React Js Notes Author By Shudhakar Sharma (Naresh IT Hyd)

Features of React

1. Virtual DOM

a) What is DOM

b) What is shadow DOM

c) What is Virtual DOM

2. Faster in rendering

3. Modular

4. Loosely coupled and Extensible

5. Asynchronous

6. Component Based

Cons of React:

- React is not designed for what you are using.

- Hence lot of GAP'S.

- It requires lot of 3rd Party Integrations

Using React Library in Building Complex U

You can use react in an existing web application.

a) React upto 18 version provides CDN library for using react in existing app.

b) React 18+ version required a library to setup for existing app.

Note: To use react in any web page you need following libraries

a) react

b) react-dom

c) babel

react : It is the core library of react.

react-dom : It is virtual DOM library.

babel : It is a JavaScript compiler

- Upto React 18 version [16, 17, 18] CDN links are provided for library

https://legacy.reactjs.org/docs/cdn-links.html

"react.development.js"

"react-dom.development.js"

- Get CDN link of Babel Standalone compiler

https://babeljs.io/docs/babel-standalone

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/babel">
        ReactDOM.render("Welcome to
React",document.getElementById("root"));
    </script>
</head>
<body>
    <noscript>Please enable JavaScript in your browser</noscript>
    <h2>Home</h2>
    <div id="root"></div>
    <a href="index.html">Index</a>
</body>
</html>
```

What is React?

Why we need React?

What are the challenges in modern web development?

Build SPA

What are the issues in JavaScript and jQuery?

- Require lot of DOM interactions

- Direct DOM interactions will take more time.

- Slow in rendering.

- Require lot of AJAX explicitly

- Lengthy operations are required for

    Data Binding

    Style Binding

    Class Binding

    Event Binding

What are the features of React?

- It uses Virtual DOM.

a) What is DOM?

- It is a hierarchy of elements in browser.

- It is reffered as Document Object Model.

b) What is Shadow DOM?

- It is a hierarchy of elements in component.

- HTML component is a combination of markup, styles and logic.

c) What is Virtual DOM?

- It is a copy of DOM in browser memory.

- React uses virtual DOM to handle manipulations, which later updates the

  actual DOM.

- Hence it is faster in rendering when compared to other technolgies.

- It is modular.

    * Application specific library

    * Light weight

    * Faster

//

- It uses Async techniques [Promises]

- It is component based. Easy to extend and Loosely coupled

- It reduces the compatibility issues

What are the issues with react?

2. Using React in existing application by downloading all libraries [version upto 18]

- Install the following libraries in your project using package manager NPM

```
>npm  install  react
>npm  install  react-dom
>npm  install  @babel/standalone
```

- All library files are copied into "node_modules"

- Link the following file into page

node_modules/react/umd/react.development.js

node_modules/react-dom/umd/react-dom.development.js

node_modules/@babel/standalone/babel.js

Ex: home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../node_modules/react/umd/react.development.js"></script>
    <script src="../node_modules/react-dom/umd/react-dom.development.js"></script>
    <script src="../node_modules/@babel/standalone/babel.js"></script>
    <script type="text/babel">
        ReactDOM.render("Welcome to React JS",document.getElementById("root"));
    </script>
</head>
<body>
    <noscript>Please enable JavaScript in your browser</noscript>
    <h2>Home</h2>
    <div id="root"></div>
    <a href="index.html">Index</a>
</body>
```

</html>

React 18+ Version

- It removed all CDN links for using react in existing application.

- It recommends to use any bundling tool for creating a complete environment for react.

- There are various bundling tools

Vite

Webpack

Parcel

Create a react 18 application with pre-defined environment:

---

1. Open any drive location on your PC in command prompt

2. Run the following command

D:\>npx create-react-app react-shopping

[appName]

3. **Open your project folder in VS code**

File / Folder Description

---

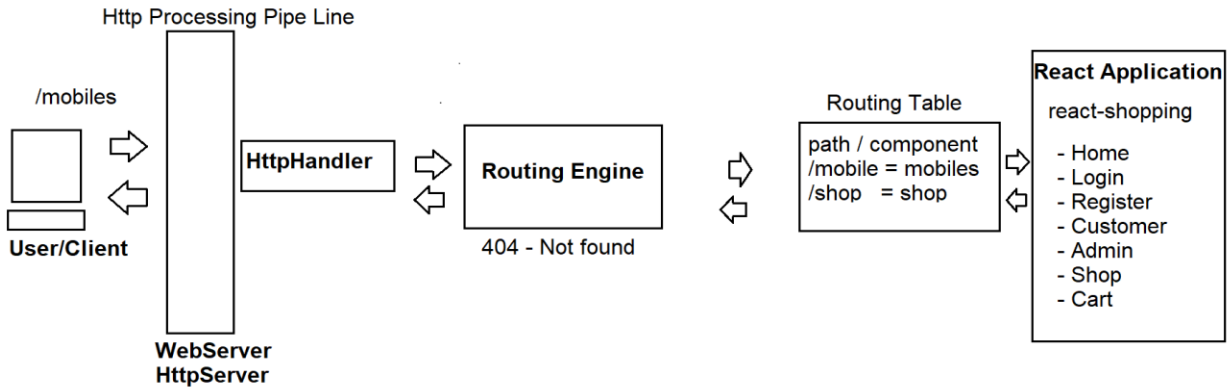
node_modules It comprises of all library files installed using NPM

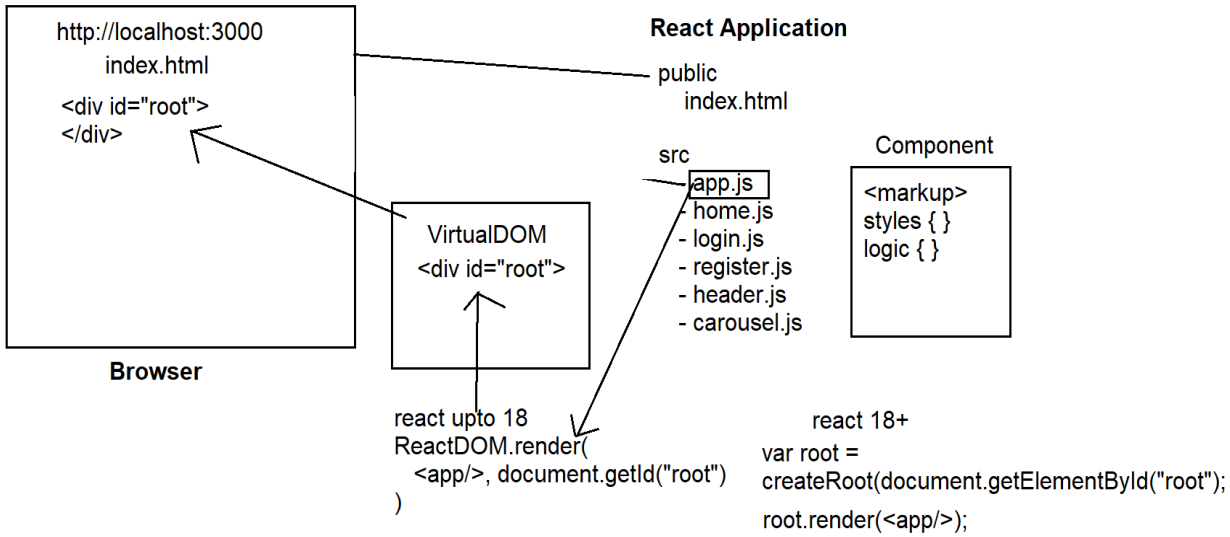
public	It comprises of all static resources [html, images, docs, multimedia]
src	It comprises of all dynamic resources [.css, .scss, .js, .jsx, .ts, .tsx, .test.js, .spec.js]
.gitignore	It configures the files to ignore while pushing the project on to GIT.
package.json	It comprises project meta data
package.lock.json	It comprises of library meta data
readme.md	It a help document for developers

FAQ: What is "npm start" ? Why you use npm start to start a project?

Ans: Batch Operation | Batch Job

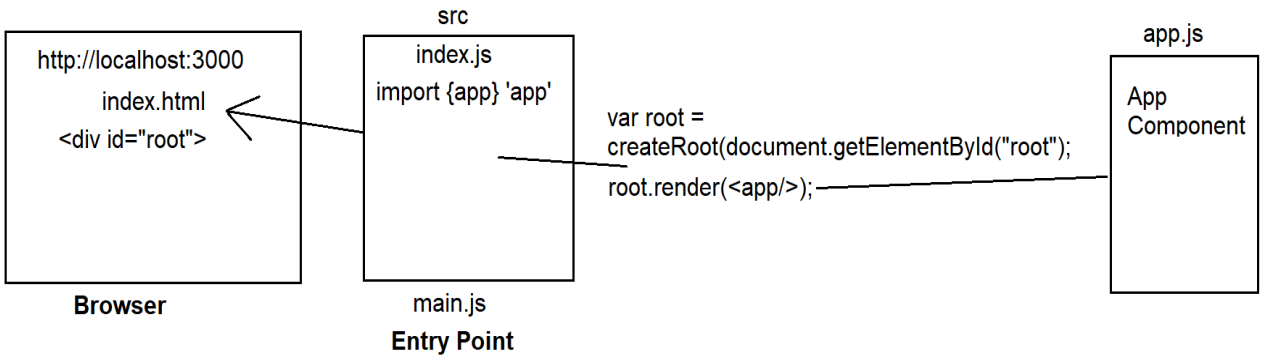
React Application Architecture


Http Processing Pipe Line
/mobiles
User/Client
HttpHandler
WebServer
HttpServer
Routing Engine
404 - Not found
Routing Table
path / component
/mobile = mobiles
/shop = shop
React Application
react-shopping
- Home
- Login
- Register
- Customer
- Admin
- Shop
- Cart


http://localhost:3000
index.html

Browser
VirtualDOM

React Application
public
index.html
src
app.js
- home.js
- login.js
- register.js
- header.js
- carousel.js
Component
<markup>
styles { }
logic { }
react upto 18
ReactDOM.render(
<app/>, document.getId("root")
)
react 18+
var root =
createRoot(document.getElementById("root");
root.render(<app/>);


http://localhost:3000
index.html

Browser
src
index.js
import {app} 'app'
main.js
Entry Point
var root =
createRoot(document.getElementById("root");
root.render(<app/>);
app.js
App
Component

FAQ's

1. What is a bundler?

A. It is a software tool used to configure an end to end application.

2. What is end to end configuration?

A. It is the process of configuring application for

a) Design

b) Build

c) Debug

d) Test

e) Deploy

FAQ: How in react without linking any file in index page those all file are working?

Ans: By using Bundling

3. Name some bundling tools?

a) Webpack

b) Vite

c) Parcel

4. What is NPX?

A. It is used to create a react application using webpack as bundler.

[Node Package]

React Components

- A component a like a template, which provides design, styles and logic that you can implement and customize according to requirements.

- Every component is built with 3 basic elements

    a) Presentation

    b) Styles

    c) Functionality

- Presentation is designed by using Markup. [HTML]

- Styles are defined by using [CSS]

- Functionality is configured by using [JavaScript | TypeScript]

Designing Components:

- React components can be designed by using JavaScript Functions or JavaScript Classes.

JavaScript Functions

---

1. Function Declaration
2. Function Expression
3. Function Definition
4. Function Parameters

    a) Parameter Data Types

    b) REST Parameter

    c) Spread Syntax

5. Function Return

6. Function Closure

7. Function Recursion

8. Function Debounce

9. Function Throttle

10. Arrow Functions

11. Anonymous Functions

12. IIFE Pattern

13. Function Callbacks

14. Function Promise

15. Async Functions [Obsolete]

Function Declaration & Expression

- Declaration comprises of function name and parameters.

```
function  Name()
{
}
```

- Expression comprises of a reference when function is stored.

```
const  Name = function() {

 }
let  Name = function() {
```

```
            return "A";
        }
        Name = function() {
            return 10 + 20;
        }
f
```

Ex:

```
<script>
   var password = prompt("Enter password");
   var response = function(){
      return "Welcome";
   }
   if(password=="admin"){
      response = function(){
         return "Login Success";
      }
   } else {
      response = function(){
         return "Invalid login";
      }
   }
   document.write(response()); // IIFE [Immediately Invoking Function Expression]
</script>
```

FAQ : How to call aynomumus function(aynomumus = a function without name)?

ANS: we can call response(); //IIFE [Immediately Invoking function Expression]

Function Definition

- It defines the actions to perform.

```
function  PrintName()
{ }
```

FAQ : What is difference between function Declaration & Expression?

ANS:

```
function PrintName()          => Declaration
PrintName()                   => Signature
{ }                           => Definition
```

Function Parameters

- Parameters are used to modify a functions functionality.

Ex:

```
<script>
    function Login(type){
        document.write(`
            <input type=${type}>
        `);
    }
    Login("button");
```

```
    Login("radio");
    Login("color");
    Login("date");
</script>
```

- **Function Parameters are 2 types**
    a) Formal Parameters
    b) Actual Parameters

```
function  Login(type) {                    type => formal  parameter

}
Login("button")                            "button" => actual parameter
```

formal parameter = actual parameter;
type="button"

- You can configure multiple parameters
- A parameter can be any type
    a) Primitive Data Type
        - number, string, boolean, symbol, null, undefined
    b) Non Primitive Data Type
        - array, object, map
    c) Function

Ex:

```
<script>
```

```
    function Login(type, val){
        document.write(`
            <input type=${type} value=${val}>
        `);
    }
    Login("button", "Register");
    Login("radio");
    Login("color", "#ff0000");
    Login("date", "2023-08-15");
</script>
```

- ECMA Standards refer max 1024 parameters.
- ES5+ introduced "rest" parameter.
- A single rest parameter allows multiple arguments.
- It is defined by using "...paramName"

Ex:

```
<script>
    function Details(...product)
    {
        var [id, name, price, stock] = product;

document.write(`Id=${id}<br>Name=${name}<br>Price=${price}<br>Stock=${st
ock}`);
    }
    Details(101,"Mobile",40000.44, true);
</script>
```

- Every function can have only one rest parameter.

- rest parameter must be the last parameter in formal list.

Ex:

```
<script>
    function Details(title,...product)
    {
        var [id, name, price, stock] = product;

document.write(`<h2>${title}</h2>Id=${id}<br>Name=${name}<br>Price=${price}<br>Stock=${stock}`);
    }
    Details("Product Details",101,"Mobile",40000.44, true);
</script>
```

- Spread is an approach of configuring one actual value that spreads content into multiple formal parameters.

- Spread value must be a collection. [ ]

Ex:

```
<script>
    function Details(id, name, price, stock)
    {

document.write(`Id=${id}<br>Name=${name}<br>Price=${price}<br>Stock=${stock}`);
    }
    var values = [101,"Mobile",40000.44, true];
```

```
    Details(...values);
</script>
```

FAQ: what is difference between rest and spread parameter?

ANS: Rest is about formal parameter & spread is about actual parameter

Function with Return

- "return" is a jump statement.
- It allows the function to keep its memory alive.
- It terminates the function before it disposes memory.
- Hence function memory can be used to store the result returned by function operations.

Syntax:

```
function Name()
{
  return value;
}
```

Ex:

```
<script>
  function Add(a,b){
     return a + b;
  }
  function Print(){
     document.write("Addition=" + Add(20,10));
  }
```

```
   Print();
</script>
```

- A function with return is allocating memory, where you can store any type of value.

Syntax:

```
function Name(){
        return  name | string | boolean | array | object | date | regExp ..;
}
```

Ex:

```
function values() {
  return  [ ] | { } | new Date() | /regExp/;
}
```

FAQ: Can a function have a multiple return keywords?

- A function can have multiple returns, which is required to handle "conditional rendering".

Syntax:

```
function Name(param)
{
    if(condition) {
        return  "A";
    }
   else {
       return "B";
```

```
        }
    }
```

Function Closure

FAQ: what is function closure?

- Closure is a mechanism where the members of outer function are accessible to inner function is called closure function.

Ex:

```
<script>
    function Outer(){
        var x = 10;
        function Inner(){
            var y = 20;
            var z = x + y;
            return z;
        }
        document.write("Z=" + Inner());
    }
    Outer();
</script>
```

Function Recursion

- Recursion is a technique of calling a function with in the context of same function.

Syntax:

```
function f1(){
   f1();
}
```

FAQ: what is return keyword in computer programming?

ANS: it a jump statement

It is terminate {} block

Ex:

```
<script>
    function Fact(n){
       if(n==0){
         return 1;
       } else {
          return n * Fact(n-1);
       }
    }
    document.write(`Factorial of 7 is  ${Fact(7)}`);
</script>
```

Function Debounce

- Keyboard uses bounce mechanism when key is pressed.
- You can delay and make it debounce by using an event "setTimeout()"

Ex:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
    <script>
      function Msg1(){
         document.querySelector("p").innerHTML = "0";
      }
      function Msg2(){
         document.querySelector("p").innerHTML = "1";
      }
      function Msg3(){
         document.querySelector("p").innerHTML = "100";
      }
      function ClickMe(){
         setTimeout(Msg1, 2000);
         setTimeout(Msg2, 3000);
         setTimeout(Msg3, 4000);
      }
    </script>
</head>
<body>
   <button onclick="ClickMe()">Volume</button>
   <p align="center"></p>
</body>
```

</html>

Function Throttle

- Function Throttle is used to limit the number of times a function have to execute when any event is triggered.

Syntax:

```
const throttleFunction =  function (callBack, dealy) {

        // timer events ...

}
```

Anonymous Function

- A function without name.
- It is configured using ( ).
- It is invoked using IIFE pattern.

Syntax:

```
<script>
   (function(){
      document.write("Hello ! React");
   })();
</script>
```

Arrow Function

- It is a short hand technique of writing a function expression.

Syntax:

```
function Name(uname) {
        return  "Hello ! " + uname;
 }
```

Arrow:

```
const  Name = (uname)=> "hello! " + uname;
const  Add = (a,b) => a + b;
```

() function

=> return and definition

{ }

Ex:

```
<script>
   const add = (a,b) => a + b;
   document.write(add(10,20));
</script>
```

Function Callbacks

- It is a mechanism of configuring set of functions that execute according to state and situation.
- Callbacks use "Sync" technique. [Synchronous]
- It is always slow.
- They use blocking technique.

Ex:

```
<script>
```

```
    function Fetch(url, success, failure){
      if(url=="http://api.com/products") {
         success();
      } else {
         failure();
      }
    }
    Fetch(
      prompt("Enter URL"),
      function(){
         document.write("Data Fetched Successfully..");
      },
      function(){
         document.write("Invalid Url- Unable to Fetch Data");
      }
    )
</script>
```

Promise

- Promise is an alternative for callbacks in modern Javascript.
- Promise is Asynchronous.
- It is faster when compared to callbacks.
- Promise comprises of 3 phases
    a) Initial
    b) Resolved
    c) Rejected
- Initial phase defines the actions to perform
- Resolved specifies that promise fullfilled.
- Rejected specifies that promise failed.
- Resolved in defined with "then()"
- Rejected is defines with "catch()"

Syntax:

```
promise.then().catch().finally()
```

Ex:

```
<script>
   var Fetch = new Promise((resolve, rejected)=>{
      var url = prompt("Enter Url");
      if(url=="http://api.com/products"){
         resolve("Data Fetched Successfully..");
```

```
        } else {
            rejected("Invalid URL: Unable to fetch data");
        }
    });
    Fetch().then((msg)=>{
        document.write(`Success: ${msg}`);
    }).catch((msg)=>{
        document.write(`Error: ${msg}`);
    })
</script>
```

Ex:

```
<script>
    fetch("http://fakestoreapi.com/products")
    .then((response)=>{
        return response.json();
    })
    .then((products)=>{
        products.map(product=>{
            document.write(product.title + "<br>");
        })
    })
    .catch((err)=>{
        document.write(err);
    })
</script>
```

Function Generator

- Generator is function technique used for configuring iterations.
- It returns a value by using "yeild".
- It moves to next iteration by using "next()"
- It initializes the reading by using "value = defined | undefined"
- It stops when "done()" is true.

Ex:

```
<script>
     function* GetValues(){
         yield 10;
         yield 20;
         yield 30;
     }
```

```
    var obj = GetValues();
    console.log(obj.next());
    console.log(obj.next());
    console.log(obj.next());
    console.log(obj.next());
</script>
```

JavaScript Module System

- A JavaScript module comprises of values, functions and classes.
- You can configure a module with set of functions, values or classess. So that you can import and implement from various locations in project or across projects.
- Every JavaScript file is considered as a Module.

Home.js => Home Module

- Module comprises of
    a) values
    b) functions
    c) classes
- Every member of module is private in access.
- If you want any member accessible outside module then you have to mark it as
    "export"
- To use modules in JavaScript you need a special module system installed and invoked.
- There are various JavaScript module systems that you can configure
    a) Common JS
    b) Require JS
    c) AMD [Asyncrhonous Module Distribution]
    d) UMD [Universal Module Distribution]
    e) ESModule

```
    <script type="text/javascript">

    </script>

    <script type="module">

    </script>
```

- You can access members of any module in HTML page by importing the module.

```
import { functionName, className } from "moduleName"; => UMD, ES
var refName = require("ModuleName");
refName.functionName()
refName.className
```

Build React Components using Functions 08/08/2023

- A component comprises of
    a) Presentation
    b) Styles
    c) Logic
- Presentation is defined with Markup [HTML]
- Styles are defined with css or scss [or less]. [scss is sass file]
- Logic is defined with JavaScript or TypeScript.

FAQ: Why we need components?

Ans:

- Components are building blocks for react application.
- You can create reusable and extendable templates.
- You can configure components using
    a) Functions
    b) Classes

Function Component Rules:

- Every function component name must start with uppercase letter. [Pascal Case]
- You can configure as declaration or expression.

Syntax:

```
function  LoginComponent()
{
}

const  LoginComponent = ()=> {
```

```
}

const  LoginComponent = function() {

}
```

- Function Name can't be a pre-defined JavaScript library method names.
- Component function must be defined with export.
- You can configure as "default" export. It uses eager loading approach.
- Every module can have only one default export.
- Every function component must return a Markup.
- Component file can be ".js, .jsx, .ts, .tsx"
- JSX refers to JavaScript Extention Library, babel can use both JS & JSX.
- JSX configures loosely coupled UI.

Syntax:

```
export function Login()
{
    return(
          <markup> {  } </markup>
     )
}
```

- JSX will not allow multiple lines, everything must be in one container.

(or)

JSX can return only one Fragment of markup.

```
export function Login()
{
  return (
          <h2>User Login </h2>                    => invalid
          <p> user can login </p>
     )
}


export function Login()
{
  return(
          <div>
```

```
                                    <h2>User Login </h2>                    =>
valid
                                    <p> user can login </p>
                            </div>
                        )
                }
```

- JSX fragment can be configured using 3 basic methods
    a) Any HTML container element which is not RC data type.
    b) <> </>

```
                        return(
                            <>
                                ...your markup ...
                            </>
                        );
```

    c) React.Fragment

```
                        return(
                            <React.Fragment>
                                ....your content ....
                            </Recat.Fragment>
                        )
```

- JSX will not allow void elements. Hence every element must be configured with end token.

```
<img>               => invalid
<img> </img>        => valid
<img />             => valid
<br>                        => invalid
<br />              => valid
<input type="text" />
<input type="text"> </input>
```

- JSX will not allow attributes, it allows only properties.

FAQ: What is difference between attribute & property?
Ans:
Attribute is immutable.

Property is mutable

Note: HTML element is configured with attribute statically.
HTML element is configured with property dynamically.
HTML element attribute and property name may not match everytime.

attribute	property
src	src
class	className
value	value

Few HTML elements attributes are not available as properties

table	attribute	property
	width	width
	class	className
	border	border
	height	N/A

- You can embed dynamic values directly into markup by using data binding expression
"{ }"

```
<h2> Addition = {10 + 20} </h2>

<div> Stock : {(stock==true)?"Available":"Out of Stock"} </div>
```

Ex:
1. Go to "src" folder
2. Add a new folder "components"
3. Add a child folder
"login"
- login.jsx
- login.css

4. login.css

```
.container {
   display: flex;
   justify-content: center;
   align-items: center;
   height: 100vh;
}
form {
   border:1px solid gray;
   padding: 10px;
   border-radius: 20px;
   box-shadow: 3px 2px 2px gray;
}
```

5. login.jsx

```
import "./login.css";

export function Login(){
   return(
      <div className="container">
         <form>
         <h2>User Login</h2>
         <dl>
            <dt>User Name</dt>
            <dd><input type="text" /></dd>
            <dt>Password</dt>
            <dd><input type="password"/></dd>
         </dl>
         <button>Login</button>
         <button>Cancel</button>
       </form>
      </div>
   )
}
```

6. Go to Index.js

```
import { Login } from './components/login/login';
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Login />
  </React.StrictMode>
);
```

7. start project

>npm

Using bootstrap

Setup React Project with Bootstrap:
1. Install bootstrap library in your project

```
>npm  install  bootstrap --save
>npm  install  bootstrap-icons --save
```

2. All bootstrap css is copied into "node_modules"

3. Import bootstrap css files into "index.js"

```
import   '../node_modules/bootstrap/dist/css/bootstrap.css';
import   '../node_modules/bootstrap-icons/font/bootstrap-icons.css';
```

4. You can apply bootstrap classes to elements in component

```
<button  className="btn btn-primary w-100">
<span className="bi bi-cart4"> </span>
```

Ex: Login.jsx

```
export function Login(){
   return(
      <div className="container-fluid d-flex justify-content-center">
         <form className="border mt-4 w-25 p-2 border-2 border-primary
rounded rounded-2">
         <h2> <span className="bi bi-person-fill"></span> User Login</h2>
         <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 w-100">Login</button>
        </form>
      </div>
    )
}
```

Netfilx Example 09/08/2023

Netfilx-index.css

```
#body {
   background-image: url("/public/netflixbanner.jpg");
   height: 100vh;
}
#shade {
   background-color: rgba(0,0,0,0.6);
   height: 100vh;
}
```

Netfilx-index.jsx

```
import './netflix-index.css';

import { NetflixHeader } from './netflix-header';
import { NetflixMain } from './netflix-main';

export function NetflixIndex(){
```

```
    return(
        <div id="body">
          <div id="shade">
             <NetflixHeader />
             <section className="text-white text-
 center mt-4">
                 <NetflixMain />
             </section>
          </div>
        </div>
    )
 }
```

Netfilx header.css

```
.brand {
    font-size: 40px;
    color:red;
    font-weight: bold;
}
```

Netfilx header.jsx

```
import './netflix-header.css';

export function NetflixHeader(){
    return(
        <header className="d-flex justify-content-
between p-4">
            <div>
```

```
                <span
className="brand">NETFLIX</span>
            </div>
            <div>
                <div className="input-group">
                   <span className="bi bi-globe
input-group-text"></span>
                   <select className="form-select">
                    <option>Language</option>
                   </select>
                   <button className="btn btn-danger
ms-2">Signin</button>
                </div>
            </div>
        </header>
    )
}
```

Netflix-Main.css

```
h1 {
    font-size: 50px;
}
p {
    font-size: 25px;
}
main {
```

```
    margin-top: 150px;
}
```

Netflix-Main.jsx

```
export function NetflixMain(){
    return(
        <main>
            <h1>Unlimited movies, TV shows and
more</h1>
            <p>Watch anywhere. Cancel anytime.</p>
            <div>
            <NetflixRegister />
            </div>
        </main>
    )
}
```

Netfilx-register.jsx

```
export function NetflixRegister(){
    return(
       <div className="w-100 d-flex justify-content-
center">
         <div>
            <div className="input-group input-group-
lg">
                <input type="email" placeholder="Your
email address" className="form-control bg-dark text-
white" />
```

```
                <button className="btn btn-danger ms-
2">
                    Get Started <span className="bi
bi-chevron-right"></span>
                </button>
            </div>
        </div>
    </div>
    )
}
```

Data Binding in React 10/08/2023

- Data Binding is a technique used in web applications, where data is accessed from a source and updated into UI.
- JavaScript depends on lot of DOM manipulations to handle data binding.
- JavaScript required lot of event binding techniques to bind data.
- React uses simple data binding expression that can bind data to UI.
- The data binding expression in react is configured with " { } ".

Syntax:

```
var  userName = "John";

<p> Hello ! { userName } </p>
```

- Data Binding in applications is classified into 2 types

1. One Way Binding
2. Two Way Binding

- React supports only One Way Binding

- One Way Binding is a technique where data is accessed from source and update to UI. Any modification in UI will not update back into source.
- Two Way Binding allows to access data from source and bind to UI. It identifies the changes in UI and updates back to source.
- React requires event binding techniques to handle 2 way binding explicitly.
- One Way Binding is fast and secured.

Syntax:

```
var userName = "John";

<input type="text"  value={userName} />    => Not editable
```

Binding Primitive and Non-Primitive data types to UI:

Primitive Types
- number
- string
- boolean
- null
- undefined
- bigint
- symbol

Non Primitive
- Array
- Object
- Map
- Set
- Dated

Boolean Type: 11/08/2023

* React boolean types configured with true or false can handle any value only using a condition.
* You can't present boolean "true or false" directly in UI.

* React uses boolean true & false for conditional rendering.
* In order to display boolean true or false, you have to convert it into string.

```
(true)?"true":"false"
```

Ex:
data-binding.jsx

```
export function DataBinding(){

    var id = 1;
    var name = "Nike Casuals";
    var stock = true;

    return(
        <div className="container-fluid">
            <h2>Product Details</h2>
            <dl>
                <dt>Product Id</dt>
                <dd>{id}</dd>
                <dt>Name</dt>
                <dd>{name}</dd>
                <dt>Stock</dt>
                <dd>{(stock==true)?"Available":"Out of Stock"}</dd>
            </dl>
        </div>

    )
}
```

Number Type
- React can display the numeric values directly.
- If you want to format a number, then you can use "Math" object of JavaScript.

Syntax:

```
    var result = 10/3;
```

```
<p> Division : ${Math.round(result)} </p>
```

String Type:
- You can use all JavaScript string methods.
- String formatting methods:
  bold()
  italics()
  fontcolor()
  fontsize()
  toUpperCase()
  toLowerCase()
- String manipulation methods:
  charAt()
  charCodeAt()
  indexOf() etc..

Syntax:

```
var msg = "Welcome to React";

<p> {msg.toUpperCase().slice(7)} </p>
```

Undefined Type
- It checks if value is defined in a reference during compile time.
- If value is not defined then it returns "undefined".
- React verifies "0" as undefined.

Syntax:

```
var price = 0;

<p> Price = {(price)?price: "Price not defined"} </p>
```

FAQ: How to verify value define or not?

ANS :

```
if(price)
        {
          }
```

Null Type
- It defines that value is not provided into reference during runtime.

Syntax:
```
var price = prompt("Enter Price");

<p> Price = {(price==null)?"Please Enter Price":price} </p>
```

BigInt Type
- It is used to configure large integer values safely.
- It is defined with suffix "n".
- You can also use
  Number.MAX_SAFE_INTEGER()

Syntax:
```
var  price = 999993882828;          => not safe
var  price = 999993882828n;          => safe integer
```

Symbol Type:
- It refers to a hidden property in object.
- It is not accessed over iterations or loops.
- But it is present in object and available to access individually.

Syntax:
```
var  ProductId = Symbol();

var product = {
   [ProductId] : 2,
   Name : "TV"
 }

product[ProductId]       => accessible

Object.keys(product);    => ProductId is not returned

for(var key in product)    => ProductId is not returned
{
```

```
    }
```

Ex:

```
<script>
    var ProductId = Symbol();
    var product = {
        [ProductId]: 1,
        Name : "TV",
        Price : 46000.33,
        Rating:4.2
    }
    Object.keys(product).map((key)=>{
      document.write(key + "<br>");
    })
    document.write("Product Id:" + product[ProductId]);
</script>
```

Summary

- number
- string
- boolean
- null
- undefined
- bigint
- symbol

Non-Primitive Types

---

1. Array

- You can use all array methods to bind array values in UI.
- You can't use explicit iterations or loops in react.
- Basic Array methods for reading data from array

Categories=[“All”, “Fashion”,”footwear”]

toString() it will return all item seprated with “,”.

join() it uses a your own separator (/).
slice() it uses a index number
find()
filter()
map()
forEach() etc..

Syntax

```
var categories = ["A", "B", "C"];

{
  categories.map(category=> <p> { category } </p>)
}
```

Ex:
data-binding.jsx

```
export function DataBinding(){
   var categories = ["All", "Electronics", "Footwear", "Fashion"];
   return(
      <div className="container-fluid">
         <nav className="btn-group-vertical">
            {
               categories.map(category=><button className="btn btn-danger mb-
1">{category}</button>)
            }
         </nav>
         <h2>Categories</h2>
         <ol>
          {
            categories.map((category)=><li>{category}</li>)
          }
         </ol>
         <select>
           {
```

```
            categories.map((category)=><option
value={category}>{category}</option>)
          }
        </select>
        <ul>
          {
            categories.map((category)=><li><input type="checkbox"/>
<span>{category}</span> </li>)
          }
        </ul>
      </div>

   )
}
```

```
export function DataBinding(){

var categories = [“All”, “Fashion” ,”footwear” ,”Electronics”];
```

```
return(
        <div className=”container-fluid”>
<h2>Select Categories</h2>
<ol>
        {
        Categories.map((category)=><li>{category}</li>)
}
</ol>
<h2>Select Categories</h2>
<select>
       {
Categories.map((category)=><option>{category}</li>)
}
</select>
<h2>IN category use checkbox</h2>
<ul>
        {
                Categories.map((category)=><li><input
type=”checkbox”/><span>{category}</span>)
}
</ul>
)
}
```

Binding Examples with Object, Array of objects

Object Type
- Object comprises of key and value collection.
- Key is "string" type and value is "any" type.
- JSON [JavaScript Object Notation]

Syntax:

```
{
  Key: value,
  Key: value
}
```

Ex:

```
var product = {
    "Name": "TV",
    "Price": 45000.33,
    "Stock": true,
    "Cities": ["Delhi", "Hyd"],
    "Rating": { "Rate":4.2, "Count": 5000 }
}

<p> Name : { product.Name } </p>
```

Ex:
data-binding.jsx

```
export function DataBinding(){
    var mobile = {
        "title": "realme C53 (Champion Black, 64 GB)",
        "price": 10999,
        "image": "realmeBlack.png ",
        "rating": {"rate":4.5, "count":12644, "reviews":575},
        "features": [
            "6 GB RAM | 64 GB ROM | Expandable Upto 2 TB",
            "17.12 cm (6.74 inch) HD Display",
            "108MP + 2MP | 8MP Front Camera",
            "5000 mAh Battery",
```

```
        "T612 Processor",
        "1 Year Manufacturer Warranty for Phone and 6 Months
Warranty for In the Box Accessories"
      ]
  }
  return
    <div className="container-fluid">
      <div className="mt-3 row">
        <div className="col-3">
          <img src={mobile.image} />
        </div>
        <div className="col-6">
          <h3 className="text-primary">{mobile.title}</h3>
          <div>
            <span className="bg-success p-2 rounded rounded-2
text-white">
              {mobile.rating.rate} <span className="bi bi-star-
fill"></span>
            </span>
            <span className="ms-2 fw-bold">
              {mobile.rating.count} Ratings & {mobile.rating.reviews}
Reviews
            </span>
          </div>
          <div>
            <ul className="mt-3">
              {
                mobile.features.map(feature=>
                  <li>{feature}</li>
                  )
              }
            </ul>
          </div>
        </div>
        <div className="col-2">
          <h2>
            &#8377; {mobile.price}
          </h2>
```

```
            </div>
          </div>
        </div>

    )
}
```

Array of Objects

- It is a collection of objects.

Syntax:

```
    [
      {
        "Key":value,
      },
      {
        "Key": value
      }
    ]
```

Ex:

data-binding.jsx

```
export function DataBinding(){
    var products = [
        {Name: "Samsung TV", Price:45000.44},
        {Name: "Nike Casuals", Price:6700.42}
    ];

    return(
        <div className="container-fluid">
            <table className="table table-hover caption-top">
                <caption>Products List</caption>
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Price</th>
```

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

  )
}
```

Ex: Card Style
data-binding.jsx

```
export function DataBinding(){
  var products = [
    {
      "title": "realme C53 (Champion Black 64 GB)",
      "image": "realmeBlack.jpg",
      "price": 10999
    },
    {
      "title": "realme C53 (Champion Gold 128 GB)",
      "image": "realme.jpg",
      "price": 15999
    }
  ];

  return(
    <div className="container-fluid">
```

```
        <div className="mt-3 d-flex">
          {
            products.map(product=>
              <div className="card p-2 m-2 w-25" key={product.title}>
                <div className="card-header bg-dark text-white">
                  <h3>{product.title}</h3>
                </div>
                <div className="card-body">
                  <img src={product.image} />
                </div>
                <div className="card-footer">
                  <h3>&#8377; {product.price}</h3>
                  <button className="btn btn-danger w-100">
                    <span className="bi bi-cart4"></span>
                    Buy
                  </button>
                </div>
              </div>
              )
          }
        </div>
      </div>

    )
}
```

Ex:
data-binding.jsx

```
export function DataBinding(){

    var menu = [
        {"Category":"Electronics", "Products": ["TV", "Mobile", "Watch"]},
        {"Category":"Footwear", "Products": ["Casuals", "Sneakers",
"Boots"]}
    ];
```

```
    return(
        <div className="container-fluid">
            <h2>Menu</h2>
            <ol>
                {
                    menu.map(item=>
                        <li>{item.Category}
                            <ul>
                                {
                                    item.Products.map(product=>
                                        <li>{product}</li>
                                        )
                                }
                            </ul>
                        </li>
                        )
                }
            </ol>
        </div>

    )
}
```

Ex:
data-binding.jsx

```
export function DataBinding(){

    var menu = [
        {"Category":"Electronics", "Products": ["TV", "Mobile", "Watch"]},
        {"Category":"Footwear", "Products": ["Casuals", "Sneakers",
"Boots"]}
    ];

    return(
        <div className="container-fluid">
            <h2>Menu</h2>
```

```
            <select>
                {
                    menu.map(item=>
                        <optgroup label={item.Category}>
                          {
                              item.Products.map(product=>
                                 <option>{product}</option>
                                 )
                          }
                        </optgroup>
                       )
                }
            </select>
        </div>

    )
}
```

Ex:
data-binding.jsx

```
export function DataBinding(){

    var topics = [
        {"title": "HTML", "description": "It is a markup language"},
        {"title": "CSS", "description": "It configures styles for HTML"}
    ];

    return(
        <div className="container-fluid">
            <h2>Topics</h2>
            <dl>
                {
                    topics.map(topic=>
                       <>
                        <dt>{topic.title}</dt>
                        <dd>{topic.description}</dd>
```

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

    )
}
```

Ex:

```
export function DataBinding(){

    var data = [[10,20,30], [40,50,60]];
    return(
        <div className="container-fluid">
          <table className="table table-bordered">
          {
              data.map(row=>
                  <tr>
                    {
                        row.map(value=>
                            <td>{value}</td>
                            )
                    }
                  </tr>
                )
           }
          </table>
        </div>

    )
}
```

Note: Every repeating element in react dynamic code must have a unique reference "key".

State in React 16/08/2023

Note:
- Don't use variables in a component for handling dynamic data.
- Variables are immutable, their structure can't change according state and situation
- Component requires mutable references for customization.
- Variables are disposed after the defined scope. You can't carry data across
  requests.
- Http is a state less protocol.
- Http uses the mechanism "Go-Get-Forget".
- Http can't remember information between requests.
- Stateless nature of Http is an advantage for Server, as it manages memory by disposing after the response.
- State less is a drawback for client, as client have to remind server about the previous details everytime when requested.

What is the solution?
- Web applications implement "State Management Techniques" to handle state.
- State Management can be handled both
  a) Client Side
  b) Server Side

FAQ : Why you need State Management technique?
ANS: Because Http is state less every web application need state management technique
This state management help allow to maintain state.
FAQ: What is state?

What is Client Side State Management?
- Application uses client's memory to store data and make it available across requests.
- Various client storages include
  a) localStorage
  b) sessionStorage
  c) Cookies
  d) QueryString etc..

What is difference between various client side state techniques?

a) Local Storage

- It is permanent.
- It keeps the data even after client system shut down or restart.
- It is accessible across tabs.
- You have to manually clear the local storage.

Syntax:

```
localStorage.setItem("Key", value);
localStorage.getItem("Key");
```

b) Session Storage

- It is temporary.
- It is only for current tab in browser.
- Data is removed once tab is closed.
- It is not accessible across tabs in browser.

Syntax:

```
sessionStorage.setItem("key", value);
sessionStorage.getItem("key");
```

Ex:

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        function SubmitClick(){
            var username = document.getElementById("UserName").value;
            sessionStorage.setItem("uname", username);
            location.href ="welcome.html";
        }
```

```
    </script>
</head>
<body>
    <h2>Home</h2>
    <input type="text" id="UserName"> <button
onclick="SubmitClick()">Submit</button>
</body>
</html>
```

welcome.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        function bodyload(){
            document.querySelector("div").innerHTML = "Hello ! " +
sessionStorage.getItem("uname");
        }
    </script>
</head>
<body onload="bodyload()">
    <h2>Welcome</h2>
    <div></div>
</body>
</html>
```

c) Cookies

- Cookie is a simple text document where client details are stored.
- Cookie can be
  a) In-memory [Temporary]
  b) Persistent [Permanent]
- You can set expiry for cookie.

Syntax
```
    document.cookie = "value;attributes";
```

d) Query String
  - It appends data into URL and keeps in address bar.
  - It is not safe, easy to hack, stored in browser logs, can be bookmarked.
  - You can't handle complex data [like binary]
  - Max limit is 2048.
  - You can access querystring using "location.search".

Ex:
home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>
<body>
    <h2>Home</h2>
    <form action="welcome.html">
        <input type="text" name="UserName"> <button>Submit</button>
    </form>
</body>
</html>
```

welcome.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        function bodyload(){
            var str = location.search;
            var uname = str.substring(str.indexOf("=")+1);
            document.querySelector("div").innerHTML = " Hi ! " + uname;
        }
    </script>
</head>
<body onload="bodyload()">
    <h2>Welcome</h2>
    <div></div>
</body>
</html>
```

React Component State

- React 17+ version introduced "useState()" hook for configuring state in function component.
- Early versions of React component can have state only for class component.
- State is mutable.
- State must be initialized, It can't be assigned or declared.
- Hence state in react is configure using "const"

```
    const [getter, setter] = useState(anyValue);
    setter(newValue);
```

Ex:
data-binding.jsx

```
import { useState } from "react";

export function DataBinding(){

    const [categories] = useState(["All", "Electronics", "Footwear"]);
```

```
    return(
        <div className="container-fluid">
            <ol>
              {
                 categories.map(category=>
                    <li key={category}>{ category }</li>
                    )
              }
            </ol>
        </div>

    )
}
```

Two Way Data Binding 16/08/2023

Data Binding
1. One Way Binding
2. Two Way Binding
- It is a data binding technique where data is accessed from source and binded to UI.
- It identifies the changes in UI and updates the source.
- Changes in UI are detected by using "onChange" event.
- It uses "Change Detection" pattern, which verifies the previous and current value.

```
previousValue == currentValue    => No Change Detected
previousValue !== currentValue    => Change Detected
```

- OnChange you can configure a function to execute.

```
<input type="text"  onChange={handleNameChange}>

function handleNameChange()
{
}
```

- JavaScript event have default arguments
  a) this   : sends information about current element
  b) event   : sends information about current event

- React event uses only one default argument that is "event".

event.eventDetails
event.target.elementDetails

Syntax:

```
function handleNameChange(e)
{
    e.target.name;
    e.target.value
    e.target.className;
    e.target.id;
    e.clientX;
    e.ctrlKey;
}
```

Ex:

data-binding.jsx

```
import { useState } from "react";

export function DataBinding(){

    const [userName, setUserName] = useState('John');

    function handleNameChange(event){
        setUserName(event.target.value);
    }

    return(
        <div className="container-fluid">
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" value={userName}
onChange={handleNameChange} /></dd>
            </dl>
            <p>Hello ! {userName} </p>
        </div>

    )
```

}

Ex:
data-binding.jsx

```
import { useState } from "react";

export function DataBinding(){

    const [product, setProduct] = useState({Name:'TV', Price:0, City:
'Select City', Stock: true})

    function NameChange(e){
        setProduct({
            Name: e.target.value,
            Price: product.Price,
            City: product.City,
            Stock: product.Stock
        })
    }
    function PriceChange(e){
        setProduct({
            Price: e.target.value,
            Name: product.Name,
            City: product.City,
            Stock: product.Stock
        })
    }
    function CityChange(e){
        setProduct({
            City: e.target.value,
            Name: product.Name,
            Price: product.Price,
            Stock: product.Stock
        })
    }
    function StockChange(e){
        setProduct({
            Stock: e.target.checked,
            Name: product.Name,
```

```
        Price: product.Price,
        City: product.City
      })
    }

    return(
      <div className="container-fluid">
        <div className="row mt-4">
          <div className="col-4">
            <dl>
              <dt>Name</dt>
              <dd><input onChange={NameChange} type="text"
value={product.Name} className="form-control"/></dd>
              <dt>Price</dt>
              <dd><input onChange={PriceChange} type="number"
value={product.Price} className="form-control"/></dd>
              <dt>City</dt>
              <dd>
                <select onChange={CityChange} className="form-
select" value={product.City}>
                  <option>Select City</option>
                  <option>Delhi</option>
                  <option>Hyd</option>
                  <option>Chennai</option>
                </select>
              </dd>
              <dt>Stock</dt>
              <dd className="form-switch">
                <input type="checkbox" onChange={StockChange}
checked={product.Stock} className="form-check-input"/> <label>
{(product.Stock==true)?"Available":"Out of Stock"} </label>
              </dd>
            </dl>
          </div>
          <div className="col-8">
            <h2>Product Details</h2>
            <dl>
              <dt>Name</dt>
              <dd>{product.Name}</dd>
              <dt>Price</dt>
              <dd>{product.Price}</dd>
```

```
                    <dt>City</dt>
                    <dd>{product.City}</dd>
                    <dt>Stock</dt>
                    <dd>
                       {(product.Stock==true)?"Available":"Out of Stock"}
                    </dd>
                  </dl>
                </div>
              </div>
            </div>

        )
}
```

API Data 18/06/2023

Task:
data-binding.jsx

```
import { useState } from "react";

export function DataBinding(){

    const [product, setProduct] = useState({Name:'TV', Price:0, City:
'Select City', Stock: true})
    const [updatedproduct, setUpdatedProduct] = useState({Name:'TV',
Price:0, City: 'Select City', Stock: true});
    function NameChange(e){
        setProduct({
            Name: e.target.value,
            Price: product.Price,
            City: product.City,
            Stock: product.Stock
        })
    }
    function PriceChange(e){
        setProduct({
            Price: e.target.value,
```

```
        Name: product.Name,
        City: product.City,
        Stock: product.Stock
    })
}
function CityChange(e){
    setProduct({
        City: e.target.value,
        Name: product.Name,
        Price: product.Price,
        Stock: product.Stock
    })
}
function StockChange(e){
    setProduct({
        Stock: e.target.checked,
        Name: product.Name,
        Price: product.Price,
        City: product.City
    })
}

function UpdateClick(){
   setUpdatedProduct({
      Name: product.Name,
      Price: product.Price,
      City: product.City,
      Stock: product.Stock
   })
}

return(
    <div className="container-fluid">
       <div className="row mt-4">
           <div className="col-4">
              <dl>
                 <dt>Name</dt>
                 <dd><input onChange={NameChange} type="text"
```

```
value={product.Name} className="form-control"/></dd>
                    <dt>Price</dt>
                    <dd><input onChange={PriceChange} type="number"
value={product.Price} className="form-control"/></dd>
                    <dt>City</dt>
                    <dd>
                       <select onChange={CityChange} className="form-
select" value={product.City}>
                          <option>Select City</option>
                          <option>Delhi</option>
                          <option>Hyd</option>
                          <option>Chennai</option>
                       </select>
                    </dd>
                    <dt>Stock</dt>
                    <dd className="form-switch">
                       <input type="checkbox" onChange={StockChange}
checked={product.Stock} className="form-check-input"/> <label>
{(product.Stock==true)?"Available":"Out of Stock"} </label>
                    </dd>
                 </dl>
                 <button onClick={UpdateClick} className="btn btn-primary
w-100">Update</button>
              </div>
              <div className="col-8">
                 <h2>Product Details</h2>
                 <dl>
                    <dt>Name</dt>
                    <dd>{updatedproduct.Name}</dd>
                    <dt>Price</dt>
                    <dd>{updatedproduct.Price}</dd>
                    <dt>City</dt>
                    <dd>{updatedproduct.City}</dd>
                    <dt>Stock</dt>
                    <dd>
                       {(updatedproduct.Stock==true)?"Available":"Out of
Stock"}
                    </dd>
```

```
                </dl>
              </div>
            </div>
          </div>

    )
}
```

Distributed Computing Architecture

- Distributed computing allows 2 different applications running or 2 different devices to share information. or 2 different applications running in 2 different process of same device can share information.

- There are a various distributed computing technologies

    CORBA - Common Request Broken Architecture [14 lang]
    DCOM - Distribute Component Object Model [VB]
    RMI - Remote Method Invocation [J2EE]
    EJB - Enterprise Java Beans [Java]
    Web Service - All technologies [Java, .NET, PHP, python...]
    Remoting - .NET

- Web Service handle communication in 3 ways

    a) SOAP
    b) REST
    c) JSON

- SOAP [Service Oriented Architecture Protocol]

    consumer => XML request
    provider => XML response

- REST [Representational State Transfer]

    consumer => Query Request
    provider => XML Response or JSON Response

?category=mobiles <> { }

- JSON [JavaScript Object Notation]

consumer => JSON Request
provider => JSON Response

- API is Application Programming Interface that handle communication between client and server by transporting data.
- API intention is to make data available for any device or any OS services.
- JavaScript can access data from API using
    a) fetch() promise
    b) XMLHttpRequest object
- React can use both JavaScript methods to handle communication with API data.

Syntax:
    fetch("url").then(onsuccess).catch(onfailure).finally(always)

Ex:
1. Go to public folder and add
    "product.json"

```
{
  "title": "realme C53 (Champion Gold, 64 GB)",
  "price": 10999,
  "ratings": {"rate":4.5, "count":14909, "reviews":705},
  "features": [
    "6 GB RAM | 64 GB ROM | Expandable Upto 2 TB",
    "17.12 cm (6.74 inch) HD Display",
    "108MP + 2MP | 8MP Front Camera",
    "5000 mAh Battery",
    "T612 Processor",
    "1 Year Manufacturer Warranty for Phone and 6 Months Warranty for In the Box Accessories"
  ],
  "photo": "realme.jpg"
```

```
}
```

2. Add a component

flipkart.jsx

```
import { useState } from "react";

export function Flipkart(){

    const [product, setProduct] = useState({title:'', price:0, ratings:{rate:0, count:0, reviews:0}, features:[], photo:''});

    function LoadClick(){
        fetch("product.json")
        .then(response=>{
            return response.json();
        })
        .then(product=> {
            setProduct(product);
        })
    }

    return(
        <div className="container-fluid">
            <button onClick={LoadClick} className="btn btn-primary mt-3">Load Data</button>
            <div className="row mt-4">
                <div className="col-3">
                    <img src={product.photo} width="250"/>
                </div>
                <div className="col-7">
                    <p className="text-primary h4">{product.title}</p>
                    <div>
                        <span className="bg-success text-white p-2 rounded">
                            {product.ratings.rate} <span className="bi bi-star-fill"></span>
                        </span>
```

```
                    <span className="ms-3">
                        <b>{product.ratings.count} Ratings &
{product.ratings.reviews} Reviews</b>
                    </span>
                </div>
                <ul className="mt-4">
                    {
                        product.features.map(feature=>
                            <li key={feature}>{feature}</li>
                            )
                    }
                </ul>
            </div>
            <div className="col-2">
                <p className="h3">&amp;#8377; {product.price} </p>
            </div>
        </div>
    </div>
  )
}
```

Ex:
products.json

```
[
  {
    "title": "realme C53 (Champion Gold, 64 GB)",
    "price": 10999,
    "ratings": {"rate":4.5, "count":14909, "reviews":705},
    "features": [
      "6 GB RAM | 64 GB ROM | Expandable Upto 2 TB",
      "17.12 cm (6.74 inch) HD Display",
      "108MP + 2MP | 8MP Front Camera",
      "5000 mAh Battery",
      "T612 Processor",
      "1 Year Manufacturer Warranty for Phone and 6 Months
Warranty for In the Box Accessories"
```

```
        ],
        "photo": "realme.jpg"
    },
    {
        "title": "realme C53 (Champion Black, 164 GB)",
        "price": 15999,
        "ratings": {"rate":4.5, "count":14909, "reviews":705},
        "features": [
            "6 GB RAM | 164 GB ROM | Expandable Upto 2 TB",
            "17.12 cm (6.74 inch) HD Display",
            "108MP + 2MP | 8MP Front Camera",
            "5000 mAh Battery",
            "T612 Processor",
            "1 Year Manufacturer Warranty for Phone and 6 Months
Warranty for In the Box Accessories"
        ],
        "photo": "realmeBlack.jpg"
    }
]
```

flipkart.jsx

```
import { useState } from "react";

export function Flipkart(){

    const [products, setProducts] = useState([{title:'', price:0,
ratings:{rate:0, count:0, reviews:0}, features:[], photo:''}]);

    function LoadClick(){
        fetch("products.json")
        .then(response=>{
            return response.json();
        })
        .then(products=> {
            setProducts(products);
        })
    }
```

```
   return(
      <div className="container-fluid">
         <button onClick={LoadClick} className="btn btn-primary mt-
3">Load Data</button>
          {
            products.map(product=>
               <div className="row mt-4" key={product.title}>
               <div className="col-3">
                  <img src={product.photo} width="250"/>
               </div>
               <div className="col-7">
                  <p className="text-primary h4">{product.title}</p>
                  <div>
                     <span className="bg-success text-white p-2
rounded">
                        {product.ratings.rate} <span className="bi bi-star-
fill"></span>
                     </span>
                     <span className="ms-3">
                        <b>{product.ratings.count} Ratings &
{product.ratings.reviews} Reviews</b>
                     </span>
                  </div>
                  <ul className="mt-4">
                     {
                        product.features.map(feature=>
                           <li key={feature}>{feature}</li>
                           )
                     }
                  </ul>
               </div>
               <div className="col-2">
                  <p className="h3">&amp;#8377; {product.price} </p>
               </div>

               </div>
```

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

Nasa and Fakestore 19/08/2023

Component Mounting & Unmounting
- Mount is a phase in which component is created and rendered.
- Unmount is a phase which occurs when user switches from one component to another.
- In unmount phase the memory of component is destroyed and component request ends.

FAQ: What is the event or method used by browser to load and render your HTML page?
Ans: DOMContentLoaded()

- If you are using React function components then the mount and unmount are defined
  by using the hook "useEffect()".

- Every component can mount only once.
- If you want the component to mount again you have to define the dependency.
- There can be zero or more dependencies.

Syntax:

```
useEffect(()=>{
    //actions on mount;
    return(){
      //actions on unmount - optional
    }
},[ dependencies ]);
```

Ex:
flipkart.jsx

```
import { useEffect, useState } from "react";

export function Flipkart(){

    const [products, setProducts] = useState([{title:'', price:0,
ratings:{rate:0, count:0, reviews:0}, features:[], photo:''}]);

    function LoadProducts(){
        fetch("products.json")
        .then(res=>res.json())
        .then(products=>{
            setProducts(products);
        })
    }

    useEffect(()=>{
      LoadProducts();
    },[]);

    return(
        <div className="container-fluid">
            {
                products.map(product=>
                    <div className="row mt-4" key={product.title}>
                    <div className="col-3">
                        <img src={product.photo} width="250"/>
                    </div>
                    <div className="col-7">
                        <p className="text-primary h4">{product.title}</p>
                        <div>
                            <span className="bg-success text-white p-2
rounded">
                                {product.ratings.rate} <span className="bi bi-star-
fill"></span>
                            </span>
                            <span className="ms-3">
                                <b>{product.ratings.count} Ratings &
{product.ratings.reviews} Reviews</b>
                            </span>
                        </div>
                        <ul className="mt-4">
```

```
                    {
                        product.features.map(feature=>
                            <li key={feature}>{feature}</li>
                            )
                    }
                </ul>
            </div>
            <div className="col-2">
                <p className="h3">&#8377; {product.price} </p>
            </div>

            </div>

            )
        }
    </div>
  )
}
```

Nasa API
(api.nasa.gov)

Ex:
nasa.jsx

```
import { useEffect, useState } from "react"

export function Nasa(){
  const [mars, setMars] = useState({});

  useEffect(()=>{
    fetch("https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=DEMO_KEY&amp;quot;)
    .then(res=> res.json())
    .then(obj=>{
      setMars(obj);
    })
  },[]);


  return(
```

```
        <div className="container-fluid">
          <h2>Mars Rover Photos</h2>
          <table className="table table-hover">
            <thead>
              <tr>
                <th>Photo Id</th>
                <th>Preview</th>
                <th>Camera Name</th>
                <th>Rover Name</th>
              </tr>
            </thead>
            <tbody>
              {
                mars.photos.map(photo=>
                  <tr key={photo.id}>
                    <td>{photo.id}</td>
                    <td>
                      <a href={photo.img_src} target="_blank">
                      <img src={photo.img_src} width="100"
height="100" />
                      </a>
                    </td>
                    <td>{photo.camera.full_name}</td>
                    <td>{photo.rover.name}</td>
                  </tr>
                  )
              }
            </tbody>
          </table>
        </div>
    )
}

Ex: Card Style
nasa.jsx

import { useEffect, useState } from "react"

export function Nasa(){
    const [mars, setMars] = useState({});
```

```
    useEffect(()=>{
        fetch("https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=DEMO_KEY&quot;)
        .then(res=> res.json())
        .then(obj=>{
            setMars(obj);
        })
    },[]);


    return(
        <div className="container-fluid">
            <h2>Mars Rover Photos</h2>
            <main className="d-flex flex-wrap">
                {
                    mars.photos.map(photo=>
                        <div key={photo.id} className="card p-2 m-2" style={{width:'200px'}}>
                            <img src={photo.img_src} className="card-img-top" height="150"/>
                            <div className="card-header">
                                <h3>{photo.id}</h3>
                            </div>
                            <div className="card-body">
                                <dl>
                                    <dt>Camera</dt>
                                    <dd>{photo.camera.full_name}</dd>
                                    <dt>Rover</dt>
                                    <dd>{photo.rover.name}</dd>
                                </dl>
                            </div>
                            <div className="card-footer">
                                <a className="btn btn-primary w-100" target="_blank" href={photo.img_src}>View Photo</a>
                            </div>
                        </div>
                        )
                }
            </main>
        </div>
```

```
    )
}
```

fakestoreapi.com

```
import { useEffect, useState } from "react"

export function Fakestore(){
    const [products, setProducts] = useState([{id:0, title:'', price:0,
description:'', image:'', rating:{rate:0, count:0}, category:''}]);

    function LoadProducts(){
        fetch("http://fakestoreapi.com/products&quot;)
        .then(res=>res.json())
        .then(products=>{
            setProducts(products);
        })
    }


    useEffect(()=>{
        LoadProducts();
    },[]);

    return(
        <div className="container-fluid">
            <header className="d-flex justify-content-between bg-danger
text-white p-2">
                <div>
                    <h3>Shopper.</h3>
                </div>
                <div>
                    <span className="me-3">Home</span>
                    <span className="me-3">Electronics</span>
                    <span className="me-3">Men's Fashion</span>
                    <span className="me-3">Women's Fashion</span>
                    <span className="me-3">Jewelery</span>
                </div>
                <div>
                    <span className="bi bi-search me-3"></span>
```

```
                <span className="bi bi-person me-3"></span>
                <span className="bi bi-cart4"></span>
              </div>
            </header>
            <section className="row mt-3">
              <nav className="col-2">

              </nav>
              <main className="col-8 d-flex flex-wrap">
                {
                  products.map(product=>
                    <div key={product.id} className="card m-2 p-2"
style={{width:'250px'}}>
                      <img src={product.image} className="card-img-top"
height="140" />
                      <div className="card-header overflow-auto"
style={{height:'140px'}}>
                        <p>{product.title}</p>
                      </div>
                      <div className="card-body">
                        <dl>
                          <dt>Price</dt>
                          <dd>{product.price}</dd>
                          <dt>Rating</dt>
                          <dd>{product.rating.rate} <span className="bi bi-
star-fill text-success"></span> </dd>
                        </dl>
                      </div>
                    </div>
                    )
                }
              </main>
            </section>
          </div>
      )
}
```

Fakestore API 22/08/2023

http://fakestoreapi.com

Request Methods:
GET /products [ ] all products - object type
GET /products/1 { } specific id product
GET /products/categories [ ] all categories - string type
GET /products/category/electronics [ ] specific category products

Currency Format for Number:

```
<script>
   var price = 46700000.45;
   document.write((price).toLocaleString('en-US',{style:'currency', currency:'USD'}));
</script>
```

FAQ: What is key ?

ANS: unique reference for every occurence //Every repeating element we use key

FAQ: What is text?

ANS: what user have see

FAQ: What is value?

ANS: what user submits // some time what he see and what is submit different.

Ex:
shopping.jsx

```
import { useState, useEffect } from "react"


export function Shopping(){
```

```
  const [categories, setCategories] = useState([]);
  const [products, setProducts] = useState([{id:0, title:'', image:'', price:0, description:'', category:'', rating:{rate:0, count:0}}]);
  const [cartCount, setCartCount] = useState(0);
  const [cartItems, setCartItems] = useState([]);

  function LoadCategories(){
    fetch("http://fakestoreapi.com/products/categories")
    .then(res=>res.json())
    .then(categories=>{
      categories.unshift("all");
      setCategories(categories);
    })
  }

  function LoadProducts(url){
    fetch(url)
    .then(res=> res.json())
    .then(products=>{
      setProducts(products);
    })
  }

  useEffect(()=>{
    LoadCategories();
    LoadProducts("http://fakestoreapi.com/products");
```

```
    },[]);


    function handleCategoryChange(e){
        if(e.target.value=="all"){
            LoadProducts("http://fakestoreapi.com/products");
        } else {
LoadProducts(`http://fakestoreapi.com/products/category/${e.target.value}`);
        }
    }

    function handleAddToCartClick(e){
        fetch(`http://fakestoreapi.com/products/${e.target.value}`)
        .then(res=> res.json())
        .then(product=>{
            cartItems.push(product);
            setCartCount(cartItems.length);
            alert(`${product.title}\nAdded to Cart`);
        })
    }

    return(
        <div className="container-fluid">
            <header className="d-flex justify-content-between p-3 bg-dark text-
white">
                <div>
```

```
            <span className="h4">Shopper.</span>
          </div>
          <div>
            <span className="me-3">Home</span>
            <span className="me-3">Electronics</span>
            <span className="me-3">Jewelery</span>
            <span className="me-3">Men's Fashion</span>
            <span className="me-3">Women's Fashion</span>
          </div>
          <div>
            <button className="btn btn-light position-relative">
              <span className="bi bi-cart4"></span> Your Cart
              <span className="badge rounded-circle bg-danger position-
absolute top-0 end-0">{cartCount}</span>
            </button>
          </div>
        </header>
        <section className="row mt-3">
          <nav className="col-2">
            <div>
              <label className="form-label fw-bold">Select Category</label>
              <div>
                <select onChange={handleCategoryChange} className="form-
select">
                  {
                    categories.map(category=>
```

```
                        <option key={category}
value={category}>{category.toUpperCase()}</option>
                        )
                    }
                </select>
            </div>
        </div>
    </nav>
    <main className="col-8">
        <div className="d-flex flex-wrap overflow-auto"
style={{height:'550px'}}>
        {
            products.map(product=>
            <div className="card p-2 m-2" style={{width:'200px'}}>
                <img src={product.image} className="card-img-top"
height="140" />
                <div className="card-header overflow-auto"
style={{height:'140px'}}>
                    {product.title}
                </div>
                <div className="card-body">
                    <dl>
                        <dt>Price</dt>
                        <dd>{(product.price).toLocaleString('en-IN',{style:
'currency', currency: 'INR'})}</dd>
                    </dl>
                </div>
                <div className="card-footer">
```

```
                    <button value={product.id} onClick={handleAddToCartClick}
classNameName="btn btn-danger w-100">
                      <span classNameName="bi bi-cart3"></span> Add to Cart
                    </button>
                  </div>
                </div>
                )
              }
             </div>
            </main>
            <aside classNameName="col-2">
              <table classNameName="table table-hover caption-top">
                <caption>Your Cart Summary</caption>
                <thead>
                  <tr>
                    <th>Title</th>
                    <th>Preview</th>
                  </tr>
                </thead>
                <tbody>
                  {
                    cartItems.map(item=>
                      <tr>
                        <td>{item.title}</td>
                        <td>
                          <img  src={item.image} width="50" height="50"/>
                        </td>
```

```
                        </tr>
                        )
                    }
                </tbody>
            </table>
        </aside>
    </section>
</div>
)
}
```

Style and Class Binding 23/08/2023

Style Binding
- It is the technique used to bind inline styles to any element in react component.
- So that you can change the appearence of any element dynamically.

Syntax: JavaScript

```
<div>

document.querySelector("div").style.attributeName = "value";

CSS Attribute           Dynamic Name
-------------------------------------------------
color               color
background-color       backgroundColor
font-size              fontSize
text-align              textAlign

document.querySelector("div").style.fontSize = "20px";
```

- React uses a data binding expression to bind the dynamic styles "{ }"

Syntax: React Style Binding

```
<div  style={ {styleBlock} }>
<div  style="color:red">           // invalid
<div  style={ {color:'red'} }>       // valid

<div style={ {backgroundColor:'red', color:'white', textAlign:'center'} }>
```

Note: Style attributes in React are defined as "style object", which is a key and value
collection.

```
{attribute1:value, attribute2:value}
```

FAQ: What is difference between display:none and visibility:hidden?
Ans: display:none will hide the element and removes allocated space.

visibility:hidden will hide the element but keeps the allocated space.

Syntax:

```
<div  style={ {display:'none' } }>
<div  style={ {visibility:'hidden'} }>
```

Ex: Validation
style-binding.jsx

```
import { useEffect, useState } from "react"

export function StyleBinding(){

   const [styleObject, setStyleObject] = useState({border:'', boxShadow:''})

   useEffect(()=>{
      setStyleObject({
         border: '1px solid red',
         boxShadow: '2px 2px 2px red'
      })
   },[]);

   function handleNameChange(e){
      if(e.target.value==""){
         setStyleObject({
            border: '1px solid red',
            boxShadow: '2px 2px 2px red'
         })
      } else {
         setStyleObject({
            border: '1px solid green',
            boxShadow: '2px 2px 2px green'
         })
      }
   }
```

```
    return(
        <div className="container-fluid">
            <h3>User Login</h3>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" placeholder="Name Required"
onChange={handleNameChange} style={styleObject} /></dd>
            </dl>
        </div>
    )
}
```

Ex: Hide Toggle
style-binding.jsx

```
import { useEffect, useState } from "react"

export function StyleBinding(){

    const [styleObj, setStyleObj] = useState({display:'none'});

    function  handlePreviewChange(e){
        if(e.target.checked){
            setStyleObj({
                display:'block'
            })
        } else {
            setStyleObj({
                display:'none'
            })
        }
    }

    return(
        <div className="container-fluid">
            <h2>Product Details</h2>
            <dl>
             <dt>Name</dt>
```

```
          <dd>Realme C55</dd>
          <dt>Preview <span className="form-switch"> <input
type="checkbox" onChange={handlePreviewChange}
className="form-check-input" /> </span> </dt>
          <dd style={styleObj}>
             <img src="realme.jpg" />
          </dd>
         </dl>
       </div>
   )
}
```

Class Binding

- Class binding is the technique used to bind any CSS class to HTML element dynamically.
- Usually class comprises of a set of css attributes.

Syntax:

```
       .effects {
            color:red;
            font-size: 20px;
            border: 2px solid red;
        }
       <div> </div>

       document.querySelector("div").className = "effects";
       document.querySelector("div").className="class1 class2
class3..";
```

- React uses "className" property with a data binding expression to bind multiple classes dynamically.

```
       <div className={ 'className' } >
```

Ex:

1. Add following file into folder "class-binding"
   class-binding.css
   class-binding.jsx

2. class-binding.css

```
.dark-theme {
    background-color: black;
    color:white;
    padding: 20px;
    border: 2px solid gray;
    border-radius: 10px;
}
.light-theme {
    background-color: white;
    color:black;
    padding: 20px;
    border:2px solid black;
    border-radius: 10px;
}
```

3. class-binding.jsx

```
import { useState } from "react";
import './class-binding.css';

export function ClassBinding(){

    const [colorTheme, setColorTheme] = useState('light-theme');
    const [buttonTheme, setButtonTheme]= useState('btn btn-dark w-
100');

    function handleThemeChange(e){
        if(e.target.checked) {
            setColorTheme('dark-theme');
            setButtonTheme('btn btn-light w-100');
        } else {
            setColorTheme('light-theme');
            setButtonTheme('btn btn-dark w-100');
        }
    }
```

```
    return(
        <div className="container-fluid">
            <div className="d-flex justify-content-center align-items-
center" style={{height:'100vh'}}>
                <form className={colorTheme}>
                    <div className="form-switch">
                        <input type="checkbox"
onChange={handleThemeChange}  className="form-check-input"/>
Dark Theme
                    </div>
                    <h3>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={buttonTheme}>Login</button>
                </form>
            </div>
        </div>
    )
}
```

Ex: Class Binding with bootstrap classes

```
import { useState } from "react";


export function ClassBinding(){

    const [colorTheme, setColorTheme] = useState('bg-light text-dark p-
4 border border-3 border-dark rounded rounded-3');
    const [buttonTheme, setButtonTheme]= useState('btn btn-dark w-
100');
```

```
    function handleThemeChange(e){
        if(e.target.checked) {
        setColorTheme('bg-dark text-white p-4 border border-3 border-
warning rounded rounded-3');
            setButtonTheme('btn btn-light w-100');
        } else {
            setColorTheme('bg-light text-dark p-4 border border-3 border-
dark rounded rounded-3');
            setButtonTheme('btn btn-dark w-100');
        }
    }

    return(
        <div className="container-fluid">
            <div className="d-flex justify-content-center align-items-
center" style={{height:'100vh'}}>
                <form className={colorTheme}>
                    <div className="form-switch">
                        <input type="checkbox"
onChange={handleThemeChange}  className="form-check-input"/>
Dark Theme
                    </div>
                    <h3>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={buttonTheme}>Login</button>
                </form>
            </div>
        </div>
    )
}
```

Event Binding

- Event is a message sent by sender to it subscriber in order to notify the change.
- Event follows a software design pattern called "observer", which is a behavioural pattern. [communication pattern]
- Event uses "Delegate Mechanism" [Function Pointer]

Syntax:

```
function InsertClick()    => Subscriber
{
}

<button onclick="InsertClick()">   => Sender
```

Event Binding in React 24/08/2023

Event Binding
- Event is a message sent by sender to its subscriber in order notify the change.
- Event follows a software design pattern called "Observer".
- It is a behavioural pattern. Usually behavioural patterns are communication patterns.
- A communication pattern defines how a object communicates with its class.
- Event uses a delegate mechanism, which is a function pointer mechanism.

Syntax:

```
function InsertClick()              => Subscriber
{

}

<button  onClick={InsertClick}>        => Sender
```

- Subscriber comprises actions to perform.
- Sender uses trigger that notifies the subscriber.
- Subscriber executes the function when event is triggered by Sender.

```
onClick={InsertClick}
```

```
onClick               => Event
onClick={InsertClick}    => Event Handler
function InsertClick() {}    => Event Listener
```

- All JavaScript events are Browser Events. [BOM]
- React can't use browser events directly.
- React uses "SyntheticEvents" library, which is a virtual DOM events library.
- SyntheticEvents map to browser events.

Browser Event	SyntheticEvent
onclick	onClick
onchange	onChange
onblur	onBlur
onkeyup	onKeyup

Configuring Events in React:
- Events in React can be configured in 2 ways
    1. Inline Events
    2. Embedded Events

Inline Events:
- In this technique the events and listeners are configured in side element directly.

```
<button onClick={alert("Hello React !")}> Hello </button>
```

Note: The actions configured for event will trigger automatically at the time of mounting
the component.

- You have to "refactor" the functionality in order to execute only when event is triggered

```
<button  onClick={ function(){ alert("Hello React !");} }> Hello </button>
```

```
   <button  onClick={ () => alert("Hello React !") }>  Hello </button>

   <button  onClick={ () => { alert("Hi !"); alert("Welcome"); } }> Hello
</button>
```

Ex:
event-binding.jsx

```
export function EventBinding(){
   return(
      <div className="container-fluid">
         <h3>Event Binding</h3>
         <button onClick={()=> {alert("Hello ! React");
alert("Welcome")}}>Hello</button>
      </div>
   )
}
```

Handling Generic Elements:
- Generic elements are the HTML elements that have a default functionality configured.
- When you define events both generic and custom events will fireup.
- You have to disable the default event configured for element by using "preventDefault()"
- preventDefault() is a method of event argument. You have to access with reference of event.

Syntax:

```
  <form  onSubmit={ (e)=> { alert("Submitted"); e.preventDefault(); } }>

    <button> Submit </button>

  </form>
```

Ex:

event-binding.jsx

```
export function EventBinding(){
   return(
      <div className="container-fluid">
         <h3>Event Binding</h3>
         <form onSubmit={(e)=>{alert("Form Submits its data to API");
e.preventDefault()}}>
            <dl>
               <dt>Name</dt>
               <dd><input type="text" name="Name"/></dd>
               <dt>Age</dt>
               <dd><input type="number" name="Age"/></dd>
            </dl>
            <button>Submit</button>
         </form>
      </div>
   )
}
```

Handling Propagation:
- It is the process of notifying the subscribers of child and to its parent.
- If user clicks on child element, it propagates notification to its parent.
- You can stop the propagation by using the event argument method.
  e.stopPropagation()
- It is defined only for child element event.

Syntax:

```
<div onClick={ ()=> alert("div clicked") }>

  <button onClick={ (e) => { alert("button clicked"); e.stopPropagation()
} }>
    Submit
  </button>

</div>
```

Ex:

event-binding.jsx

```
export function EventBinding(){
    return(
        <div className="container-fluid">
            <h3>Event Binding</h3>
            <div onClick={()=>{alert("Div Clicked")}} className="bg-dark
text-white p-3">
                <p>Parent Element</p>
                <button onClick={(e)=>{alert("Button Clicked");
e.stopPropagation()}}>Child Element</button>
            </div>
        </div>
    )
}
```

Handling Event Arguments:
- JavaScript events have 2 default arguments
    a) this
    b) event
- "this" sends information about current element, which includes
    id, name, className, src, href etc..

- "event" sends information about current event, which includes
    clientX, clientY, which, shiftKey, altKey, ctrlKey etc..

Syntax:

```
    <button  onclick="InsertClick(this, event)">
```

- JavaScript event can have custom arguments

Syntax:

```
    <button onclick="InsertClick(...[10, "A", true, [ ], { }])">

    function InsertClick(id, name, stock, cities, rating)
    {
    }
```

```
<button onclick="InsertClick(this,...[10, "A", true, [ ], { }])">
```

- React can't have 2 default arguments like this and event.
- It uses only a SyntheticEvent argument reference "e", which can access both element and event details.

Syntax:

```
<button  id="btnInsert"  name="Insert"  onClick={ (e) =>
alert(`${e.target.id}\n ${e.clientX}`) }>
```

e.clientX
e.clientY
e.ctrlKey
e.target.id
e.target.name
e.target.className

Embedded Technique and Mouse Events 25/08/2023

Custom Event Arguments:

- React events can handle custom arguments along with default arguments.
- A custom argument can be any data type.
  a) Primitive
  b) Non Primitive
- You can configure custom argument in embedded technique.

Syntax: JavaScript

```
<button  onclick="DetailsClick({Name:'TV', Price:50000.44})">

 function DetailsClick(product)
{
   console.log(product.Name + "\n" + product.Price);
}
```

Embedded Technique

- It allows developers to configure functions for events in component and point towards the functions using event handlers.

Syntax:

```
function InsertClick(e)
{

}

<button onClick={InsertClick}>  Insert </button>
```

FAQ: Configuring different functions for a sequence of operations & Configuring single function for a sequence of operations, which one is good?

Ans: Configuring different functions for a sequence of operations is good when you are handling discreet operations. [Disconnected]

Configuring single function for a sequence of operations is good when you are handling contineous operations. [Connected].

Ex:
event-binding.jsx

```
import { useState } from "react"

export function EventBinding(){

   const [msg, setMsg] = useState('');

   function dataOperations(e){
      switch(e.target.name){
         case "Insert":
         setMsg("Record Inserted");
```

```
            break;
            case "Update":
            setMsg("Record Updated");
            break;
            case "Delete":
            setMsg("Record Deleted");
            break;
        }
    }

    return(
        <div className="container-fluid">
            <h3>Event Binding</h3>
            <button onClick={dataOperations}
name="Insert">Insert</button>
            <button onClick={dataOperations}
name="Update">Update</button>
            <button onClick={dataOperations}
name="Delete">Delete</button>
            <p>{msg}</p>
        </div>
    )
}
```

React Custom Arguments for Event:

Syntax:

```
    function DetailsClick(e)
    {
    }
    <button onClick={DetailsClick}>              => only default args are
passed.
                                        e.ref,  e.target.ref

    <button  onClick={DetailsClick('Welcome')}>    => function executes
on mount.

    <button  onClick={()=>{ DetailsClick('Welcome') }}>
```

```
      <button onClick={(e)=> { DetailsClick('custom', e) }}>
```

Ex:

event-binding.jsx

```
import { useState } from "react"

export function EventBinding(){

   function DetailsClick(product,e){
      alert(`${product.Name}\n${product.Price}\n${e.clientX}`);
   }

   return(
      <div className="container-fluid">
         <h3>Event Binding</h3>
         <button onClick={(e)=>{ DetailsClick({Name:'TV',
Price:45660.33}, e) }}>Details</button>
      </div>
   )
}
```

React Synthetic Events

- React provides a Synthetic Events library that handles various types of browser events with Virtual DOM.
- They are classified into various groups

1. Mouse Events
2. Keyboard Events
3. Button Events
4. Clipboard Events
5. Element State Events
6. Form Events
7. Timer Events
8. Touch Events etc...

Mouse Events

- onMouseOver
- onMouseOut

- onMouseUp
- onMouseDown
- onMouseMove

Note: Mostly used mouse event args are
clientX
clientY
screenX
screenY etc..

Ex:
1. Add a new JSON file into project public folder
"mobiles.json"

```
[
   {
      "img_src": "images/m1.jpg"
   },
   {
      "img_src": "images/m2.jpg"
   },
   {
      "img_src": "images/m3.jpg"
   },
   {
      "img_src": "images/m4.jpg"
   },
   {
      "img_src": "images/m5.jpg"
   }
]
```

2. event-binding.css

```
img:hover {
   cursor: grab;
}
```

3. event-binding.jsx

```
import { useEffect } from "react";
import { useState } from "react"
import './event-binding.css';

export function EventBinding(){

    const [mobiles, setMobiles ] = useState([{img_src:''}]);
    const [preview, setPerview] = useState('images/m1.jpg');

    useEffect(()=>{
        fetch("mobiles.json")
        .then(res=> res.json())
        .then(pics=>{
            setMobiles(pics);
        })
    },[]);

    function DisplayImage(e){
        setPerview(e.target.src);
    }

    return(
        <div className="container-fluid">
          <h2>Realme</h2>
          <div className="row">
           <div className="col-2">
             {
                 mobiles.map(mobile=>
                    <div className="mb-4 border border-primary border-2"
style={{width:'105px'}}>
                        <img src={mobile.img_src} width="100" height="100"
onMouseOver={DisplayImage} />
                    </div>
                    )
             }
           </div>
```

```
            <div className="col-10">
                <img width="500" src={preview} height="600"/>
            </div>
          </div>
        </div>
    )
}
```

Ex: Mouse Move

mouse-demo.jsx

```
import { useState } from "react"

export function MouseDemo(){

    const [styleObject, setStyleObject] = useState({position:'', top:'', left:''});

    function GetPosition(e){
        setStyleObject({
            position: 'fixed',
            top: e.clientY + 'px',
            left: e.clientX + 'px'
        })
    }

    return(
        <div className="container-fluid" onMouseMove={GetPosition}>
            <div style={{height:'1000px'}}>
                <p>Move your mouse pointer to test</p>
            </div>
            <img style={styleObject} src="images/flag.gif" width="50" height="50" />
        </div>
    )
}
```

Ex: Marquee Animation

```
import { useState } from "react"

export function MouseDemo(){

    return(
        <div className="container-fluid">
            <marquee scrollamount="15" className="mt-4"
onMouseOut={(e)=>{e.target.start()}}
onMouseOver={(e)=>{e.target.stop()}} >
                <img src="images/m1.jpg" width="200" height="200"
className="me-4" />
                <img src="images/m2.jpg" width="200" height="200"
className="me-4"  />
                <img src="images/m3.jpg" width="200" height="200"
className="me-4"  />
                <img src="images/m4.jpg" width="200" height="200"
className="me-4"  />
                <img src="images/m5.jpg" width="200" height="200"
className="me-4"  />
            </marquee>
        </div>
    )
}
```

Events Continued. 26/08/2023

Keyboard Events
- onKeyup
- onKeyDown
- onKeyPress

Note: KeyUp and Down will fireup immediately when your release a key.
KeyPress will fireup when your finish the current key and keyin another.
KeyEvents can get the code of key by using event args

a) keyCode
b) charCode
c) which (is good for all types of keyboard layouts)

Ex:

key-demo.jsx

```
import { useState } from "react"

export function KeyDemo(){

    const [error, setError] = useState('');
    const [errorClass, setErrorClass] = useState('');
    const [pwdError, setPwdError] = useState('');

    function VerifyUserName(e){
        fetch("data/users.json")
        .then(res=>res.json())
        .then(users=>{
            for(var user of users){
                if(user.UserName===e.target.value){
                    setError('User Name Taken - Try Another');
                    setErrorClass('text-danger');
                    break;
                } else {
                    setError('User Name Available');
                    setErrorClass('text-success');
                }
            }
        })
    }
    function VerifyPassword(e){
        if(e.which>=65 && e.which<=90){
            setPwdError('Warning : Caps ON');
        } else {
            setPwdError('');
        }
    }
    return(
```

```
        <div className="container-fluid">
           <h3>Register User</h3>
           <dl>
              <dt>User Name</dt>
              <dd><input type="text" onKeyUp={VerifyUserName} /></dd>
              <dd className={errorClass}> {error} </dd>
              <dt>Password</dt>
              <dd><input type="password" onKeyPress={VerifyPassword}
/></dd>
              <dd className="text-warning">{pwdError}</dd>
           </dl>
        </div>
    )
}
```

Button Events

- onClick
- onDoubleClick
- onContextMenu [Right Click]

Note: The context menu in virtual dom can be handled using Synthetic Events.

But if you want to manage window object functionality, you have to take help of

document browser events.

Syntax:

```
document.oncontextmenu = function() { }
document.oncut = function() { }
```

To disable any event, the function have to return false.

Ex:

button-demo.jsx

```
export function ButtonDemo(){

   function ViewLarge(){
```

```
        window.open('kids.png','Kids','width=300 height=400');
    }
    function DisableContext(){

        document.oncontextmenu = function(){
            alert("Right Click Disabled");
            return false;
        }
    }
    return(
        <div className="container-fluid"
onContextMenu={DisableContext}>
            <h2>Button Events</h2>
            <img onDoubleClick={ViewLarge} src="kids.png" width="100"
height="100" />
            <p>Double Click to View Large</p>
        </div>
    )
}
```

Element State Events

- onChange
- onFocus
- onBlur
- onSelectStart

Ex:

```
import { useState } from "react"

export function ElementState(){
    const [msg, setMsg] = useState('');
    const [cityError, setCityError] = useState('');

    function NameBlur(e){
        if(e.target.value==""){
            setMsg('User Name Required');
        } else {
            setMsg('');
```

```
        }
    }
    function VerifyCity(e){
        if(e.target.value=="-1"){
            setCityError('Please Select Your City');
        } else {
            setCityError('');
        }
    }

    return(
        <div className="container-fluid" >
            <div>
                <p>Selecting Text is disable in this page.</p>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" onBlur={NameBlur}
placeholder="Name in Block Letters"/></dd>
                    <dd className="text-danger">{msg}</dd>
                    <dt>Your City</dt>
                    <dd>
                        <select onChange={VerifyCity}>
                            <option value="-1">Select Your City</option>
                            <option value="Delhi">Delhi</option>
                            <option value="Hyd">Hyd</option>
                        </select>
                    </dd>
                    <dd className="text-danger">{cityError}</dd>
                </dl>
            </div>
        </div>
    )
}
```

Clipboard Events

- onCut
- onCopy
- onPaste

Ex:

```
import { useState } from "react"

export function ElementState(){
   const [msg, setMsg] = useState('');

   function Cut(){
      setMsg('Removed - Copied to Clipboard');
   }
   function Copy(){
      setMsg('Copied to Clipboard');
   }
   function Paste(){
      setMsg('Inserted from Clipboard');
   }

   return(
      <div className="container-fluid" >
         <div>
           <textarea onCut={Cut} onCopy={Copy} onPaste={Paste}
rows="4" cols="40"></textarea>
           <p>{msg}</p>
         </div>
      </div>
   )
}
```

Events and Component Properties 28/08/2023

Timer Events
setInterval()
setTimeout()
clearInterval()
clearTimeout()

- These events are used in the technique like debounce and throttle.

Syntax:
```
setTimeout(function(){ }, interval)
```

JavaScript Date Type:
- It is configure using "Date()" constructor.

```
var  now = new Date();        // loads current date.
var  mfd  = new Date("year-month-day  hrs:min:sec.milliSec");
```

- JavaScript provides various date and time functions to read and store date.

```
getHours()        0 to 23
getMinutes()        0 to 59
getSeconds()        0 to 59
getMilliSeconds()    0 to 99
getDate()        1 to 31
   getDay()            0 to 6 [0=Sunday....6=Saturday]
getMonth()        0 to 11 [0=Jan.....11=Dec]
getFullYear()        4 digits year number
toLocaleDateString()
toLocaleTimeString()
toString()

setHours()
setMinutes()
setSeconds()
setMilliSeconds()
setDate()
setMonth()
setYear()
```

Ex:
```
import { useEffect, useState } from "react"

export function ElementState(){

   const [clock, setClock] = useState(Date());
```

```
    const [msg, setMsg] = useState('');

    function ShowClock(){
        var time = new Date();
        setClock(time.toLocaleTimeString());
    }

    useEffect(()=>{
        setInterval(ShowClock,1000);
        var date = new Date();
        var hrs = date.getHours();
        if(hrs>=0 && hrs<=12){
            setMsg('Good Morning');
        } else if(hrs>12 && hrs<=15) {
            setMsg('Good Afternoon');
        } else if(hrs>15 && hrs<=23){
            setMsg('Good Evening');
        }
    },[])

    return(
        <div className="container-fluid" >
            <p className="text-center mt-3">
                {clock} <br />
                {msg}
            </p>
        </div>
    )
}
```

Ex: Timer Events

```
import { useEffect, useState } from "react"

export function ElementState(){

    const [buttonContainer, setButtonContainer] =
useState({display:'block'});
```

```
    const [loadingContainer, setLoadingContainer] =
useState({display:'none'});
    const [imageContainer, setImageContainer] =
useState({display:'none'});
    const [count, setCount] = useState(0);

    var i = 0;
    function LoadImage(){
        i++;
        setCount(i);
        if(i==100){
            setLoadingContainer({display:'none'});
            setImageContainer({display:'block'});
            return;
        }
    }

    function LoadClick(){
        setInterval(LoadImage,200);
        setButtonContainer({display:'none'});
        setLoadingContainer({display:'block'});
    }

    useEffect(()=>{

    },[])

    return(
        <div className="container-fluid d-flex justify-content-center align-
items-center" style={{height:'100vh'}}>
            <div>
                <div style={buttonContainer}>
                    <button onClick={LoadClick} className="btn btn-
primary">Load Image</button>
                </div>
                <div style={loadingContainer} className="text-center">
                    <div className="spinner-border text-success"></div>
                    <div>{count} % completed</div>
```

```
            </div>
            <div style={imageContainer}>
               <img src="women.jpg" width="300" height="400"/>
            </div>
         </div>
      </div>
   )
}
```

Touch Events

- onTouchStart()
- onTouchEnd()
- onTouchMove()

Ex:
touch-demo.jsx

```
import { useState } from "react"

export function TouchDemo(){

   const [msg, setMsg] = useState('');

   function GetDetails(e){
      switch(e.target.id){
         case "men":
         setMsg('40% OFF on Men Clothing - Offer Ends 31-Aug-2023');
         break;
         case "women":
         setMsg('SALE on Women Fashion - 50% OFF');
         break;
      }
   }

   return (
      <div className="container-fluid">
         <h3>Fashion Store</h3>
         <img src="men.jpg" id="men" onTouchStart={GetDetails}
width="200" height="300"/>
```

```
        <img src="women.jpg" id="women" onTouchStart={GetDetails}
width="200" height="300" />
        <h3>{msg}</h3>
      </div>
   )
}
```

Form Events

- onSubmit
- onReset

- "onSubmit" is used for form element to submit all the form data to server.
- It firesup only by using Generic Submit button.
- "onReset" is used to reset the form data.

Syntax:

```
   <form  onSubmit={ }   onReset={ }>

      <button type="submit"> submit </button>
      <button type="reset"> cancel </button>

   </form>
```

Ex:

```
import { useState } from "react"

export function TouchDemo(){

   function FormSubmit(e){
      e.preventDefault();
      alert(JSON.stringify({'Name':'TV', 'Price':45000.33}));
   }

   return (
      <div className="container-fluid">
         <form onSubmit={FormSubmit}>
            <button>Submit</button>
         </form>
```

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

Summary
- Function Component
- Data Binding
- Style Biniding
- Class Binding
- Event Binding

Component Properties

- Properties are the parameters configured into a function to modify the functionality.
- Component uses properties to modify the component behaviour.
- A component can change its behaviour according to state and situation.
- Every component property is an "object" type, with key and value reference.

Syntax:

```
    export function ComponentName(props)        => props.key
    {

    }

    <ComponentName  key="value" />
    <img src=" "/>
```

Ex:
1. navbar.jsx

```
export function NavBar(props){
    return(
        <div className={`${props.theme} text-white mt-2 mb-2 p-2 d-flex
justify-content-between`}>
            <div>
```

```
            <span className="h3">{props.brandname}</span>
          </div>
          <div>
            {
             props.menuitems.map((item)=>
              <span className="me-3">{item}</span>
             )
            }
          </div>
          <div>
            <span className="bi bi-search"></span>
            <span className="bi bi-person"></span>
            <span className="bi bi-cart3"></span>
          </div>
        </div>
    )
}
```

2. props-demo.jsx

```
import  { NavBar } from '../navbar/navbar';

export function PropsDemo(){
    return(
        <div className="container-fluid">
            <h3>Properties Demo</h3>
            <NavBar brandname="Shopper" theme="bg-dark"
menuitems={['Home', 'About', 'Contact']} />
            <NavBar brandname="Super Market" theme="bg-primary"
menuitems={['Home', 'Shop', 'Accessories', 'Help']} />
        </div>
    )
}
```

Component Properties 30/08/2023

Ex:
1. data-grid.jsx

```
export function DataGrid(props){
   return(
      <div className="container-fluid table-responsive">
         <table className={`table table-hover caption-top
${props.theme}`}>
            <caption>{props.caption}</caption>
            <thead>
               <tr>
                  {
                     props.fields.map(field=>
                        <th key={field}>{field} <button className="bi bi-sort-
alpha-down btn btn"></button> </th>
                        )
                  }
               </tr>
            </thead>
            <tbody>
               {
                  props.data.map(item=>
                     <tr key={item}>
                        {
                           Object.keys(item).map(key=>
                              <td key={key}>{item[key]}</td>
                              )
                        }
                     </tr>
                     )
               }
            </tbody>
         </table>
      </div>
   )
}
```

2. Props-Demo.jsx

```
import  { NavBar } from '../navbar/navbar';
import  { DataGrid } from '../data-grid/data-grid';
import { useState } from 'react';
import { useEffect } from 'react';

export function PropsDemo(){

    const [columns, setColumns] = useState([]);
    const [rows, setRows] = useState([]);

    function LoadColumns(){
        fetch('http://fakestoreapi.com/products/1&#39;)
        .then(res=> res.json())
        .then(product=> {
            delete product.rating;
            setColumns(Object.keys(product));
        })
    }

    function LoadData(){
        fetch('http://fakestoreapi.com/products&#39;)
        .then(res=>res.json())
        .then(products=>{
            products.map(product=>{
                delete product.rating;
            })
            setRows(products);
        })
    }

    useEffect(()=>{
        LoadColumns();
        LoadData();
    },[]);
```

```
    return(
        <div className="container-fluid">
            <h3>Fake Store Data</h3>
            <DataGrid caption="Fakestore Data" fields={columns}
data={rows} />
            <h3>Products Grid</h3>
            <DataGrid theme="table-primary" caption="Product Details"
fields={['Name','Price','Stock']} data={[{'Name':'Samsung TV',
'Price':35000.44, 'Stock':'Available'},{'Name':'Mobile', 'Price':12000.33,
'Stock':'Out of Stock'}]} />
            <h3>Employees Grid</h3>
            <DataGrid theme="table-success" caption="Employee Details"
fields={['First Name', 'Last Name', 'Designation', 'Salary']} data={[{'First
Name':'Raj', 'Last Name':'Kumar', 'Designation':'Developer',
'Salary':50000.33}]} />
        </div>
    )
}
```

Conditional Rendering

- It is a technique used in components to render various behaviours according to the state and situation.

Ex:

```
import { useState } from "react"

function ViewComponent(props){
    return(
        <tr>
            <td>Your Name</td>
            <td><label>{props.uname}</label></td>
            <td><button>Edit</button></td>
        </tr>
    )
}
function EditComponent(props){
    return(
```

```
        <tr>
            <td>Your Name</td>
            <td><input type="text" value={props.uname} /></td>
            <td><button>Save</button></td>
        </tr>
    )
}

export function RenderDemo(){

    const [component, setComponent] = useState(<ViewComponent
uname='John'/>);

    function ViewClick(){
        setComponent(<ViewComponent uname='John' />);
    }
    function EditClick(){
        setComponent(<EditComponent uname='John' />)
    }

    return(
        <div className="container-fluid">
            <h3>User Details <button onClick={ViewClick} className="bi
bi-eye-fill"></button> <button onClick={EditClick} className="bi bi-
pen-fill"></button> </h3>
            <table className="table table-hover w-50">
                <tbody>
                    {component}
                </tbody>
            </table>
        </div>
    )
}
```

FAQ: How to transfer data one component to another?

How to transfer data from parent to child ?

ANS: By using hooks

Conditional Rendering 30/08/2023

Ex:
render-demo.jsx

```
import { useEffect, useState } from "react"


function ViewComponent(props){
    return (
        <label>{props.UserName}</label>
    )
}
function EditComponent(props){
    return(
        <input type="text"  value={props.UserName} />
    )
}

export function RenderDemo(){

    const [userName, setUserName] = useState('John');
    const [btnText, setBtnText] = useState('Edit');
    const [component, setComponent] = useState('');

    useEffect(()=>{
        setComponent(<ViewComponent UserName={userName} />)
    },[]);

    function ToggleComponent(){
        if(btnText==="Edit"){
            setBtnText("Save");
            setComponent(<EditComponent UserName={userName} />)
        } else {
            setBtnText("Edit");
```

```
            setComponent(<ViewComponent UserName={userName} />)
        }
    }

    return(
        <div className="container-fluid">
            <h3>User Details  </h3>
            <table className="table table-hover w-50">
                <tbody>
                  <tr>
                   <td>Your Name</td>
                   <td> {component} </td>
                   <td>
                      <button onClick={ToggleComponent} className="btn
btn-primary me-2"> {btnText} </button>
                      <button className="btn btn-danger">Cancel</button>
                   </td>
                  </tr>
                </tbody>
            </table>
        </div>
    )
}
```

Ex: Conditional Render without explicit components

render-demo.jsx

```
import { useEffect, useState } from "react"


export function RenderDemo(){

    const [userName, setUserName] = useState('John');
    const [btnText, setBtnText] = useState('Edit');
    const [component, setComponent] = useState('');
```

```
    useEffect(()=>{

    },[]);

    function ToggleComponent(){
        if(btnText==="Edit") {
            setBtnText("Save");
        } else {
            setBtnText("Edit");
        }
    }

    function NameChange(e){
        setUserName(e.target.value);
    }

    return(
        <div className="container-fluid">
            <h3>User Details  </h3>
            <table className="table table-hover w-50">
                <tbody>
                  <tr>
                   <td>Your Name</td>
                   <td>
                      {
                         (btnText==="Edit")?<label
onDoubleClick={ToggleComponent}>{userName}</label>:<input
type="text" onChange={NameChange} value={userName} />
                      }
                   </td>
                   <td>
                      <button onClick={ToggleComponent} className="btn
btn-primary me-2"> {btnText} </button>
                      <button className="btn btn-danger">Cancel</button>
                   </td>
                  </tr>
                </tbody>
            </table>
```

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

Ex: Conditional Rendering using component properties

1. dynamic-grid.jsx

```
export function DynamicGrid(props){
    if(props.view==="grid"){
        return(
            <div>
                <h2>Grid View</h2>
                <table className="table table-hover w-50">
                  <thead>
                   <tr>
                      <th>Mobile Images</th>
                   </tr>
                  </thead>
                   <tbody>
                      {
                          props.data.map(item=>
                              <tr>
                                  <td>
                                      {item.title}
                                  </td>
                                  <td>
                                      <img width="200" height="200"
src={item.image} />
                                  </td>
                                  <td>{item.price}</td>
                                  <td><button className="btn btn-
primary">Buy</button></td>
                              </tr>
                              )
                      }
                  </tbody>
```

```
            </table>
        </div>
    )
} else if(props.view==="cards") {
    return(
        <div>
            <h2>Card View</h2>
            <div className="d-flex justify-content-between flex-wrap">
            {
                props.data.map(item=>
                    <div className="card m-2 p-2" style={{width:'200px'}}>
                        <img src={item.image} height="100" className="card-
img-top" />
                        <div className="card-body overflow-auto"
style={{height:'140px'}}>
                            {item.title}
                        </div>
                        <div className="card-footer">
                            {item.price}
                            <div>
                                <button className="btn btn-danger bi bi-cart4">
Add to Cart </button>
                            </div>
                        </div>
                    </div>

                    )
            }
            </div>
        </div>
    )
} else {
    return(
        <div>
            <h4>Please Select a View</h4>
        </div>
    )
}
```

```
}
```

2. render-demo.jsx

```
import { useEffect, useState } from "react"
import { DynamicGrid } from "../dynamic-grid/dynamic-grid"


export function RenderDemo(){

    const [viewName, setViewName] = useState('grid');
    const [mobiles, setMobiles] = useState([{img_src:''}]);

    function handleViewChange(e){
        setViewName(e.target.value);
    }
    useEffect(()=>{
        fetch("http://fakestoreapi.com/products&amp;quot;)
        .then(res=>res.json())
        .then(data=>{
            setMobiles(data);
        })
    },[]);

    return(
        <div className="container-fluid">
            <div className="mt-2 w-25">
                <label>Select View</label>
                <div>
                    <select onChange={handleViewChange} className="form-
select">
                        <option value="-1">Choose View</option>
                        <option value="grid">Grid</option>
                        <option value="cards">Cards</option>
                    </select>
                </div>
            </div>
```

```
        <div className="mt-4">
          <DynamicGrid view={viewName} data={mobiles} />
        </div>
      </div>
    )
}
```

Ex:
conditional-render.jsx

```
import { useState } from "react"
import { Login } from "../login/login";
import { NetflixRegister } from "../netflix/netflix-register";

export function ConditionalRender(){

    const [component, setComponent] = useState('');

    function handleNavClick(e){
        if(e.target.id==='login'){
            setComponent(<Login />);
        } else {
            setComponent(<NetflixRegister />);
        }
    }

    return(
        <div className="container-fluid">
            <div className="d-flex justify-content-center align-items-top
mt-4" style={{height:'100vh'}}>
                <div className="w-50">
                    <ul className="nav nav-tabs">
                        <li className="nav-item active"><a  id='login' href="#"
onClick={handleNavClick} className="nav-link active">Login</a></li>
                        <li className="nav-item"><a id='register' href="#"
onClick={handleNavClick} className="nav-link">Register</a></li>
```

```
                </ul>
                <div className="mt-4 p-4">
                   {component}
                </div>
             </div>
          </div>
       </div>
    )
}
```

Hooks 31/08/2024

React Hooks

- React library introduced hooks from 16.8x versions.
- You can use hooks only in function components.
- React provides few built-in hooks and also allows to create custom hooks.
- React uses lot of 3rd party hooks for various interactions.
- Hook is JavaScript that configure specific functionality you can use across various locations.

FAQ: Why hooks are not allowed in class component?

ANS: Because function can not be a class member.

FAQ: What can be class member in javascript?

ANS: Property Method constructor accessor

FAQ: Why a function can't be a class member?

ANS: class Is a template and it can contain only mutable members.

Function Is immutable, hence it is not allowed as class member.

- React hooks must start with “Use” word and have to follow camel case.

Syntax:

```
Function useState()
    {
        }
```

EX: Custom Hook -Captcha

1. Add a new folder “hooks” into src
2. Add a new file

```
Captcha.hook.js
Export function use(){

    Var a = Math.random()*10;
    Var b = Math.random()*10;
    Var c = Math.random()*10;
    Var d = Math.random()*10;
    Var e = Math.random()*10;
    Var f = Math.random()*10;
    Return `${Math.round(a)} ${Math.round(b)}
${Math.round(c)}${Math.round(d)} ${Math.round(e)}
${Math.round(f)}`
}
```

3. Login.jsx

```
Import “./login.css”;
Import{ useCaptcha } from “../../hooks/captcha.hook”;


Export function Login(){


Export function Login(){


Const captch = useCaptcha();


Return(
```

<div className=”container-fluid d-flex justify-content-center”>

<form className=” border mt-4 w-25 p-2 border-2 border-primary rounded rounded circle>

<h2> <span className=”bi bi-person-fill”>User Login</h2>

<dl>

<dt>User Name</dt>

<dd><input type=”text” className=”form-control” /></dd>

<dt>Password</dt>

<dd><input type=”password” className=”form-control” /></dd>

<dt>Verify Code</dt>

<dd>{captcha}

</dl>

<button className=”btn btn-primary w-100”>Login</button>

</form>

)}

1. useState() : It configures state for component, where you can store data and use across requests.

   Syntax :

   Const[getter, setter] = useState(any);
2. useEffect() : It can configure the action to perform on mount and unmount

Hooks 1/09/2023

useEffect

- It is a react hook used by component to handle any external system.

- If a component have to change by using external systems, then you need useEffect.
- It can configure the functionality to execute at the time of mount and unmount.
- Mount is a phase where memory is allocated for component and all the dependencies are loaded into memory.
- Unmount is a phase where the function memory is disposed, events and unregistered and component ends.
- The unmount phase firesup when application is close or when you switch from one component to another.

FAQ: What is use of useEffect ?

ANS: to design a component which will be modified external system.

Note: Always use the hook "useEffect" when your component need to be effected by
external systems like, browser DOM, dynamic CSS, animations etc.

Syntax:

```
useEffect(()=>{
  ..on mount..
  return () => {
  ..on unmount..
  }
},[])                [ ] dependencies
```

Ex:

```
import { useEffect, useState } from "react";

export function Login(){
   useEffect(()=>{
      alert(`Login Mounted`);
      return()=>{
         console.log(`Login Unmounted`);
      }
```

```
    },[]);
    return(
        <div>
            <h3>Login Component</h3>
        </div>
    )
}
export function Register(){
    useEffect(()=>{
        alert(`Register Mounted`);
        return()=>{
            console.log(`Register Unmounted`);
        }
    },[]);
    return(
        <div>
            <h2>Register Component</h2>
        </div>
    )
}

export function HooksDemo(){

    const [component, setComponent] = useState(<Login/>);

    function handleLogin(){
        setComponent(<Login />);
    }

    function handRegister(){
        setComponent(<Register />);
    }

    return(
        <div className="container-fluid">
            <button onClick={handleLogin}>Login</button>
            <button onClick={handRegister}>Register</button>
            <hr />
```

```
        {component}
      </div>
   )
}
```

FAQ: How to define unmount in function component?
Ans: By configuring "return" is useEffect hook.

Syntax:

```
      useEffect(()=>{

          return()=>{
          ..unmount phase..
        }

      },[])
```

useContext

- Context is the memory allocated for parent component and accessible to all other components that run within the context of parent.

- It is accessible to any level of hierarchy.

Note: You can send the data of parent into child by using "props".
    [component properties]

- Context allows to transport data from parent to child without using "props".
- "useContext()" is a hook that uses the context memory, which is created by component.
- To create context memory you need the method "createContext()"

Steps:
1. Create a new context memory

```
    let  contextName = React.createContext(null);
```

2. Context is service that uses
   a) Provider   : It locates the value in memory
   b) Injector   : It injects the value into component.

   <contextName.Provider   value={ }>

      ....only component defines in this area can use context...

   </contextName.Provider>

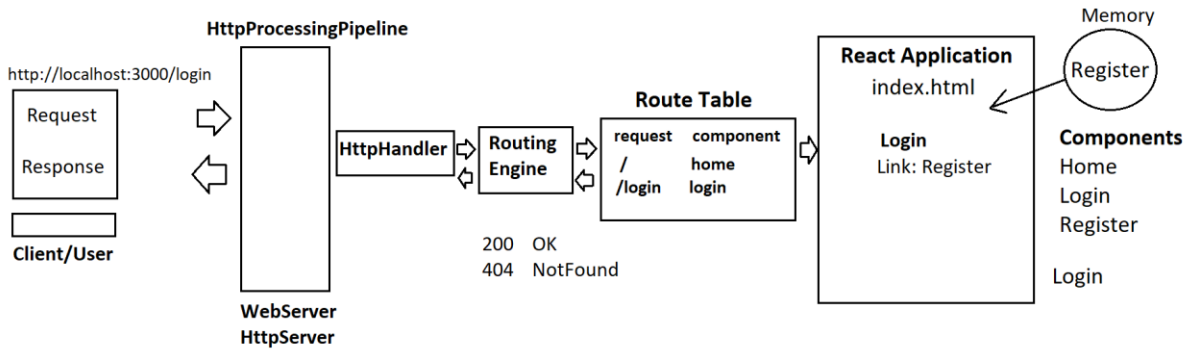
3. Component in context provider area can use the context values by using
      "useContext()"

      let ref = useContext(contextName);

Context-Demo.jsx

```
import { createContext, useContext, useState } from "react"

let UserContext = createContext(null);

export function VideosComponent(){
   let context = useContext(UserContext);
   return(
      <div className="container-fluid bg-danger text-white p-3">
         <h4>Videos Component - {context.UserName}</h4>
      </div>
   )
}


export function HomeComponent(){
   let context = useContext(UserContext);
   return(
      <div className="container-fluid bg-light text-dark p-3">
         <h3>Home Component - {context.UserName}</h3>
```

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


export function ContextDemo()
{
    const [userName, setUserName] = useState('');

    function handleNameChange(e){
        setUserName(e.target.value);
    }

    return(
        <div className="container-fluid bg-dark text-white p-3">
            <UserContext.Provider value={{UserName:userName}}>
             <h2>Index Component <input onChange={handleNameChange}
type="text" placeholder="UserName"/> </h2>
             <HomeComponent />
            </UserContext.Provider>
        </div>
    )
}
```



useReducer Hook 05/09/2023

useState
useEffect

useContext

useReducer

- It is a state hook that can store data for application or session.
- useReducer hook allows to configure a predictable state container for application.
- It is predictable as it exactly knows what is the data stored and when it is changed.
- It is a state container as it can store a value and use across requests.
- The major components of reducer store
    1. store
    2. state
    3. actions
    4. reducer
- Store is the location where data is kept.
- State is responsible for binding the data to component UI.
- Actions identify the change in data and notify the reducer.
- Reducer identifies the action type and the changed data, which it will update into store.

Configuring and use the Reducer:

1. You have to define initial state.

```
let  initialState = { count: 0 };
```

2. You have to define a reducer, which is a function that contains state and actions.

```
function  reducer(state, action)
{
    switch(action.type)
   {
     case "a":
         //actions to update state;
     case "b":
      //actions to update state;
   }
}
```

3. You have to use reducer in any component, where component can dispatch action and modify the state value.

```
const  [state, dispatch] = useReducer(reducer, initialState);
```

4. On any specific event you have to dispacth the action

```
function handleClick()
{
   dispatch( {type: 'A'} );
}

<p> {state.count} </p>
```

Ex: useReducer

```
import { useReducer } from "react";

let initialState = {count: 0};

function reducer(state, action){
   switch(action.type){
      case "join":
      return {count : state.count + 1};
      case "exit":
      return {count : state.count - 1};
   }
}

export function ReducerDemo(){

   const [state, dispatch] = useReducer(reducer, initialState);

   function handleJoinClick(){
      dispatch({type: 'join'});
      console.log(state);
   }
```

```
    function handleExitClick(){
        dispatch({type: 'exit'});
    }

    return(
        <div className="container-fluid">
            <h2>NareshIT - Youtube</h2>
            <div className="mt-4 mb-4">
                <button onClick={handleJoinClick} className="btn btn-
primary"> <span className="bi bi-youtube"></span> Join</button>
                <button onClick={handleExitClick} className="btn btn-
danger ms-2"> Exit</button>
            </div>
            <iframe src="https://www.youtube.com/embed/Ft-
v5aLDWqc&amp;quot; width="400" height="300" />
            <div className="mt-4">
                <div>
                    Live Viewers : {state.count}
                </div>
            </div>
        </div>
    )
}
```

useRef Hook 06/09/2023

useRef
- It is used to configure values in a component which are not used in rendering.
- You can configure reference, store value in reference, you use across various functions in component but not in rendering.

Syntax:

```
let   ref  = useRef(null);       => configure a reference , default null
ref.current;              => access the value in reference
```

Ex: JavaScript Clear Interval

```
        var ref  =    setInterval(()=>{ }, interval)
      clearInterval(ref);
```

Ex: React Clear Interval

```
      let  ref =  useRef(null);
      ref.current = setInterval(()=> { }, interval);
      clearInterval(ref.current);
```

Ex: reference-demo.jsx

```
import { useRef, useState } from "react"

export function ReferenceDemo(){
   const [msg, setMsg] = useState('');
   let m1 = useRef(null);
   let m2 = useRef(null);
   let m3 = useRef(null);

   function Message1(){
      setMsg("Hello !");
   }
   function Message2(){
      setMsg("How are you ?");
   }
   function Message3(){
      setMsg("Welcome to React");
   }

   function handleStartClick(){
      console.log(`Start Click`);
      m1 = setTimeout(Message1, 3000);
      m2.current = setTimeout(Message2, 6000);
      m3 = setTimeout(Message3, 10000);
   }
```

```
    function handleCancelClick(){
        clearInterval(m2.current);
        console.log('Cancel Clicked');
    }

    return(
        <div className="container-fluid">
            <button className="mt-4"
onClick={handleStartClick}>Start</button>
            <button className="mt-4" onClick={handleCancelClick}>Cancel
Message 2</button>
            <p className="mt-4 text-center h3">{msg}</p>
        </div>
    )
}
```

Ex:
slide-show.jsx

```
import { useEffect, useRef, useState } from "react"

export function SlideShow(){

    const [product, setProduct] = useState({});
    const [status, setStatus] = useState('');
    let productId = useRef(1);
    let slideshow = useRef(null);

    function LoadProduct(){
        fetch(`http://fakestoreapi.com/products/${productId.current}`)
        .then(res=>res.json())
        .then(product=>{
            setProduct(product);
        })
    }

    function ChangeImagesAuto(){
        productId.current = productId.current + 1;
```

```
        fetch(`http://fakestoreapi.com/products/${productId.current}`)
        .then(res=>res.json())
        .then(product=>{
            setProduct(product);
        })
    }

    useEffect(()=>{
        LoadProduct();
    },[]);

    function handlePrevClick(){
        productId.current = productId.current - 1;
        LoadProduct();
    }
    function handleNextClick(){
        productId.current = productId.current + 1;
        LoadProduct();
    }

    function handlePlayClick(){
        slideshow.current = setInterval(ChangeImagesAuto, 5000);
        setStatus('Slide Show - Started');
    }
    function handlePauseClick(){
        clearInterval(slideshow.current);
        setStatus('Slide Show - Paused');
    }


    return(
        <div className="container-fluid d-flex justify-content-center">
            <div className="card p-2 mt-4 w-50">
                <div className="card-header text-center"
style={{height:'80px'}}>
                    <span>{product.title}</span>
                    <div className="text-danger fw-bold">[{status}]</div>
                </div>
```

```
            <div className="card-body text-center">
              <div className="row">
                <div className="col-1">
                  <button  onClick={handlePrevClick} className="btn btn-
primary"> &lt; </button>
                </div>
                <div className="col-10">
                <img width="100%" src={product.image}  height="300"/>
                </div>
                <div className="col-1">
                  <button onClick={handleNextClick} className="btn btn-
primary"> &gt; </button>
                </div>
              </div>
            </div>
            <div className="card-footer text-center">
               <button onClick={handlePlayClick} className="btn btn-
outline-primary">
                  <span className="bi bi-play"></span>
               </button>
               <button onClick={handlePauseClick} className="btn btn-
outline-danger ms-2">
                  <span className="bi bi-pause"></span>
               </button>
            </div>
         </div>
      </div>
   )
}
```

useMemo, useCallBack and Classes 7/09/2023

useCallback & useMemo

- useMemo is a hook provided by react to handle memoized data.
- It can cache the data and save round trips.

- If complex data is required to load frequently you can cache and load.
- The UI is updated and rendered only when any change occured in data.

```
useMemo(()=>{}, [ dependencies ])
```

- useCallback is similar to useMemo but it executes a function when dependency changes instead of returning a value.

```
useCallback(()=>function(){}, [dependencies])
```

Summary:
- useState()
- useEffect()
- useMemo()
- useCallback()
- useRef()
- useReducer()
- useContext()

React Class Components

JavaScript Class:
- Class is a program template.
- A program template provides data and logic which you implement and customize according to requirements.
- A class is considered as Model when it is mapping to data requirements.
- A class is considered as Entity when it is mapping to business requirements.
- In general class is also known as Blue Print. [As is the single source for several objects]

Configuring Class:
1. Class Declaration

```
class  Name
{
```

```
}
```

2. Class Expression

```
const  Name = class {

}
```

Class Members:
- A JavaScript class member can be a
  - Property
  - Method
  - Constructor
  - Accessor

FAQ: Can we have a variable or function as class member?
Ans: No.

FAQ: Can we have a variable of function in class?
Ans: Yes.

FAQ: Why we can't have variable and function as class member?
Ans: They are immutable, and class member must be mutable.

Property

- A property is used to store data.
- You can store any type of data, primive or non-primitive.
- Property is mutable.

Syntax:

```
class Name
{
  Property = value|any ;
}
```

Ex:

```
class Product
```

```
{
  Name = "Samsung TV";
  Price = 45000.44;
  Stock = true;
  Cities = [ "Delhi" , "Hyd" ];
  Rating = { Rate:3.5, Count: 4000 }
}
```

Accessors

- An accessor gives fine grained control over property in class.
- They handle read and write operations on property.
- Accessors are 2 types

a) get() getter, used to read and return value of property
b) set() setter, used to store value into property

- In general we read and write value into property by using instance of class.

```
let  obj = new Product();
obj.Name = "Shoes";       // assign
console.log(obj.Name);       // read
```

Ex:

```
<script>
  var username = prompt("Enter Name");
  var role = prompt("Enter your role","customer|admin|manager");
  var productname = prompt("Enter New Name for Product");

  class Product
  {
    _productName;

    get ProductName(){
      return this._productName;
    }
    set ProductName(newName){
      if(role==="admin"){
```

```
            this._productName = newName;
          } else {
            document.write(`Unauthorized: Hello ! ${username} your role
${role} is not authorized to set product name`);
          }
       }
    }
    let obj = new Product();
    obj.ProductName = productname;
    if(obj.ProductName){
       document.write(`Product Name : ${obj.ProductName}`);
    }
</script>

Ex:
<script>
     class Product
     {
        Name = "Samsung TV";
        Rating = {
            Vendor: {
              Direct : { Rate : 4.2 },
              InDirect: { Rate: 4.6}
            }
        }
        get DirectVendor(){
          return this.Rating.Vendor.Direct.Rate;
        }
        set DirectVendor(newRate){
          this.Rating.Vendor.Direct.Rate = newRate;
        }
     }
     let obj = new Product();
     obj.DirectVendor = prompt("Enter New Rating");
     document.write(`Direct Vendor Rating : ${obj.DirectVendor}`);
</script>
```

Class Configuration
Class Members
- Properties
- Accessors

Methods

- The actions in a class are defined by using methods.
- Method is mutable.

Syntax:

```
class Name
{
  MethodName(params)
  {
  }
}
```

- All methods rules are similar to a function.

Ex:

```
<script>
  class Product
  {
    Name = "Samsung TV";
    Price = 45000.44;
    Qty = 2;
    Total(){
      return this.Qty * this.Price;
    }
    Print(){
      document.write(`Name=${this.Name}<br>Price=${this.Price}<br>
Qty=${this.Qty}<br>Total=${this.Total()}`);
    }
  }
  let obj = new Product();
  obj.Print();
</script>
```

Constructor

- Constructor is a special type of sub-routine used for instantiation.
- Instantiation is the process of creating an object for class.
- Every class have an implicit constructor, which is known "Default Constructor".
- Every constructor implicitly executes when an instance is created for class.
- JavaScript constructor is Anonymous type.

Syntax:

```
class Name
{
   constructor() {
   }
   }
```

- A constructor executes only once per object.
- Constructor parameters are same as function parameters.
- The parameters into constructor are passed using IIFE pattern while creating instance of class.

```
let obj  = new ClassName(values);    =>  () constructor
                                        Immediately Invoked Function
                                    Expression
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script>
    class Database
    {
      constructor(dbName){
        document.write(`Connected to ${dbName}<br>`);
      }
```

```
        Insert(){
          document.write("Record Inserted<br>");
        }
        Update(){
          document.write("Record Updated<br>");
        }
        Delete(){
          document.write("Record Deleted");
        }
      }
      function InsertClick(){
        let obj = new Database(document.querySelector("select").value);
        obj.Insert();
        obj.Insert();
      }
      function UpdateClick(){
        let obj = new Database(document.querySelector("select").value);
        obj.Update();
      }
      function DeleteClick(){
        let obj = new Database(document.querySelector("select").value);
        obj.Delete();
      }
    </script>
</head>
<body>
    <select>
      <option value="-1">Select Database</option>
      <option>Oracle</option>
      <option>MySql</option>
      <option>MongoDB</option>
    </select>
    <button onclick="InsertClick()">Insert</button>
    <button onclick="UpdateClick()">Update</button>
    <button onclick="DeleteClick()">Delete</button>
</body>
</html>
```

Ex:

```
<script>
  var values = [["Delhi","Hyd"], function(){document.write("Array
Function")}];
  document.write(values[0][1] + "<br>");
  values[1]();
</script>
```

Code Resuability & Extensibility

- Code resuablity is the process of accessing members of one class and using in another class.
- Code extensibility is the process of extending a class and adding new features without modifying the class.
- Reusability and Extensibility can be configure using 2 techniques
  1. Aggregration
  2. Inheritance

Aggregration

- It is a technique of accessing members of one class in another class by creating an instance of class.
- It is known as "Object-To-Object" communication.
- It is often reffered as "Has-A-Relation".
- You can access class members without configuring any relation between classes.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
     class HDFC_Version1
     {
       Personal = "Personal Banking Services <br>";
       NRI = "NRI Banking Services <br>";
```

```
  Print(){
    document.write(`${this.Personal}<br>${this.NRI}<br>`);
  }
}
class HDFC_Version2
{
  Loans = "Personal & Car Loans <br>";
  Print(){
    let obj = new HDFC_Version1();
    obj.Print();
    document.write(`${this.Loans}<br>`);
  }
}
class HDFC_Version3
{
  AGRI = "Govt. Bank Schemes <br>";
  Print(){
    let obj = new HDFC_Version2();
    obj.Print();
    document.write(`${this.AGRI}<br>`);
  }
}
function InstallClick(){
 var version = document.querySelector("select").value;
 switch(version){
     case "ver1":
     let obj1 = new HDFC_Version1();
     obj1.Print();
     break;
     case "ver2":
     let obj2 = new HDFC_Version2();
     obj2.Print();
     break;
     case "ver3":
     let obj3 = new HDFC_Version3();
     obj3.Print();
     break;
     default:
```

```
            document.write("Please Select a Version");
            break;
        }
      }
  </script>
</head>
<body>
  <fieldset>
    <legend>Install HDFC Bank App</legend>
    <select>
      <option value="-1">Select Version</option>
      <option value="ver1">Version-1 [2021]</option>
      <option value="ver2">Version-2 [2022]</option>
      <option value="ver3">Version-3 [2023]</option>
    </select>
    <button onclick="InstallClick()">Install</button>
  </fieldset>
</body>
</html>
```

Inheritance:
- It is the process of configuring relationship between classes so that you can access the members of one class in another without creating instance of class.
- It is reffered as "Is-A-Relation".
- The existing class is known as "Super Class".
- The newly created class that extends super class is known as "Derived Class".
- The keyword "extends" is used to extend the super class.
- The keyword "super" is used to access the members of super class.

Syntax:

```
class A
{
}
class B extends A
{
}
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    class HDFC_Version1
    {
      Personal = "Personal Banking Services <br>";
      NRI = "NRI Banking Services <br>";
      Print(){
        document.write(`${this.Personal}<br>${this.NRI}<br>`);
      }
    }
    class HDFC_Version2 extends HDFC_Version1
    {
      Loans = "Personal & Car Loans <br>";
      Print(){
        super.Print();
        document.write(`${this.Loans}<br>`);
      }
    }
    class HDFC_Version3 extends HDFC_Version2
    {
      AGRI = "Govt. Bank Schemes <br>";
      Print(){
        super.Print();
        document.write(`${this.AGRI}<br>`);
      }
    }
    function InstallClick(){
      var version = document.querySelector("select").value;
      switch(version){
          case "ver1":
```

```
            let obj1 = new HDFC_Version1();
            obj1.Print();
            break;
            case "ver2":
            let obj2 = new HDFC_Version2();
            obj2.Print();
            break;
            case "ver3":
            let obj3 = new HDFC_Version3();
            obj3.Print();
            break;
            default:
            document.write("Please Select a Version");
            break;
        }
      }
  </script>
</head>
<body>
  <fieldset>
    <legend>Install HDFC Bank App</legend>
    <select>
      <option value="-1">Select Version</option>
      <option value="ver1">Version-1 [2021]</option>
      <option value="ver2">Version-2 [2022]</option>
      <option value="ver3">Version-3 [2023]</option>
    </select>
    <button onclick="InstallClick()">Install</button>
  </fieldset>
</body>
</html>
```

Note: A derived class constructor must call super class constructor.

```
      super()
```

Ex:

```
<script>
   class SuperClass
```

```
    {
      constructor(){
        document.write("Super Class Constructor <br>");
      }
    }
    class DerivedClass extends SuperClass
    {
      constructor(){
        super();
        document.write("Derived Class Constructor<br>");
      }
    }
    let obj = new DerivedClass();
</script>
```

Class Components in React 12/09/2023

Summary
- JavaScript classes
- Declaration & Expression
- Class Members
    a) Property
    b) Method
    c) Accessor
    d) Constructor
- Aggregation
- Inheritance
- Ploymorphism

Polymorphism

- Poly means "Many" and Morphos means "Forms".
- Poly morphism is the ability of a component to work for various situations.
- A single base class reference can use the memory of multiple derived classes, which is reffered as polymorphism.

Note:

- You can create a single base class reference using the memory of derived class.
- You can't create a derived class reference using the memory of base class.

Syntax:

```
let  reference = new Array(new class1(), new class2(),..);
```

Ex:

```
<script>
   class Employee
   {
      FirstName;
      LastName;
      Designation;
      Print(){
        document.write(`${this.FirstName} ${this.LastName} -
${this.Designation}<br>`);
      }
   }

   class Developer extends Employee
   {
      FirstName = "Raj";
      LastName = "Kumar";
      Designation = "Developer";
      Role = "Developer Role : Build, Debug, Test";
      Print(){
        super.Print();
        document.write(`${this.Role}`);
      }
   }
   class Admin extends Employee
   {
      FirstName = "Kiran";
      LastName = "Rao";
      Designation = "Admin";
      Role = "Admin Role : Authorizations & Authentications";
```

```
    Print(){
      super.Print();
      document.write(`${this.Role}`);
    }
  }
  class Manager extends Employee
  {
    FirstName = "Tom";
    LastName = "Hanks";
    Designation = "Manager";
    Role = "Manager Role : Approvals";
    Print(){
      super.Print();
      document.write(`${this.Role}`);
    }
  }

  let employees = new Array(new Developer(), new Admin(), new
Manager());
  var designation = prompt("Enter Designation");
  for(var employee of employees) {
    if(employee.Designation===designation){
      employee.Print();
    }
  }
</script>
```

Programming Systems
1. POPS
2. OBPS
3. OOPS

POPS
- Process Oriented Programming System
- It supports low level features.
- It can directly interact with hardware services.
- It uses less memory.
- It is faster.

Ex: C, Pascal
- Code reusability issues
- Code extensibility issues
- Dymamic memory allocation issues

OBPS
- Object Based Programming System
- Supports dynamic memory, code reusability, extensibility etc.

Ex: VB, JavaScript

- No Dynamic polymorphism
- Limitied extensibility
- No code level security. etc..

OOPS
- Object Oriented Programming
- Reusability
- Extensibility
- Security
- Contracts
- Templates
- Abstraction etc..

Ex: C++, C#, Java

FAQ: What are the issues with OOP?
- It will not support low level features
- It can't directly interact with hardware services.
- It uses more memory
- It is slow.
- It heavy on application.

React Class Component

- React class component is created by using JavaScript class

```
class Login
{
}
```

- A class gets component behaviour by extending "React.Component" base class.

```
export class Login extends React.Component
{

}
```

- React provides 2 super classes for creating components

```
a) React.Component        [Impure Component]
b) React.PureComponent    [Pure Component]
```

- Impure component loads all content again when any change occured. It is legacy type.
- Pure component is modular & async, it loads only the changes.
- Every React component must render a node, and class component renders a node or fragment by using "render()" method.

Syntax:

```
export class Login extends React.Component
{
      constructor(){
    super();
    }
    render() {

    }
}
```

- React class component render() method will return markup. | fragment | node.

Syntax:

```
export class  Login  extends React.Component
{
    constructor(){
      super();
    }
    render(){
        return(
            <React.Fragment>

            </React.Fragment>
        )
    }
}
```

Ex:

- Add a new folder into src
  "class-components"

- Add component folder "login"

- Add a new file
  login.jsx

```
import React from "react";

export class Login extends React.Component
{
    constructor(){
      super();
    }
    render(){
        return(
          <div className="container-fluid">
            <h3>User Login</h3>
            <dl>
              <dt>User Name</dt>
              <dd><input type="text"/></dd>
```

```
                <dt>Password</dt>
                <dd><input type="password"/></dd>
              </dl>
              <button className="btn btn-primary">Login</button>
            </div>
        )
    }
}
```

State in Class Component

- Class component have a default state configured.
- Hence class component as known as statefull components.
- "React.Component" base provides a state.
- You have to configure state for component at the time of instantiation.
- State is object type with key and value reference.
- You can access the current class state using "this" keyword.

```
    constructor() {
      super();
      this.state = {
       key : value
       }
    }

    <p> { this.state.key } </p>
```

Ex: State

```
import React from "react";

export class Login extends React.Component
{
    constructor(){
      super();
      this.state = {
         Id : 1,
```

```
            Name: "Samsung TV",
            Price: 45000.44,
            Stock: true,
            Cities: ["Delhi", "Hyd"],
            Rating: {Rate:4.2}
        }
    }
    render(){
        return(
            <div className="container-fluid">
              <h2>Product Details</h2>
              <dl>
               <dt>Product Id</dt>
               <dd>{this.state.Id}</dd>
               <dt>Name</dt>
               <dd>{this.state.Name}</dd>
               <dt>Price</dt>
               <dd>{this.state.Price}</dd>
               <dt>Stock</dt>
               <dd>{(this.state.Stock==true)?"Avialable":"Out of Stock"}</dd>
               <dt>Cities</dt>
               <dd>
                  <ol>
                     {
                        this.state.Cities.map(city=>
                            <li key={city}>{city}</li>
                          )
                     }
                  </ol>
               </dd>
               <dt>Rating</dt>
               <dd>{this.state.Rating.Rate}</dd>
              </dl>
            </div>
        )
    }
}
```

FAQ: can a useState or useEffect in a class component?

ANS: NO.

FAQ: Why useState or useEffect not allow in class component?

ANS: Becouse they are function and will not allow in class.

Class Component Events and Props 12/09/2023

- Designing Class Components

State in Class Components

Syntax:

```
constructor(){
    this.state = {
      key : value
    }
}

<p> { this.state.key } </p>
```

- To assign a value into state you can use "setState()"

Syntax:

```
this.setState({
    key : value          // assign value into state reference
})
```

Events in Class Component

- All Synthetic Events of React are same in class component.
- In class component event points to a method.
- All browser events of JavaScript you can use as synthetic events in class component.

Syntax:

```
export class  Login  extends   React.Component
{
    constructor() {
```

```
        super();
      }
      handleButtonClick() {

      }
      render(){
        return(
        <button onClick={this.handleButtonClick}> Click </button>
        )
      }
  }
```

Ex:
login.jsx

```
import React from "react";

export class Login extends React.Component
{
    constructor(){
      super();
    }

    handleClick(){
      alert("Button Clicked");
    }

    render(){
       return(
         <div className="container-fluid">
           <button onClick={this.handleClick}>Click</button>
         </div>
       )
    }
}
```

Note: If event in class component is using class state, then event must bind with current state. If binding is not defined then event fails in

accessing state.

Syntax:

```
constructor()
{
   this.handleClick = this.handleClick.bind(this);
}
        (or)

<button onClick={ this.handleClick.bind(this) }>  Insert </button>
```

Ex:
login.jsx

```
import React from "react";

export class Login extends React.Component
{
    constructor(){
      super();
      this.state = {
         message: ''
      }
      this.handleInsertClick = this.handleInsertClick.bind(this);
    }

    handleInsertClick(){
      this.setState({
         message: 'Record Inserted'
      })
    }

    handleDeleteClick(){
      this.setState({
         message: 'Record Deleted Successfully..'
      })
    }
```

```
    render(){
        return(
          <div className="container-fluid mt-4">
            <button onClick={this.handleInsertClick}>Insert</button>
            <button
onClick={this.handleDeleteClick.bind(this)}>Delete</button>
            <p>{this.state.message}</p>
          </div>
        )
    }
}
```

FAQ: Can we use react class events with state, without using bind()?
Ans: Yes. By configuring an expression that returns the method.

Syntax:

```
   <button  onClick={ ()=> { this.handleClick() } }> Insert </button>
```

Class Event Args:
- Every event is class component can use the default argument "event", which contains information about element and event.

```
       handleClick(event)
       {
         event.clientX;
         event.target.id;
       }
```

- In class component the mount phase is defined by using the method "componentDidMount()"

Note: Setting state values in constructor is not recommended, as it is identified as a bug. Hence dynamic values are assigned to state on mount phase.

Ex:
shopping-class-demo.jsx

```
import React from "react";


export class ShoppingClassDemo extends React.Component
{
    constructor(){
      super();
      this.state = {
          categories : [],
          products : []
      }
    }

    LoadCategories(){
      fetch("http://fakestoreapi.com/products/categories&quot;)
      .then(res=>res.json())
      .then(categories => {
          this.setState({
              categories: categories
          })
      })
    }

    LoadProducts(){
      fetch("http://fakestoreapi.com/products&quot;)
      .then(res=>res.json())
      .then(products=>{
          this.setState({
              products : products
          })
      })
    }

    componentDidMount(){
      this.LoadCategories();
      this.LoadProducts();
    }
```

```
    render(){
      return(
        <div className="container-fluid mt-2">
          <section className="row">
            <nav className="col-2">
              <ol>
                {
                  this.state.categories.map(category=>
                    <li key={category}>{category}</li>
                    )
                }
              </ol>
            </nav>
            <main className="col-10">
              <ol>
                {
                  this.state.products.map(product=>
                    <li key={product.id}>{product.title}</li>
                    )
                }
              </ol>
            </main>
          </section>
        </div>
      )
    }
}
```

Properties in Class Component

- Everything same as function component.
- The "props" are configured in constructor.
- "props" is an object, which you can access using "this" keyword.

Syntax:

```
    export class Navbar extends React.Component
    {
       constructor(props){
```

```
        super();
    }
    render(){
        return(
            <p> { this.props.dynamicName } </p>
        )
    }
}
```

Ex:
navbar-class-demo.jsx

```
import React from "react";


export class NavBarClassDemo extends React.Component
{
    constructor(props){
        super();
    }
    render(){
        return(
            <nav className="p-3 bg-light">
                <span className="h3">{this.props.title}</span>
                {
                    this.props.items.map(item=>
                        <button className="btn btn-link me-3">{item}</button>
                        )
                }
            </nav>
        )
    }
}
```

shopping.jsx

```
import React from "react";
import { NavBarClassDemo } from "../navbar-class-demo/navbar-class-
```

```
demo";


export class ShoppingClassDemo extends React.Component
{
    constructor(){
      super();
      this.state = {
          categories : [],
          products : []
      }
    }

    LoadCategories(){
      fetch("http://fakestoreapi.com/products/categories&quot;)
      .then(res=>res.json())
      .then(categories => {
          this.setState({
              categories: categories
          })
      })
    }

    LoadProducts(){
      fetch("http://fakestoreapi.com/products&quot;)
      .then(res=>res.json())
      .then(products=>{
          this.setState({
              products : products
          })
      })
    }

    componentDidMount(){
      this.LoadCategories();
      this.LoadProducts();
    }
```

```
    render(){
      return(
        <div className="container-fluid mt-2">
          <NavBarClassDemo title="Shopper."
items={['Home','About','Shop','Contact']} />
          <section className="row">
            <nav className="col-2">
              <ol>
                {
                  this.state.categories.map(category=>
                    <li key={category}>{category}</li>
                    )
                }
              </ol>
            </nav>
            <main className="col-10">
              <ol>
                {
                  this.state.products.map(product=>
                    <li key={product.id}>{product.title}</li>
                    )
                }
              </ol>
            </main>
          </section>
        </div>
      )
    }
}
```

Class Components in React 12/09/2023

Summary
- JavaScript classes
- Declaration & Expression
- Class Members
  a) Property
  b) Method
  c) Accessor

d) Constructor
- Aggregation
- Inheritance
- Ploymorphism

Polymorphism

- Poly means "Many" and Morphos means "Forms".
- Poly morphism is the ability of a component to work for various situations.
- A single base class reference can use the memory of multiple derived classes, which is reffered as polymorphism.

Note:
- You can create a single base class reference using the memory of derived class.
- You can't create a derived class reference using the memory of base class.

Syntax:

```
let  reference = new Array(new class1(), new class2(),..);
```

Ex:

```
<script>
   class Employee
   {
      FirstName;
      LastName;
      Designation;
      Print(){
        document.write(`${this.FirstName} ${this.LastName} -
${this.Designation}<br>`);
      }
   }

   class Developer extends Employee
   {
      FirstName = "Raj";
      LastName = "Kumar";
      Designation = "Developer";
      Role = "Developer Role : Build, Debug, Test";
```

```
    Print(){
      super.Print();
      document.write(`${this.Role}`);
    }
  }
  class Admin extends Employee
  {
    FirstName = "Kiran";
    LastName = "Rao";
    Designation = "Admin";
    Role = "Admin Role : Authorizations & Authentications";
    Print(){
      super.Print();
      document.write(`${this.Role}`);
    }
  }
  class Manager extends Employee
  {
    FirstName = "Tom";
    LastName = "Hanks";
    Designation = "Manager";
    Role = "Manager Role : Approvals";
    Print(){
      super.Print();
      document.write(`${this.Role}`);
    }
  }

  let employees = new Array(new Developer(), new Admin(), new
Manager());
  var designation = prompt("Enter Designation");
  for(var employee of employees) {
    if(employee.Designation===designation){
      employee.Print();
    }
  }
</script>
```

Programming Systems
1. POPS
2. OBPS

3. OOPS

POPS
- Process Oriented Programming System
- It supports low level features.
- It can directly interact with hardware services.
- It uses less memory.
- It is faster.

Ex: C, Pascal
- Code reusability issues
- Code extensibility issues
- Dymamic memory allocation issues

OBPS
- Object Based Programming System
- Supports dynamic memory, code reusability, extensibility etc.

Ex: VB, JavaScript

- No Dynamic polymorphism
- Limitied extensibility
- No code level security. etc..

OOPS
- Object Oriented Programming
- Reusability
- Extensibility
- Security
- Contracts
- Templates
- Abstraction etc..

Ex: C++, C#, Java

FAQ: What are the issues with OOP?
- It will not support low level features
- It can't directly interact with hardware services.
- It uses more memory
- It is slow.

- It heavy on application.

React Class Component

- React class component is created by using JavaScript class

```
class Login
{
}
```

- A class gets component behaviour by extending "React.Component" base class.

```
export class Login extends React.Component
{

}
```

- React provides 2 super classes for creating components

a) React.Component [Impure Component]
b) React.PureComponent [Pure Component]

- Impure component loads all content again when any change occured. It is legacy type.
- Pure component is modular & async, it loads only the changes.
- Every React component must render a node, and class component renders a node or fragment by using "render()" method.

Syntax:

```
export class Login extends React.Component
{
    constructor(){
  super();
}
render() {

}
}
```

- React class component render() method will return markup. |

fragment | node.

Syntax:

```
export class  Login  extends React.Component
{
   constructor(){
     super();
   }
   render(){
      return(
         <React.Fragment>

         </React.Fragment>
      )
   }
}
```

Ex:

- Add a new folder into src
  "class-components"

- Add component folder "login"

- Add a new file
  login.jsx

```
import React from "react";

export class Login extends React.Component
{
    constructor(){
      super();
    }
    render(){
       return(
         <div className="container-fluid">
            <h3>User Login</h3>
            <dl>
               <dt>User Name</dt>
               <dd><input type="text"/></dd>
               <dt>Password</dt>
```

```
                <dd><input type="password"/></dd>
              </dl>
              <button className="btn btn-primary">Login</button>
            </div>
        )
    }
}
```

State in Class Component

- Class component have a default state configured.
- Hence class component as known as statefull components.
- "React.Component" base provides a state.
- You have to configure state for component at the time of instantiation.
- State is object type with key and value reference.
- You can access the current class state using "this" keyword.

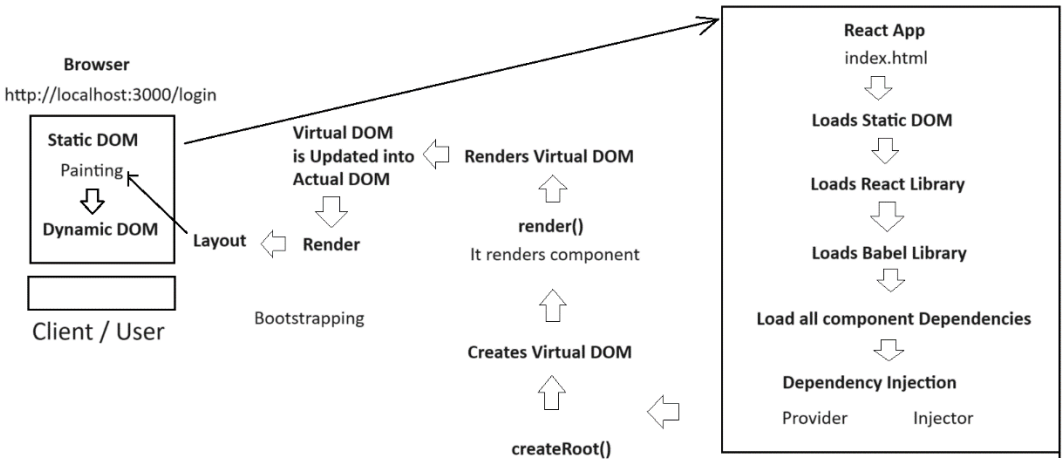
```
        constructor() {
          super();
          this.state = {
           key : value
           }
        }

        <p> { this.state.key } </p>
```

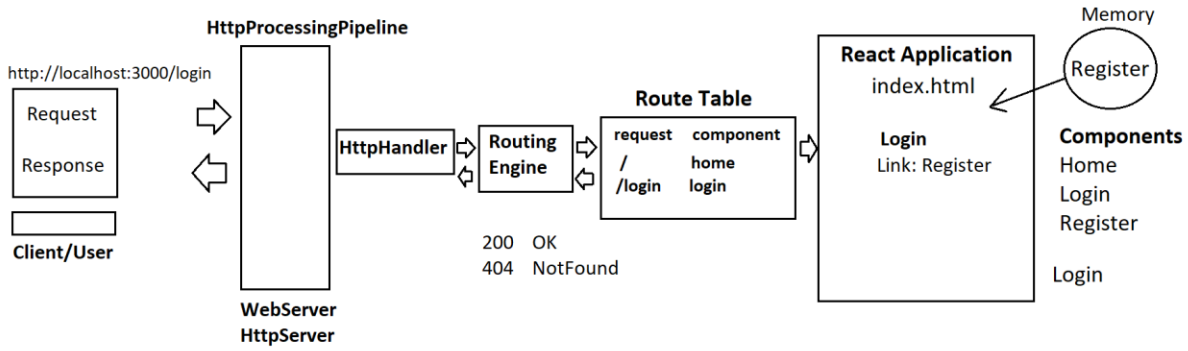
Ex: State

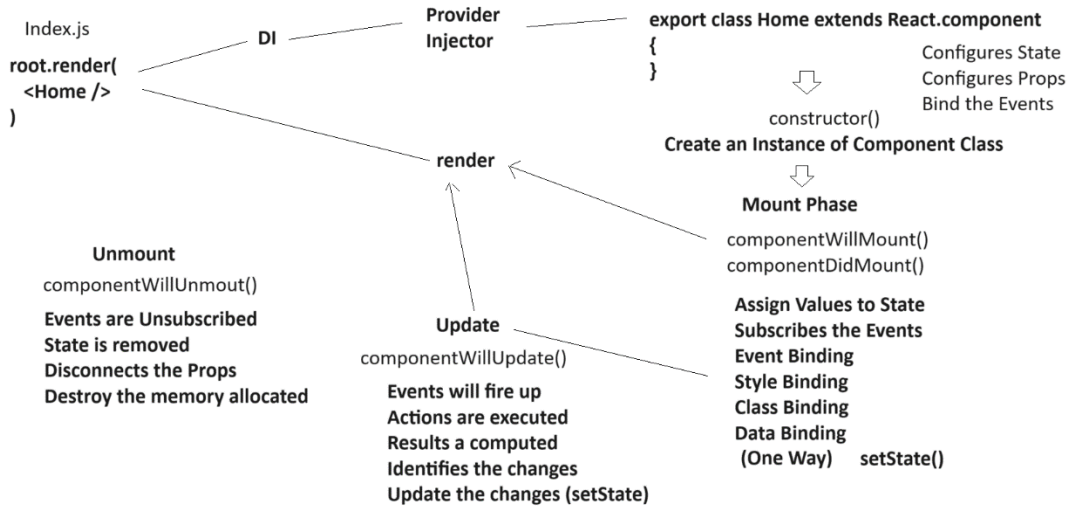
```
import React from "react";

export class Login extends React.Component
{
    constructor(){
        super();
        this.state = {
            Id : 1,
            Name: "Samsung TV",
            Price: 45000.44,
            Stock: true,
            Cities: ["Delhi", "Hyd"],
```

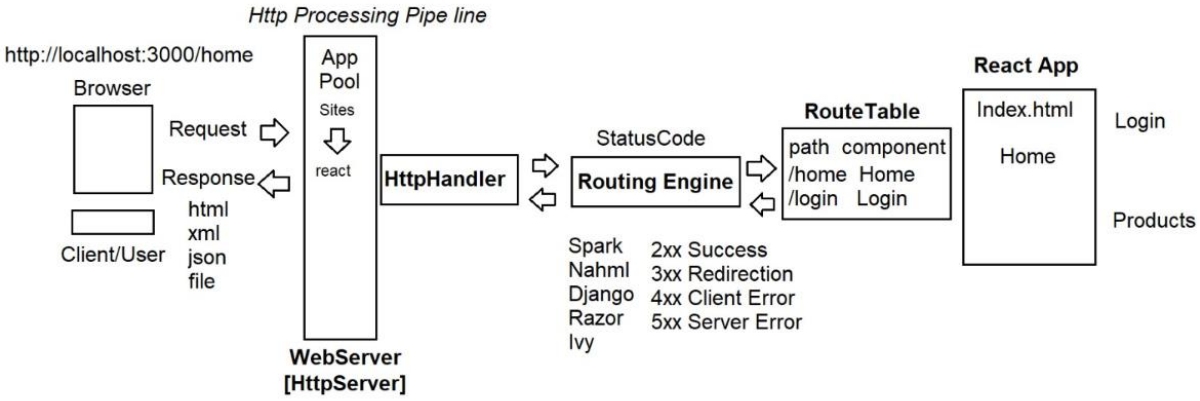
```
            Rating: {Rate:4.2}
          }
        }
        render(){
           return(
             <div className="container-fluid">
               <h2>Product Details</h2>
               <dl>
                <dt>Product Id</dt>
                <dd>{this.state.Id}</dd>
                <dt>Name</dt>
                <dd>{this.state.Name}</dd>
                <dt>Price</dt>
                <dd>{this.state.Price}</dd>
                <dt>Stock</dt>
                <dd>{(this.state.Stock==true)?"Avialable":"Out of Stock"}</dd>
                <dt>Cities</dt>
                <dd>
                   <ol>
                      {
                         this.state.Cities.map(city=>
                             <li key={city}>{city}</li>
                          )
                      }
                   </ol>
                </dd>
                <dt>Rating</dt>
                <dd>{this.state.Rating.Rate}</dd>
               </dl>
             </div>
           )
        }
}
```

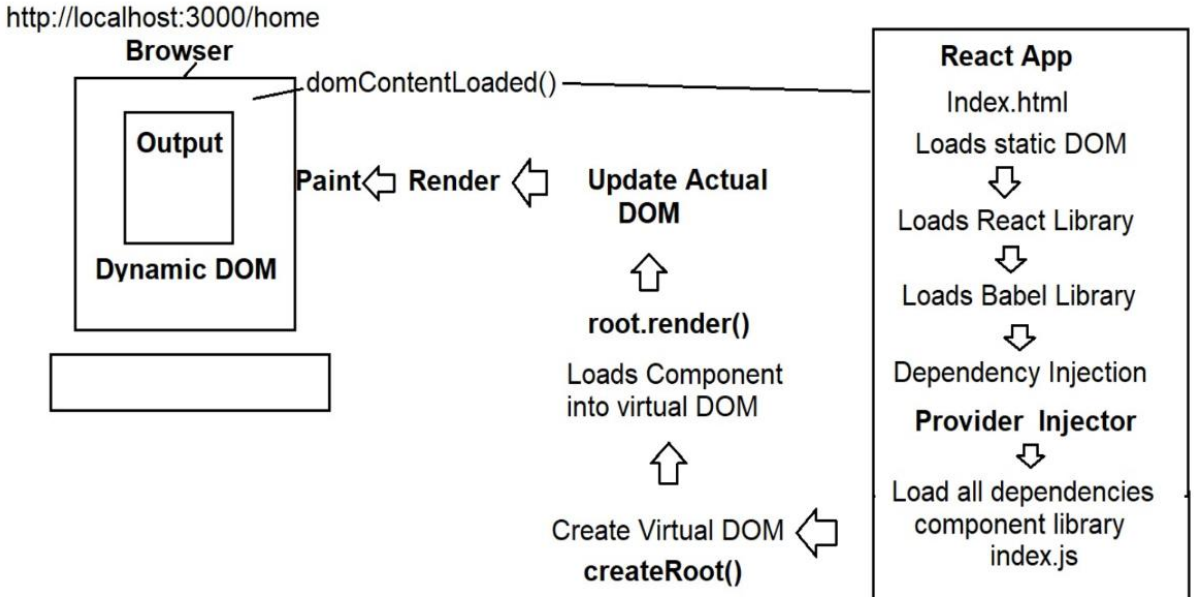
React Component Life Cycle 13/09/2023

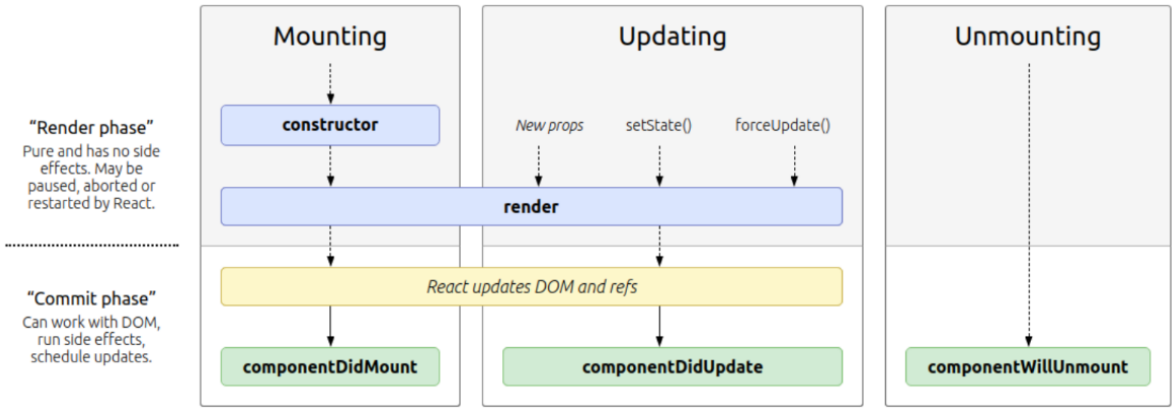

Browser
http://localhost:3000/login
Static DOM
Painting
Dynamic DOM
Layout
Render
Virtual DOM
is Updated into
Actual DOM
Renders Virtual DOM
render()
It renders component
Creates Virtual DOM
createRoot()
Bootstrapping
Client / User
React App
index.html
Loads Static DOM
Loads React Library
Loads Babel Library
Load all component Dependencies
Dependency Injection
Provider
Injector


HttpProcessingPipeline
http://localhost:3000/login
Request
Response
Client/User
HttpHandler
Routing
Engine
Route Table
request
component
/
home
/login
login
200 OK
404 NotFound
WebServer
HttpServer
React Application
index.html
Login
Link: Register
Memory
Register
Components
Home
Login
Register
Login


Index.js
root.render(
<Home />
)
DI
Provider
Injector
export class Home extends React.component
{
}
Configures State
Configures Props
Bind the Events
constructor()
Create an Instance of Component Class
render
Mount Phase
componentWillMount()
componentDidMount()
Assign Values to State
Subscribes the Events
Event Binding
Style Binding
Class Binding
Data Binding
(One Way) setState()
Update
componentWillUpdate()
Events will fire up
Actions are executed
Results a computed
Identifies the changes
Update the changes (setState)
Unmount
componentWillUnmout()
Events are Unsubscribed
State is removed
Disconnects the Props
Destroy the memory allocated


Http Processing Pipe line
http://localhost:3000/home
Browser
Request
Response
html
xml
json
file
Client/User
App Pool
Sites
react
WebServer
[HttpServer]
HttpHandler
StatusCode
Routing Engine
Spark
Nahml
Django
Razor
Ivy
2xx Success
3xx Redirection
4xx Client Error
5xx Server Error
RouteTable
path component
/home Home
/login Login
React App
Index.html
Home
Login
Products


http://localhost:3000/home
Browser
Output
Dynamic DOM
domContentLoaded()
Paint
Render
Update Actual DOM
root.render()
Loads Component into virtual DOM
Create Virtual DOM
createRoot()
React App
Index.html
Loads static DOM
Loads React Library
Loads Babel Library
Dependency Injection
Provider Injector
Load all dependencies
component library
index.js


"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Commit phase"
Can work with DOM, run side effects, schedule updates.
Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
render
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

React Forms 14/09/2023

React Forms

- Form is used for configuring UI from where user can interact with the data.
- Form is user for CRUD operations.

C	Create
R	Read
U	Update
D	Delete

- React requires lot of event handling manually to manage for and its validations.
- React form is design by using the generic container "<form>".
- Generic container have default validations and can submit data to server.

Note: If you are designing a form for interactions in container, then you can configure
without generic container.
If you are designing a form to handle interactions with server side data then
you have to configure within generic container <form>.

Ex: React Form

form-demo.jsx

```
import { useState } from "react"

export function FormDemo(){

    const [user, setUser] = useState({Name:'',Age:0, Mobile:'', City:''});

    function handleNameChange(e){
        setUser({
            Name: e.target.value,
            Age: user.Age,
```

```
        Mobile: user.Mobile,
        City: user.City
    })
}

function handleAgeChange(e){
    setUser({
        Name: user.Name,
        Age: e.target.value,
        Mobile: user.Mobile,
        City: user.City
    })
}

function handleMobileChange(e){
    setUser({
        Name: user.Name,
        Age: user.Age,
        Mobile: e.target.value,
        City: user.City
    })
}

function handleCityChange(e){
    setUser({
        Name: user.Name,
        Age: user.Age,
        Mobile: user.Mobile,
        City: e.target.value
    })
}

function handleSubmitClick(){
    alert(JSON.stringify(user));
}

return(
    <div className="container-fluid">
```

```
        <form onSubmit={handleSubmitClick}>
         <h3>Register User</h3>
            <dl>
               <dt>User Name</dt>
               <dd><input type="text" onChange={handleNameChange}
/></dd>
               <dt>Age</dt>
               <dd><input type="number"
onChange={handleAgeChange}/></dd>
               <dt>Mobile</dt>
               <dd><input type="text" onChange={handleMobileChange}
/></dd>
               <dt>City</dt>
               <dd>
                  <select onChange={handleCityChange}>
                     <option value="-1">Select City</option>
                     <option value="Delhi">Delhi</option>
                     <option value="Hyd">Hyd</option>
                  </select>
               </dd>
            </dl>
            <button>Submit</button>
         </form>
      </div>
   )
}
```

Form Validation

- Validation is the process of verifying user input.
- Validation is required to ensure that contradictionary and unauthorized data is not get stored into database.
- React doesn't provide any validation services or library for form validation.
- You have to validate manually by using JavaScript methods.

Ex:

```
import { useState } from "react"
```

```
export function FormDemo(){

    const [user, setUser] = useState({Name:'',Age:0, Mobile:'', City:''});
    const [error, setError] = useState({Name:'', Age:'', Mobile:'', City:''});



    function handleNameChange(e){
        setUser({
            Name: e.target.value,
            Age: user.Age,
            Mobile: user.Mobile,
            City: user.City
        })
    }

    function handleAgeChange(e){
        setUser({
            Name: user.Name,
            Age: e.target.value,
            Mobile: user.Mobile,
            City: user.City
        })
    }

    function handleMobileChange(e){
        setUser({
            Name: user.Name,
            Age: user.Age,
            Mobile: e.target.value,
            City: user.City
        })
    }

    function handleCityChange(e){
        setUser({
            Name: user.Name,
            Age: user.Age,
```

```
        Mobile: user.Mobile,
        City: e.target.value
    })
}

function ValidateUser(){
    if(user.Name==""){
        setError({
            Name: 'User Name Required',
            Age: error.Age,
            Mobile: error.Mobile,
            City: error.City
        })
    } else {
        setError({
            Name: '',
            Age: error.Age,
            Mobile: error.Mobile,
            City: error.City
        })
    }

    if(isNaN(user.Age)) {
        setError({
            Name: error.Name,
            Age: 'Age must be a number',
            Mobile: error.Mobile,
            City: error.City
        })
    } else {
        setError({
            Name: error.Name,
            Age: '',
            Mobile: error.Mobile,
            City: error.City
        })
    }
}
```

```
    function handleSubmitClick(e){
        e.preventDefault();
        ValidateUser();
    }

    function handleNameBlur(){
        ValidateUser();
    }
    function handleNameKeyUp(){
        ValidateUser();
    }

    return(
        <div className="container-fluid">
          <form onSubmit={handleSubmitClick}>
           <h3>Register User</h3>
              <dl>
                 <dt>User Name</dt>
                 <dd><input type="text" onKeyUp={handleNameKeyUp}
onBlur={handleNameBlur} onChange={handleNameChange} /></dd>
                 <dd className="text-danger">{error.Name}</dd>
                 <dt>Age</dt>
                 <dd><input type="text"
onChange={handleAgeChange}/></dd>
                 <dd className="text-danger">{error.Age}</dd>
                 <dt>Mobile</dt>
                 <dd><input type="text" onChange={handleMobileChange}
/></dd>
                 <dt>City</dt>
                 <dd>
                    <select onChange={handleCityChange}>
                       <option value="-1">Select City</option>
                       <option value="Delhi">Delhi</option>
                       <option value="Hyd">Hyd</option>
                    </select>
                 </dd>
              </dl>
```

```
            <button>Submit</button>
          </form>
        </div>
    )
}
```

Ex: Validation
form-demo.jsx

```
import { useState } from "react"

export function FormDemo(){

    const [user, setUser] = useState({Name:'',Age:0, Mobile:'', City:''});
    const [error, setError] = useState({Name:'', Age:'', Mobile:'', City:''});



    function handleNameChange(e){
        setUser({
            Name: e.target.value,
            Age: user.Age,
            Mobile: user.Mobile,
            City: user.City
        })
    }

    function handleAgeChange(e){
        setUser({
            Name: user.Name,
            Age: e.target.value,
            Mobile: user.Mobile,
            City: user.City
        })
    }

    function handleMobileChange(e){
        setUser({
```

```
        Name: user.Name,
        Age: user.Age,
        Mobile: e.target.value,
        City: user.City
    })
}

function handleCityChange(e){
    setUser({
        Name: user.Name,
        Age: user.Age,
        Mobile: user.Mobile,
        City: e.target.value
    })

    if(e.target.value=="-1"){
        setError({
            Name: error.Name,
            Age: error.Age,
            City: 'Please Select Your City',
            Mobile: error.Mobile
        })
    } else {
        setError({
            Name: error.Name,
            Age: error.Age,
            City: '',
            Mobile: error.Mobile
        })
    }
}

function ValidateUser(){
    if(user.Name==""){
        setError({
            Name: 'User Name Required',
            Age: error.Age,
            Mobile: error.Mobile,
```

```
            City: error.City
        })
    } else {
        setError({
            Name: '',
            Age: error.Age,
            Mobile: error.Mobile,
            City: error.City
        })
    }

    if(isNaN(user.Age)) {
        setError({
            Name: error.Name,
            Age: 'Age must be a number',
            Mobile: error.Mobile,
            City: error.City
        })
    } else {
        setError({
            Name: error.Name,
            Age: '',
            Mobile: error.Mobile,
            City: error.City
        })
    }
}

function handleSubmitClick(e){
    e.preventDefault();
    ValidateUser();
}

function handleNameBlur(){
    ValidateUser();
}
function handleNameKeyUp(){
    ValidateUser();
```

```
    }

    return(
        <div className="container-fluid">
            <form onSubmit={handleSubmitClick}>
             <h3>Register User</h3>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" onKeyUp={handleNameKeyUp}
onBlur={handleNameBlur} onChange={handleNameChange} /></dd>
                    <dd className="text-danger">{error.Name}</dd>
                    <dt>Age</dt>
                    <dd><input type="text"
onChange={handleAgeChange}/></dd>
                    <dd className="text-danger">{error.Age}</dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" onChange={handleMobileChange}
/></dd>
                    <dt>City</dt>
                    <dd>
                        <select onChange={handleCityChange}>
                            <option value="-1">Select City</option>
                            <option value="Delhi">Delhi</option>
                            <option value="Hyd">Hyd</option>
                        </select>
                    </dd>
                    <dd className="text-danger">
                        {error.City}
                    </dd>
                </dl>
                <button>Submit</button>
            </form>
        </div>
    )
}
```

3rd Party Form Libraries in React

- React provides support and integration of various 3rd party forms like

- Telerik Forms [Kendo UI Forms]
- DevExpress Forms
- Formik

Formik Library

1. Install Formik

```
>npm install  formik --save
```

2. Import useFormik hook

```
import   { useFormik } from  "formik";
```

3. Create formik reference with formik settings

```
const formik = useFormik({
   initialValues: { },
   onSubmit: ( values)=> { },
   validate: () => { },
   validationSchema: ()=>{ }, etc..
})
```

4. Bind formik methods with elements in component

```
<input type="text" onChange={formik.handleChange}>
<form onSubmit={formik.handleSubmit}>
```

Ex:

formik-demo.jsx

```
import { useFormik } from "formik";

export  function FormikDemo(){

   const formik = useFormik({
       initialValues : {
          Name: '',
          Age: 0,
```

```
            Mobile:",
            City:"
        },
        onSubmit : (values) => {
            alert(JSON.stringify(values));
        }
    })

    return(
        <div className="container-fluid">
            <form onSubmit={formik.handleSubmit}>
            <h3>Register User - Formik Form</h3>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" name="Name"
onChange={formik.handleChange} /></dd>
                    <dt>Age</dt>
                    <dd><input type="text" name="Age"
onChange={formik.handleChange} /></dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" name="Mobile"
onChange={formik.handleChange} /></dd>
                    <dt>City</dt>
                    <dd>
                        <select name="City" onChange={formik.handleChange}>
                            <option value="-1">Select City</option>
                            <option value="Delhi">Delhi</option>
                            <option value="Hyd">Hyd</option>
                        </select>
                    </dd>
                </dl>
                <button type="submit">Submit</button>
            </form>
        </div>
    )
}
```

Formik Validations 15/09/2023

Formik Validation
- Formik can handle validation using
  a) Validation Function
  b) Validation Schema

Validation Function:
- Developer can write a function manually that verifies the input values and returns an error object.
- Error object can report all errors in form.

Syntax:

```
function  ValidateUser(FormCollection)
{
    var errors = { };

     if(FormCollection.UserName=="")
     {
       errors.UserName = "User Name Required";
     }

    return errors;
}

const formik = useFormik({
     InitialValues: { },
     validate: ValidateUser,          => errors { UserName, Age,.. }
     onSubmit: ()=>{}
})

{formik.errors.UserName}
{formik.errors.Age}
```

Ex:
Formik-Validation.jsx

```
import { Form, useFormik } from "formik";
```

```
export function FormikValidation(){

    function ValidateUser(FormCollection){
        var errors = {};

        if(FormCollection.UserName==""){
            errors.UserName = "User Name Required";
        } else {
            if(FormCollection.UserName.length<4){
                errors.UserName = "Name too short.. min 4 chars";
            } else {
                errors.UserName = "";
            }
        }

        if(FormCollection.Age==""){
            errors.Age = "Age Required";
        } else {
            if(isNaN(FormCollection.Age)){
                errors.Age = "Age must be a Number";
            } else {
                errors.Age = "";
            }
        }

        if(FormCollection.Mobile==""){
            errors.Mobile = "Mobile Required";
        } else {
            if(FormCollection.Mobile.match(/\+91\d{10}/)) {
                errors.Mobile = "";
            } else {
                errors.Mobile = "Invalid Mobile : +91 and 10 digits"
            }
        }

        if(FormCollection.City=="-1"){
            errors.City = "Please Select Your City";
        } else {
```

```
            errors.City = "";
        }

        return errors;
    }


    const formik = useFormik({
        initialValues: {
            UserName:'',
            Age: 0,
            Mobile:'',
            City:''
        },
        validate : ValidateUser,
        onSubmit: (values)=>{
            alert(JSON.stringify(values));
        }
    })

    return(
        <div className="container-fluid">
            <form onSubmit={formik.handleSubmit}>
                <h2>Register User</h2>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" onBlur={formik.handleBlur}
onChange={formik.handleChange} name="UserName" /></dd>
                    <dd className="text-
danger">{formik.errors.UserName}</dd>
                    <dt>Age</dt>
                    <dd><input type="text" onChange={formik.handleChange}
name="Age" /></dd>
                    <dd className="text-danger">{formik.errors.Age}</dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" onChange={formik.handleChange}
name="Mobile" /></dd>
                    <dd className="text-danger">{formik.errors.Mobile}</dd>
```

```
                <dt>City</dt>
                <dd>
                   <select name="City" onChange={formik.handleChange}>
                      <option value="-1">Select City</option>
                      <option value="Delhi">Delhi</option>
                      <option value="Hyd">Hyd</option>
                   </select>
                </dd>
                <dd className="text-danger">{formik.errors.City}</dd>
             </dl>
             <button type="submit">Submit</button>
          </form>
       </div>
    )
}
```

- You can use formik method "getFieldProps()" , which can access all the properties and methods of a field and configure dynamically.
- It uses a "spread" syntax.

```
   <input type="text"  {...formik.getFieldProps("UserName") }
name="UserName" />

   ...formik.getFieldProps()  => onChange, onBlur, onSubmit
```

Ex:
formik-validation.jsx

```
import { Form, useFormik } from "formik";

export function FormikValidation(){

   function ValidateUser(FormCollection){
      var errors = {};

      if(FormCollection.UserName==""){
         errors.UserName = "User Name Required";
      } else {
```

```
        if(FormCollection.UserName.length<4){
            errors.UserName = "Name too short.. min 4 chars";
        } else {
            errors.UserName = "";
        }
    }

    if(FormCollection.Age==""){
        errors.Age = "Age Required";
    } else {
        if(isNaN(FormCollection.Age)){
            errors.Age = "Age must be a Number";
        } else {
            errors.Age = "";
        }
    }

    if(FormCollection.Mobile==""){
        errors.Mobile = "Mobile Required";
    } else {
        if(FormCollection.Mobile.match(/\+91\d{10}/)) {
            errors.Mobile = "";
        } else {
            errors.Mobile = "Invalid Mobile : +91 and 10 digits"
        }
    }

    if(FormCollection.City=="-1"){
        errors.City = "Please Select Your City";
    } else {
        errors.City = "";
    }

    return errors;
}


const formik = useFormik({
```

```
        initialValues: {
            UserName:'',
            Age: 0,
            Mobile:'',
            City:''
        },
        validate : ValidateUser,
        onSubmit: (values)=>{
          alert(JSON.stringify(values));
        }
    })

    return(
        <div className="container-fluid">
            <form onSubmit={formik.handleSubmit}>
                <h2>Register User</h2>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text"
{...formik.getFieldProps("UserName")} name="UserName" /></dd>
                    <dd className="text-
danger">{formik.errors.UserName}</dd>
                    <dt>Age</dt>
                    <dd><input type="text" {...formik.getFieldProps("Age")}
name="Age" /></dd>
                    <dd className="text-danger">{formik.errors.Age}</dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" {...formik.getFieldProps("Mobile")}
name="Mobile" /></dd>
                    <dd className="text-danger">{formik.errors.Mobile}</dd>
                    <dt>City</dt>
                    <dd>
                        <select name="City" {...formik.getFieldProps("City")}>
                            <option value="-1">Select City</option>
                            <option value="Delhi">Delhi</option>
                            <option value="Hyd">Hyd</option>
                        </select>
                    </dd>
```

```
                <dd className="text-danger">{formik.errors.City}</dd>
              </dl>
              <button type="submit">Submit</button>
            </form>
        </div>
    )
}
```

- Formik provides support for validation schema, which is a pre-defined validation library called "yup".

```
>npm  install yup  --save
```

- yup provides validation library with a set of predefined functions

```
yup.required()
yup.minlength()
yup.maxlength()
yup.number()
yup.string()
yup.matches()  etc..
```

- You can import any specific function or all functions from yup.

```
import  * as yup  from  "yup";

import  {required, minlength} from yup
```

- All yup functions are defined in "yup.object({ })"

Note: If you are using pre-defined validation functions, then you have to configure using "ValidationSchema".

```
const  formik = useFormik({
    initialValues : { },
    validationSchema: yup.object({
      UserName: yup.required("Name Required").minlength(4,
"Message"),
      Age: yup.number("Age must be number").required("Message")
```

```
    })
  })
```

Ex: Schema

formik-validation.jsx

```
import { useFormik } from "formik";
import * as yup from "yup";

export function FormikValidation(){

  const formik = useFormik({
      initialValues: {
        UserName:'',
        Age: 0,
        Mobile:'',
        City:''
      },
       validationSchema : yup.object({
          UserName: yup.string().required('Name Required').min(4,
"Name too short.."),
          Age: yup.number().required("Age Required"),
          Mobile: yup.string().required("Mobile
Required").matches(/\+91\d{10}/,"Invalid Mobile")
       })
      ,
      onSubmit: (values)=>{
        alert(JSON.stringify(values));
      }
  })

  return(
    <div className="container-fluid">
      <form onSubmit={formik.handleSubmit}>
        <h2>Register User</h2>
        <dl>
          <dt>User Name</dt>
```

```
                <dd><input type="text"
{...formik.getFieldProps("UserName")} name="UserName" /></dd>
                <dd className="text-
danger">{formik.errors.UserName}</dd>
                <dt>Age</dt>
                <dd><input type="text" {...formik.getFieldProps("Age")}
name="Age" /></dd>
                <dd className="text-danger">{formik.errors.Age}</dd>
                <dt>Mobile</dt>
                <dd><input type="text" {...formik.getFieldProps("Mobile")}
name="Mobile" /></dd>
                <dd className="text-danger">{formik.errors.Mobile}</dd>
                <dt>City</dt>
                <dd>
                  <select name="City" {...formik.getFieldProps("City")}>
                    <option value="-1">Select City</option>
                    <option value="Delhi">Delhi</option>
                    <option value="Hyd">Hyd</option>
                  </select>
                </dd>
                <dd className="text-danger">{formik.errors.City}</dd>
              </dl>
              <button type="submit">Submit</button>
            </form>
        </div>
    )
}
```

- Formik provides pre-defined components, which can handle the elements and validations.
- Formik components include

```
<Formik>
<Form>
<Field>
<ErrorMessage>
```

Syntax:

```
   <Formik   initialValues={}  onSubmit={
}  validation={}  validationSchema={} >
      <Form>
         <Field type="text" name="UserName" />
         <ErrorMessage name="UserName" />
      </Form>
   </Formik>
```

Ex:
formik-validation.jsx

```
import { useFormik, Formik, Form, Field, ErrorMessage } from "formik";
import * as yup from "yup";

export function FormikValidation(){

   return(
      <div className="container-fluid">
         <Formik
            initialValues={{
               UserName:'',
               Age: 0,
               Mobile:'',
               City:''
            }}

            validationSchema={
               yup.object({
                  UserName: yup.string().required('Name Required').min(4,
"Name too short.."),
                  Age: yup.number().required("Age Required"),
                  Mobile: yup.string().required("Mobile
Required").matches(/\+91\d{10}/,"Invalid Mobile")
               })
            }

            onSubmit={(values)=>{
               alert(JSON.stringify(values));
```

```
        }}
      >
        <Form>
          <dl>
            <dt>User Name</dt>
            <dd>
              <Field name="UserName" type="text" />
            </dd>
            <dd className="text-danger">
              <ErrorMessage name="UserName" />
            </dd>
            <dt>Age</dt>
            <dd>
              <Field name="Age" type="text" />
            </dd>
            <dd>
              <ErrorMessage name="Age" />
            </dd>
            <dt>Mobile</dt>
            <dd>
              <Field name="Mobile" type="text" />
            </dd>
            <dd>
              <ErrorMessage name="Mobile" />
            </dd>
            <dt>City</dt>
            <dd>
              <Field as="select" name="City">
                <option>Select City</option>
                <option>Delhi</option>
              </Field>
            </dd>
          </dl>

          <button type="submit">Submit</button>
        </Form>
      </Formik>
    </div>
```

```
    )
}
```