” is used for Schema based validation.
- Formik uses various methods/events for validating the values
    - After Change events/methods
        - handleChange
        - setFieldValue
        - setValues
    - After Blur events/methods
        - handleBlur
        - setTouched
        - setFieldTouched
    - After Submit events/methods
        - handleSubmit
        - submitForm

Ex:

```
import React from 'react';
import ReactDOM from 'react-dom';
import {useFormik} from 'formik';

const validateProduct = productData => {
    const errors = {}
    if(!productData.Name) {
        errors.Name = 'Product Name Required';
    } else if(productData.Name.length<4) {
        errors.Name = 'Name too short..Min 4 required';
    }

    if(!productData.Price) {
        errors.Price = 'Product Price Required';
    }

    if(!productData.Code) {
        errors.Code = 'Product Code Required';
    } else if (!/[A-Z]{3}[0-9]{2}/.test(productData.Code)) {
```

```
        errors.Code = 'Invalid Code..';
    }
   return errors;
}

const RegisterProductComponent = () => {
    const formik = useFormik({
        initialValues: {
            Name: '',
            Price: '',
            Code: ''
        },
        validate: validateProduct,
        onSubmit: values => {
            alert(JSON.stringify(values));
        }
    })
    return(
        <div>
            <h2>Register Product</h2>
```

```
        <form onSubmit={formik.handleSubmit}>
          <div className="form-group">
            <label>Name</label>
            <div>
              <input name="Name"
onBlur={formik.handleBlur}
onChange={formik.handleChange}
value={formik.values.Name} type="text" />
              <div className="text-danger">
                {(formik.touched.Name &&
formik.errors.Name)?formik.errors.Name:null}
              </div>
            </div>
          </div>
          <div className="form-group">
            <label>Price</label>
            <div>
              <input type="text"
onBlur={formik.handleBlur}
onChange={formik.handleChange} name="Price"
value={formik.values.Price} />
              <div className="text-danger">
```

```
                    {(formik.touched.Price &&
formik.errors.Price)?formik.errors.Price:null}
                  </div>
                </div>
              </div>
              <div class="form-group">
                <label>Code</label>
                <div>
                  <input onBlur={formik.handleBlur}
onChange={formik.handleChange} name="Code"
value={formik.values.Code} type="text" />
                  <div className="text-danger">
                    {(formik.touched.Code &&
formik.errors.Code)?formik.errors.Code:null}
                  </div>
                </div>
              </div>
              <div className="form-group">
                <button>Register</button>
              </div>
            </form>
```

```
        </div>
    )
}
ReactDOM.render(
    <RegisterProductComponent />,
    document.getElementById("root")
);
export default RegisterProductComponent;
```

Yup Library

- It provides object schema validation.
- It provides a validationSchema object.
- Schema object comprises of key and value reference.
- It uses an error object that can bind with any HTML element and configure the validations for element.
- It comprises of Key and Value reference, where key refers to validation error and value refers to validation message.

Syntax:

```
validationSchema: yup.object({
```

Name:
yup.DataType().required().max().email()
})

- Yup library uses “formik.getFieldProps()” that can give access to values of elements.

Syntax:

<input type=”text” name=”Name” {…formik.getFieldProps(“Name”)} />

Ex:

YupValidationDemo.js

```
import React from 'react';
import {useFormik} from 'formik';
import * as yup from 'yup';

const YupValidationComponent = () => {
  const formik = useFormik({
    initialValues: {
      Name: '',
      Salary: '',
      Email: ''
```

```
    },
    validationSchema: yup.object({
        Name: yup.string()
                .min(4, "Name too short..min 4 chars")
                .max(10, "Name too long.. max 10 chars")
                .required("User Name Required"),
        Salary: yup.number()
                 .required("Salary Required"),
        Email: yup.string()
                .required("Email Required")
                .email("Invalid Email")
    }),
    onSubmit: values => {
        alert(JSON.stringify(values))
    }
})
return(
    <div>
        <h2>Register User</h2>
        <form onSubmit={formik.handleSubmit}>
```

```
        <dl>
          <dt>Name</dt>
          <dd>
            <input type="text" name="Name"
{...formik.getFieldProps("Name")} />
          </dd>
          <dd className="text-danger">
            {(formik.touched.Name &&
formik.errors.Name)? formik.errors.Name: null }
          </dd>
          <dt>Salary</dt>
          <dd>
            <input type="text" name="Salary"
{...formik.getFieldProps("Salary")} />
          </dd>
          <dd className="text-danger">
            {(formik.touched.Salary &&
formik.errors.Salary)? formik.errors.Salary: null}
          </dd>
          <dt>Email</dt>
          <dd>
```

```
                <input type="text" name="Email"
{...formik.getFieldProps("Email")} />
            </dd>
            <dd className="text-danger">
                {(formik.touched.Email &&
formik.errors.Email)? formik.errors.Email: null}
            </dd>
          </dl>
          <button>Register</button>
        </form>
      </div>
    )
}
export default YupValidationComponent;
```

Formik Components

- Formik library provides pre-defined component used for designing a form and its validations.
- The built-in components of Formik library
    - <Formik />
    - <Form />
    - <Field />

- <ErrorMessage />

Syntax:

```
<formik  initialValues = {
    { Name: ‘ ‘, Salary: ‘ ‘, Email: ‘ ‘ }
 },
 validationSchema = {
 },
onsubmit = {
 }
 { props => (
      <Form>
           <Field name=”Name” type=”text”>
</Field>
           <ErrorMessage name=”Name”>
</ErrorMessage>
      </Form>
  )
 }
```

Syntax:

<Formik initialValues={} validationSchema={} onSubmit={}>

{

props=> ()

}

</**Formik>**

Props function in Formik is responsible for identifying the state of every field.

dirty : It returns true when any field in form is modified.

touched **:** It returns true when any field is touched but value not modified.

isValid **:** It returns true when all form fields are valid.

Ex:

ValidationComponent.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import { useFormik, Formik, Form, Field, ErrorMessage } from 'formik';
```

```
import * as yup from 'yup';

const ValidationComponent = () => {

    return(
        <Formik initialValues={
            {
                Name: '',
                Salary: '',
                Email: '',
                City: ''
            }
        } validationSchema={
            yup.object({
                Name: yup.string()
                        .required("Name Required")
                        .min(4, "Name too short")
                        .max(10, "Name too long"),
                Salary: yup.number()
                         .required("Salary Required"),
```

```
            Email: yup.string()
                    .required("Email Required")
                    .email("Invalid Email")
            })
        } onSubmit = {values=> {
            alert(JSON.stringify(values))
        }} >

            {
                props => (
                    <div>
                        <h3>Register User</h3>
                        <Form>
                            <dl>
                                <dt>Name</dt>
                                <dd>
                                    <Field name="Name"
type="text"></Field>
                                </dd>
                                <dd className="text-danger">
```

```
                        <ErrorMessage
name="Name"></ErrorMessage>
                    </dd>
                    <dt>Salary</dt>
                    <dd>
                        <Field name="Salary"
type="text"></Field>
                    </dd>
                    <dd className="text-danger">
                        <ErrorMessage
name="Salary"></ErrorMessage>
                    </dd>
                    <dt>Email</dt>
                    <dd>
                        <Field name="Email"
type="text"></Field>
                    </dd>
                    <dd className="text-danger">
                        <ErrorMessage
name="Email"></ErrorMessage>
                    </dd>
```

```
                        <dt>Your City</dt>
                        <dd>
                            <Field name="City" as="select">
                                <option>Delhi</option>
                                <option>Hyd</option>
                            </Field>
                        </dd>
                    </dl>
                    <button
disabled={props.isValid==false}>Register</button>
                    <button
disabled={props.dirty==false}>Save</button>
                </Form>
            </div>
        )
    }

</Formik>
)
}
ReactDOM.render(
```

```
  <ValidationComponent/>,
  document.getElementById("root")
)
export default ValidationComponent;
```

Styles using Validation Attributes

- You can dynamically apply styles based on validation properties

Syntax:

```
<Form
style={props.isValid?{backgroundColor:'green'}:{backgroundColor:'red'}}>
```

Client-Side Routing in React JS

What is Routing?

- Routing is technique used in web applications to configure User and SEO friendly URL.
- User friendly URL allows to query any content from URL.

  Without Routing:

  http://www.amazon.in/electronics.html?category=electronics&model=samsung

  With Routing:

http://www.amazon.in/electronics/mobile/samsung

- SEO friendly URL allows the web crawlers to identify the exact location in page.
- In SPA routing allows to access everything from one page.
- New content will be added into page without reloading the page.
- Routing can be handled Server-Side and Client-Side.
- React JS uses routing client-side.

Routing Architecture:

Install Router Module

> npm install –save react-router-dom

Ex:

AppHome.js

```
import React from 'react';
```

```
import { BrowserRouter as Router, Route} from 'react-
router-dom';

const Home = () => (
   <h2>App Home</h2>
)

const Contact = () => (
   <div>
      <address>
         NareshIT - Hyd <br />
         hr@nareshit.in
      </address>
   </div>
)

export default class AppHome extends
React.Component
{
   render(){
      return(
```

```
      <Router>
      <Route path="/" component={Home} />
      <Route path="/contact" component={Contact} />
      </Router>
     )
   }
}
```

Switch for Routes

- A <Switch> looks through all its children <Route> elements and renders the first one whose path is matching with specified URL.
- Always use a switch to configure set of routes but only one to render at a time.

Syntax:

```
 <Switch>
  <Route  path=”” component={ }/>
  <Route path=””>
    <Home />
  </Route>
</Switch>
```

Redirection of Route

- The element <Redirect> is used in Route to define redirection to an existing path instead of re-defining the path and its parameters.

Syntax:

<Route path=”/home”>

<Redirect to=”/” />

</Route>

Route for No-Match

- If client request can’t be processed due to “404” status code, which is not-found by default it renders empty page.
- You have configure a generic route with path defined as “*” , which can render alternative if the match not found.

Syntax:

<Route path=”*”>

<NotFound />

</Route>

Note: Always the generic path must be configured at the end.

- Router library provides “useLocation”, which can get the current location details
  let location = useLocation();
- You can also use browser location object
  window.location.pathname

Configuring Links for Navigation

- Router provides <Link> element , which creates an anchor that can redirect to specified route path.
  Syntax:
  <Link to=”/path”> Text </Link>

Ex:

AppHome.js

```
import React from 'react';
import {BrowserRouter as Router, Redirect, Route, Switch, useLocation, Link} from 'react-router-dom';

function Home(){
   return(
      <div>
```

```
            <h2>Amazon Home</h2>
            <p>Shopping Home Page</p>
        </div>
    )
}
function Electronics(){
    return(
        <div>
            <h2>Electronics Home</h2>
            <img src="Images/speaker.jpg" width="100"
height="100" />
            <img src="Images/earpods.jpg" width="100"
height="100" />
        </div>
    )
}
function Footwear(){
    return(
        <div>
            <h2>Footwear Home</h2>
```

```
        <img src="Images/shoe.jpg" width="100"
height="100" />
        <img src="Images/shoe1.jpg" width="100"
height="100" />
      </div>
   )
}
function Fashion(){
   return(
      <div>
         <h2>Fashion Home</h2>
         <img src="Images/shirt.jpg" width="100"
height="100" />
         <img src="Images/jeans.jpg" width="100"
height="100" />
      </div>
   )
}
function NotFound(){

   return(
```

```
        <div>
            <h2>Not-Found</h2>
            <p>Page you requested <code>
{window.location.href} </code> - Not Found</p>
            <p><Link to="/home">Back to
Home</Link></p>
        </div>
    )
}
export default class AppHome extends
React.Component
{
    render(){
        return(
            <Router>
                <header>
                    <h1>Amazon Shopping</h1>
                </header>
                <div>
                    <ul style={{display:'flex', listStyle:'none'}}>
                        <li><Link to="/home">Home</Link></li>
```

```
                    <li><Link
to="/electronics">Electronics</Link></li>
                    <li><Link
to="/footwear">Footwear</Link></li>
                    <li><Link
to="/fashion">Fashion</Link></li>
                </ul>
            </div>
            <hr />
            <Switch>
                <Route exact path="/">
                    <Home />
                </Route>
                <Route path="/home">
                    <Redirect to="/" />
                </Route>
                <Route path="/electronics">
                    <Electronics />
                </Route>
                <Route path="/footwear">
                    <Footwear />
```

```
            </Route>
            <Route path="/fashion">
              <Fashion />
            </Route>
            <Route path="*" >
              <NotFound />
            </Route>
          </Switch>
        </Router>
      )
  }
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
```

```
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon"
href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your
web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/we
b-app-manifest/
    -->
    <link rel="manifest"
href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public`
folder during the build.
      Only files inside the `public` folder can be
referenced from the HTML.
```

```
    Unlike "/favicon.ico" or "favicon.ico",
"%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a
non-root public URL.
    Learn how to configure a non-root public URL by
running `npm run build`.
  -->
  <title>React App</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0s
zH/Ev+nYRRuWlolflfl" crossorigin="anonymous">
  <style>
    li {
      margin-right: 30px;
      border:2px solid tomato;
      border-radius: 10px;
      padding: 5px;
      width: 100px;
```

```
      text-align: center;
    }
    a{
      text-decoration: none;
      color:tomato;
      font-weight: bold;
    }
  </style>
 </head>
 <body class="container">
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
```

The build step will place the bundled scripts into the <body> tag.

To begin the development, run `npm start` or `yarn start`.

To create a production bundle, use `npm run build` or `yarn build`.

-->

</body>

</html>

Route Parameters

- Route Parameters are used to query any content in the component directly from URL.
- Route Parameters are used to transport data across requests.
- Route Parameters are used in the place of query strings

  Syntax:

  http://localhost:3200/product?id=1&name=tv

  http://localhost:3200/1/tv
- Parameters are configured in <Route>

  Syntax:

<Route path=”/product/:param1/:param2”>
</Route>

- Parameters are passed into URL
  Syntax:
  http://localhost:3200/product/param1val/param2val
- To access the route parameters you can use the method
  “useParams()”

Ex:

AppHome.js

```
import React from 'react';

import {BrowserRouter as Router, Redirect, Route, Switch, useLocation, Link, useParams} from 'react-router-dom';


function Home(){
   return(
      <div>
         <h2>Amazon Home</h2>
         <p>Shopping Home Page</p>
```

```
        </div>
    )
}
function Electronics(){
    return(
        <div>
            <h2>Electronics Home</h2>
            <img src="Images/speaker.jpg" width="100"
height="100" />
            <img src="Images/earpods.jpg" width="100"
height="100" />
        </div>
    )
}
function Footwear(){
    return(
        <div>
            <h2>Footwear Home</h2>
            <img src="Images/shoe.jpg" width="100"
height="100" />
```

```
            <img src="Images/shoe1.jpg" width="100"
height="100" />
        </div>
    )
}
function Fashion(){
    return(
        <div>
            <h2>Fashion Home</h2>
            <img src="Images/shirt.jpg" width="100"
height="100" />
            <img src="Images/jeans.jpg" width="100"
height="100" />
        </div>
    )
}
function Details(){
    let {id, name, price} = useParams();
    return(
        <div>
            <h2>Details Page</h2>
```

```
            <dl>
                <dt>Product Id</dt>
                <dd>{id}</dd>
                <dt>Name</dt>
                <dd>{name}</dd>
                <dt>Price</dt>
                <dd>{price}</dd>
            </dl>
        </div>
    )
}
function NotFound(){

    return(
        <div>
            <h2>Not-Found</h2>
            <p>Page you requested <code>
{window.location.href} </code> - Not Found</p>
            <p><Link to="/home">Back to
Home</Link></p>
        </div>
```

```
    )
}
export default class AppHome extends
React.Component
{
   render(){
      return(
         <Router>
            <header>
               <h1>Amazon Shopping</h1>
            </header>
            <div>
               <ul style={{display:'flex', listStyle:'none'}}>
                  <li><Link to="/home">Home</Link></li>
                  <li><Link
to="/electronics">Electronics</Link></li>
                  <li><Link
to="/footwear">Footwear</Link></li>
                  <li><Link
to="/fashion">Fashion</Link></li>
```

```
                    <li><Link
to="/details/1/mobile/5600">Details</Link></li>
                </ul>
            </div>
            <hr />
            <Switch>
                <Route exact path="/">
                    <Home />
                </Route>
                <Route path="/home">
                    <Redirect to="/" />
                </Route>
                <Route path="/electronics">
                    <Electronics />
                </Route>
                <Route path="/footwear">
                    <Footwear />
                </Route>
                <Route path="/fashion">
                    <Fashion />
```

```
                </Route>
                <Route path="/details/:id/:name/:price">
                   <Details/>
                </Route>
                <Route path="*" >
                   <NotFound />
                </Route>
             </Switch>
          </Router>
       )
   }
}
```

Ex: Nested Routes

AppHome.js

```
import React from 'react';
import {BrowserRouter as Router, Redirect, Route, Switch, useLocation, Link, useParams} from 'react-router-dom';

function Home(){
```

```
    return(
        <div>
            <h2>Amazon Home</h2>
            <p>Shopping Home Page</p>
        </div>
    )
}
function Electronics(){
    return(
        <div>
            <h2>Electronics Home</h2>
            <img src="Images/speaker.jpg" width="100"
height="100" />
            <img src="Images/earpods.jpg" width="100"
height="100" />
        </div>
    )
}
function Footwear(){
    return(
        <div>
```

```
            <h2>Footwear Home</h2>
            <img src="Images/shoe.jpg" width="100"
height="100" />
            <img src="Images/shoe1.jpg" width="100"
height="100" />
        </div>
    )
}
function Fashion(){
    return(
        <div>
            <h2>Fashion Home</h2>
            <img src="Images/shirt.jpg" width="100"
height="100" />
            <img src="Images/jeans.jpg" width="100"
height="100" />
        </div>
    )
}

function Categories(){
```

```
    let categories = [
        {CategoryId: 1, CategoryName: "Electronics"},
        {CategoryId: 2, CategoryName: "Footwear"}
    ];
    return(
      <div>
        <h2>Categories</h2>
        <ol>
            {
                categories.map(cat =>
                    <li key={cat.CategoryId}><Link
to={'/products/' +
cat.CategoryId}>{cat.CategoryName}</Link></li>
                    )
            }
        </ol>
      </div>
    )
}

let products = [
```

```
    {ProductId:1, Name: 'Samsung TV', Price: 45000.33, CategoryId: 1},
    {ProductId:2, Name: 'Nike Casuals', Price: 6000.44, CategoryId: 2},
    {ProductId:3, Name: 'Earpods', Price: 3500.44, CategoryId: 1},
    {ProductId:4, Name: 'Lee Boot', Price: 6300.33, CategoryId: 2}
];

function Products(){
    let {id} = useParams();

    let results = products.filter(x=>x.CategoryId==id);
    return(
        <div>
            <h3>Products List</h3>
            <ol>
                {
                    results.map(prod=>(
```

```
                    <li><Link to={'/details/' +
prod.ProductId}>{prod.Name}</Link></li>
                ))
            }
        </ol>
        <div>
            <Link to="/categories">Back to
Categories</Link>
        </div>
    </div>
  )
}

function Details(){
  let {id} = useParams();
  let searchedProduct =
products.find(x=>x.ProductId==id);
  return(
    <div>
      <h2>Details Page</h2>
      <dl>
```

```
            <dt>Name</dt>
            <dd>{searchedProduct.Name}</dd>
            <dt>Price</dt>
            <dd>{searchedProduct.Price}</dd>
            <dt>ProductId</dt>
            <dd>{searchedProduct.ProductId}</dd>
          </dl>
        </div>
    )
}
function NotFound(){

    return(
        <div>
          <h2>Not-Found</h2>
          <p>Page you requested <code>
{window.location.href} </code> - Not Found</p>
          <p><Link to="/home">Back to
Home</Link></p>
        </div>
    )
```

```
}
export default class AppHome extends
React.Component
{
   render(){
      return(
         <Router>
            <header>
               <h1>Amazon Shopping</h1>
            </header>
            <div>
               <ul style={{display:'flex', listStyle:'none'}}>
                  <li><Link to="/home">Home</Link></li>
                  <li><Link
to="/electronics">Electronics</Link></li>
                  <li><Link
to="/footwear">Footwear</Link></li>
                  <li><Link
to="/fashion">Fashion</Link></li>
                  <li><Link
to="/categories">Categories</Link></li>
```

```
                    <li><Link
to="/details/1/mobile/5600">Details</Link></li>
                </ul>
            </div>
            <hr />
            <Switch>
                <Route exact path="/">
                    <Home />
                </Route>
                <Route path="/home">
                    <Redirect to="/" />
                </Route>
                <Route path="/electronics">
                    <Electronics />
                </Route>
                <Route path="/footwear">
                    <Footwear />
                </Route>
                <Route path="/fashion">
                    <Fashion />
```

```
            </Route>
            <Route path="/categories">
              <Categories />
            </Route>
            <Route path="/products/:id">
              <Products />
            </Route>
            <Route path="/details/:id">
              <Details/>
            </Route>
            <Route path="*" >
              <NotFound />
            </Route>
          </Switch>
        </Router>
      )
    }
}
```

Authentication / Secured Routes

- Application-level security includes

- XSS [Cross Site Scripting Attack]
- Request Forgery
- Authorization / Authentication

- Authorization is preventing access to resources.
- Authentication is verifying user credentials like userid, password, security token etc.

Securing React JS Application

- Protection from XSS with Data binding
- Detecting Dangerous URLs
- Rendering HTML
- Direct DOM Access
- Server-side rendering
- Detecting Vulnerabilities in Dependencies
- Injecting JSON state
- Detecting Vulnerable Versions of react.
- Authentication and Authorization

Precautions:

Cross Site Scripting Attack [XSS]

Syntax:

<td> { productName } </td> // Good

<td innerHTML={productName}> </td> // Bad – XSS

Unsecured URLs

- URL may contain dynamic logic or interaction
  Syntax:
   Text
- URL based script injection
- Use HTTP for unsecured and HTTPS for secured
- This requires parsing and verifying URL.
- You can use JavaScript “URL()”
  Syntax:

```
function SecurityTest(){
    function Hello() {
        alert("Hello ! React");
    }
    return(
        <div>
            <h2>Security Test</h2>
            <button onClick={Hello}>Test</button>
                // Secured
            <a href={`javascript:${Hello}`}>Click
Here..</a>          //Not Secured
        </div>
    )
}
```

- If you want HTML to use the string with script then you have to use “DOM Sanitizer”
- It allows to use the functions safely in DOM HTML elements.
- It requires “dangerouslySetInnerHTML” attribute for DOM element.
- It is defined in the library “dompurify”

Ex:

- Install “dompurify”
  > npm install dompurify

- Import dompurify

  Import purify from ‘dompurify’

- Use “sanitize()”

```
function SecurityTest(){
   function Hello() {
      alert("Hello ! React");
   }
   return(
      <div>
         <h2>Security Test</h2>
         <button onClick={Hello}>Test</button>
```

```
        <a
href={purify.sanitize(`javascript:${Hello()}`)}>Click
Here..</a>
      </div>
   )
}
```

Syntax:

```
const msg = “<b>Hello</b>”;
<div
dangerouslySetInnerHTML={purify.sanitize(msg)}></
div>
```

Ex:

- Install dompurify
  > npm install dompurify
- Import dompurify
  import purify from ‘dompurify’
- Use DOM purify with sanitize() method

```
function SecurityTest(){
   const msg = "<b>Hello</b>";
   return(
      <div>
```

```
        <h2>Security Test</h2>
        <div
dangerouslySetInnerHTML={{__html:purify.sanitize(
msg)}} ></div>
      </div>
   )
}
```

Cookies in React

- Cookie is a simple text document
- Cookies comprises of client data
- Cookies can be stored
    - In browser memory [In Memory]
    - In client hard drive [persistent]
- Cookies store client authentication details and re-use across requests.
- Cookies can be defined with expiry so that they are deleted automatically after specific duration.
- JavaScript manages cookies by using
    - document.cookie
- In React you can use local storage

Note: By default, all cookies are in-memory, to make them persistent you have to set expiry for cookie.

Ex:

- Import “useState”

  import React, {useState} from 'react';

- Create a component

```
function Login(){
    const [message, setMessage] = useState('');
    const CreateCookie = (email) => {
        localStorage.setItem('email', email);
        setMessage("Cookie Created");
    }
    return(
        <div>
          <label>Email :</label>
          <input
onChange={(e)=>CreateCookie(e.target.value)}
type="text"/>
          <div>
              {message}
          </div>
        </div>
    )
}
```

React-Cookie Library

- Install library

  > npm install react-cookie

- Import

```
import {useCookies} from 'react-cookie';
```

- Add function

```
function Login(){
   const [message, setMessage] = useState('');
   const [cookies, setCookie] =
useCookies(['useremail']);

   const CreateCookie = (email) => {
      localStorage.setItem('email', email);
      setMessage("Cookie Created");
      let now = new Date();
      now.setTime(now.getTime() +
(24*60*60*1000));
      setCookie('useremail',email, {path: '/'});
   }
   return(
      <div>
        <label>Email :</label>
        <input
onChange={(e)=>CreateCookie(e.target.value)}
type="text"/>
        <div>
            {message} <br />
            {cookies.useremail}
```

```
            </div>
          </div>
        )
    }
```

Authorized Access to Resources

- Secured Routes
- A secured route will verify the credentials and provide access to any resource location when authentication passed.
- You can restrict access to certain resources in application.
- You have to configure a custom router for authorization.

Ex:

SecurityDemo.js

```
import React, {useState, useContext, createContext} from 'react';

import {BrowserRouter as Router, Route, Switch, Redirect, useHistory, useLocation, Link} from 'react-router-dom';
```

```
function PublicPage() {
    return(
        <h2>Public Page</h2>
    )
}

function ProtectedPage(){
    return(
        <h2>Protected Page</h2>
    )
}

const authContext = createContext();

function useAuth() {
    return useContext(authContext);
}

function LoginPage(){
    let auth = useAuth();
```

```
    let history = useHistory();
    let location = useLocation();
    let {from} = location.state || {from: {pathname: "/"}};
    let login = () => {
        auth.signin(()=> {
            history.replace(from);
        })
    }
    return(
        <div>
            <p>Please Login to View Protected Page</p>
            <button onClick={login}>Login</button>
        </div>
    )
}

function AuthButton(){
    let history = useHistory();
    let auth = useAuth();
```

```
    return auth.user ? (<p>Welcome login success.. ! {"
"} <button onClick={()=>{auth.signout(()=>
history.push("/"))}} >Logout</button> </p>) : (<p>You
are not logged in..Please Login..</p>)
}

function PrivateRoute({children, ...rest}) {
    let auth = useAuth();
    return(
        <Route
        {...rest} //path, component
        render={ ({location}) =>  auth.user ? (children) :
(<Redirect to={{pathname: "/login", state:
{from:location}}}/>)}
        />
    )
}

function useProvideAuth(){
    const [user, setUser] = useState(null);
    const signin = callback => {
```

```
        return fakeAuth.signin(()=> {
            setUser("user");
            callback();
        })
    }
    const signout = callback => {
        return fakeAuth.signout(()=> {
            setUser(null);
            callback();
        })
    }
    return {
        user,
        signin,
        signout
    }
}
const fakeAuth = {
    isAuthenticated : false,
    signin(callback) {
```

```
        fakeAuth.isAuthenticated = true;
        setTimeout(callback, 100);
    },
    signout(callback) {
        fakeAuth.isAuthenticated = false;
        setTimeout(callback, 100);
    }
}

function ProvideAuth({children}) {
    const auth = useProvideAuth();
    return (
      <authContext.Provider value={auth}>
          {children}
      </authContext.Provider>
    )
}

export default function SecurityDemo(){
    return(
```

```
      <ProvideAuth>
        <Router>
          <div>
            <AuthButton />
          </div>
          <ul>
            <li>
              <Link to="/public">Public Page</Link>
            </li>
            <li>
              <Link to="/protected">Protected
Page</Link>
            </li>
          </ul>
          <Switch>
            <Route path="/public">
              <PublicPage />
            </Route>
            <Route path="/login">
              <LoginPage />
```

```
            </Route>
            <PrivateRoute path="/protected">
              <ProtectedPage />
            </PrivateRoute>
          </Switch>
        </Router>
      </ProvideAuth>
    )
}
```

Building Cross Platform Mobile Applications in React

[Progressive Web Applications - PWA**]**

- You can develop and design app like experience in browser.
- You can use React Native with Material UI components.
- You can use frameworks like
    - Apache Cordova
    - Ionic
    - Native Script etc..

Building PWA using Ionic in React:

- Ionic is a framework used to build cross platform and PWA, which can run on any device.
- It provides pre-defined components and hooks for React.
- You can use Ionic instead of React Native and Material UI components.

Download and Install Ionic CLI on your PC:

- CLI is Command Line Tool, which provides set of commands used to create and manage your application.

> npm install -g @ionic/cli

Create a new React application integrated with Ionic Framework

> **ionic start appName tabs --type=react**

Note: It will create a react application integrated with ionic framework that provides library of pre-defined components and UX.

- Unified UX
    - Your application will have same behaviour across all devices

    - Mobile users can get access to everything.
    - It will not optimize your application.
- Fluid UX
    - User will stay on one page and gets access to everything from the page.
    - SPA and PWA

Update your react application to use react hooks and PWA

> npm install @ionic/react-hooks @ionic/pwa-elements

[Ionic provides various components derived from “pwa-elements”]

Note: Ionic react uses “TypeScript” – tsx [.ts]

Setup “CustomElements” from window

- Go to “index.tsx” in “**src”** folder
- Import and configure “pwa-elements”

```
import { defineCustomElements } from '@ionic/pwa-elements/loader';

ReactDOM.render(
  <React.StrictMode>
```

```
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
defineCustomElements(window);
```

Run Your React Application with Ionic

```
> ionic serve
```

App.tsx is the startup component, you can customize App.tsx in “src”

```
import { Redirect, Route } from 'react-router-dom';
import {
  IonApp,
  IonIcon,
  IonLabel,
  IonRouterOutlet,
  IonTabBar,
  IonTabButton,
  IonTabs,
```

```
} from '@ionic/react';
import { IonReactRouter } from '@ionic/react-router';
import { ellipse, square, triangle, call, images, camera } from 'ionicons/icons';
import Tab1 from './pages/Tab1';
import Tab2 from './pages/Tab2';
import Tab3 from './pages/Tab3';

/* Core CSS required for Ionic components to work properly */
import '@ionic/react/css/core.css';

/* Basic CSS for apps built with Ionic */
import '@ionic/react/css/normalize.css';
import '@ionic/react/css/structure.css';
import '@ionic/react/css/typography.css';

/* Optional CSS utils that can be commented out */
import '@ionic/react/css/padding.css';
import '@ionic/react/css/float-elements.css';
import '@ionic/react/css/text-alignment.css';
```

```
import '@ionic/react/css/text-transformation.css';
import '@ionic/react/css/flex-utils.css';
import '@ionic/react/css/display.css';

/* Theme variables */
import './theme/variables.css';

const App: React.FC = () => (
  <IonApp>
    <IonReactRouter>
      <IonTabs>
        <IonRouterOutlet>
          <Route exact path="/tab1">
            <Tab1 />
          </Route>
          <Route exact path="/tab2">
            <Tab2 />
          </Route>
          <Route path="/tab3">
            <Tab3 />
```

```
    </Route>
    <Route exact path="/">
      <Redirect to="/tab1" />
    </Route>
  </IonRouterOutlet>
  <IonTabBar slot="bottom">
    <IonTabButton tab="tab1" href="/tab1">
      <IonIcon icon={call} />
      <IonLabel>Call</IonLabel>
    </IonTabButton>
    <IonTabButton tab="tab2" href="/tab2">
      <IonIcon icon={camera} />
      <IonLabel>Camera</IonLabel>
    </IonTabButton>
    <IonTabButton tab="tab3" href="/tab3">
      <IonIcon icon={images} />
      <IonLabel>Gallery</IonLabel>
    </IonTabButton>
  </IonTabBar>
</IonTabs>
```

```
    </IonReactRouter>
  </IonApp>
);

export default App;
```

- **Add a new folder “hooks”**
- **Add a new file into Hooks “usePhotoGallery.ts”**

```
import { useCamera } from '@ionic/react-hooks/camera';

import { CameraResultType, CameraSource, CameraPhoto, Capacitor, FilesystemDirectory } from '@capacitor/core';

export function usePhotoGallery(){
    const { getPhoto } = useCamera();
    const takePhoto = async () => {
        const cameraPhoto = await getPhoto({
            resultType : CameraResultType.Uri,
            source: CameraSource.Camera,
            quality: 100
```

```
        })
      }
      return{
        takePhoto
      };
    }
```

- **Go to “tab-2.tsx”**

```
import { IonContent, IonHeader, IonPage, IonTitle,
IonToolbar } from '@ionic/react';
import ExploreContainer from
'../components/ExploreContainer';
import './Tab2.css';
import { usePhotoGallery } from
'../hooks/usePhotoGallery';

const Tab2: React.FC = () => {
  const { takePhoto } = usePhotoGallery();
  takePhoto();
  return (
    <IonPage>
      <IonHeader>
```

```
        <IonToolbar>
          <IonTitle>Camera</IonTitle>
        </IonToolbar>
      </IonHeader>
      <IonContent fullscreen>
        <IonHeader collapse="condense">
          <IonToolbar>
            <IonTitle size="large">Camera</IonTitle>
          </IonToolbar>
        </IonHeader>
        <ExploreContainer name="Tab 2 page" />
      </IonContent>
    </IonPage>
  );
};

export default Tab2;
```

Material UI

- It provides pre-defined components for react application.
- You can customize and inject into application.

- Material UI is similar to bootstrap, but native to react application.
- More compatible with React application.
- Material UI provides
    - Container
    - Grid
    - Modal
    - Accordion
    - Alerts etc..

Download and Install Material UI for your React Application

- Change to your project folder and run the following command
  > npm install @material-ui/core
- You can also download material icons
  > npm install @material-ui/icons

Ex: **Card in Material-UI**

- Material UI provides various components for card
    - <CardHeader>
    - <CardMedia>
    - <CardContent>
    - <CardActions>
    - <Collapse>
    - <Avatar>

- You have to import all the library required for components

Ex: Card

```
import React from 'react';
import { makeStyles } from '@material-ui/core/styles';
import clsx from 'clsx';
import Card from '@material-ui/core/Card';
import CardHeader from '@material-ui/core/CardHeader';
import CardMedia from '@material-ui/core/CardMedia';
import CardContent from '@material-ui/core/CardContent';
import CardActions from '@material-ui/core/CardActions';
import Collapse from '@material-ui/core/Collapse';
import Avatar from '@material-ui/core/Avatar';
import IconButton from '@material-ui/core/IconButton';
import Typography from '@material-ui/core/Typography';
```

```
import { red } from '@material-ui/core/colors';
import FavoriteIcon from '@material-ui/icons/Favorite';
import ShareIcon from '@material-ui/icons/Share';
import ExpandMoreIcon from '@material-ui/icons/ExpandMore';
import MoreVertIcon from '@material-ui/icons/MoreVert';

const useStyles = makeStyles((theme) => ({
  root: {
    maxWidth: 345,
  },
  media: {
    height: 0,
    paddingTop: '56.25%', // 16:9
  },
  expand: {
    transform: 'rotate(0deg)',
    marginLeft: 'auto',
    transition: theme.transitions.create('transform', {
```

```
      duration: theme.transitions.duration.shortest,
    }),
  },
  expandOpen: {
    transform: 'rotate(180deg)',
  },
  avatar: {
    backgroundColor: red[500],
  },
}));

export default function RecipeReviewCard() {
  const classes = useStyles();
  const [expanded, setExpanded] =
React.useState(false);

  const handleExpandClick = () => {
    setExpanded(!expanded);
  };
```

```
  return (
    <Card className={classes.root}>
      <CardHeader
        avatar={
          <Avatar aria-label="recipe"
classNameclasses.avatar}>
            R
          </Avatar>
        }
        action={
          <IconButton aria-label="settings">
            <MoreVertIcon />
          </IconButton>
        }
        title="Shrimp and Chorizo Paella"
        subheader="September 14, 2016"
      />
      <CardMedia
        className={classes.media}
        image="/static/images/cards/paella.jpg"
```

```
        title="Paella dish"
      />
      <CardContent>
        <Typography variant="body2"
color="textSecondary" component="p">
          This impressive paella is a perfect party dish and
a fun meal to cook together with your
          guests. Add 1 cup of frozen peas along with the
mussels, if you like.
        </Typography>
      </CardContent>
      <CardActions disableSpacing>
        <IconButton aria-label="add to favorites">
          <FavoriteIcon />
        </IconButton>
        <IconButton aria-label="share">
          <ShareIcon />
        </IconButton>
        <IconButton
          className={clsx(classes.expand, {
            [classes.expandOpen]: expanded,
```

```
          })}
          onClick={handleExpandClick}
          aria-expanded={expanded}
          aria-label="show more"
        >
          <ExpandMoreIcon />
        </IconButton>
      </CardActions>
      <Collapse in={expanded} timeout="auto"
unmountOnExit>
        <CardContent>
          <Typography paragraph>Method:</Typography>
          <Typography paragraph>
            Heat 1/2 cup of the broth in a pot until
simmering, add saffron and set aside for 10
            minutes.
          </Typography>
          <Typography paragraph>
            Heat oil in a (14- to 16-inch) paella pan or a
large, deep skillet over medium-high
```

heat. Add chicken, shrimp and chorizo, and cook, stirring occasionally until lightly

browned, 6 to 8 minutes. Transfer shrimp to a large plate and set aside, leaving chicken

and chorizo in the pan. Add pimentón, bay leaves, garlic, tomatoes, onion, salt and

pepper, and cook, stirring often until thickened and fragrant, about 10 minutes. Add

saffron broth and remaining 4 1/2 cups chicken broth; bring to a boil.

</Typography>

<Typography paragraph>

Add rice and stir very gently to distribute. Top with artichokes and peppers, and cook

without stirring, until most of the liquid is absorbed, 15 to 18 minutes. Reduce heat to

medium-low, add reserved shrimp and mussels, tucking them down into the rice, and cook

again without stirring, until mussels have opened and rice is just tender, 5 to 7

minutes more. (Discard any mussels that don’t open.)

```
          </Typography>
          <Typography>
            Set aside off of the heat to let rest for 10
minutes, and then serve.
          </Typography>
        </CardContent>
      </Collapse>
    </Card>
  );
}
```

Ex: Avtar

```
import React from 'react';
import { makeStyles } from '@material-ui/core/styles';
import Avatar from '@material-ui/core/Avatar';

const useStyles = makeStyles((theme) => ({
  root: {
    display: 'flex',
```

```
    '& > *': {
      margin: theme.spacing(1),
    },
  },
}));

export default function ImageAvatars() {
  const classes = useStyles();

  return (
    <div className={classes.root}>
      <Avatar alt="Remy Sharp"
src="/static/images/avatar/1.jpg" />
      <Avatar alt="Travis Howard"
src="/static/images/avatar/2.jpg" />
      <Avatar alt="Cindy Baker"
src="/static/images/avatar/3.jpg" />
    </div>
  );
}
```

Ex: Dialog

```
import React from 'react';
import Button from '@material-ui/core/Button';
import Dialog from '@material-ui/core/Dialog';
import DialogActions from '@material-ui/core/DialogActions';
import DialogContent from '@material-ui/core/DialogContent';
import DialogContentText from '@material-ui/core/DialogContentText';
import DialogTitle from '@material-ui/core/DialogTitle';

export default function AlertDialog() {
  const [open, setOpen] = React.useState(false);

  const handleClickOpen = () => {
    setOpen(true);
```

```
  };

  const handleClose = () => {
    setOpen(false);
  };

  return (
    <div>
      <Button variant="outlined" color="primary"
onClick={handleClickOpen}>
        Open alert dialog
      </Button>
      <Dialog
        open={open}
        onClose={handleClose}
        aria-labelledby="alert-dialog-title"
        aria-describedby="alert-dialog-description"
      >
        <DialogTitle id="alert-dialog-title">{"Use Google's
location service?"}</DialogTitle>
        <DialogContent>
```

```
        <DialogContentText id="alert-dialog-
description">
          Let Google help apps determine location. This
means sending anonymous location data to
          Google, even when no apps are running.
        </DialogContentText>
      </DialogContent>
      <DialogActions>
        <Button onClick={handleClose} color="primary">
          Disagree
        </Button>
        <Button onClick={handleClose} color="primary"
autoFocus>
          Agree
        </Button>
      </DialogActions>
    </Dialog>
  </div>
 );
}
```

React Native

- It is a cross platform library for building mobile native applications.
- Unified experience across devices [Unified-UX].
- Android and iOS native applications.
- React native provides native components, which can run on any device with unified UX.

Core Components of React Native

React Native UI Component	Android View	IOS View	Web View	Description
<View>	<ViewGroup>	<UIView>		It is a container that supports layout with flexbox, style, touch handling etc.
<Text>	<TextView>	<UITextView>	<p>	Display styles, strings of text, handling touch

				events.
<Image>	<ImageView>	<UIImageView>	<img>	Displaying different types of images.
<ScrollView>	<ScrollView>	<UIScrollView>		It is scrollable container, overflow effects
<TextInput>	<EditText>	<UITextField>	<input type=”text”>	Allows text input.
<StyleSheet>				CSS Stylesheets
<button>				
<Switch>				
<FlatList>				Scrollable lists
<SectionList>				Sectioned list
<Alert>				
<Modal>				
<StatusBar>				

Install “Expo-CLI” for building React Native Application

> npm install -g expo-cli

Create a React Native Application

> expo init mobile-shopping

Run Application

C:\mobile-shopping> npm start

[Application starts development environment where you can choose the target platform]

Syntax: React Native Component

Header.js

```
import React from “react”;
import {StyleSheet, View, Text } from ‘react-native’;
const styles = StyleSheet.create({
    header: {
    }
   headerTitle: {
   }
})
const Header = props => {
```

```
  return(
      <View style={styles.header}>
       <Text style={styles.headerTitle} > {props.title}
</Text>
     </View>
  )
}
```