React 19v

============

- HTML, CSS, JavaScript, Bootstrap

1. What is React?
A. React is a JavaScript library used to build UI for web and native applications.

2. Can we build any application using React?
A. No. It requires frameworks like
   a) Next.js
   b) Remix
   c) Gatsby
   d) Expo (Mobile Native Apps)

3. What are the languages used for React?
A. JavaScript, Typescript.

4. What is difference between React & Angular?
A. React is a library.
   Angular is a framework.

5. Where React is recommended?
A. If your application in Backend is enabled with lot of frameworks and you want only a good UI for application, then it is recommended to use React.

6. Where Angular is used?
A. If application is using limited backend frameworks and they need both front end and backend enabled with framework to control application flow, then Angular is recommended.

7. What is difference between React & React JS?
A. Both are same. [Modern name is React]

8. Why we need technologies like Angular & React?
A. TO build SPA & PWA

9. What are the challenges in modern web development?
A.
   a) Unified UX
      - Application must have same experience across devices.
      - Mobile users must get access to everything.

   b) Fluid UX
      - User stays on one page and gets access to everything from the page.
      - It is a SPA [Single Page Application].

   c) Loosely Coupled & Extensible

- Allows to build new features outside the scope and integrate into application
  without leading to catastrophic failures.

10. What is the solution?
A. Better build "SPA" & "Progressive Web Application" [PWA].

Summary
- What is React?
- Difference between React & Angular
- Why we need React?
- What are the challenges in modern web development?
      Unified UX
      Fluid UX
      Loosely Coupled and Extensible
- What is solution?
      SPA
      PWA
- Can we build SPA using JavaScript & jQuery?
   Yes

FAQ: What are the issues with JavaScript & jQuery?
Ans:
      - JS & jQuery use lot of DOM manipulations
      - They are slow & heavy on application
      - It requires to use lot of Ajax explicitly.

Features of React JS:
1. React uses Virtual DOM.
2. It is modular.
3. It is light weight and faster.
4. Loosely coupled & extensible
5. It is component based.
6. Easy to reuse and extend.
7. It is easy for mantainability and testability.

Issue with React:
- Lot of GAP's.
- Lot of 3rd party library are required

FAQ's:
1. What is DOM?
A. It is a hierarchy of elements in page.
   Browser engine presents content in a hierarchical order by using HTML parsing.

2. HTML Parsing
A.

```
Markup=>Bytes => Chars => Tokens => Nodes => DOM => Render => Layout => Paint

3. Browser Architecture
      - User Interface
      - UI Backend
      - Browser Engine
      - Rendering Engine
      - JavaScript Interpreter
      - Networking
      - Data Persistence

4. Browser Engines
      - V8
      - Chakra
      - Chromium
      - Webkit
      - Gecko
      - Spider Monkey

5. What is Shadow DOM?
A. It is a hierarchy of elements in a component. HTML have various components
like
            <input type="date">
            <input type="email">
            <input type="number">
            <input type="range">
            etc..

6. What is Virtual DOM?
A. It is a copy of DOM in browser memory.
    Page updates virtual and reflects output to user. However it is updated into
actual
    DOM later.
------------------------------------------------

Summary
- Features of React
      Virtual DOM [DOM, Shadow DOM, Virtual DOM]
      Modular
      Loosely Coupled & Extensible
      Component Based
- Issues with React
Setup Environment for React:
1. Your PC must have OS windows 8x, Mac, Linux etc.

2. Download and Install "Node JS" on your PC.

            https://nodejs.org/en

    - Node JS is required to build servers, web apps, scripts & command line
tools.
    - Node JS provides a package manager called NPM.
            [ Yarn, Bower, Grunt, Compose, Ruby Gems, NuGet ]
```

```
3. Check the version of Node & NPM from command prompt

      C:\> node  -v
      C:\> npm   -v

    Note: Make sure that the node version is 18x and higher &
          npm version is 8x and higher.

4. Download and Install "Visual Studio Code" editor

      https://code.visualstudio.com/

     - VS code provides an IDE [Integrated Development Environments] for
building,
      debugging, testing and deploying apps.

5. Add Extensions to VS Code editor

      - Live Server
      - VS-Code Icons
      - IntelliSense for CSS class names in HTML

Create a new Web Application:
1. Create a new folder for your project

            D:\web-app

2. Open your project folder in VS Code

3. Setup the file system for Web Application

      a) Generate package.json
           - Open Terminal & run the command

                 > npm init  -y

           - package.json comprises of project meta data.

      b) Add "README.md"  into project

          - It is a help document designed by developers for developers.

      c) Add  ".gitignore", It is required for publishing on GIT and ignoring
specific set
          of resources.

      d) Add folders
                 - public         : It comprises of static resources [html,
images, docs..]
                 - src            : It comprises of dynamic resources [.css, .js,
.ts, .scss..]

      e) Add following pages into public folder
```

```
                  - index.html
                  - home.html
```

Setup React for your web application:

1. Setup library using CDN for legacy React [up to 17x]

     - React in page requires following libraries

```
          a) React               [Core Library]
          b) React DOM           [Virtual DOM Library]
          c) Babel               [Compiler]
```

     - React libraries you can get from "https://legacy.reactjs.org/"

```
<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>
```

     - Babel library you can get from "https://babeljs.io/"

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

     - Copy the babel @standalone library

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

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Index</title>
</head>
<body>
    <h2>Index Page</h2>
    React is in <a href="./home.html">Home</a> page.
</body>
</html>
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</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>
    <h2>Home</h2>
    <noscript>Please enable JavaScript on your browser.</noscript>
    <div id="root"></div>
</body>
</html>
```

How to use react in existing web application?
1. Install Node JS
2. VS Code
3. Create a project
4. Setup file system for project
5. Enable React in page using CDN
    a) React
    b) React DOM
    c) Babel

Install react & babel library for project:

1. Run the following commands in your project terminal

```
            >npm  i  react   react-dom   @babel/standalone  --save   [ latest -
19]

            >npm  i  react@18.2  react-dom@18.2  @babel/standalone --save
```

2. All library files are copied into "node_modules".

3. You web application requires react libraries from "UMD" [Universal Module Distribution] system.

4. Link library files to your HTML page.

```
<head>
<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>
</head>
```

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>Home</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", document.getElementById("root"));
    </script>
</head>
<body>
    <h2>Home</h2>
    <noscript>Please enable JavaScript on your browser.</noscript>
    <div id="root"></div>
</body>
</html>
```

FAQ: What is "ReactDOM" ?
Ans: It is a property that creates "Virtual DOM" and "render()" is a method that renders
virtual DOM into actual DOM.

Syntax:

```
      ReactDOM.render( <your_component>,  dom_target_element );
```

React Components

- Component is a template with pre-defined design, styles and functionality.
- Design is created using "HTML".
- Styles are configured using "CSS".
- Functionality is defined by using "JavaScript / TypeScript".
- React uses JavaScript Extension known as "JSX".
- React component can be created by using 2 techniques
  a) JavaScript Class
  b) JavaScript Function

JavaScript Function Topics:
- Function Declaration
- Function Expression
- Function Definition
- Function Parameters
- Rest Parameters
- Spread Operator
- Function Closure
- Function Return
- Function Currying
- Higher Order Functions
- Function Recursion
- Function Signature
- Function Generator
- Function Call back
- Function Promise
- Async Functions
- Anonymous Functions

- IIFE Pattern
- Arrow Functions

Function Component Rules:
1. A component function can't be void.
2. A component function must return JSX element.
3. Component function name must start with Uppercase letter.
4. Component JSX must return only one fragment.
5. Every element in component must have an end token.

```
           <img> </img>
           <img />
           <input type="text">  </input>
           <input type="text" />
```

Syntax:

```
       function   Component()
       {
         return (
                 <fragment>

                 </fragment>
                 );
       }
```

- Component is accessed and used as a token "<Component />".

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>Home</title>
    <style>
        form {
            border: 1px solid gray;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 2px 2px 2px black;
            width: 200px;
            margin: 20px;
        }
        nav {
            display: flex;
            justify-content: space-between;
            border: 1px solid gray;
            padding: 20px;
        }
        footer {
            background-color: black;
            color:white;
            text-align: center;
            padding: 10px;
        }
    </style>
    <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">

        function Login()
        {
            return (
                <form>
                    <h3>User Login</h3>
                    <dl>
                        <dt>User Name</dt>
                        <dd><input type="text" /></dd>
                        <dt>Password </dt>
                        <dd><input type="password"/> </dd>
                    </dl>
                    <button>Login</button>
                </form>
            );
        }

        const SearchBar = ()=> (
            <div>
                <input type="text" placeholder="Search Netflix.in" />
                <button>Search</button>
            </div>
        )

        const Navbar = () => (
            <nav>
                <div>Netflix</div>
                <div>
                    <SearchBar />
                </div>
                <div>
                  <select>
                    <option>Language</option>
                    <option>English</option>
                  </select>
                  <button>Sign In</button>
                </div>
            </nav>
        )

        function Footer()
        {
            return(
                <footer>
                    &copy; 2025  All right reserved for Netflix
                </footer>
            )
        }

        ReactDOM.render(<section> <Navbar/> <Login/>  <Footer /> </section>,
document.getElementById("root"));
    </script>
</head>
<body>
    <noscript>Please enable JavaScript on your browser.</noscript>
    <div id="root"></div>
</body>
</html>
----------------------------------------------
```

Note: JSX elements can't use attributes. You have to configure properties.

```
                 <img  src=""  width=""  height="" class="img-fluid">
attributes

                 document.querySelector("img").src="";         property
                 document.querySelector("img").className=""
```

Syntax:

```
           <img  src=""  className=""  />
```

All JSX elements from HTML must be in lowercase.

```
            <form> <button> <select>          valid
            <Form> <Button> <Select>          Invalid
```

Setup Bootstrap for project:

1. Open Terminal
2. Run the following command in project terminal

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

3. Link the bootstrap files to your web page.

```
<head>
 <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-
icons.css">
 <link rel="stylesheet" href="../node_modules/bootstrap/dis/css/bootstrap.css">
 <script src="../node_modules/bootstrap/dist/js/bootstrap.bundle.js">
</head>
```

Ex: Netflix.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Netflix</title>
    <style>
        .bg-image {
            background-image: url("netflix-banner.jpg");
            height: 100vh;
            background-size: cover;
        }
        .bg-shade {
            background-color: rgba(0,0,0,0.7);
            height: 100vh;
        }
        .brand-title {
            font-size: 35px;
            font-weight: bold;
            color:red;
        }
        main {
            padding-top: 100px;
            font-family: Arial;
        }
        .main-title {
            font-size: 45px;
            font-weight: bold;
        }
        .main-subtitle {
            font-size: 22px;
        }
    </style>
    <!-- Bootstrap Library -->
    <link  rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-
icons.css">
    <link  rel="stylesheet"
href="../node_modules/bootstrap/dist/css/bootstrap.css">
    <script src="../node_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>
    <!-- React Library -->
    <script src="../node_modules/react/umd/react.development.js"></script>
    <script src="../node_modules/react-dom/umd/react-
dom.development.js"></script>
    <!-- Compiler Library -->
    <script src="../node_modules/@babel/standalone/babel.js"></script>
    <!-- React Application Library -->
    <script type="text/babel">
        function NetflixIndex(){
            return(
                <div className="bg-image">
                    <div className="bg-shade">
                        <NetflixHeader />
                        <NetflixMain />
                    </div>
                </div>
            )
        }
        function NetflixHeader(){
            return(
                <header className="p-4 text-white d-flex justify-content-
between">
                    <div className="brand-title"> NETFLIX </div>
                    <div className="d-flex">
                        <div>
                            <div className="input-group">
                            <span className="bi bi-translate input-group-
text"></span>
                            <select className="form-select">
                                <option>Language</option>
                                <option>English</option>
                            </select>
                        </div>
                        </div>
                        <div>
                            <button data-bs-target="#signin" data-bs-
toggle="modal" className="btn btn-danger ms-2">Sign In</button>
                            <div className="modal fade" id="signin">
                                <div className="modal-dialog modal-dialog-
centered">
                                    <div className="modal-content">
                                        <div className="modal-header">
                                            <h3 className="bi bi-person-fill
text-danger">User Login </h3>
                                            <button data-bs-dismiss="modal"
className="btn btn-close"></button>
                                        </div>
                                        <div className="modal-body text-dark">
                                            <dl>
                                                <dt>User Name </dt>
                                                <dd><input type="text"
className="form-control"/> </dd>
                                                <dt> Password </dt>
                                                <dd><input type="password"
className="form-control"/> </dd>
                                            </dl>
                                        </div>
                                        <div className="modal-footer">
                                            <button className="btn btn-danger w-
100">Login </button>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </header>
            )
        }
        function NetflixMain(){
            return(
                <main className="text-white text-center">
```

```
                    <div className="main-title">Unlimited movies, TV <br /> shows 
and more</div>
                    <div className="main-subtitle">Starts at ₹149. Cancel at any 
time.</div>
                    <NetflixRegister />
                </main>
            )
        }
        function NetflixRegister(){
            return(
                <form className="d-flex justify-content-center mt-4">
                    <div>
                        <p>Ready to watch? Enter your email to create or restart 
your membership.</p>
                     <div>
                        <div className="input-group input-group-lg">
                        <input className="form-control" type="email" 
placeholder="Email address" />
                        <button className="btn btn-danger"> Get Started <span 
className="bi bi-chevron-right"></span> </button>
                        </div>
                      </div>
                    </div>
                </form>
            )
        }

        ReactDOM.render(<NetflixIndex />, document.getElementById("root"));

    </script>
</head>
<body>
    <noscript> Please enable JavaScript on your browser </noscript>
    <div id="root"></div>
</body>
</html>
----------
```

React 19

React Application:
- You can create react application without any framework. It requires any bundling tool like "Webpack, parcel, vite etc.
- You can create production grade react application using any framework like
  a) Next.js
  b) Remix
  c) Gatsby
  d) Expo etc.

Creating a React application without framework: [using Webpack bundler]

1. Go to any physical location on your PC using command prompt

```
           D:\> npm config set legacy-peer-deps true
```

2. Create a new application using the command

```
          D:\> npx   create-react-app    your_app_name

          D:\> npx   create-react-app    demo-react-app
```

3. Open project folder in VS code   "D:\demo-react-app"

4. Project file system comprises of following files & folders

```
     node_modules                : It contains all library files installed using
NPM.
     public                          : It contains static resources like html,
image, docs etc
     src                             : It contains dynamic resources like js,
jsx, ts, css etc.
     .gitignore                      : It configures folder & file to ignore
for GIT.
     package.json                    : It comprises of project meta data.
     package.lock.json           : It comprises of dependencies meta data.
     README.md                   : It is a help document for developers.
```

5. Open Terminal in your project and install JavaScript Validation module.

```
      >npm  install  ajv  --save
```

6. Start project using the command

```
     > npm start
```

- You project starts on local server at 3000 port
- Open any browser and request

```
               http://127.0.0.1:3000
                         (or)
               http://localhost:3000
```

React Application Flow [High Level Design]:

1. React application starts with "index.html" defined in "public" folder

```
          <div id="root"> </div>
```

2. The logic for index page is defined in "src/index.js". It creates virtual DOM and renders into actual DOM.

Version 18x & 19x:

```
          const  root = ReactDOM.createRoot(document.getElementById("root");
          root.render(
                     <App />
                      );

   Version up to 17x:

          ReactDOM.render(<App />, document.getElementById("root"));
```

FAQ: What is <React.StrictMode> ?
Ans: Strict Mode is used in JavaScript application to avoid code inconsistency.

Adding a new component in to project:

1. Add a new file into "src" folder with extension ".jsx or .js"

```
                       login.jsx



export function Login()
{
    return(
        <div>
            <h3>User Login</h3>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" /></dd>
                <dt>Password</dt>
                <dd><input type="password" /></dd>
            </dl>
            <button>Login</button>
        </div>
    )
}
```

2. Go to "index.js"

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

      render(
             <Login />
       )
```

Data Binding & State

Component in React Application
1. Every component in react application comprises of 3 files
            a) JSX
            b) CSS
            c) test.js

2. ".jsx" is for designing the markup [presentation]. It also contains the application logic.

3. ".css" comprises of styles.
4. "test.js" or "spec.js" comprises of test cases, which are used to test the component functions and UI.

```
                            login.jsx
                            login.css
                            login.test.js
```

5. CSS files are linked to components by using "import" statement.

```
                            login.jsx

                import   './login.css';
```

Note: Always use class & id selectors for styles in component.
Type selector applies effects to all elements across all components.

Ex:
1. Go to src folder and add a new folder by name "components".

2. Add a new folder for "register".

3. Add following files into register folder

```
            register.jsx
            register.css
            register.test.js
```

Ex:
register.jsx

```
import './register.css';

export function Register()
{
    return(
        <div className="form-container">
            <form className="form-register">
                <h3>Register User</h3>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" /></dd>
                    <dt>Password</dt>
                    <dd><input type="password" /></dd>
                    <dt>Email</dt>
                    <dd><input type="email" /></dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" /></dd>
                </dl>
                <button>Register</button>
            </form>
        </div>
    )
}
```

register.css

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

Enable bootstrap for React application:
1. Install bootstrap from terminal

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

2. Go to "index.js" in src folder

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

3. register.jsx

```
export function Register()
{
    return(
        <div className="d-flex justify-content-center align-items-center">
            <form className="mt-4 w-25 p-4 alert alert-warning alert-dismissible
border border-secondary rounded rounded-2">
                <h3 className="bi bi-person-fill"> Register User</h3>
                <button className="btn btn-close"></button>
                <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>Email</dt>
                    <dd><input type="email" className="form-control"  /></dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" className="form-control"  /></dd>
                </dl>
                <button className="btn btn-warning w-100">Register</button>
            </form>
        </div>
    )
}
```

Data Binding

- Data binding is the process of accessing data from source and binding to UI elements.
- Data binding is classified into 2 types
    a) One Way Binding
    b) Two Way Binding

- React supports "One way binding".
- One way is secured and will not allow accidental changes in data from UI.
- However you can implement "Two way binding" using explicit techniques.
- React uses a data binding expression "{ }".

Syntax:
```
            var  uname = "John";

            <p> Hello !  { uname } </p>

            <input type="text"  value={uname} />
```

Note: Don't use variable for storing data in a component. Variables are immutable.
        It is always recommend to store data using "state" in component.

FAQ: What is state? Why we need state?
Ans:
- Web applications use "http" as protocol.
- "http" is a stateless protocol.
- It can't remember data between pages.
- It requires various state management techniques both client side & server side.

React Component State:
- Component are state less.
- You have to configure state explicitly by using React hooks
    a) useState
    b) useContext
    c) userReducer
    d) useMemo
    e) useCallBack
    f) useRef

----

useState Hook

- Hook is a function.
- It is a predefined function provided by React library.
- useState can configure a local state for component.
- You can store any value in local state and use across requests.

Syntax:
```
           const [getter, setter] = useState();
```

- Getter allows to read value from state.

- Setter allows to assign value into state.

Ex:
```
        const [userName  , setUserName] = useState( "John" );

        <p>  {  userName   }  </p>

        setUserName("David");           // initializing memory
        setUserName = "David";          // assigning  - invalid
```

- State can handle any type of data.

            a) Primitive
            b) Non Primitive

- JavaScript Primitive Types

            - number
            - string
            - Boolean
            - undefined
            - null
            - symbol
            - bigint

FAQ's:
1. Can we declare state with var or let keywords?
A. Yes. But not recommended.

2. Why state is configured with "const"?
A. State must be initialized for react component. The keywords var & let allows assignment. State can't be assigned it must be initialized.

3. If you declare state using const then how you can assign a new value?
A. Const will not allow to assign, however we can change the constant by initializing memory for value.

Syntax:
```
                  setter = value;              // invalid - const will not allow to
assign
                  setter(value);               // valid - const initializes memory
```

Binding Number Type:
- Number type can be any one of the following

Signed Integer	- 10
Unsigned Integer	10
Floating Point	3.5
Double	345.423
Decimal	2345.554592
Exponent	2e3
Hexadecimal	0x3314
Octa	0o753
Binary	0b1010

```
            Bigint                        994288182n
```

- The formatting methods of JavaScript are same in React

```
            toFixed()
            toLocaleString()
```

Syntax:

```
           const [value] = useState(450000.00);

            {  value.toFixed(2)  }
            {  value.toLocaleString() }             450,000
            {  value.toLocaleString('en-in') }      4,50,000
            {  value.toLocaleString('en-in', { style: 'currency', currency: 'INR'
})
```

https://www.instagram.com/sharmatechgroup/

Ex:
data-binding.jsx

```
import { useState } from "react"

export  function DataBinding()
{

    const [value] = useState(450000.00);

    return(
        <div className="container-fluid">
            <h2>Data Binding</h2>
            <p> value = {value.toLocaleString('en-in',{style:'currency',
currency:'INR'})} </p>
        </div>
    )
}
```

- You can convert a numeric string into number by using following methods

```
            a) parseInt()
            b) parseFloat()
```

- You can verify number type by using "isNaN()" [boolean]

Note: JSX will not allow any statements in Markup. You have to handle only with
       operators and functions.

Syntax:

```
            <p>
                  {
                        if(condition) { }            // invalid

                  }
            </p>
            <p>
                  {
                    (condition)?true:false           // valid
                  }
            </p>
```

Ex:
data-binding.jsx

```
import { useState } from "react"

export  function DataBinding()
{

    const [age] = useState("A");

    return(
        <div className="container-fluid">
            <h2>Data Binding</h2>
            <p>
                {
                    (isNaN(age))?'Age must be a number':age
                }
            </p>
        </div>
    )
}
```

Binding String Type:
- JS string can be defined using 3 techniques

      a) Double Quote   " "
      b) Single Quote   '   '
      c) Backtick   `  `

- Backtick allows embedded expression.
- Databinding expression of JavaScript string is "${ }".

Syntax:

```
         const [str] = useState('text-danger');

        <h2  className={`border border-1 p-2 ${str}`}>
```

Ex:
data-binding.jsx

```
import { useState } from "react"

export  function DataBinding()
{

    const [validationClass] = useState('text-success')

    return(
        <div className="container-fluid">
            <h2 className={`border border-2 text-center mt-2 p-2
${validationClass}`}>Data Binding</h2>

        </div>
    )
}
```

- All string formatting and manipulation methods are same as in JavaScript.

```
            bold()
            italics()
            fontsize()
            toUpperCase()
            toLowerCase()
            fontcolor() etc..

            charAt(), charCodeAt()
            indexOf(), lastIndexOf(),
            substr(), slice(), substring()
            replace(), replaceAll(), match()
            trim(), split() etc..
```

Binding Boolean Types
- Boolean type are defined with true or false.
- React will not recommend to use Boolean with 0 & 1.

```
            0 = false
            1 = true
```

- React can't show the Boolean value, it can just use the value.

Syntax:

```
      const  [stock] = useState(true);

      <p>  {  (stock==true)?"true":"false" } </p>
```

Undefined Type:
- React will not display undefined value but it can use the type.
- You can verify by using "undefined" or by using defined technique.

Syntax:

```
       const  [price] = useState();

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

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

Null Type
- Null is a type defined for values at runtime of application.
- It is verified by using "null" keyword.

Syntax:
```
      const [ price ]  = useState(prompt('enter price'));

      <p>  {  (price===null) ? "Not defined" : price }  </p>
```

Note: Remove "<React.Strictmode>" from index.js to avoid compiling 2 times.
        [for development and production]

Symbol Type:
- It is a primitive JavaScript type.
- It is used to configure hidden properties.
- Hidden property is not displayed over iterations, but accessible individually.

Syntax:
```
      var [id] = Symbol();

      var product = {
           [id] : 1,
           Name: "TV"
       }
```

Non Primitive Types
- They a mutable types
- They are stored in memory heap
- They don't have any fixed range for value.
- Value range varies according to memory available.
- JavaScript non-primitive types
      a) Array
      b) Object
      c) Map / WeakMap

Arrays in React:
- Array configuration is same as in JavaScript.
- All array methods are used similarly in React.
```
      pop()       indexOf()
      push()            lastIndexOf() etc..
      shift()
      unshift()
      forEach()
      map()
      find()
      filter()
      slice()
```

Syntax:
```
        const  [values] = useState([ ]);
        const  [values] = useState(new Array());
        values.map(function(value) { })       // not recommended

        values.map(value => <element> </element>)

        values.map(value => { <element> </element> })     // invalid
```

Ex:
data-binding.jsx

```
import { useState } from "react"

export  function DataBinding()
{

    var [categories] = useState(['All','Electronics', 'Footwear', 'Fashion']);

    return(
        <div className="container-fluid">
           <h3>Categories</h3>
           <nav className="bg-danger btn-group">
              {
                categories.map(category=><button className="btn btn-
danger">{category}</button>)
              }
           </nav>
           <ol>
             {
                categories.map(category=> <li>{category}</li>)
             }
           </ol>
           <select>
              {
                 categories.map(category=> <option>{category}</option> )
              }
           </select>

        </div>
    )
}
```

Note: If any element is repeating in JSX, then every repeating element must have a unique key.

Syntax:

```
           {
             categories.map((category,index)=> <li key={index}> {category} </li>
           }

           {
             categories.map( category => <option key={category}> {category}
</option>
           }
```

Ex:

data-binding.jsx

```
import { useState } from "react"

export  function DataBinding()
{

    var [categories] = useState(['All','Electronics', 'Footwear', 'Fashion']);

    return(
        <div className="container-fluid">
           <h3>Categories</h3>
           <nav className="bg-danger btn-group">
              {
                categories.map((category,index)=><button key={index}
className="btn btn-danger">{category}</button>)
              }
           </nav>
           <ol>
             {
                categories.map(category=> <li key={category}>{category}</li>)
             }
           </ol>
           <select>
              {
                 categories.map(category=> <option
key={category}>{category}</option> )
              }
           </select>

        </div>
    )
}
```

Ex:

```
import { useState } from "react"

export  function DataBinding()
{

    const [sales] = useState([45000, 68000, 32000, 78000, 35000]);

    return(
        <div className="container-fluid">
            <h3>Sales above 40k</h3>
            <ol>
                {
                    sales.filter(value=> value>40000).map(item=> <li
key={item}>{item}</li>).reverse().sort()
                }
            </ol>
        </div>
    )
```

}

Object Type [JSON]

Binding Object Type:
- Object is a key and value collection.

```
            {
              Key: value
            }
```

- Key is string type & value can be any type.
- The value are accessed with reference of Key.

```
      <p> { objectName.key } </p>
```

Object Manipulations:
1. How to read all keys from object?
A. Object.keys()

2. How to remove any specific key?
A. By "delete" operator

3. How to check any key?
A. By using "in" operator

4. How to check the data type of value in a key?
A. By using "typeof" operator

Ex:
data-binding.jsx

```
import { useState } from "react"

export  function DataBinding()
{

    const [product] = useState({Id:1, Name:"TV", Price:4000.33, Stock:true,
Cities:['Delhi', 'Hyd'], Rating:{Rate:4.5, Count:500}});

    return(
        <div className="container-fluid">
            <h2>Product Details</h2>
            <dl>
                <dt>Product Id</dt>
                <dd>{product.Id}</dd>
                <dt>Name</dt>
                <dd>{product.Name}</dd>
                <dt>Price</dt>
                <dd>{product.Price}</dd>
                <dt>Stock</dt>
                <dd>{(product.Stock===true)?"Available":"Out of Stock"}</dd>
                <dt>Cities</dt>
                <dd>
                    <ol>
                        {
                            product.Cities.map(city=><li key={city}>{city}</li>)
                        }
                    </ol>
                </dd>
                <dt>Rating</dt>
                <dd>
                    {product.Rating.Rate} <span className="bi bi-star-
fill"></span> {product.Rating.Count}
                </dd>
            </dl>
        </div>
    )
}
```

Ex:

```
import { useState } from "react"

export  function DataBinding()
{

    const [product] = useState({Id:1, Name:"TV", Price:4000.33, Stock:true,
Cities:['Delhi', 'Hyd'], Rating:{Rate:4.5, Count:500}});

    return(
        <div className="container-fluid">
            <h2>Product Details</h2>
            <ol>
                {
                    Object.keys(product).map(key=> <li key={key}>{key}</li>)
                }
            </ol>
        </div>
    )
}
```

Array of Objects:
- Data from API is accessed in JSON format.
- Usually JSON comprises of an Object with key & value or an Array with a collection of object.

Syntax:

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

- The basic Array iterators are used to read and present data.

```
            map()
            forEach()
```

Ex:
data-binding.jsx

```
import { useState } from "react"

export  function DataBinding()
{

    const [products] = useState([
        {Name: "TV", Price:45000.44},
        {Name: "Mobile", Price:12000.33},
        {Name: "Watch", Price:3000.44},
        {Name: "Jeans", Price:3600.55}
    ]);

    return(
        <div className="container-fluid">
            <h2>Product Details</h2>
            <table className="table table-hover">
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Price</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        products.map(product=><tr key={product}>
<td>{product.Name}</td> <td>{product.Price}</td> <td> <button className="btn btn-
warning bi bi-pen-fill mx-2"></button> <button className="btn btn-danger bi bi-
trash"></button> </td> </tr>)
                    }
                </tbody>
            </table>
        </div>
    )
}
```

FAQ: What are the issues with Object type?
Ans:

- Key must be always string type.
- It requires all explicit methods for manipulation
- It is slow in interactions
- It is structured. [not good for schema less data]

Map Type
- It is a key and value collection.
- Key can be any type and value can be any type.

- It provides implicit methods.
- It is faster than object.

Syntax:

```
        var data = new Map();

        data.set(key, value)
        data.get(key)
        data.remove(key)
        data.clear()
        data.has(key)
        data.size
        data.keys()
        data.values()
        data.entries() etc..
```

Date Type:
- Date values are stored using a Date() constructor.
- JavaScript date & time functions are used to present date.

```
            getHours()                    setHours()
            getMinutes()                        setMinutes()
            getSeconds()                        setSeconds()
            getMilliSeconds()             setMilliSeconds()
            getDate()                           setDate()
            getDay()                            setMonth()
            getMonth()                    setFullYear()
            getFullYear()
            toLocaleDateString()
            toDateString()
            toLocaleTimeString()
            toTimeString()
```

- React can't use the date & time functions of JavaScript to present in desired format.
- React requires various 3rd party library to handle date & time.
- The popular date & time libraries of JavaScript are
    a) moment
    b) dayjs
    c) luxon

Setup Moment in your project:
1. Open Terminal
2. Install Moment

```
      > npm  install  moment --save
```

3. Import moment into component

```
      import   moment  from  "moment";
```

4. Define format for your date value using moment.

```
      const [mfd] = useState(Date());
      <p>  {  moment(mdf).format('date_time_format')  }  </p>

            do            5th
            dd            05
            ddd           Fri
            dddd    Friday               https://momentjs.com/

            MM            02
            MMM     Feb
            MMMM    February

            YY           25
            YYYY   2025
```

Ex:
data-binding.jsx

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

export  function DataBinding()
{

    const [Mfd] = useState(Date());


    return(
        <div className="container-fluid">
           <p>{moment(Mfd).format('dddd do MMMM YYYY')}</p>
        </div>
    )
}
```

Regular Expression Type:
- Regular expression uses Meta characters and quantifiers.
- Expression is enclosed in "/ /".
- Expression is verified by using "match()" method.

Syntax:

```
        const [mobile] = useState("9876543210");

        <p>
           {
              (mobile.match(/\+91\d{10}/) ? "Verified": "Invalid Mobile"
           }
      </p>
```

Ex:
data-binding.jsx

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

```
export  function DataBinding()
{

    const [password] = useState(prompt("Enter Password"));

    return(
        <div className="container-fluid">
           <p>
              {
                 (password.match(/(?=.*[A-Z])\w{4,15}/))?"Verified":"Invalid
Password"
              }
           </p>
        </div>
    )
}

Summary:
1. Primitive Types
           number
           string
           Boolean
           null
           undefined
           symbol
           bigint

2. Non Primitive Type
           array
           object
           map
3. Additional
           date
           regular expression
-------------------------------------------
React Application
Components
Data Binding
- One Way Binding
- How to binding different data types?
- useState()

Set State:
- State is configured while creating component.
- You can't set a new value into state while creating component.
- You can only initialized a value.
- Every component have a mount phase. [load phase]
- You can initialize new values or set new values into state while mounting
component.
- Component mount phase is defined using "useEffect()" hook.

Syntax:
           useEffect(()=>{
            },[ dependencies ])
```

- Dependencies specify when to load the component again.

Ex:
data-binding.jsx

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

export  function DataBinding()
{

    const [productName, setProductName] = useState('TV');
    const [price, setPrice] = useState(0);

    useEffect(()=>{

        setProductName('Samsung TV');
        setPrice(5000.44);

    },[])



    return(
        <div className="container-fluid">
           <p>
              Product Name : {productName} <br />
              Price : {price}
           </p>
        </div>
    )
}
```

Working with API's

FAQ: What is distributed computing?
Ans: Distributed computing allows communication between two applications running
        on two different machines.

        It also allows communication between two different objects of two different
        applications running in different process of same machine.

FAQ: What are the various distributed computing technologies?
Ans:

```
            CORBA [Common Object Request Broken Architecture]   14 languages
            DCOM  [Distribute Component Object Model]   - Visual Basic
            RMI        [Remote Method Invocation]  - J2EE
            EJB        [Enterprise Java Beans]           - Java
            Webservice                                    - all technologies
            Remoting                                      - .NET
```

Note: The most popular distributed computing technology used across all languages is
       Webservice. However it have issues for modern web, hence the alternative is API.

FAQ: What is API?
Ans: - Application Programming Interface
       - It handles communication between applications in distributed architecture.
       - It can run on any protocol.
       - It can run on any server.
       - It uses XML & JSON communication.

Ex:
1. Add a new file into public folder
            "products.json"

```
[
    {
        "Name": "Samsung TV",
        "Price": 45000.44
    },
    {
        "Name": "Mobile",
        "Price": 12000.43
    },
    {
        "Name":"Watch",
        "Price": 5000.55
    }
]
```

2. Data-binding.jsx

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

export  function DataBinding()
{

    const [products, setProducts] = useState([]);

    function LoadData(){

        var http = new XMLHttpRequest();
        http.open("get","products.json",true);
        http.send();

        http.onreadystatechange = function(){

            if(http.readyState===4){
                 setProducts(JSON.parse(http.responseText));
            }
        }

    }


    useEffect(()=>{

        LoadData();

    },[])



    return(
        <div className="container-fluid">
            <h2>Products</h2>
            <table className="table table-hover">
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Price</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        products.map(product=> <tr key={product}>
                            <td>{product.Name}</td>
                            <td>{product.Price}</td>
                        </tr>)
                    }
                </tbody>
            </table>
        </div>
    )
}
```

JavaScript AJAX Techniques
- Asynchronous JavaScript and XML.
- It enables "Partial Post Back".
- It allows to add new content into page without reloading the page.
- Browser handle AJAX by using "XMLHttpRequest" object.

Communication Specification in Service:
1. SOAP
2. REST
3. JSON

SOAP
- Service Oriented Architecture Protocol
- Service consumer sends an XML request.

```
      <Products>
```

```
         <Product>
         <Category> Electronics </Category>
         </Product>
     </Products>
```

- Service Provider sends an XML response.

```
     <Products>
        <Product>
            <ProductId> 1 </ProductId>
            <Name> Mobile </Name>
            <Category> Electronics </Category>
        </Product>
        ....
        ....
    </Products>
```

REST:
- Representational State Transfer
- Service consumer sends a simple query request.

```
    https://sever.com/products?category=electronics
```

- Service provider sends response in XML or JSON format.

```
     [
        {
        ProductId: 1,
        Name: "Mobile",
        Category: "Electronics"
        },
        ...
        ...
    ]
```

JSON:
- JavaScript Object Notation
- Service consumer sends an JSON request.

```
        {
          Category: "Electronics"
        }
```

- Service provider sends an JSON response.

```
        [
          {   },
          {   }
        ]
```

XMLHttpRequest:
- It is a browser window object member.
- It is the native Ajax method used by browser.
- It "Sync" by default.

- You have to explicitly make it "Async".
- It returns response in XML, HTML or Text.
- It requires explicit conversion methods to convert data into JSON.
- It is not good in error handling.

1. Create a new XMLHttpRequest object

```
     var http = new XMLHttpRequest();
```

2. Configure the request by using "open()" method

```
    http.open("method-type", "url", async:boolean);

    method-type        : GET, POST
    Url                : API URL
    async            : true / false
```

3. Send the request into process

```
    http.send();
```

4. Execute the request in process by using "onreadystatechange" event.

```
    http.onreadystatechange = function()  {

    }
```

5. Get response on ready state.

```
     if (http.readyState===4)
    {

    }
```

6. Response is returned by using following techniques

```
        http.responseText
        http.responseHTML
        http.responseXML
```

7. If your data is in JSON format then convert data by using "JSON.parse()".

Ex:
1. public/product.json

```
{
    "title": "Apple iPhone 16 (Green, 256 GB)",
    "price": 79999,
    "rating": {
        "rate": 4.5,
        "ratings": 15329,
        "reviews": 3225
    },
    "offers": [
        "Bank Offer5% Unlimited Cashback on Flipkart Axis Bank Credit CardT&C",

        "Bank Offer10% off up to ₹750 on Canara Bank Credit and Credit EMI
Transactions, on orders of ₹5,000 and aboveT&C",

        "Bank Offer10% off up to ₹750 on DBS Bank Debit Card Transactions, on
orders of ₹5,000 and aboveT&C",

        "Special PriceGet extra ₹9901 off (price inclusive of
cashback/coupon)T&C"
    ],
    "image": "iphone-green.jpg"
}
```

2. src/component/flipkart-mobile

        flipkart-mobile.jsx

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


export function FlipkartMobile()
{

    const [product, setProduct] = useState({title:'', price:0, rating:{rate:0,
ratings:0, reviews:0}, offers:[], image:''});


    function LoadProduct(){

         var http = new XMLHttpRequest();

         http.open("get", "product.json", true);

         http.send();

         http.onreadystatechange = function(){

              if(http.readyState===4){
                   setProduct(JSON.parse(http.responseText));
              }

         }

    }

    useEffect(()=>{

        LoadProduct();

    },[])
    return(
        <div className="container-fluid">
            <div className="row mt-4">
                <div className="col-3">
                    <img src={product.image} width="100%" />
                </div>
                <div className="col-9">
                    <div className="h4 my-2">{product.title}</div>
                    <div>
                        <span className="badge bg-success text-
white">{product.rating.rate} <span className="bi bi-star-fill"></span> </span>
                        <span className="text-secondary fw-bold">
{product.rating.ratings.toLocaleString()} ratings & {product.rating.reviews}
reviews </span>
                    </div>
                    <div className="my-3">
                        <div className="fs-2 fw-bold"> &#8377;
{product.price.toLocaleString('en-in')}</div>
                    </div>
                    <div>
                        <h5>Available Offers</h5>
                        <ul className="list-unstyled">
                            {
                                product.offers.map(offer=><li className="bi bi-
tag-fill my-3 text-success" key={offer}> <span className="text-
secondary">{offer}</span> </li>)
                            }
                        </ul>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

JavaScript "fetch()" API:
- Fetch is a promise of JavaScript.
- It is async by default.
- It is good in error handling.
- It provides simplified approach for AJAX.

Syntax:
```
    fetch("url")
    .then(function(response){
        // response is in binary
        // convert response into JSON
    })
    .then(function(data){

    })
    .catch(function(error){
    })
    .finally(function(){

    })
```

```
- It returns data in binary.
- Explicit conversion is required.
- It have CORS issues.
- It have security issues. [XSRF, XSS]
```

Ex: Fetch

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


export function FlipkartMobile()
{

    const [product, setProduct] = useState({title:'', price:0, rating:{rate:0,
ratings:0, reviews:0}, offers:[], image:''});


    function LoadProduct(){

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

    }

    useEffect(()=>{

        LoadProduct();

    },[])


    return(
        <div className="container-fluid">
            <div className="row mt-4">
                <div className="col-3">
                    <img src={product.image} width="100%" />
                </div>
                <div className="col-9">
                    <div className="h4 my-2">{product.title}</div>
                    <div>
                        <span className="badge bg-success text-
white">{product.rating.rate} <span className="bi bi-star-fill"></span> </span>
                        <span className="text-secondary fw-bold">
{product.rating.ratings.toLocaleString()} ratings & {product.rating.reviews}
reviews </span>
                    </div>
                    <div className="my-3">
                        <div className="fs-2 fw-bold"> &#8377;
{product.price.toLocaleString('en-in')}</div>
                    </div>
                    <div>
                        <h5>Available Offers</h5>
                        <ul className="list-unstyled">
                            {
                                product.offers.map(offer=><li className="bi bi-
tag-fill my-3 text-success" key={offer}> <span className="text-
secondary">{offer}</span> </li>)
                            }
                        </ul>
                    </div>
                </div>
            </div>
        </div>
    )
}
-----------------------------------------
```

JavaScript Ajax Methods
- XMLHttpRequest
- Fetch Promise

FAQ: What are the issues with "fetch()"?
Ans:
- It returns data in binary format.
- You have to parse the data explicitly.
- CORS issues [Cross Origin Resource Sharing]
- It allows only GET.
- It not good handling XSRF & XSS.
  [Cross Site Request Forgery & Cross Site Scripting Attacks]

jQuery Ajax
- jQuery is a JavaScript library.
- It provides various Ajax methods to handle Async requests.
- It returns data directly in the native format of source.
- It doesn't require explicit parsing of data.
- It provides various life cycle methods that allow to track the request.
- It is good in error handling.

```
- CORS, XSRF & XSS have issues. [Limited]
- You can't cancel the requests.
- jQuery Ajax methods
    a) getJSON()
    b) $.ajax()

1. Install jQuery for your project

        >npm install jquery --save

2. jQuery library is accessed in any component using "$" reference.

        import  $  from  "jquery";

3. Create Ajax request

        $.ajax( {
                method: "get | post | put | delete ..",
                url: "api_url",
                data: data_to_submit,
                success: ()=>{ on success },
                error: ()=> { on failure }
              })

4. On Success Ajax returns data in native format. [JSON, XML, Text, HTML..]

5. You can use the response data directly in component.

Ex:
flipkart-mobile.jsx

import { useEffect, useState } from "react";
import $ from "jquery";


export function FlipkartMobile()
{

    const [product, setProduct] = useState({title:'', price:0, rating:{rate:0,
ratings:0, reviews:0}, offers:[], image:''});


    function LoadProduct(){

         $.ajax({
             method: "get",
             url: "product.json",
             success: (product) => {
                 setProduct(product);
             }
         })

    }
    useEffect(()=>{

        LoadProduct();

    },[])


    return(
        <div className="container-fluid">
            <div className="row mt-4">
                <div className="col-3">
                    <img src={product.image} width="100%" />
                </div>
                <div className="col-9">
                    <div className="h4 my-2">{product.title}</div>
                    <div>
                        <span className="badge bg-success text-
white">{product.rating.rate} <span className="bi bi-star-fill"></span> </span>
                        <span className="text-secondary fw-bold">
{product.rating.ratings.toLocaleString()} ratings & {product.rating.reviews}
reviews </span>
                    </div>
                    <div className="my-3">
                        <div className="fs-2 fw-bold"> &#8377;
{product.price.toLocaleString('en-in')}</div>
                    </div>
                    <div>
                        <h5>Available Offers</h5>
                        <ul className="list-unstyled">
                            {
                                product.offers.map(offer=><li className="bi bi-
tag-fill my-3 text-success" key={offer}> <span className="text-
secondary">{offer}</span> </li>)
                            }
                        </ul>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

                        3rd Party Ajax Libraries

- React can use various 3rd Party libraries to handle Ajax requests.
- The popular 3rd party for React are
    a) Axios
    b) WHATWGFetch
    c) React Relay
    d) Redux React Ajax methods etc..
    e) Telerik

    www.npmjs.com

Axios for Ajax in React:

- It is promise based.
- It is async by default.
- It can handle multiple requests simultaneously at the same time.
- It is good in error handling.
- It provides better features to manage CORS, XSRF & XSS.
- It doesn't require explicit parsing.
- It is built for React.
- It uses virtual DOM.
- It can cancel the requests.

1. Install Axios for project

```
        > npm  install axios --save
```

2. Import axios into component

```
        import  axios  from  "axios";
```

3. Configure axios request

Syntax:

```
        axios.get("url").then(response=>{}).catch(error=>{}).finally(()=>{ })
```

4. "response" returns various details like

```
        a) statusCode            200, 404
        b) statusText            OK, Not Found
        c) data                 XML, JSON, Text, HTML
        d) headers             Request & Response details
```

Ex:
flipkart-mobile.jsx

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

export function FlipkartMobile()
{

    const [product, setProduct] = useState({title:'', price:0, rating:{rate:0,
ratings:0, reviews:0}, offers:[], image:''});


    function LoadProduct(){

         axios.get('product.json')
         .then(response=>{
             setProduct(response.data);
         })

    }

    useEffect(()=>{

        LoadProduct();
    },[])


    return(
        <div className="container-fluid">
            <div className="row mt-4">
                <div className="col-3">
                    <img src={product.image} width="100%" />
                </div>
                <div className="col-9">
                    <div className="h4 my-2">{product.title}</div>
                    <div>
                        <span className="badge bg-success text-
white">{product.rating.rate} <span className="bi bi-star-fill"></span> </span>
                        <span className="text-secondary fw-bold">
{product.rating.ratings.toLocaleString()} ratings & {product.rating.reviews}
reviews </span>
                    </div>
                    <div className="my-3">
                        <div className="fs-2 fw-bold"> &#8377;
{product.price.toLocaleString('en-in')}</div>
                    </div>
                    <div>
                        <h5>Available Offers</h5>
                        <ul className="list-unstyled">
                            {
                                product.offers.map(offer=><li className="bi bi-
tag-fill my-3 text-success" key={offer}> <span className="text-
secondary">{offer}</span> </li>)
                            }
                        </ul>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Ex: api.nasa.gov

nasa.jsx

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


export function Nasa(){

    const [marsObject, setMarsObject] = useState({photos:[]});
    useEffect(()=>{

        axios.get('https://api.nasa.gov/mars-
photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=DEMO_KEY&#39;)
        .then(response=>{
            setMarsObject(response.data);
        })

    },[])

    return(
        <div className="container-fluid">
            <h3>Mars Rover Photos</h3>
            <table className="table table-hover">
                <thead>
                    <tr>
                        <th className="bi bi-key"> Photo Id</th>
                        <th className="bi bi-camera"> Camera Name </th>
                        <th className="bi bi-rocket"> Rover Name </th>
                        <th className="bi bi-eye"> Preview </th>
                    </tr>
                </thead>
                <tbody>
                    {
                        marsObject.photos.map(item=>
                            <tr key={item.id}>
                                <td>{item.id}</td>
                                <td>{item.camera.full_name}</td>
                                <td>{item.rover.name}</td>
                                <td><a href={item.img_src} target="_blank"><img
src={item.img_src} width="100" height="100" /></a></td>
                            </tr>
                        )
                    }
                </tbody>
            </table>
        </div>
    )
}
```

Ex: Cards

Nasa.jsx

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


export function Nasa(){

    const [marsObject, setMarsObject] = useState({photos:[]});
    useEffect(()=>{

        axios.get('https://api.nasa.gov/mars-
photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=DEMO_KEY&#39;)
        .then(response=>{
            setMarsObject(response.data);
        })

    },[])

    return(
        <div className="container-fluid">
            <h3>Mars Rover Photos</h3>
            <main className="d-flex flex-wrap">
                {
                    marsObject.photos.map(item=>
                    <div key={item.id} className="card m-3 p-2 w-25">
                        <img src={item.img_src} className="card-img-top"
height="100" />
                        <div className="card-header">
                            <div className="h3">{item.id}</div>
                        </div>
                        <div className="card-body">
                                <dl>
                                    <dt>Camera Name</dt>
                                    <dd>{item.camera.full_name}</dd>
                                    <dt>Rover Name</dt>
                                    <dd>{item.rover.name}</dd>
                                </dl>
                        </div>
                    </div>
                   )
                }
            </main>
        </div>
    )
}
```

Ajax Techniques
- XMLHttpRequest
- fetch()
- jQuery Ajax
- axios

Two Way Data Binding

- React key feature is One Way Binding.
- It allows to bind the data with UI elements.
- It will not allow changes in data.
- It is more secured.
- To enable two-way-binding, it requires explicit actions to configure.

```
- Actions are defined by using "Events".

1. What is Event?
A. Event is a message sent by sender to its subscriber in order to notify the
change.
    Event follows a "Delegate" mechanism, which is a function pointer mechanism.
    Event uses a software design pattern called "Observer", which is a
communication
    pattern.

2. What is Sender?
A. Sender is trigger that identifies the changes.

3. What is Subscriber?
A. It defines the actions to perform when event triggers.

Syntax:
        function InsertClick() { }        => Subscriber

        onclick="InsertClick()"        => Sender

4. What is Event Handler?
A. Every element can have an event handler, which defines the trigger and action.

    onclick                     => Event
    onclick="InsertClick()"        => Event Handler

5. What is Event Listener?
A. A listener is configured dynamically for elements in page. So that it can
trigger
    a functionality dynamically. [runtime]

Syntax:
       document.querySelector("button").addEventListener("eventName", function(){

       })

FAQ: What is difference between onclick & click?
Ans :  onclick is an handler name.
      click is a listener name.

Syntax:
      document.getElementById("button").addEventListener("click", ()=>{});

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 InsertClick(){
         document.write("Inserted..");
      }
      function bodyload(){
         var btn = document.createElement("button");
         btn.innerHTML = "Update";
         btn.addEventListener("click",()=>{
             document.write("Updated..");
         })
         document.querySelector("body").appendChild(btn);

         document.getElementById("btnDelete").addEventListener("click",()=>{
             document.write("Deleted..");
         })

      }
   </script>
</head>
<body onload="bodyload()">
   <button id="btnDelete">Delete</button>
   <button onclick="InsertClick()">Insert</button>
</body>
</html>
```

5. What are Event Arguments?
A. Every arguments refer to payload.
    Payload refers to data carried from one location to another.

6. What are JavaScript Event Arguments?
A. JavaScript event arguments transport information about element and event.
     Elements information includes details like:
        a) id
        b) name
        c) value
        d) src
        e) href etc..
      Event information includes details like:
        a) clientX
        b) clientY
        c) keycode
        e) charCode
        f) which etc..
      JavaScript allows default arguments and custom arguments.

7. What are JavaScript default arguments?
A. this & event.

```
        this        : It contains information about element
        event    : It contains information about event
```

Ex: this

```
<!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 Player(button){
         switch(button.name){
             case "Play":
               document.querySelector("p").innerHTML = "Playing..";
               break;
               case "Pause":
               document.querySelector("p").innerHTML = "Paused..";
               break;
               case "Stop":
               document.querySelector("p").innerHTML = "Stopped..";
               break;
         }
      }
   </script>
</head>
<body>
   <button onclick="Player(this)" name="Play">Play</button>
   <button onclick="Player(this)" name="Pause">Pause</button>
   <button onclick="Player(this)" name="Stop">Stop</button>
   <p></p>
</body>
</html>
```

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 LinkClick(e, label){
          if(e.ctrlKey){
              window.open('https://www.amazon.in','Amazon&#39;);
          }
          console.log(label.title);
      }
   </script>
</head>
<body>
    <label onclick="LinkClick(event, this)" title="Ctrl+Click to follow
link">https://www.amazon.in</label&gt;
</body>
</html>
```

8. What are Custom arguments?
A. JavaScript event allows to send any custom arguments, which is user defined data.
     It can be any type of data:
    a) Primitive
    b) Non Primitive

```
Syntax:
        <button onclick="DetailsClick(1, 'TV', {rate:4.3}, ['Delhi', 'Hyd'])">

        function DetailsClick(...product)
        {
        }

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 DetailsClick(obj, e, ...product){
         var [id, name, rating] = product;
         document.write(`
               Button Id : ${obj.id} <br>
               X Position: ${e.clientX} <br>
               Product Id : ${id} <br>
               Name : ${name} <br>
               Rating : ${rating.rate}
         `);
      }
   </script>
</head>
<body>
   <button onclick="DetailsClick(this, event, 1, 'TV', {rate:4.5})"
id="btnDetails">Details</button>
</body>
</html>

----------------------------------------------

JavaScript Events

1. What is Event?
     Observer
2. What is Event Handler?
3. What is Event Listener?
4. What are Event Arguments?
5. Default Event Arguments
      this, event
6. Custom Event Arguments

Note: Event Listener can have only one default argument "event". However it
provides
      access to both event and element details.

Syntax:
     document.querySelector("button").addEventListener("click", (e)=>{
            // event details are accessible directly using event reference

              e.clientX, e.clientY, e.keyCode, e.ctrlKey..

               // element details are accessible by using "target" reference

              e.target.id, e.target.name, e.target.src, e.target.className ..

     }
```

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 bodyload(){
         document.getElementById("btnInsert").addEventListener("click", (e)=>{
               console.log(`
                   X Position : ${e.clientX} \n
                   Ctrl Key  : ${e.ctrlKey} \n
                   Button Id : ${e.target.id} \n
                   Button Name: ${e.target.name}
               `);
         })
      }
   </script>
</head>
<body onload="bodyload()">
   <button id="btnInsert" name="Insert">Insert</button>
</body>
</html>
```

7. What is Event Bubbling?
            (or)
    What is Event Propagation?

A. Event Bubbling or Propagation is a mechanism where the child event can simulate the parent. It leads to propagation of events in a parent child bubble.

8. How to prevent propagation of events?
A. By using event argument method "stopPropagation()".

Syntax:

```
        function onChildEvent(e) {

             e.stopPropagation();

        }
```

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>
   <style>
      nav {
         border: 1px solid black;
         padding: 20px;
      }
   </style>
   <script>
      function NavClick(){
         alert("Navbar Clicked");
      }
      function SearchClick(e){
         e.stopPropagation();
         alert("Search Clicked");
      }
   </script>
</head>
<body>
   <nav onclick="NavClick()">
       <h2>Navbar</h2>
       <button onclick="SearchClick(event)">Search</button>
   </nav>
</body>
</html>
```

9. How to disable the default action of element?
A. By using the event argument method "preventDefault()".

Syntax:
```
        function FormSubmit(e)
        {
            e.preventDefault();
        }
```

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 FormSubmit(e){
          e.preventDefault();
          alert('Form Submitted to Server');
      }
   </script>
</head>
<body>
    <form onsubmit="FormSubmit(event)">
```

```
        User Name  : <input type="text" name="UserName"> <button>Submit</button>
    </form>
</body>
</html>
```

10. What are browser events?
A.   All events of window object are used for elements, which are known as "Browser Events". They are classified into various groups

        - Mouse Events
        - Keyboard Events
        - Button Events
        - Clipboard Events
        - Timer Events
        - Touch Events
        - Element State Events
        - Form Events etc..

React Events

- React uses Virtual DOM.
- React virtual DOM can't use the browser events.
- React provides a library called "SyntheticEvents".
- Synthetic Events are virtual DOM events that map to actual DOM.

Syntax:
    Developer writes synthetic event "onClick" => It maps to actual event "onclick"

- SyntheticEvent is the base for all events in React.
- It handles all browser events using event map technique.
- React SynthenticEvents includes

        Mouse Events
        Keyboard Events
        Clipboard Events
        Button Events
        Touch Events
        Timer Events
        Form Events etc..

- React have support for various events but it can manage "Two-Way-Binding" only with "onChange".

```
            <input type="text"  onChange={functionName} />
            <select onChange={functionName}>
```

- The delegate function can access element value by using default event argument.

Syntax:
```
         <input type="text"  onChange={GetName} />

         function GetName(e)
         {
             e.target.value;
```

```
            e.keyCode;
        }
```

Note: To handle two-way binding every form element must be configured with
      "onChange". If onChange is not defined then React 18x version marks it as
       "read-only".

Ex:
form-demo.jsx

```
import { useState } from "react"

export function FormDemo()
{
    const [product, setProduct] = useState({Name:'TV', Price:0, City:'Delhi',
Stock:true});

    function handleNameChange(e){
        setProduct({
            Name: e.target.value,
            Price: product.Price,
            City: product.City,
            Stock: product.Stock
        })
    }
    function handlePriceChange(e){
        setProduct({
            Name: product.Name,
            Price: parseFloat(e.target.value),
            City: product.City,
            Stock: product.Stock
        })
    }
    function handleCityName(e){
        setProduct({
            Name: product.Name,
            Price: product.Price,
            City: e.target.value,
            Stock: product.Stock
        })
    }
    function handleStockChange(e){
        setProduct({
            Name: product.Name,
            Price: product.Price,
            City: product.City,
            Stock: e.target.checked
        })
    }

    function handleSubmitClick(){
        console.log(product);
    }

    return(
        <div className="container-fluid">
            <div>
                <h3>Register Product</h3>
                <dl>
                    <dt>Product Name</dt>
                    <dd><input type="text" value={product.Name}
onChange={handleNameChange} /></dd>
                    <dt>Price</dt>
                    <dd><input type="number" value={product.Price}
onChange={handlePriceChange} /></dd>
                    <dt>City</dt>
                    <dd>
                        <select value={product.City} onChange={handleCityName}>
                            <option>Select City</option>
                            <option>Delhi</option>
                            <option>Hyd</option>
                        </select>
                    </dd>
                    <dt>Stock</dt>
                    <dd><input type="checkbox" checked={product.Stock}
onChange={handleStockChange} /> <label>Available</label> </dd>
                </dl>
                <button onClick={handleSubmitClick}>Submit</button>
            </div>
        </div>
    )
}
```

Ex: Weather API

1. Visit
    https://openweathermap.org/api

2. Register a new account [free]

3. Login with registered account and copy your API key and save in any text document.

4. Go to Current Weather data in "API" category

5. Select "Built-in API request by City Name".

https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

Ex:
weather.jsx

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

```
export function Weather(){


    const url = 'https://api.openweathermap.org/data/2.5/weather&#39;;
    const api_key = '1318ca6725c69160d346c41fc0612596';

    const [cityName, setCityName] = useState('');
    const [weatherData, setWeatherData] = useState({name:'', main:{temp:0},
weather:[{description:''}]})

    function handleCityChange(e){
        setCityName(e.target.value);
    }

    function handleSearchClick(){

        // axios.get(`url?q=${cityName}&appid=${api_key}`);

        axios.get(url, {params:{
             q: cityName,
             appid: api_key,
             units:'metric'
        }})
        .then(response=>{
            setWeatherData(response.data);
            console.log(response.data);
        })
    }

    return(
        <div className="container-fluid">
            <div className="mt-4 d-flex justify-content-center">
                <div className="w-50">
                    <div className="input-group">
                        <input type="text" onChange={handleCityChange}
placeholder="Enter City Name" className="form-control" />
                        <button onClick={handleSearchClick} className="bi bi-
search btn btn-warning"></button>
                    </div>
                    <div className="mt-4">
                        <h2>{weatherData.name} -
{weatherData.weather[0].description.toUpperCase()} </h2>
                        <p className="fs-4">{Math.round(weatherData.main.temp)}
&deg; C <span className="bi bi-sun"></span> </p>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Events
- Mouse Events
- Keyboard Events

- Button Events
- Clipboard Events
etc..

Style Binding

- It is a technique of configuring inline styles for JSX elements dynamically.
- Styles in JSX are defined by using "style" property.
- It is an object type with key & value reference.
- Key is a style attribute defined in Camel Case.
- Value is always a string.

Syntax: HTML

```
    <h2  style="color:red; background-color:yellow; border:1px solid black">
```

Syntax: JSX

```
    <h2  style={ { color:'red', backgroundColor:'yellow', border:'1px solid
black' } }>
```

Ex:
weather.jsx

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


export function Weather(){


    const url = 'https://api.openweathermap.org/data/2.5/weather&#39;;
    const api_key = '1318ca6725c69160d346c41fc0612596';

    const [cityName, setCityName] = useState('');
    const [weatherData, setWeatherData] = useState({name:'', main:{temp:0},
weather:[{description:''}]})

    function handleCityChange(e){
        setCityName(e.target.value);
    }

    function handleSearchClick(){

        // axios.get(`url?q=${cityName}&appid=${api_key}`);

        axios.get(url, {params:{
             q: cityName,
             appid: api_key,
             units:'metric'
        }})
        .then(response=>{
            setWeatherData(response.data);
            console.log(response.data);
        })
    }

    return(
        <div className="container-fluid">
            <div className="mt-4 d-flex justify-content-center">
                <div className="w-50">
                    <div className="input-group">
                        <input type="text" onChange={handleCityChange}
placeholder="Enter City Name" className="form-control" />
                        <button onClick={handleSearchClick} className="bi bi-
search btn btn-warning"></button>
                    </div>
                    <div style={{marginTop:'50px', boxShadow:'2px 2px 2px black',
padding:'20px', border:'1px solid black', textAlign:'center'}}>
                        <h2>{weatherData.name} -
{weatherData.weather[0].description.toUpperCase()} </h2>
                        <p className="fs-4">{Math.round(weatherData.main.temp)}
&deg; C <span className="bi bi-sun"></span> </p>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Ex: Change Color

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


export function Weather(){


    const url = 'https://api.openweathermap.org/data/2.5/weather&#39;;
    const api_key = '1318ca6725c69160d346c41fc0612596';

    const [cityName, setCityName] = useState('');
    const [weatherData, setWeatherData] = useState({name:'', main:{temp:0},
weather:[{description:''}]});


    function handleCityChange(e){
        setCityName(e.target.value);
    }

    function handleSearchClick(){

        // axios.get(`url?q=${cityName}&appid=${api_key}`);
        axios.get(url, {params:{
             q: cityName,
             appid: api_key,
             units:'metric'
        }})
        .then(response=>{
            setWeatherData(response.data);
            console.log(response.data);
        })
    }

    return(
        <div className="container-fluid">
            <div className="mt-4 d-flex justify-content-center">
                <div className="w-50">
                    <div className="input-group">
                        <input type="text" onChange={handleCityChange}
placeholder="Enter City Name" className="form-control" />
                        <button onClick={handleSearchClick} className="bi bi-
search btn btn-warning"></button>
                    </div>
                    <div style={{marginTop:'50px', boxShadow:'2px 2px 2px black',
padding:'20px', border:'1px solid black', textAlign:'center',
backgroundColor:`${(weatherData.weather[0].description==='mist')?'lightcyan':'#00
0044'}`, color:'white'}}>
                        <h2>{weatherData.name} -
{weatherData.weather[0].description.toUpperCase()} </h2>
                        <p className="fs-4">{Math.round(weatherData.main.temp)}
&deg; C <span className="bi bi-sun"></span> </p>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Ex: Change Image

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


export function Weather(){


    const url = 'https://api.openweathermap.org/data/2.5/weather&#39;;
    const api_key = '1318ca6725c69160d346c41fc0612596';

    const [cityName, setCityName] = useState('');
    const [weatherData, setWeatherData] = useState({name:'', main:{temp:0},
weather:[{description:''}]});


    function handleCityChange(e){
        setCityName(e.target.value);
    }

    function handleSearchClick(){

        // axios.get(`url?q=${cityName}&appid=${api_key}`);

        axios.get(url, {params:{
             q: cityName,
             appid: api_key,
             units:'metric'
        }})
        .then(response=>{
            setWeatherData(response.data);
            console.log(response.data);
        })
    }

    return(
        <div className="container-fluid">
            <div className="mt-4 d-flex justify-content-center">
                <div className="w-50">
                    <div className="input-group">
                        <input type="text" onChange={handleCityChange}
placeholder="Enter City Name" className="form-control" />
                        <button onClick={handleSearchClick} className="bi bi-
search btn btn-warning"></button>
                    </div>
                    <div style={{marginTop:'50px', boxShadow:'2px 2px 2px black',
padding:'20px', border:'1px solid black', textAlign:'center',
backgroundImage:`url(${(weatherData.weather[0].description==='mist')?'mist.jpg':'
smoke.jpg'})` , color:'white', backgroundSize:'cover'}}>
                        <h2>{weatherData.name} -
{weatherData.weather[0].description.toUpperCase()} </h2>
                        <p className="fs-4">{Math.round(weatherData.main.temp)}
&deg; C <span className="bi bi-sun"></span> </p>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Class Binding

- It is a technique of binding CSS classes to JSX elements.
- A CSS class comprises of set of styles defined by using a class name.
- You can apply the class dynamically to JSX elements using "className" property.

Syntax:

```
     <h2 className='css_class_name'>
     <h2 className={ (condition)?' ' : ' ' }>
```

Ex:

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

```
export function ClassBinding(){


    const [theme, setTheme] = useState('border border-2 p-4 rounded');
    const [btnTheme, setbtnTheme] = useState('btn btn-dark w-100');

    function handleThemeChange(e){
        if(e.target.checked){
            setTheme('border border-2 p-4 rounded bg-dark text-white');
            setbtnTheme('btn btn-warning w-100');
        } else {
            setTheme('border border-2 p-4 rounded');
            setbtnTheme('btn btn-dark w-100');
        }
    }

    return(
        <div className="container-fluid d-flex justify-content-center">
            <div className="mt-4">
                <form className={theme}>
                    <div className="form-switch">
                        <input type="checkbox" onChange={handleThemeChange}
className="form-check-input" /> <label> Dark Theme </label>
                    </div>
                    <h3 className="bi bi-person-fill"> 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={btnTheme}>Login</button>
                </form>
            </div>
        </div>
    )
}
```

```
Mouse Event Binding
    onMouseOver
    onMouseOut
    onMouseDown
    onMouseUp
    onMouseMove

Ex:
1. Add a new JSON file in public folder
```

```
        mobiles.json
[
    {
        "img_src": "m1.jpg"
    },
    {
        "img_src": "m2.jpg"
    },
    {
        "img_src": "m3.jpg"
    }

]
```

2. Add component files

```
     mouse-demo.css


.col-1 img {
    border: 2px solid gray;
    padding: 5px;
}
.col-1 img:hover {
    border : 2px solid blue;
    cursor: grab;
}
```

```
    mouse-demo.jsx

import axios from "axios";
import { useEffect, useState } from "react";
import './mouse-demo.css';


export function MouseDemo(){

    const [images, setImages] = useState([{img_src:''}]);
    const [previewImage, setPreviewImage] = useState('m1.jpg');

    useEffect(()=>{

            axios.get('mobiles.json')
            .then(response=>{
                setImages(response.data);
            })

    },[])

    function handleMouseOver(e){
        setPreviewImage(e.target.src);
    }

    return(
        <div className="container-fluid">
            <div className="row">
                <div className="col-1">
                    {
                        images.map(image=>
                            <div key={image} className="my-4">
                                <img onMouseOver={handleMouseOver}
src={image.img_src}  width="100" height="100"/>
                            </div>
                        )
                    }
                </div>
                <div className="col-11 ps-4 mt-4">
                    <img height="400" src={previewImage} width="300" />
                </div>
            </div>
        </div>
    )
}
```

Ex: Mouse Animation

mouse-animation.css

```
@keyframes Spin {
    from {
        transform: rotate(0deg) scale(1);
    }
    to {
        transform: rotate(360deg) scale(2);
    }
}
.react-logo {
    animation-name: Spin;
    animation-duration: 5s;
    animation-iteration-count: infinite;
    animation-timing-function: linear;
}
```

mouse-animation.jsx

```
import { useState } from 'react';
import './mouse-animation.css';

export function MouseAnimation(){

    const [animationSpeed, setAnimationSpeed] = useState('5s');

    function handleMouseOver(){
        setAnimationSpeed('1s');
    }
    function handleMouseout(){
        setAnimationSpeed('5s');
    }

    return(
        <div className="container-fluid d-flex bg-dark justify-content-center
align-items-center" style={{height:'100vh'}}>
            <img onMouseDown={handleMouseOver} onMouseUp={handleMouseout}
style={{animationDuration:animationSpeed}} className='react-logo'
src='logo192.png' />
        </div>
    )
}
```

Mouse Events

- onMouseOver
- onMouseOut
- onMouseDown
- onMouseUp
- onMouseMove

            https://cssloaders.github.io/

Ex:
mouse-animation.jsx

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

export function MouseAnimation(){


    const [styleObj, setStyleObj] = useState({position:'fixed', left:'',
top:''});

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

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

Keyboard Events:
    onKeyUp
    onKeyDown

```
    onKeyPress
```

- KeyUp & KeyDown are recommended to handle interactions with the characters that user key-in.
- KeyPress is recommended to handle the character ASCII code.
- The common key event argument properties are
    a) keycode
    b) charCode
    c) which
    d) shiftKey
    e) ctrlKey
    f) altKey etc..

Ex:
key-demo.jsx

```
import axios from "axios";
import { useState, useTransition } from "react"

export function KeyDemo(){


    const [username, setUsername] = useState('');
    const [statusMsg, setStatusMsg] = useState('');
    const [errorClass, setErrorClass] = useState('');
    const [toggle, setToggle] = useState('d-none');

    function handleUserName(e){
        setUsername(e.target.value);
    }

    function VerifyName(){
        axios.get('users.json')
        .then(response=>{
              for(var user of response.data){
                  if(user.UserName===username){
                       setStatusMsg('User Name Taken - Try Another');
                       setErrorClass('text-danger');
                       break;
                  } else {
                      setStatusMsg('User Name Available');
                      setErrorClass('text-success');
                  }
              }
        })
    }

    function VerifyCaps(e){
        if(e.which >=65 && e.which<=90) {
            setToggle('d-block');
        } else {
            setToggle('d-none');
        }
    }

    return(
        <div className="container-fluid">
            <h3>Register User</h3>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" onKeyUp={VerifyName}
onChange={handleUserName} /></dd>
                <dd className={errorClass} >{statusMsg}</dd>
                <dt>Password</dt>
                <dd><input type="password" onKeyPress={VerifyCaps} /></dd>
                <dd className={toggle}>
                    <span className="bi bi-exclamation-triangle text-warning">
Warning : Caps ON </span>
                </dd>
            </dl>
        </div>
    )
}
```

Ex: Password with Meter

```
import axios from "axios";
import { useState, useTransition } from "react"

export function KeyDemo(){


    const [username, setUsername] = useState('');
    const [statusMsg, setStatusMsg] = useState('');
    const [errorClass, setErrorClass] = useState('');
    const [pwdMsg, setPwdMsg] = useState('');
    const [strength, setStrength] = useState(1);

    function handleUserName(e){
        setUsername(e.target.value);
    }

    function VerifyName(){
        axios.get('users.json')
        .then(response=>{
              for(var user of response.data){
                  if(user.UserName===username){
                       setStatusMsg('User Name Taken - Try Another');
                       setErrorClass('text-danger');
                       break;
                  } else {
                      setStatusMsg('User Name Available');
                      setErrorClass('text-success');
                  }
              }
        })
    }
    function handlePasswordChange(e){
         if(e.target.value.match(/(?=.*[A-Z])\w{4,15}/)) {
              setPwdMsg('Strong Password');
              setStrength(100);
         } else {
              if(e.target.value.length<4) {
                  setPwdMsg('Poor Password');
                  setStrength(20);
              } else {
                  setPwdMsg('Weak Password');
                  setStrength(70);
              }
         }
    }

    return(
        <div className="container-fluid">
            <h3>Register User</h3>
            <dl className="w-25">
                <dt>User Name</dt>
                <dd><input type="text" className="form-control"
onKeyUp={VerifyName} onChange={handleUserName} /></dd>
                <dd className={errorClass} >{statusMsg}</dd>
                <dt>Password</dt>
                <dd><input type="password" className="form-control"
onKeyUp={handlePasswordChange} /></dd>
                <dd><meter className="w-100" min={1} max={100} value={strength}
></meter></dd>
                <dd>{pwdMsg}</dd>
            </dl>
        </div>
    )
}
```

Ex: Password with Bootstrap Progress Bar

```
import axios from "axios";
import { useState, useTransition } from "react"

export function KeyDemo(){


    const [username, setUsername] = useState('');
    const [statusMsg, setStatusMsg] = useState('');
    const [errorClass, setErrorClass] = useState('');
    const [pwdMsg, setPwdMsg] = useState('');
    const [progressClass, setProgressClass] = useState('');
    const [progressWidth, setProgressWidth] = useState({width:''});

    function handleUserName(e){
        setUsername(e.target.value);
    }
    function VerifyName(){
        axios.get('users.json')
        .then(response=>{
              for(var user of response.data){
                  if(user.UserName===username){
                       setStatusMsg('User Name Taken - Try Another');
                       setErrorClass('text-danger');
                       break;
                  } else {
                      setStatusMsg('User Name Available');
                      setErrorClass('text-success');
                  }
              }
        })
    }

    function handlePasswordChange(e){
         if(e.target.value.match(/(?=.*[A-Z])\w{4,15}/)) {
              setPwdMsg('Strong Password');
              setProgressWidth({width:'100%'});
              setProgressClass('progress-bar bg-success progress-bar-striped
progress-bar-animated');
         } else {
              if(e.target.value.length<4) {
                  setPwdMsg('Poor Password');
                  setProgressWidth({width:'20%'});
                  setProgressClass('progress-bar bg-danger progress-bar-striped
progress-bar-animated');
              } else {
                  setPwdMsg('Weak Password');
                  setProgressWidth({width:'70%'});
                   setProgressClass('progress-bar bg-warning progress-bar-striped
progress-bar-animated');
              }
         }
    }

    return(
        <div className="container-fluid">
            <h3>Register User</h3>
            <dl className="w-25">
                <dt>User Name</dt>
                <dd><input type="text" className="form-control"
onKeyUp={VerifyName} onChange={handleUserName} /></dd>
                <dd className={errorClass} >{statusMsg}</dd>
                <dt>Password</dt>
                <dd><input type="password" className="form-control"
onKeyUp={handlePasswordChange} /></dd>
                <dd>
                    <div className="progress">
                        <div className={progressClass} style={progressWidth}>
                            {pwdMsg}
                        </div>
                    </div>
                </dd>
            </dl>
        </div>
    )
}
```

```
Mouse Events
    onMouseOver
    onMouseOut
    onMouseDown
    onMouseUp
    onMouseMove
Keyboard Events
    onKeyUp
    onKeyDown
    onKeyPress [Deprecated]
Button Events
    onClick
    onDoubleClick  [ondblclick]
    onContextMenu
```

FAQ: How to disable right click in page?
Ans: You can disable any JavaScript event, by using a function that returns "false".

```
      <body oncontextmenu="return false">

       document.contextmenu = function() {
        return false;
       }
```

Ex:

```
export function ButtonDemo(){

    function handleDoubleClick(){
        window.open('iphone-green.jpg','Iphone','width=300 height=400');
    }
    function handleContextMenu(){
        document.oncontextmenu = function(){
             alert('Right Click Disabled on this page');
             return false;
        }
    }

    return(
        <div className="container-fluid" onContextMenu={handleContextMenu} >
            <h2>Right Click not allowed</h2>
            <img src="iphone-green.jpg" onDoubleClick={handleDoubleClick}
width="100" height="100" />
            <p>Double Click to view large</p>
        </div>
    )
}
```

Element State Events:
    onChange
    onFocus
    onBlur

- React "Two Way Binding" is handled by using "onChange" event.
- Blur defines actions to perform when element lost focus.

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

export function ButtonDemo(){

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

    function handleFocus(){
        setTip('Name in Block Letters');
    }
    function handleBlur(){
        setTip('');
        setUserName(userName.toUpperCase());
    }

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

    return(
        <div className="container-fluid">
            <h2>Register User</h2>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" value={userName} onChange={handleChange}
onBlur={handleBlur} onFocus={handleFocus} /></dd>
                <dd>{tip}</dd>
            </dl>
        </div>
    )
}
```

Ex: EMI Calculator

emi-calculator.jsx

```
import { useState } from "react"

export function EmiCalculator(){
    const [principle, setPrinciple] = useState(100000);
    const [years, setYears] = useState(1);
    const [rate, setRate] = useState(10.45);
    const [emi, setEMI] = useState(0);

    function handlePrincipleChange(e){
        setPrinciple(e.target.value);
    }

    function handleYearChange(e){
        setYears(e.target.value);
    }

    function handleRateChange(e){
        setRate(e.target.value);
    }

    function handleCalculateClick(){

            var P = principle;
            var r = rate/12/100;
            var n = years * 12;
            var EMI = P * r * (Math.pow(1+r,n)) / (Math.pow(1+r,n)) - 1;

            setEMI(EMI);

    }

    return(
        <div className="container-fluid bg-secondary">
            <h3 className="text-center mt-3 text-white">Personal Loan EMI
Calculator</h3>
            <div className="p-4 m-4 bg-light border border-1 border-dark">
                <div className="row my-4">
                    <div className="col">
                        Amount you need &#8377; <input type="text"
value={principle} />
                    </div>
                    <div className="col">
                        for <input type="text" size="2" value={years} /> years
                    </div>
                    <div className="col">
                        Interest Rate <input type="text" size="2" value={rate} />
%
                    </div>
                </div>
                <div className="row my-4">
                    <div className="col">
                        <input type="range" onChange={handlePrincipleChange}
min={100000} max={500000} value={principle} className="form-range" />
                        <span> &#8377; 1,00,000</span>
                        <span className="float-end"> &#8377; 5,00,000</span>
                    </div>
                    <div className="col">
```

```
                        <input type="range" onChange={handleYearChange} min={1}
max={5} value={years} className="form-range" />
                        <span>1</span>
                        <span className="float-end"> 5 </span>
                    </div>
                    <div className="col">
                        <input type="range" onChange={handleRateChange}
min={10.45} max={18.45} value={rate} className="form-range" />
                        <span> 10.45%</span>
                        <span className="float-end"> 18.45% </span>
                    </div>
                </div>
                <div className="row my-4">
                    <div className="col text-end">
                            <button onClick={handleCalculateClick} className="btn
btn-primary">Calculate</button>
                    </div>
                </div>
                <div className="row my-4">
                    <div className="col">
                        <p className="text-center"> Your Monthly EMI is <span
className="fw-bold fs-4 text-primary">{Math.round(emi).toLocaleString('en-in',
{style:'currency', currency:'INR'})} </span> for next {years * 12} months.  </p>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Clipboard Events:

```
    onCut
    onCopy
    onPaste
```

- Clipboard is temporary memory of your PC.
- The content that you copy or cut is kept on clipboard.
- You can disable clipboard event with a function that "return false".

Timer Events

```
setTimeout()
clearTimeout()
setInterval()
clearInterval()
```

setTimeout():
- It is a timer event used to handle "debounce" in application.
- Bounce is a mechanism where every task is released into process immediately from memory.
- Debounce allows to lock the task in memory for a specific duration.

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


clearTimeout():
- It is used to clear the task from memory before released into process.
- It requires a memory reference of task to clear from memory.

Syntax:
        clearTimeout(ref_name);
```

React "useRef()" hook

```
- React 18x version introduced a new hook "useRef()".
- It is used to configure a reference memory, which is native to the process.
- It allocates memory and stores data, so that it can be used by the process.
- The data in reference memory is not intended to present (UI). It is intended to
use in the background process.

Syntax:
        let  thread = useRef(null);

        thread.current = value / function;

        setTimeout(thread.current);

Ex:
timeout-demo.jsx

import { useRef, useState } from "react"

export function TimeoutDemo(){

    const [msg, setMsg] = useState('');
    let thread = useRef(null);

    function Msg1(){
        setMsg('10% Volume Increased');
    }
    function Msg2(){
        setMsg('50% Volume Increased');
    }
    function Msg3(){
        setMsg('100% Volume Increased');
    }



    function handleShowClick(){
        setTimeout(Msg1, 3000);
        thread.current = setTimeout(Msg2, 6000);
        setTimeout(Msg3,10000);
    }
    function handleMuteClick(){
        clearTimeout(thread.current);
    }

    return(
        <div className="container-fluid text-center">
            <button className="my-4 btn btn-primary bi bi-volume-up"
onClick={handleShowClick}></button>
            <button onClick={handleMuteClick} className="my-4 mx-2 btn btn-danger
bi bi-volume-mute">50%</button>
            <p>{msg}</p>
        </div>
    )
}
```

setInterval()
- It is used to handle "throttle".
- Throttle is a mechanism of executing specified task repeatedly until removed from memory.
- It loads task into memory and releases a copy of task into process.

Syntax:

```
           setInterval(function(){},interval);
```

clearInterval()
- It removes the task from memory, that is loaded using setInterval().

Syntax:

```
        clearInterval(ref_name);
```

Debounce & Throttle
- setTimeout()
- clearTimeout()
- setInterval()
- clearInterval()

React Hooks
- useRef()
- useEffects()
- useState()

Ex:

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


export function IntervalDemo(){

    const [toggleButton, setToggleButton] = useState('d-block');
    const [toggleProgress, setToggleProgress] = useState('d-none');
    const [toggleImage, setToggleImage] = useState('d-none');
    const [pValue, setPvalue] = useState(1);
    const [status, setStatus] = useState('');
    let thread = useRef(null);
    let progressValue = useRef(null);

    var count = 1;
    function StartProgress(){
        count++;
        setPvalue(count);
        progressValue.current = count;
        if(count===100) {
            clearInterval(thread.current);
            setToggleProgress('d-none');
            setToggleImage('d-block');
        }
    }

    function LoadImageClick(){
        setToggleButton('d-none');
        setToggleProgress('d-block');
        thread.current = setInterval(StartProgress,100);
    }

    function PauseClick(){
        clearInterval(thread.current);
        setStatus('Paused');
    }
    function PlayClick(){
        thread.current = setInterval(StartProgress,100);
        setStatus('');
    }

    return(
        <div className="container-fluid d-flex text-center justify-content-center
align-items-center" style={{height:'100vh'}}>
            <div>
                <div className={toggleButton}>
                   <button onClick={LoadImageClick} className="btn btn-
primary">Load Image</button>
                </div>
                <div className={toggleProgress}>
                    <progress min={1} max={100} value={progressValue.current}
style={{width:'300px', height:'30px'}} ></progress>
                    <div className="my-2">
                        <button onClick={PlayClick} className="btn btn-primary bi
bi-play mx-2"></button>
                        <button onClick={PauseClick} className="btn btn-danger bi
bi-pause"></button>
                    </div>
                    <div>
                        {pValue} % Completed {status}
                    </div>
                </div>
                <div className={toggleImage}>
                    <img src="iphone-green.jpg" width="300" height="400" />
                </div>
            </div>

        </div>
    )
}
```

Fakestore API
(https://fakestoreapi.com)

https://fakestoreapi.com/products/1

Ex:

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

export function CarouselDemo(){


    const [product, setProduct] = useState({id:0, title:'', price:0,
description:'', image:'', rating:{rate:0, count:0}, category:''});

    let productId = useRef(1);

    function LoadProduct(id){
        axios.get(`https://fakestoreapi.com/products/${id}`)
        .then(response=>{
             setProduct(response.data);
        })
    }

    useEffect(()=>{
         LoadProduct(1);
    },[])

    function NextClick(){
        productId.current = productId.current + 1;
        LoadProduct(productId.current);
    }

    function PreviousClick(){
        productId.current = productId.current - 1;
        LoadProduct(productId.current);
    }


    return(
        <div className="container-fluid d-flex justify-content-center">
            <div className="card m-4 p-4 w-50">
                <div className="card-header text-center">
                    {product.title}
                </div>
                <div className="card-body row">
```

```
                    <div className="col-1 d-flex flex-column justify-content-
center align-items-center">
                        <button onClick={PreviousClick} className="btn btn-dark
bi bi-chevron-left"></button>
                    </div>
                    <div className="col-10 position-relative">
                        <div className="position-absolute badge bg-danger p-2 fs-
5 text-white rounded rounded-circle end-0 top-0">
                            {product.price.toLocaleString('en-us',
{style:'currency', currency:'USD'})}
                        </div>
                        <img height="300" src={product.image} width="100%" />
                    </div>
                    <div className="col-1 d-flex flex-column justify-content-
center align-items-center">
                    <button onClick={NextClick} className="btn btn-dark bi bi-
chevron-right"></button>
                    </div>
                </div>
                <div className="card-footer text-center">
                    <input type="range" className="form-range" />
                    <div className="my-2">
                        <button className="btn btn-primary bi bi-play"></button>
                        <button className="btn btn-warning bi bi-pause mx-
2"></button>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Ex: Carousel Demo

carousel-demo.jsx

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

export function CarouselDemo(){


    const [product, setProduct] = useState({id:0, title:'', price:0,
description:'', image:'', rating:{rate:0, count:0}, category:''});
    const [status, setStatus] = useState('');

    let productId = useRef(1);
    let thread = useRef(null);

    function LoadProduct(id){
        axios.get(`https://fakestoreapi.com/products/${id}`)
        .then(response=>{
                setProduct(response.data);
           })
    }

    function LoadProductAuto(){
        productId.current = productId.current + 1;
        LoadProduct(productId.current);
        // stop timer when product id = 20;
    }

    useEffect(()=>{
         LoadProduct(1);
    },[])

    function NextClick(){
        productId.current = productId.current + 1;
        LoadProduct(productId.current);
        // if id=20 then set end of show and set id = 20;
    }

    function PreviousClick(){
        productId.current = productId.current - 1;
        LoadProduct(productId.current);
        // if id=0 then set id=1;
    }

    function handleSeekbarChange(e){
        productId.current = e.target.value;
        LoadProduct(productId.current);
    }

    function handlePlayClick(){
        thread.current = setInterval(LoadProductAuto,5000);
        setStatus('Slide Show Playing');
    }
    function handlePauseClick(){
        clearInterval(thread.current);
        setStatus('Slide Show Paused');
    }


    return(
        <div className="container-fluid d-flex justify-content-center">
            <div className="card m-4 p-4 w-50">
                <div className="card-header text-center" style={{height:'70px'}}>
                    {product.title}
                    <div className="fw-bold">{status}</div>
                </div>
                <div className="card-body row">
                    <div className="col-1 d-flex flex-column justify-content-
center align-items-center">
                        <button onClick={PreviousClick} className="btn btn-dark
bi bi-chevron-left"></button>
                    </div>
                    <div className="col-10 position-relative">
                        <div className="position-absolute badge bg-danger p-2 fs-
5 text-white rounded rounded-circle end-0 top-0">
                            {product.price.toLocaleString('en-us',
{style:'currency', currency:'USD'})}
                        </div>
                        <img height="300" src={product.image} width="100%" />
                    </div>
                    <div className="col-1 d-flex flex-column justify-content-
center align-items-center">
                    <button onClick={NextClick} className="btn btn-dark bi bi-
chevron-right"></button>
                    </div>
                </div>
                <div className="card-footer text-center">
                    <input onChange={handleSeekbarChange} min={1} max={20}
type="range" value={productId.current} className="form-range" />
                    <div className="my-2">
                        <button onClick={handlePlayClick} className="btn btn-
primary bi bi-play"></button>
                        <button onClick={handlePauseClick} className="btn btn-
warning bi bi-pause mx-2"></button>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Ex:
Time Ticking

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

export function TimeoutDemo(){

    const [timer, setTimer] = useState();


    function Clock(){
        var now = new Date();
        setTimer(now.toLocaleTimeString());
    }

    useEffect(()=>{
        setInterval(Clock,1000);
    },[])

    return(
        <div className="container-fluid text-center">
            <h1 className="mt-3">{timer}</h1>
        </div>
    )
}
```

Task : Try designing a stop watch.

```
import { useState } from "react"

export function StopWatch(){

    const [ms, setMs] = useState(0);

    const [sec, setSec] = useState(0);
    const [min, setMin] = useState(0);
    const [hrs, setHrs] = useState(0);

    const [btnText, setBtnText] = useState('Start');

    var count = 100;
    var seconds = 0;
    function Counter(){
        count++;
        if(count===900){
            count = 100;
            seconds++;
            setSec(seconds);
        }
        setMs(count);
    }

    function handleStartClick(){
        setInterval(Counter,1);
        setBtnText((btnText==='Start'?'Stop':'Start'));
    }

    return(
        <div className="container-fluid d-flex justify-content-center">

           <div>
           <div className="mt-4 border border-1 border-dark row p-2 fs-4 fw-bold"
style={{height:'60px', width:'600px'}}>
                <div className="col">
                    {hrs} hrs
                </div>
                <div className="col">
                    {min} min
                </div>
                <div className="col">
                    {sec} sec
                </div>
                <div className="col">
                    {ms} ms
                </div>
            </div>
              <button onClick={handleStartClick} className="btn btn-primary mt-
4">{btnText}</button>
           </div>

        </div>
    )
}
```

Summary
- setTimeout()
- clearTimeout()
- setInterval()
- clearInterval()

Touch Events

onTouchStart()
onTouchEnd()
onTouchMove()

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


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


    function handleTouch(e) {
        setMsg(e.target.alt);
    }

    return(
        <div className="container-fluid">
             <img onTouchStart={handleTouch} alt='iPhone Green - 128 GB
&#8377;69999' src="iphone-green.jpg"  width="200" height="300"/>
             <img onTouchStart={handleTouch} alt='iPhone white - 256 GB
&#8377;74999' src="iphone-white.jpg"  width="200" height="300"/>
             <h2>{msg}</h2>
        </div>
    )
}
```

Summary:
1. Mouse Events
    onMouseOver
    onMouseOut
    onMouseDown
    onMouseUp
    onMouseMove
2. Keyboard Events
    onKeyup
    onKeydown

```
    onKeypress
3. Button Events
    onClick
    onDoubleClick
    onContextMenu

4. Clipboard Events
    onCut
    onCopy
    onPaste

5. Element State Events
    onChange
    onFocus
    onBlur

6. Touch Events
    onTouchStart
    onTouchEnd
    onTouchMove

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

8. Form Events
    onSubmit
    onReset

  - Form is a generic element that provides submit and reset actions by default.
  - If you want any functionality to trigger on submit and reset then you can
configure custom form events.

Syntax:
        <form  onSubmit={ }   onReset={ }>

        </form>

Ex:
import { useState } from "react"


export function TouchDemo(){


    function handleSubmit(e){
        e.preventDefault();
        alert('Form data posted to server');
    }

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

```
            <form onSubmit={handleSubmit}>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" name="UserName" /></dd>
                </dl>
                <button>Submit</button>
            </form>
        </div>
    )
}
```

FAQ: How to send custom arguments using event handler?
Ans: Event handler requires to use a function that returns handler function.
     The handler function can pass both default and custom arguments.

Syntax:

```
      <button onClick={ (event) => handleClick(event, 'your custom args') }>

      function handleClick(e, ...args)
      {

      }
```

Ex:

```
import { useState } from "react"


export function TouchDemo(){


    function handleButtonClick(e,...product){
        var [id,name,stock,cities,rating] = product;
        console.log(`X=${e.clientX}\nClassName=${e.target.className}`);

console.log(`id=${id}\nname=${name}\nstock=${stock}\ncities=${cities}\nrating=${r
ating.rating}`);
    }


    return(
        <div className="container-fluid">
             <button onClick={(event)=> handleButtonClick(event, 1, 'TV', true,
['Delhi', 'Hyd'], {rating:4.5})} className="btn btn-primary mt-4"
id="btnInsert">Insert</button>
        </div>
    )
}
```

React Forms

- React supports only one-way binding.
- Handling form interactions in React is challenging for developer, as it
requires lot of explicit two-way bindings to implement.

- It is difficult to collect data from "form" and submit to server.
- React two way binding requires lot of event binding techniques to implement.
- React allows to implement 3rd party form libraries that make form interactions easy and simplified.
- The popular 3rd party form libraries are
    a) Formik
    b) React Hook Form
    c) Kendo Form [Telerik] etc..

React Formik Library

Features:
- It provides built-in two-way-binding methods.
- You have to attach to the form elements.
- It provides built-in events that can access data from elements and submit to server.
- It provides built-in security features that prevent XSRF & CORS.
- It supports built-in validation.
- You can define custom validation or use validation schemas.
- It provides built-in components that makes the form design easy.

1. Install Formik for your project

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

2. Formik library provides "useFormik()" hook, which is used to configure a form.

```
const formik = useFormik({
 initialValues: { },
 onSubmit: () =>{ },
 validate : ()=>{ },
 validationSchema: ()=>{ },
 enableReinitialize: true|false
})
```

```
 initialValues        : It refers to the values that a form have to handle.
                        It uses the form element "name".

onSubmit            : It collects the form values and submits to server.

validate            : It uses a function that can validate form values.
                      It requires custom validation.

validationSchema    : It uses pre-defined validations.


enableReinitialize    : It allows to modify the initialized values.
```

3. Bind onChange for every element with formik.handleChange.

```
<input type="text"  onChange={formik.handleChange} />
```

```
        <select onChange={formik.handleChange} />
```

Note: Every form element Name must match with initialValues.

```
        <input type="text" onChange={formik.handleChange} name="UserName">
```

4. Form must use "onSubmit" bind with "formik.handleSubmit"

```
        <form onSubmit={formik.handleSubmit}>
```

react-form.jsx

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


export function ReactForm(){


    const formik = useFormik({
        initialValues: {
            UserName: '',
            Mobile: '',
            City: '',
            Gender: ''
        },
        onSubmit: (values) => {
            console.log(values);
        }
    })

    return(
        <div className="container-fluid">
            <form onSubmit={formik.handleSubmit}>
                <h3>Register User</h3>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" onChange={formik.handleChange}
name="UserName" /></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>Select City</option>
                            <option>Delhi</option>
                            <option>Hyd</option>
                        </select>
                    </dd>
                    <dt>Gender</dt>
                    <dd>
```

```
                        <input onChange={formik.handleChange} type="radio"
name="Gender" value="Male" /> <label>Male</label>
                        <input onChange={formik.handleChange} type="radio"
name="Gender" value="Female" /> <label>Female</label>
                    </dd>
                </dl>
                <button type="submit">Register</button>
            </form>
        </div>
    )
}
```

Form Validation
- Validation is the process of verifying user input.
- Validation is required to ensure that contradictory and unauthorized data is not get stored into database.
- React Formik library allows to write custom validations and can also use validation schemas.
- HTML 5 provides default validations for form elements, which include
        a) required
        b) minlength
        c) min
        d) max
        e) step
        f) email
        g) url
        h) pattern etc..
- You can disable the default form validations using "novalidate" attribute.

```
        <form  novalidate>  </form>
```

- It is required to disable default validations when you are writing custom validations.

Formik Custom Validations:
1. Create a function that takes form data and validates data.
2. The validation function must return errors object.

Syntax:
```
        function ValidateUser(formData)
        {

                 var errors = { fieldname:' ' };

           // your validation logic for every field;

         return errors;
         }
```

3. Assign validation function to formik validation

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

4. You can access and display validation errors in UI, by using "formik.errors"

```
        { formik.errors.fieldname }
```

Ex:
form-demo.jsx

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


export function ReactForm(){

    function ValidateUser(user){
        var errors = {UserName:'', Age:'', Mobile:'', City:'', Gender:''};

        if(user.UserName.length===0){
            errors.UserName = 'User Name Required';
        } else {
            if(user.UserName.length<4){
                errors.UserName = 'Name too short';
            } else {
                errors.UserName = '';
            }
        }

        if(user.Mobile.length===0){
            errors.Mobile = 'Mobile Required';
        } else {
            if(user.Mobile.match(/\+91\d{10}/)){
                errors.Mobile = '';
            } else {
                errors.Mobile = 'Invalid Mobile';
            }
        }

        if(user.City==='-1'){
            errors.City = 'Please select your city';
        } else {
            errors.City = '';
        }

        if(user.Gender===''){
            errors.Gender = 'Please select a gender';
        } else {
            errors.Gender = '';
        }

        if(isNaN(user.Age)){
            errors.Age = 'Age must be a number';
        } else {
            errors.Age = '';
        }


        return errors;
    }

    const formik = useFormik({
        initialValues: {
            UserName: '',
            Age: 0,
            Mobile: '',
            City: '',
            Gender: ''
        },
        validate : ValidateUser,
        onSubmit: (values) => {
            console.log(values);
        }
    })

    return(
        <div className="container-fluid">
            <form onSubmit={formik.handleSubmit}>
                <h3>Register User</h3>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" onChange={formik.handleChange}
name="UserName" /></dd>
                    <dd className="text-danger">{formik.errors.UserName}</dd>
                    <dt>Age</dt>
                    <dd><input type="text" name="Age"
onChange={formik.handleChange} /></dd>
                    <dd className="text-danger">{formik.errors.Age}</dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" name="Mobile"
onChange={formik.handleChange}  /></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>
                    <dt>Gender</dt>
                    <dd>
                        <input onChange={formik.handleChange} type="radio"
name="Gender" value="Male" /> <label>Male</label>
```

```
                        <input onChange={formik.handleChange} type="radio"
name="Gender" value="Female" /> <label>Female</label>
                    </dd>
                    <dd className="text-danger">{formik.errors.Gender}</dd>
                </dl>
                <button type="submit">Register</button>
            </form>
        </div>
    )
}
```

Formik Validation Schema:
- Formik can use pre-defined validation schemas.
- Validation Schema is a structured validation with pre-defined validation services.
- Formik uses validation services from "Yup" library.

React Formik Library

Custom Validation

React Yup Library

- Yup is a validation schema library.
- It can be used with any JavaScript based libraries.
- It provides pre-defined validation services.
- Service is a pre-defined business logic, which you can customize and implement according to the requirements.

1. Install Yup library for project

```
        > npm  install yup --save
```

2. Import services from yup

```
    import  { required, min, max,.. }  from "yup";
                  (or)
    import * as yup from "yup";
```

3. Configure validation schema for formik using "yup.object()".

```
    const formik = useFormik({

            initialValues: { },
            validationSchema: yup.object({
                field : yup.datatype().required('msg').min(n, 'msg').max(n,
'msg'),
                field : ..
            })
    })
```

4. All validation errors are accessible using "formik.errors".

```
        { formik.errors.fieldname }
```

Ex:
 form-demo.jsx

```
import { useState } from "react";
import { useFormik } from "formik";
import * as yup from "yup";



export function ReactForm(){



    const formik = useFormik({
        initialValues: {
            UserName: '',
            Age: 0,
            Mobile: '',
            City: '',
            Gender: ''
        },
        validationSchema: yup.object({
             UserName: yup.string().required('Name Required').min(4, 'Name too
short'),
             Age: yup.number().min(15, 'Minimum age is 15').max(30,'Maximum age
is 30'),
             Mobile: yup.string().required('Mobile
Required').matches(/\+91\d{10}/, 'Invalid Mobile')
        }),
        onSubmit: (values) => {
            console.log(values);
        }
    })

    return(
        <div className="container-fluid">
            <form onSubmit={formik.handleSubmit}>
                <h3>Register User</h3>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" onChange={formik.handleChange}
name="UserName" /></dd>
                    <dd className="text-danger">{formik.errors.UserName}</dd>
                    <dt>Age</dt>
                    <dd><input type="text" name="Age"
onChange={formik.handleChange} /></dd>
                    <dd className="text-danger">{formik.errors.Age}</dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" name="Mobile"
onChange={formik.handleChange}  /></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>
                    <dt>Gender</dt>
                    <dd>
                        <input onChange={formik.handleChange} type="radio"
name="Gender" value="Male" /> <label>Male</label>
                        <input onChange={formik.handleChange} type="radio"
name="Gender" value="Female" /> <label>Female</label>
                    </dd>
                    <dd className="text-danger">{formik.errors.Gender}</dd>
                </dl>
                <button type="submit">Register</button>
            </form>
        </div>
    )
}
```

```
- Formik uses various events for validation.
    a) onSubmit            : on form submit
    b) onChange            : on value change in element
    c) onBlur                : on lost focus
```

- If you want the validation for blur then you have to configure "formik.onblur"

```
<input type="text" onChange={formik.handleChange}  onBlur={formik.handleBlur} />
```

- If you want multiple events to validate any element then you can use "formik" spread operator that can access all field properties.

Syntax:

```
 <input type="text" name="UserName"   {...formik.getFieldProps("UserName") } />

 <select  name="City"  {...formik.getFieldProps("City") }>
```

- Formik provides built-in components for configuring and validating form.
- These components simplify the design and functionality.
- You can't use "hooks" in class components, hence formik components play key role in class components.
- Formik components are

```
            <Formik>
            <Form>
            <Field>
            <ErrorMessage>
```

```
Syntax:
    <Formik  initialvalues={}   validationSchema={}   validate={}  onSubmit={}>

        <Form>

            <Field  type|as = "element" />
            <ErrorMessage />

        </Form>

    </Formik>
```

```
Ex:
formik-demo.jsx

import { Formik, Form, Field, ErrorMessage } from "formik";
import * as yup from "yup";

export function FormikDemo(){
    return(
        <div className="container-fluid">
            <h3>Register User</h3>
            <Formik initialValues={{UserName:'', Age:0, Mobile:''}}
validationSchema={yup.object({UserName:yup.string().required('Name Required'),
Age:yup.number().required('Age Required'), Mobile: yup.string().required('Mobile
Required').matches(/\+91\d{10}/,'Invalid Mobile')})} onSubmit={(values)=>{
console.log(values) }} >
                <Form>
                    <dl>
                        <dt>User Name</dt>
                        <dd> <Field type="text" name="UserName" /> </dd>
                        <dd className="text-danger"> <ErrorMessage
name="UserName" /> </dd>
                        <dt>Age</dt>
                        <dd> <Field type="number" name="Age" /> </dd>
                        <dd className="text-danger"> <ErrorMessage name="Age" />
</dd>
                        <dt>Mobile</dt>
                        <dd><Field type="text" name="Mobile" /></dd>
                        <dd className="text-danger"> <ErrorMessage name="Mobile"
/> </dd>
                    </dl>
                    <button type="submit">Register</button>
                </Form>
            </Formik>
        </div>
    )
}
```

- Formik library provides 2 validation states.

    a) Input State Validation
    b) Form State Validation

```
- Input state validation verifies every field individually.
- Form state validation verifies all field simultaneously at the same time.
- Form state allows to verify
    a) form values
    b) form state [ valid, invalid, pristine, dirty ]

Syntax:
        <Formik>
            {
            formik  => <Form> </Form>
            }
        </Formik>

        formik.isValid            all fields are valid
        formik.dirty             any one field modified
        formik.pristine          form is untouched
        formik.errors             returns all errors

Ex:
import { Formik, Form, Field, ErrorMessage } from "formik";
import * as yup from "yup";

export function FormikDemo(){
    return(
        <div className="container-fluid">
            <h3>Register User</h3>
            <Formik initialValues={{UserName:'', Age:0, Mobile:''}}
validationSchema={yup.object({UserName:yup.string().required('Name Required'),
Age:yup.number().required('Age Required'), Mobile: yup.string().required('Mobile
Required').matches(/\+91\d{10}/,'Invalid Mobile')})} onSubmit={(values)=>{
console.log(values) }} >
                {
                    form =>
                        <Form>
                        <dl>
                            <dt>User Name</dt>
                            <dd> <Field type="text" name="UserName" /> </dd>
                            <dd className="text-danger"> <ErrorMessage
name="UserName" /> </dd>
                            <dt>Age</dt>
                            <dd> <Field type="number" name="Age" /> </dd>
                            <dd className="text-danger"> <ErrorMessage name="Age"
/>  </dd>
                            <dt>Mobile</dt>
                            <dd><Field type="text" name="Mobile" /></dd>
                            <dd className="text-danger"> <ErrorMessage
name="Mobile" /> </dd>
                        </dl>
                        <button type="submit" className="me-2"
disabled={!form.isValid} >Register</button>
                        <button className={(form.dirty)?'d-inline':'d-none'}
>Save</button>
                        <h4>Check the following Errors</h4>
                         <ul>
```

```
                            {
                                Object.keys(form.errors).map(property=>
                                    <li key={property}> {form.errors[property]}
</li>
                                )
                            }
                         </ul>
                        </Form>
                }
            </Formik>
        </div>
    )
}
```

React Hook Form
----

https://react-hook-form.com/

1. To know about how to install and setup a library

        "Get Started"

2. Go to "API" of any library to explore everything provided by them.

3. Identify the required service for your project.

4. Explore details of required service

        - Get to know about the definition
        - Its features
        - Its syntax

5. Syntax provides all properties and methods required

        - now you can explore the purpose of every method and property
        - you must know about its type, return type and basic syntax

Implementing Hook Form:

Features:
- It provides various hooks to handle form and its state.
- It is light weight and faster.
- It uses all default HTML validations.
- It is loosely coupled and extensible.

Limitations:
- It completely depends on hooks.
- Hooks are allowed only in function component.
- Hence it is not good for class components.
- HTML validations are not enough for handling all types of validations.

1. Install Hook Form library

```
    >npm install react-hook-form --save
```

2. Hook Form library provides various hooks

```
          useForm()
          useController()
          useWatch()
          useFormContext()
          useFormState()
          useFieldArray()
          createFormControl() [beta]
```

3. You can configure a form and validations by using "useForm()".

```
        import  { useForm }  from  "react-hook-form";

            const { formName,  handleSubmit,  formState: { errors } } =
useForm();


        formName        : It refers to the form.
        handleSubmit        : It refers to the function to execute on submit
        formState        : It returns "errors" object, that contains field
errors.
```

4. Write a function for submit

```
        const  submit  = (values) =>{
            console.log(values);
        }

        <form  onSubmit={ handleSubmit(submit) }>

        </form>
```

5. You can access data from elements by using hook-form spread operator.

```
     <input type="text" name="Name"  {...formName("Name", { validations })}  />

      validations        : required, minLength, maxLength, pattern, email, url ..
```

6. Errors are handled by using "errors" object.

```
      {  errors.fieldName.type==="errorType" }    // Boolean expression
```

Note: FieldName returns undefined, as it is expecting validation to fireup at complile
time. To configure runtime validation, you have to make the field as nullable type.

The null reference character is "?".

```
      {  errors.fieldName?.type==="errortype" }
```

```
        fieldname?    => can be null or have error.

Ex:
hook-form-demo.jsx

import { useForm } from "react-hook-form";

export function HookFormDemo(){

    const {register, handleSubmit, formState:{errors}} = useForm();

    const submit = (values)=>{
        console.log(values);
    }

    return(
        <div className="container-fluid">
            <h2>Register User</h2>
            <form onSubmit={handleSubmit(submit)} >
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" name="UserName"
{...register("UserName", { required:true, minLength:4 })} /></dd>
                    <dd className="text-danger">
                        {(errors.UserName?.type==="required")?<span>User Name
Required</span>:<span></span> && (errors.UserName?.type==="minLength")?<span>Name
too short</span>:<span></span>}
                    </dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" name="Mobile" {...register("Mobile", {
required:true, pattern:/\+91\d{10}/ })} /></dd>
                    <dd className="text-danger">
                        {(errors.Mobile?.type==="required")?<span>Mobile
Required</span>:<span></span> && (errors.Mobile?.type==="pattern")?<span>Invalid
Mobile</span>:<span></span> }
                    </dd>
                </dl>
                <button type="submit">Register</button>
            </form>
        </div>
    )
}
```

Controlled Components

- Components are categorized into various types
    a) Uncontrolled
    b) Controlled
    c) Higher Order etc..
- Uncontrolled components are absolute components, with a specific design and functionality. They are independent and not dependent on other components in design.

- Controlled components are dependent on the values from other components. You can customize according to the requirements.

- Components can be controlled by using "Props" (Properties).

Syntax:
```
        function Component(props)
        {
            return(
                <div> </div>
            )
        }
```

- Every component "Props" is object type, with key and value collection

Syntax:
```
        function  Component(props)
        {
            return(
                <div>  { props.key } </div>
            )
        }
```

- Every key or property is a required property. You can ignore using optional implementation.

```
          <Component   key=value    key=value  />
```

Ex:
1. Add a new folder "controlled-components"

2. Add a new file into folder   "navbar.jsx"

```
export function Navbar(props){
    return(
        <nav className={`d-flex my-2 justify-content-between border border-1 p-2 
${props.Theme}`}>
            <div>
                <span className="fw-bold fs-4">{props.BrandTitle}</span>
            </div>
            <div>
                {
                    props.MenuItems.map(item=><span key={item} className="mx-
3">{item}</span>)
                }
            </div>
            <div className={`${props.showSignin}`}>
                <button className={`${props.btnTheme}`}>Sign In</button>
            </div>
        </nav>
    )
}
```

3. You can access from any component

hook-form-demo.jsx

```
import { useForm } from "react-hook-form";
import { Navbar } from "../../controlled-components/navbar";

export function HookFormDemo(){

    const {register, handleSubmit, formState:{errors}} = useForm();

    const submit = (values)=>{
        console.log(values);
    }

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

                <Navbar showSignin='d-none' btnTheme='btn btn-light' Theme="bg-
dark text-white" BrandTitle="Shopper."  MenuItems={['Home', 'Shop', 'Pages',
'Blog', 'Docs']} />
                <Navbar btnTheme='btn btn-warning' Theme="bg-primary text-white"
BrandTitle="Amazon" MenuItems={['Electronics', 'Fashion', 'Footwear']} />
            </header>
            <h2>Register User</h2>
            <form onSubmit={handleSubmit(submit)} >
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" name="UserName"
{...register("UserName", { required:true, minLength:4 })} /></dd>
                    <dd className="text-danger">
                        {(errors.UserName?.type==="required")?<span>User Name
Required</span>:<span></span> && (errors.UserName?.type==="minLength")?<span>Name
too short</span>:<span></span>}
                    </dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" name="Mobile" {...register("Mobile", {
required:true, pattern:/\+91\d{10}/ })} /></dd>
                    <dd className="text-danger">
                        {(errors.Mobile?.type==="required")?<span>Mobile
Required</span>:<span></span> && (errors.Mobile?.type==="pattern")?<span>Invalid
Mobile</span>:<span></span> }
                    </dd>
                </dl>
                <button type="submit">Register</button>
            </form>
        </div>
    )
}
```

Ex:
controlled-components

data-grid.jsx

```
export function DataGrid(props){
    return(
        <div>
            <table className={`table caption-top table-hover ${props.theme}`}>
                <caption>{props.caption}</caption>
                <thead>
                    <tr>
                        {
                            props.fields.map(field=><th key={field}>{field}
<button className="btn bi bi-sort-alpha-down"></button> </th>)
                        }
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        props.records.map(record=>
                            <tr key={record}>
                                {
                                    Object.keys(record).map(key=><td
key={key}>{record[key]}</td>)
                                }
                                <td>
                                    <button className="btn btn-warning bi bi-pen-
fill"></button>
                                    <button className="btn btn-danger bi bi-
trash-fill mx-2"></button>
                                </td>
                            </tr>
                        )
                    }
                </tbody>
                <tfoot>
                    <tr>
                        <td>
                            <ul className="pagination">
                                <li className="page-item"><a className="page-
link"> &laquo;</a></li>
                                <li className="page-item"><a className="page-
link"> 1</a></li>
                                <li className="page-item"><a className="page-
link"> 2</a></li>
                                <li className="page-item"><a className="page-
link"> 3</a></li>
                                <li className="page-item"><a className="page-
link"> &raquo;</a></li>
                            </ul>
                        </td>
                    </tr>
                </tfoot>
            </table>
        </div>
    )
}


components
custom-demo.jsx

import { useState } from "react";
import { DataGrid } from "../../controlled-components/data-grid";


export function CustomDemo(){


    const [products, setProducts] = useState([
        {Id:1, Name:'Samsung TV', Price:14300, Stock:'Available'},
        {Id:2, Name:'Mobile', Price:24300, Stock:'Out of Stock'},
        {Id:3, Name:'Watch', Price:4300, Stock:'Available'}
    ]);

    return(
        <div className="container-fluid">
            <h3>Grid-1</h3>
            <DataGrid  caption="Employee Details" records={[{FirstName:'Raj',
LastName:'Kumar', Designation:'Manager'},{FirstName:'Kiran', LastName:'Kumar',
Designation:'Clerk'}]} fields={['First Name', 'Last Name', 'Designation']} />
            <h3>Grid-2</h3>
            <DataGrid records={products} caption="Product Details" fields={['Id',
'Name', 'Price', 'Stock']} />
        </div>
    )
}
-------------------------------
```

Conditional Rendering

- Conditional Rendering is a technique where component can render different content according to the state and situation.
- You can use various decision making approaches to render different components.

Syntax: Uncontrolled component - conditional rendering

```
         if (condition )
         {
         setComponent(<name1 />);
         }
        else
         {
        setComponent(<name2 />);
         }
```

Ex:
conditional-demo.jsx

```
import { useState } from "react"
import { Weather } from "../weather/weather";
import { CarouselDemo } from "../carousel-demo/carousel-demo";

export function ConditionalDemo(){
    const [component, setComponent] = useState();


    function LoadComponent(cname){
        switch(cname){
            case "Weather":
            setComponent(<Weather />);
            break;
            case "SlideShow":
            setComponent(<CarouselDemo />);
            break;
        }
    }

    return(
        <div className="container-fluid">
            <h2>My Projects</h2>
            <button onClick={()=> LoadComponent('Weather')} className="btn btn-
primary mx-2">Weather API</button>
            <button onClick={()=> LoadComponent('SlideShow') } className="btn
btn-warning">Slide Show</button>
            <div className="mt-4">
                {
                    component
                }
            </div>
        </div>
    )
}
```

- A controlled component can render different content using props.
- Props and provide a property, which allows to render different content by using any decision making approach. [if, else, switch, case, default]

Syntax: Conditional Render with controlled component

```
    function ControlledComponent(props)
    {
         if (props.key === value )
         {
         return(
               <Layout1 />
         );
        }
        else
        {
        return(
              <Layout2 />
```

```
                );
            }
        }
```

Ex:
1. Go to controlled-components and add "dynamic-grid.jsx"

```
export function DynamicGrid(props){

     if(props.layout==="grid"){
        return(
            <table className="table caption-top table-hover">
                <caption>{props.caption}</caption>
                <thead>
                    <tr>
                        {
                            props.fields.map(field=><th key={field}>{field}</th>)
                        }
                    </tr>
                </thead>
                <tbody>
                    {
                        props.records.map(record=>
                        <tr key={record}>
                            {
                                Object.keys(record).map(key=><td
key={key}>{record[key]}</td>)
                            }
                        </tr>
                        )
                    }
                </tbody>
            </table>
        )
     } else {
        return(
            <div className="d-flex flex-wrap">
                {
                    props.records.map(record=>
                        <div className="card m-2 p-2" style={{width:'250px'}}>
                            <div className="card-header"
style={{height:'100px'}}>
                               <h3>
                                {
                                    record[Object.keys(record)[0]]
                                }
                                </h3>
                            </div>
                            <div className="card-body">
                                {
                                    Object.keys(record)[1]
                                }
                                :
                                {
                                    record[Object.keys(record)[1]]
                                }
                            </div>
                            <div className="card-footer">
                                <button className="btn btn-primary w-100"> View
Details </button>
                            </div>
                        </div>
                    )
                }
            </div>
        )
     }
}
```

2. Conditional-Demo.jsx

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

import { DynamicGrid } from "../../controlled-components/dynamic-grid";

export function ConditionalDemo(){

    const [products] = useState([{Name:'Samsung TV', Price:50000},
{Name:'Mobile', Price:12000}]);
    const [fields, setFields] = useState([]);
    const [layout, setLayout] = useState('');

    useEffect(()=>{

          setFields(Object.keys(products[0]));

    },[])

    function handleLayoutChange(e){
        setLayout(e.target.value);
    }

    return(
        <div className="container-fluid">
            <h2>Conditional Rendering</h2>
            <div className="my-3 w-25">
                <label className="fw-bold form-label">Select Layout</label>
                <div>
                <select onChange={handleLayoutChange} className="form-select">
                    <option>Select Layout</option>
                    <option value="grid">Grid</option>
                    <option value="card">Card</option>
                 </select>
                </div>
            </div>
            <DynamicGrid layout={layout} caption="Product Details"
fields={fields} records={products} />
        </div>
    )
}
-------------------------------------
```

Client Side State Management Techniques:
--

1. Session Storage
- It is temporary storage.
- It is memory allocated for a page in browser.
- It is deleted automatically when user closes browser tab or window.
- It is accessible only from the same tab where it is allocated.
- It is not accessible across tabs.

Syntax:

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

Ex:
conditional-demo.jsx

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



export function ConditionalDemo(){


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

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

    function handleSignInClick(){
        sessionStorage.setItem("username", userName);
        window.location.reload();

    }

    function handleSignoutClick(){
        sessionStorage.removeItem("username");
        window.location.reload();
    }


    useEffect(()=>{

    },[])
    return(
        <div className="container-fluid">
            <nav className="mt-4 p-2 border border-1 d-flex justify-content-
between">
                <div className="fs-3 fw-bold">YouTube</div>
                <div>
                   {
                     (sessionStorage.getItem("username")===null)
                     ?
                     <div>
                        <div className="input-group">
                            <input type="text" onChange={handleUserName}
placeholder="User Name" className="form-control" /> <button
onClick={handleSignInClick} className="btn btn-warning">Sign In</button>
                        </div>
                     </div>
                     :
                     <div className="fs-4 fw-bold bi bi-person-fill">
{sessionStorage.getItem("username")} <button className="btn btn-link mx-2"
onClick={handleSignoutClick}>Signout</button> </div>
                   }
                </div>
            </nav>
        </div>
    )
}
```

2. Local Storage
- It is permanent storage.
- It allocates memory for page, which is accessible across tabs.
- It is not deleted even when a browser closed.
- You have to delete explicitly from browser memory.

Syntax:
```
        localStorage.setItem("key", value);
                     .getItem("key");
                     .removeItem("key");
                     .clear();
```

Ex:
conditional-demo.jsx

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



export function ConditionalDemo(){


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

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

    function handleSignInClick(){
        localStorage.setItem("username", userName);
        window.location.reload();

    }

    function handleSignoutClick(){
        localStorage.removeItem("username");
        window.location.reload();
    }


    useEffect(()=>{

    },[])

    return(
        <div className="container-fluid">
            <nav className="mt-4 p-2 border border-1 d-flex justify-content-
between">
                <div className="fs-3 fw-bold">YouTube</div>
                <div>
                   {
                     (localStorage.getItem("username")===null)
                     ?
                     <div>
                        <div className="input-group">
                            <input type="text" onChange={handleUserName}
placeholder="User Name" className="form-control" /> <button
onClick={handleSignInClick} className="btn btn-warning">Sign In</button>
                        </div>
                     </div>
                     :
                     <div className="fs-4 fw-bold bi bi-person-fill">
{localStorage.getItem("username")} <button className="btn btn-link mx-2"
onClick={handleSignoutClick}>Signout</button> </div>
                   }
                </div>
            </nav>
        </div>
    )
}
```

3. Cookies
- Cookie is a simple text document.
- Client details can be stored in a cookie.
- Cookie is appended into browser memory or client HDD. (Hard Disk Drive)
- Hence cookies are classified into 2 types
    a) In-memory cookie
    b) Persistent cookie
- In Memory cookie is a temporary cookie. It is deleted when browser closed.
  It is accessible across tabs.

- Persistent cookie is a permanent cookie, You can set expiry date & time for cookie. So that it is deleted from memory automatically after expiry date & time.

- JavaScript can configure cookies by using "DOM" object "document".

```
        document.cookie = "name=value; expires=dateTime";
```

- React have "react-cookie" library that can manage cookies client side with virtual DOM.

- Cookie library is not a native library of React. It is a 3rd party library.

1. Install Cookie library

```
        > npm  install  react-cookie --save
```

2. Import Cookie Provider into "index.js" and configure for your startup component.

```
    import { CookiesProvider } from 'react-cookie';

    <CookiesProvider>
        <YourComponent />
    </CookiesProvider>
```

3. Configure and manipulate cookie by using "useCookies()" hook.

```
     import  { useCookies  }  from  "react-cookie";


    const [cookies, setCookie, removeCookies] = useCookies(['cookieName']);

    setCookie('cookieName', value);        // set a cookie
    removeCookies('cookieName');        // remove cookie from memory
    cookies['cookieName'];                // read cookie from memory
```

4. You can set persistent cookie by using expiry

```
    setCookie('cookieName', value, { expires : daysNumber });
```

State Management & Conditional Rendering

- Session Storage
- Local Storage
- Cookies [react-cookie]
- CookiesProvider
- useCookies()

FAQ: How to configure a persistent cookie?
Ans : Persistent cookie is a permanent cookie defined with expiry date & time.

```
Syntax:
        setCookie('name', value, { expires : new Date('yy-mm-dd hrs:min:sec') })


FAQ: How to configure in-memory cookie?
Ans:  If expires is not defined then the cookie is in-memory. [temporary]

Syntax:
        setCookie('name', value)

Ex:
1. Install  react-cookie

        > npm install react-cookie --save


2. Go to index.js

import { ConditionalDemo } from './components/conditional-demo/conditional-demo';
import { CookiesProvider } from 'react-cookie';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(

   <CookiesProvider>
        <ConditionalDemo/>
   </CookiesProvider>

);

3. Conditional-Demo.jsx

import { useEffect, useState } from "react";
import { useCookies } from "react-cookie";

export function ConditionalDemo(){


    const [userName, setUserName] = useState('');
    const [cookies, setCookie, removeCookie] = useCookies(['username']);

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

    function handleSignInClick(){

        setCookie('username', userName, { expires: new Date('2025-03-09') });

    }

    function handleSignoutClick(){
        removeCookie('username');
    }
    useEffect(()=>{

    },[])

    return(
        <div className="container-fluid">
            <nav className="mt-4 p-2 border border-1 d-flex justify-content-
between">
                <div className="fs-3 fw-bold">YouTube</div>
                    <div>
                    {
                        (cookies['username']===undefined)
                        ?
                        <div>
                            <div className="input-group">
                                <input type="text" onChange={handleUserName}
placeholder="User Name" className="form-control" /> <button
onClick={handleSignInClick} className="btn btn-warning">Sign In</button>
                            </div>
                        </div>
                        :
                        <div className="fs-4 fw-bold bi bi-person-fill">
{cookies['username']} <button className="btn btn-link mx-2"
onClick={handleSignoutClick}>Signout</button> </div>
                    }
                    </div>
            </nav>
        </div>
    )
}
```

Task:
- Create simple registration form with fields like
  UserName, Age, Mobile, City
- Collect the form data using Formik library
- Save in local storage.
- Read data from local storage and display in component UI.

Note: On submit click it must display status as saved.
      Add a view button, on view click it must display saved data.

Summary
- Function Components
- Components in React
- Nested Components
- Controlled Components
- Uncontrolled Components
- Data Binding
- Style Binding
- Class Binding
- Event Binding
- Conditional Rendering

- State Management
- Forms & Validations
- API

React Hooks

- Hook is a service in React.
- Service is a pre-defined business logic, which you can customize and implement according to the requirements.

```
        values, functions => factories => services => application
```

- Factory uses a "single call" mechanism, where an object is created for every request.
- Service uses a "single ton" mechanism, where an object is create for first request and the same is used across multiple requests.
- A service comprises of
    a) Consumer
    b) Provider
    c) Injector
- Consumer uses the service.
- Provider locates the service in memory.
- Injector injects the service into consumer location.
- Service uses "DI" mechanism, which is "Dependency Injection".
- It identifies the dependencies, locates them in memory and injects into component.
- React allows to create custom hooks, which are custom services.

Hook Rules:
- Hook must be a function.
- Its name must start with "use" prefix.
- Its name must be in camel case.

```
        useState
        useEffect
```

- It can't be void type function.
- It must be configured with return.
- It can be parameterized or parameter less.
- You can't use hooks in class components.
- Hook can't be defined in a closure.

Syntax:
```
    export function  useName(params)
    {
       return value;
    }
```

Ex:
1. Add a new folder into SRC by name "hooks"

2. Add a new file "captcha.jsx"

```
export function useCaptcha()
```

```
{
    var code = '';

    code = `${Math.round(Math.random()*10)} ${Math.round(Math.random()*10)}
${Math.round(Math.random()*10)} ${Math.round(Math.random()*10)}
${Math.round(Math.random()*10)} ${Math.round(Math.random()*10)}`;

    return code;
}
```

3. Register.jsx

```
import { useState } from "react";
import { useCaptcha } from "../../hooks/captcha";


export function Register()
{

    const [uname] = useState('John');
    const [title] = useState('User Register');

    let code = useCaptcha();

    return(
        <div className="d-flex justify-content-center align-items-center">
            <form className="mt-4 w-25 p-4 alert alert-warning alert-dismissible
border border-secondary rounded rounded-2">
                <h3 className="bi bi-person-fill"> {title} </h3>
                <button className="btn btn-close"></button>
                <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>Email</dt>
                    <dd><input type="email" className="form-control"  /></dd>
                    <dt>Mobile</dt>
                    <dd><input type="text" className="form-control"  /></dd>
                    <dt>Verify Code <button className="btn bi bi-arrow-clockwise
"></button> </dt>
                    <dd>{code}</dd>
                </dl>
                <button className="btn btn-warning w-100">Register</button>
            </form>
        </div>
    )
}
```

Ex: Custom Hook
    fetch-api.jsx

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

```
export function useFetch(url){
    const [data, setData] = useState([]);
    useEffect(()=>{
        fetch(url).then(response=> response.json()).then(items=> setData(items));
    },[url]);
    return data;
}


component.jsx

 let categories = useFetch('https://fakestoreapi.com/products/categories&#39;);
---------------------------

Hooks in React
- Rules
- Service
- Service Provider
- Service Consumer
- Injector
- Single ton
- DI
- Custom Hooks

Task:
1. Create a custom hook for converting and display text in Title Case.

            useTitleCase()

            useTitleCase('welcome to react');

        O/P:        Welcome To React

2. Create a custom hook for sorting the data.

            useSort(list, reverse:bool)

            useSort(["C","A", "B" ], true);        // C, B, A
            useSort(["C", "A", "B"]);            // A, B, C

FAQ: Can JavaScript function overload?
Ans:  No. JavaScript will not support function overloading.

        function add(a, b) {
            return a + b;
        }

        function add(a,b,c) {
            return a + b + c;
        }

        add(10, 20, 30);
        add(10, 20);        // c = undefined
```

Solution: You have to implement various technique to configure pseudo overload.

1. You can use "default" parameters in function.

Syntax:
```
        function Name(param=value)
        {
        }
        Name();        // default value
        Name(value);    // uses value
```

Ex:
```
<script>
    function Product(name="TV", price=4000){
       document.write(`Name=${name}<br>Price=${price}<br>`);
    }
    Product();
    Product("Samsung TV");
    Product("Mobile", 12000);
</script>
```

2. You can use JavaScript "arguments.length" property

Syntax:
```
        function Name(param1, param2,..)
        {
             if(arguments.length===0)
            {
            }
             else if(argument.length===1)
              {
              }
             else{
            }
        }
```
Ex:
```
<script>
    function Product(name, price){
        if(arguments.length===0){
           document.write(`Name and Price are expected<br>`);
        }else if(arguments.length===1){
           document.write(`Name=${name}<br>`);
        } else {
          document.write(`Name=${name}<br>Price=${price}<br>`);
        }
    }
    Product();
    Product("Mobile");
    Product("Samsung TV", 60000);
</script>
```

React Built-in Hooks

1. useEffect
- It is a hook used to configure actions to perform while mounting and unmounting component.

```
    Syntax:
        useEffect(()=>{

            // on mount

            return ()=>{
                // on unmount
             }

        },[dependencies])
```

- Every component mounts once for a request.
- If you want the component to mount again then you have to configure the dependencies.
- Dependencies is an array, hence you can have multiple dependencies.

FAQ: Can a component have multiple mount and unmount actions?
Ans: Yes.

FAQ: Can we have multiple useEffect in component?
Ans: Yes.

FAQ: Why you need multiple useEffect?
Ans: To handle conditional render.

----

useState

- It configures a local state for component.
- State allocates memory where you can store data and use it across requests.
- State can handle any type of data
    a) Primitive
    b) Non Primitive
- State is mutable.

```
Syntax:
    const [getter, setter] = useState(value);

    setter(newValue);        // storing value

    { getter }                // reading value
```

useContext

- Context is the memory allocated for a parent component, so that it is accessible to the child component that run within the context of parent.

- The child component at any level of hierarchy inside parent can access the context.

- Context is created explicitly by using "createContext()" method.

Syntax:
```
        let  contextName = createContext(null);

        null  => expecting data at runtime
```

- You can access and use context from any child element by using "useContext()" hookl.

Syntax:
```
        const   refName = useContext(contextName);
```

- Context provides a scope, so that child components within the context scope can access the context memory.

- Context scope is defined using DI mechanism.

Syntax:
```
        <ContextName   Provider={value}>

                // context scope   <Child1 />

        </ContextName>

         <Child2 />            => It can't access context memory
```

- Provider is a member of context service it uses the value and injects into the
  child component.

Syntax:
```
        <ContextName.Provider    value={ }>

             <Child />

        </ContextName.Provider>
```

Ex:
context-demo.jsx

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

let userContext = createContext(null);

export function Level1(){

    const username = useContext(userContext);

    return(
        <div className="m-4 p-4 bg-warning">
            <h4>Level-1 - {username} </h4>
            <Level2 />
        </div>
    )
}

export function Level2(){

    const username = useContext(userContext);

    return(
        <div className="m-4 p-4 bg-danger">
            <h4>Level-2 - {username}</h4>
        </div>
    )
}


export function ContextDemo(){
    const [user, setUser] = useState('');

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

    return(
        <div className="container-fluid p-4 bg-dark text-white">
            <h2>Parent Component - <input type="text" onChange={handleNameChange}
placeholder="User Name" /></h2>
            <userContext.Provider value={user}>
               <Level1 />
            </userContext.Provider>
        </div>
    )
}
```

Ex: Amazon.jsx

```
import axios from "axios";
import { createContext, useContext, useEffect, useState } from "react"

let SearchContext = createContext(null);


export function MainComponent(){


    const [products, setProducts] = useState([{id:0, title:'', image:''}]);

    let searchTerm = useContext(SearchContext);

    useEffect(()=>{
        if(searchTerm===""){
            axios.get(`https://fakestoreapi.com/products`)
            .then(response=>{
                 setProducts(response.data);
            })
        } else {
            axios.get(`https://fakestoreapi.com/products/category/${searchTerm}`)
            .then(response=>{
                 setProducts(response.data);
            })
        }
    },[searchTerm])

    return(
        <div className="p-4">
            <div className="d-flex flex-wrap">
                {
                    products.map(product=>
                        <div key={product.id} className="card p-2 m-2"
style={{width:'100px'}}>
                            <img src={product.image} height="100" />
                        </div>
                    )
                }
            </div>
        </div>
    )
}


export function Amazon(){

    const [searchTerm, setSearchTerm] = useState('');
    const [search, setSearch] = useState('');

    function handleSearchChange(e){
        setSearchTerm(e.target.value)
    }
    function handleSearchClick(){
        setSearch(searchTerm);
    }

    return(
        <div className="container-fluid">
             <nav className="border d-flex justify-content-between   border-1 p-
2">
                <h2>Amazon</h2>
                <div>
                    <div className="input-group">
                        <input type="text" onChange={handleSearchChange}
className="form-control" placeholder="Search Amazon.in" />
                        <button onClick={handleSearchClick} className="btn btn-
warning bi bi-search"></button>
                    </div>
                </div>
                <div>
                    <button className="btn btn-warning bi bi-person-fill"> Sign
In </button>
```

```
                </div>
             </nav>
             <section className="mt-4">
                 <SearchContext.Provider value={search}>
                     <MainComponent />
                 </SearchContext.Provider>
             </section>
        </div>
    )
}
```

----

 useReducer

- Every web application uses a "global state", which is known as application state.
- It allows to store data and make it available from application start to end.
- Application state is accessible to all sessions.
- It is difficult to debug and predict application memory.
- "useReducer" hook provides access to application memory, which is predictable and debuggable.
- You can use JavaScript library "Redux" to configure and implement global state in large scale applications.
- Reducer requires following components
    a) Store
    b) State
    c) UI
    d) Reducer

- Store is the location where data is kept.
- State allows to access data from store and display in UI.
- UI refers to component user interface where we use the global data.
- Reducer identifies the actions performed on data and updates into store.

Implementation:
1. Create initial state that refers to "store".

```
       let  initialState = {
            key: value,
            key: value [any type]
       }
```

2. Create a reducer function to identify the action and update the data.

```
      function  reducer(state, action)
      {
        switch(action.type)
        {
          case "type1":
            actions;  // updating the data into store
            break;
        }
      }
```

3. Components use the reducer function by using "userReducer()" hook.

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

    dispatch({type: 'actionName'})
```

Ex:
reducer-demo.jsx

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

let initialState = {
    ViewersCount: 0,
}

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


export function ReducerDemo(){


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

    function JoinClick(){
        dispatch({type: 'join'});
    }
    function ExitClick(){
        dispatch({type:'exit'});
    }

    return(
        <div className="container-fluid">
            <div className="card mt-4 w-50">
                <div className="card-header">
                    <iframe src="https://www.youtube.com/embed/uspNXYdKEh8&quot;
width="100%" height="300"></iframe>
                </div>
                <div className="card-body text-center">
                    <h4>Core Java Streaming Live..</h4>
                    <h5> <span className="bi bi-eye"></span>
[{state.ViewersCount}] Viewers</h5>
                </div>
                <div className="card-footer text-center">
```

```
                    <button onClick={JoinClick} className="bi bi-door-open btn
btn-warning"> Join </button>
                    <button onClick={ExitClick} className="bi bi-door-closed btn
btn-danger mx-2">Exit</button>
                </div>
            </div>
        </div>
    )
}
```

Summary
- useState
- useContext
- userReducer
- useMemo
- useRef
- useCallback
----
   useRef()

- It configures a reference memory where you can store a value or function.
- The reference memory allocates thread for value or task.
- Thread is intended to run the task or keep value in background process.
- The thread function or value is not intended for rendering in UI.

Syntax:
```
        const  thread = useRef(null);

        thread.current = value;
        thread.current = () =>{ }
```

- You can access with reference of "current" property.

FAQ: What are the techniques required for handling interactions faster that normal?
Ans: a) You can use JavaScript promises.
      b) You can implement async and await for functions.

FAQ: What is difference between callback and promise?
Ans: Callback uses "Synchronous" technique to handle interactions.
      Promise uses "Asynchronous" technique.

FAQ: What is sync?
Ans: It is a blocking technique where all other tasks in process are blocked while
     performing the given task.

Ex: Callback
```
<script>
    function FetchData(url, success, failure){
          if(url==="fakestore"){
              success([{Name:'TV', Price:13000}, {Name:'Mobile', Price:12000}]);
          } else {
              failure('Unable to Fetch - Invalid URL');
          }
    }
    FetchData(
       prompt("Enter URL"),
       function(response){
         console.log(response);
       },
       function(error){
          console.log(error);
       }
    )
</script>
```

FAQ: What is async?
Ans: It is an unblocking technique that performs all task simultaneously at the same
     time. It allows to run the task in background.

Ex: Promise

```
<script>
     var FetchData = new Promise(function(resolve, reject){
           var url = prompt("Enter Url");
           if(url==="fakestore"){
               resolve([{Name:'TV', Price:23000}, {Name:'Mobile', Price:12000}]);
           } else {
               reject('Invalid URL - Unable to Fetch Data');
           }
     })
     FetchData.then(function(response){
         console.log(response);
     })
     .catch(function(error){
         console.log(error);
     })
     .finally(function(){
         console.log('Request End');
     })
</script>
```

Ex: Promise

```
<script>
  var FetchData = new Promise(function(resolve, reject){
        var url = prompt("Enter Url");
        if(url==="fakestore"){
            resolve([{Name:'TV', Price:23000}, {Name:'Mobile', Price:12000}]);
        } else {
            reject('Invalid URL - Unable to Fetch Data');
        }
  })
  FetchData
  .then(function(response){
      var now = new Date();
      console.log(now.getMilliseconds());
      console.log(response);

  })
  .then(function(){
       var now = new Date();
      console.log(now.getMilliseconds());
      console.log('Data is ready to display');
  })
  .then(function(){
      var now = new Date();
      console.log(now.getMilliseconds());
      console.log('Data displayed successfully..');
  })
  .catch(function(error){
      console.log(error);
  })
  .finally(function(){
      console.log('Request End');
  })
</script>
```

Ex: async & await

```
<script>

    async function GetData()
    {
        return await [{Name:'TV'},{Name:'Mobile'}];
    }


    GetData().then(function(data){
        console.log(data);
    })

</script>
```

Ex: Async & Await

```
<script>
    fetch("https://fakestoreapi.com/products&quot;)
    .then(async function(response){
        return await response.json();
    })
    .then(function(products){
         products.map(async function(product){
              await document.write(product.title + "<br>");
         })
    })
</script>
```

Ex: Weather API - Async & Await

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


export function Weather(){


    const url = 'https://api.openweathermap.org/data/2.5/weather&#39;;
    const api_key = '1318ca6725c69160d346c41fc0612596';

    const [cityName, setCityName] = useState('');
    const [weatherData, setWeatherData] = useState({name:'', main:{temp:0},
weather:[{description:''}]});


    async function handleCityChange(e){
        setCityName(await e.target.value);
    }

    function handleSearchClick(){

        // axios.get(`url?q=${cityName}&appid=${api_key}`);

        axios.get(url, {params:{
             q: cityName,
             appid: api_key,
             units:'metric'
        }})
        .then(async response=>{
            setWeatherData(await response.data);
            console.log(response.data);
        })
    }

    return(
        <div className="container-fluid">
            <div className="mt-4 d-flex justify-content-center">
                <div className="w-50">
                    <div className="input-group">
                        <input type="text" onChange={handleCityChange}
placeholder="Enter City Name" className="form-control" />
                        <button onClick={handleSearchClick} className="bi bi-
search btn btn-warning"></button>
                    </div>
                    <div style={{marginTop:'50px', boxShadow:'2px 2px 2px black',
padding:'20px', border:'1px solid black', textAlign:'center',
backgroundImage:`url(${(weatherData.weather[0].description==='mist')?'mist.jpg':'
smoke.jpg'})` , color:'white', backgroundSize:'cover'}}>
                        <h2>{weatherData.name} -
{weatherData.weather[0].description.toUpperCase()} </h2>
                        <p className="fs-4">{Math.round(weatherData.main.temp)}
&deg; C <span className="bi bi-sun"></span> </p>
                    </div>
                </div>
            </div>
        </div>
    )
}
-----------------
```

useMemo
- It is used to memorize any value so that it can save round trip.
- It can cache the value and avoid re-rendering of value until it is changed.

Syntax:
```
        const cachedData = useMemo(()=> {return data}, [dependency])
```

useCallback
- It is similar to memo, but caches a function instead of value.
- It can avoid re-rendering of component until the dependency changes.

Syntax:
```
         useCallback(()=>{

         }, [dependency])
```

Routing in React
- Routing a technique used in web applications to configure user and SEO friendly URL's.
- User friendly URL allows to query any content directly through the URL in address bar.
- SEO friendly URL allows the web-crawlers to known the exact location on page.
- Routing enables navigation in SPA. [Single Page Application]
- New details are added to page without reloading the page.
- User can stay on one page and get access to everything from the page.
- Routing can be configured
    a) Server Side
    b) Client Side
- Server side routes are mostly used for API's. You can create and configure API end points using server side routes.

Creating Server Side API Routes using JSON Server:

1. Install JSON server on your device from command prompt

```
        C:\>npm install  -g  json-server
```

2. Add a new file into your project by name   "db.json" [public folder]

3. Add data into "db.json"

```
      {
         "key": [  { }, { } ],
         "key": [  { }, { } ]
      }
```

    Note: "key" is considered as API end point.

Ex:
    db.json

```
    {
    "users":[
        {
            "id": 1,
            "userid":"john_nit",
            "password":"john@nit",
            "email":"john@gmail.com"
        },
        {
            "id": 2,
            "userid":"david",
            "password":"david11",
            "email":"david@outlook.com"
        }
    ],
    "appointments":[
        {
            "id":1,
            "title":"Friend Birthday",
            "date": "2025-03-22",
            "userid":"john_nit"
        },
        {
            "id":2,
            "title":"Submit Project Document",
            "date": "2025-03-18",
            "userid":"david"
        }
    ]
}
```

4. Change to the public folder where you kept db.json and run the following command

```
        >json-server  db.json  --watch
```

5. Server starts by creating API end-points

```
        http://localhost:3000/users
        http://localhost:3000/appointments
```

6. The server side routes are configured automatically

```
    GET            /users            returns all users
    GET            /users/1            returns user by id
    POST        /users            adds new user
    PUT            /users/1            modify specific user
    DELETE        /users/1            delete specific user
    GET             /appointment        return all appointments
    ... similar..  for other requests

    > json-server   db.json  --watch   --port  4040
```

Ex:
todo.jsx

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


export function ToDO(){

    const [appointments, setAppointments] = useState([]);

    useEffect(()=>{

        axios.get(`http://localhost:4040/appointments`)
        .then(response=>{
            setAppointments(response.data);
        })

    },[])

    return(
        <div className="container-fluid">
            <h3>Your Appointments</h3>
            <table className="table table-hover">
                <thead>
                    <tr>
                        <th>Title</th>
                        <th>Date</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        appointments.map(appointment=>
                        <tr key={appointment.id}>
                            <td>{appointment.title}</td>
                            <td>{appointment.date}</td>
                            <td>
                                <button className="btn btn-warning bi bi-
pen"></button>
                                <button className="btn btn-danger mx-2 bi bi-
trash"></button>
                            </td>
                        </tr>
                        )
                    }
                </tbody>
            </table>
        </div>
    )
}
------------------------------------------------------

Client Side Routing

- React uses "react-router-dom" library for configuring client side routes.
- Router DOM library is not a native library of react.
- The popular Router DOM versions

        V5              up to React 17
        V6              React 18
        V7              React 18 & 19

- V6 & V7 are complete re-write of router-dom.

1. Install React Router DOM library for project

    >npm install react-router-dom --save
    >npm install react-router-dom@6.30.0  --save   [old version]
    >npm install react-router-dom@latest --save

2. Router DOM provides following components

    <BrowserRouter>
    <Routes>
    <Route>
    <Outlet>
    <Link>

   - BrowserRouter is responsible for configuring virtual DOM routes and
translate into
     actual DOM. React routing for SPA must be within the scope of BrowserRouter.

      Syntax:
            <BrowserRouter>

                .... all your route configurations ...

            </BrowserRouter>

   - Routes is a collection of individual routes defined for application. It
configures
     a router table.

Syntax:

        <BrowserRouter>

            <Routes>
                ... all routes ...
            </Routes>
        </BrowserRouter>

    - Route component is responsible for configuring individual route. A route
defines
      the request path and component to render.

Syntax:
        <BrowserRouter>
            <Routes>
                <Route  path=""   element={ } />
                    <Route  path=""   element={ } />
            </Routes>
        </BrowserRouter>

       - Link is a component used to configure a hyperlink that navigates user to
specified
     route path.

Syntax:
        <Link  to="home">  Home </Link>
        <Link  to={route_path_name}>  Text | Image </Link>

    - Outlet is a component that defines the location in page where the resulting
      markup must render. It is mostly used in nested and child routes.

Syntax:
        <main>
            <Outlet />
        </main>

Wild Card Routes:
- Application sends the request content as response.
- If client request is not specific or invalid then you can send a customized
response using wild card routes.

     path="/"         It refers to component that will render when user is not
                 requesting any specific. [default render]

    path="*"        It refers to component that renders when requested path not
                found.
Ex:

app.js

import logo from './logo.svg';
import './App.css';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import { Login } from './login';
import { Weather } from './components/weather/weather';

function App() {
  return (
    <div className="App">
        <BrowserRouter>
```

```
          <header className='d-flex border border-2 border-secondary my-2 mx-2
justify-content-between p-3'>
             <div className='fs-4 fw-bold'>Shopper.</div>
             <nav className='d-flex btn-group bg-primary align-items-center'>
                <span> <Link className='btn btn-primary' to='home'>Home</Link>
</span>
                <span className='mx-3'> <Link className='btn btn-primary'
to='men'>Mens' Fashion</Link> </span>
                <span> <Link className='btn btn-primary' to='women'>Women
Fashion</Link> </span>
                <span className='mx-3'> <Link className='btn btn-primary'
to='kids'>Kids Fashion</Link> </span>
             </nav>
             <div>
                <span> <Link to='weather'><span className='bi bi-
clouds'></span></Link> </span>
                <span> <Link to='login'><span className='bi bi-person-fill mx-
3'></span></Link> </span>
                <span  className='bi bi-cart4'></span>
             </div>
          </header>
          <section className='mt-4 p-4'>
              <Routes>
                   <Route path='/' element={<div><h4>Online Shopping</h4><p>30%
OFF on all fashion accessories</p></div>} />
                   <Route path='home' element={<div><h4>Online
Shopping</h4><p>30% OFF on all fashion accessories</p></div>} />
                   <Route path='men' element={<div><h4>Men's Fashion</h4><img
src='men-fashion.jpg' width='200' height='300'/></div>} />
                   <Route path='women' element={<div><h4>Women's Fashion</h4><img
src='women-fashion.jpg' width='200' height='300'/></div>} />
                   <Route path='kids' element={<div><h4>Kid's Fashion</h4><img
src='kids-fashion.jpg' width='200' height='300'/></div>} />
                   <Route path='login' element={<Login/>} />
                   <Route path='weather' element={<Weather />} />
                   <Route path='*' element={<div><h4 className='text-danger'>Not
Found</h4><code className='text-warning'>The requested path Not
found</code></div>} />
              </Routes>
          </section>
        </BrowserRouter>
    </div>
  );
}

export default App;
```

Note: In index.js set <App/> as startup

Route Parameters

- Web application uses "http" as protocol.
- Http is a state less protocol.

- In multipage applications data can be transported from one page to another by using query string.
- Query String is a key & value collection appended into URL and stored in browser address bar.

Syntax:
```
  https://www.amazon.in/products.asp?category=electronics&product=mobiles
```

- Query string is appended using "?" character.
- Multiple keys are appended using "&" character.
- Query String is complex and not much SEO friendly or user friendly.
- In modern web applications you can use "Route" parameters.
- A route parameter is configured in route path.

Syntax:
```
    <Route  path="products/:category/:product"  element={}>
```

- The parameter values are defined as a part of URL

Syntax:
```
    http://localhost:3000/products/electronics/mobiles
```

- The route parameters can be accessed from URL by using "useParams()" hook.

Syntax:
```
        let  params = useParams();
```

- "params" reference is an object with key and value.

Syntax:
```
          {  params.keyName }
          {  params.category }        // electronics
          {  params.product }         // mobiles
```

Ex:
1. Add a new component   "params-demo.jsx"

```
import { useParams } from "react-router-dom";

export function ParamsDemo(){

    let params = useParams();

    return(
        <div>
            <h3>Products</h3>
            You are searching for {params.brand} related {params.product} in
{params.category} category.
        </div>
    )
}
```

2. Go to app.js and add new route

```
    <Routes>
        ....
       <Route path='products/:category/:product/:brand' element={<ParamsDemo />}
/>

    </Routes>
```

3. Make the following request from URL

```
    http://localhost:3000/products/electronics/mobiles/apple
```

Ex: Fakestore

1. Add a new folder "fakestore" into SRC

2. Add following components

fakestore-home.jsx

```
import axios from "axios";
import { useEffect, useState } from "react"
import { Link } from "react-router-dom";


export function FakestoreHome(){

    const [categories, setCategories] = useState([]);

    useEffect(()=>{

        axios.get(`https://fakestoreapi.com/products/categories`)
        .then(response=>{
             setCategories(response.data);
        })

    },[])

    return(
        <div>
            <h5>Fakestore Home</h5>
            <ul className="list-unstyled">
                {
                    categories.map(category=><li className="my-3 p-2"
key={category}> <Link className="btn btn-dark w-25"
to={`products/${category}`}>{category.toUpperCase()}</Link> </li>)
                }
            </ul>
        </div>
    )
}
```

fakestore-products.jsx

```
import axios from "axios";
import { useEffect, useState } from "react";
import { Link, useParams } from "react-router-dom";



export function FakestoreProducts(){


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

    let params = useParams();

    useEffect(()=>{
        axios.get(`http://fakestoreapi.com/products/category/${params.category}`)
        .then(response=>{
             setProducts(response.data);
        })
    },[])

    return(
        <div>
            <h3>Fakestore Products</h3>
            <div className="d-flex flex-wrap">
                {
                    products.map(product=>
                        <div key={product.id} className="card m-2 p-2"
style={{width:'150px'}}>
                            <div className="card-header">
                                <img height="100" src={product.image}
className="card-img-top" />
                            </div>
                            <div className="card-footer">
                                <Link to={`/details/${product.id}`}
className="btn btn-primary bi bi-eye-fill"> Details</Link>
                            </div>
                        </div>
                    )
                }
            </div>
            <Link to="/">Back to Categories</Link>
        </div>
    )
}
```

fakestore-details.jsx

```
import axios from "axios";
import { useEffect, useState } from "react"
import { Link, useParams } from "react-router-dom";
```

```
export function FakestoreDetails(){


    const [product, setProduct] = useState({id:0, title:'', category:'', price:0,
image:'', rating:{rate:0, count:0}, description:''});

    let params = useParams();

    useEffect(()=>{

        axios.get(`https://fakestoreapi.com/products/${params.id}`)
        .then(response=>{
            setProduct(response.data);
        })

    },[])

    return(
        <div>
            <h3>Product Details</h3>
            <dl className="w-25">
                <img src={product.image} height="200" />
                <dt>Title</dt>
                <dd>{product.title}</dd>
                <dt>Price</dt>
                <dd>{product.price}</dd>
            </dl>
            <Link to={`/products/${product.category}`} >Back to Products</Link>
        </div>
    )
}
```

app.js

```
import logo from './logo.svg';
import './App.css';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { FakestoreHome } from './fakestore/fakestore-home';
import { FakestoreProducts } from './fakestore/fakestore-products';
import { FakestoreDetails } from './fakestore/fakestore-details';

function App() {
  return (
    <div className="App">
        <BrowserRouter>
          <header className='text-center p-2 border border-2 mt-2'>
            <h3>Fakestore</h3>
          </header>
          <section className='mt-4'>
              <Routes>
                   <Route path="/" element={<FakestoreHome />} />
                   <Route path='products/:category' element={<FakestoreProducts
/>} />
                   <Route path='details/:id' element={<FakestoreDetails />} />
              </Routes>
          </section>

        </BrowserRouter>
    </div>
  );
}

export default App;
-
 Child Routes
------------------

- You can configure nested routes with parent and child hierarchy.
- Route container can be defined with nested routes.

Syntax:
    <Route  path="parent"  element={<ParentComponent />}>

        <Route  path="child" element={<ChildComponent} />

    </Route>

- Child Route requires an outlet, which defines the location to render target
content.

Syntax:
      <ParentContainer>

            <Outlet />

     </ParentContainer>

FAQ's:
1. What is difference between Absolute & Relative path?
A. Absolute path is independent and individual in     reference. It is not
effected by existing route path, as it removes the current context and add a new
path.

        Existing Path:
             http://localhost:3000/products/category/jewelery

             <Link to="/details/2" />                  => Absolute

            http://localhost:3000/details/2

          Existing Path
            http://localhost:3000/products/category/jewelery

            <Link to="details/2" />                => Relative
            http://localhost:3000/products/category/jewelery/details/2
```

2. Can we define multiple Route outlets?
A. Yes. Resulting markup is required in multiple locations in SPA. Hence multiple outlets are allowed.

3. How to configure dynamic navigation in Routes?
4. By using useNavigate() hook.

Syntax:
```
     let navigate  = useNavigate();        // import from react-router-dom

    navigate("/path/params");
```

4. What are Search Parameters? How to access and use Search Params?
A. Search parameter refers to Query String.
    You can add a query string along with route parameters.
    Search parameter allows to transport more data from one component to another.
    It submits form data as query string.

Syntax:
```
        ?name=value&key=value

        let  [searchparams] = useSearchParams();

        { searchparams.get('name') }
```

5. Can we configure both route and search parameters in URL?
A. Yes.

```
        useParams()            // for route parameters
        useSearchParams()    // for query string
```

Syntax:
```
       http://localhost:3000/products/mobiles?brand=apple

        products/mobiles        => route params
        ?brand=apple            => search params
```

6. What is the DOM method for accessing search parameters?
A. URLSearchParams()

Syntax:
```
        URLSearchParams(location.search);
```

7. What is difference between useSearchParams() & URLSearchParams()?
A. useSearchParams is Virtual DOM method.
     URLSearchParams is actual DOM method.

Route Parameters
Search Parameters

Child Routes
Dynamic Navigation

To-Do Application

- User can register
- User can login with registered account
- User dash board shows all appointments after login
- User can add a new appointment
- User can edit existing appointments
- User can delete any appointment

Software
- JSON Server for managing data locally.
- React for designing UI.
- React Material UI for designing components.

React MUI
(Material UI)

- It is a toolkit for React applications.
- It provides templates, components and designs that allows developer to build interactive and responsive UI faster, when compared to the traditional approach.
- MUI provides several products like
    a) MUI Core
    b) MUI X
    c) Templates etc.
- MUI Core is free while others are premium services.

MUI Core

- It is a component library.
- It provides built-in components for handling various interactions.
- MUI latest version is a complete re-write of previous versions.
- The latest version of MUI uses "@emotion" library.

Setup MUI for React App:
1. Install latest MUI core library along with emotion library

```
    > npm install @mui/material @emotion/react @emotion/styled  --save
```

2. Every component is designed as "Controlled" component. You can customize by using "Props" [properties].

```
    import  { component }  from  "@mui/material";

                 (or)

     import   component   from  "@mui/material/component-module";
```

3. Implement component with props

```
     <component   property={value} />
```

Ex:

```
    import   {  Button  }  from  "@mui/material";

    <Button  variant="contained" color="error">  Login  </Button>
```

Note: All event binding techniques are same as you define for HTML elements.

Ex:
mui-demo.jsx

```
import { Button, TextField } from "@mui/material";
import { useState } from "react";

export function MUIDemo()
{

    const [username, setUserName] = useState('');

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

    function handleLoginClick(){
        console.log(username);
    }

    return(
        <div className="container-fluid">
             <h2>Bootstrap Design</h2>
               <div className="w-25">
                   <label className="form-label">User Name</label>
                   <div>
                      <input type="text" placeholder="Your name" className="form-
control" />
                   </div>
                   <button className="btn btn-danger w-100 mt-3">Login</button>
               </div>
               <br /> <br/>
             <h2>MUI Design</h2>
             <div className="w-25">
                 <div>
                 <TextField onChange={handleNameChange} type="text" label="User
Name" className="w-100" variant="standard" placeholder="Enter your name"
></TextField>
                 </div>
                 <div>
                  <TextField type="password" label="Password" className="w-100"
variant="standard" placeholder="Your password" ></TextField>
                 </div>
                 <div className="mt-4">
```

```
                   <Button onClick={handleLoginClick} variant="contained"
color="error" className="w-100" > Login </Button>
                </div>
            </div>
        </div>
    )
}
```

Task:
1. alert
2. dialog
3. card

To-Do Application

1. Configure data for application in db.json [public folder]

```
     {
       "users": [
               { "id" : 1,  "userid": ' ',  "password": ' ', "email": ' ' },
               {                                       }
             ],
       "appointments": [
                { "id": 1, "title": ' ',  "date": ' ', "description":' ',
"userid": ' ' }
             ]
      }
```

2. JSON Server creates API routes for your data

```
    GET                /users                 return all users
    GET                /users/id                 return specific user using id
    POST            /users                 add new user into data source
    PUT                /users/id                 edit existing user & update data
    DELETE            /users/id                removes user using id

    GET                /appointments
    GET                /appointments/id
    POST            /appointments
    PUT                /appointments/id
    DELETE             /appointments/id
```

3. Start your API

```
    >json-server  db.json  --watch  --port=4000

    http://127.0.0.1:4000/users        => GET / POST
```

Task for TO-DO Application:

- Setup Validation for Register & Login
    - All fields are required

    - Password must have at least one uppercase letter, number & special character
    - Email must be in proper email format.
- Design UI for login & register with MUI.
- Setup Validation for Add Appointment
- Appointment Date must use MUI date picker.
- Design user dash board with MUI.

React Class Components

JavaScript Class Concepts:
1. Class Declaration & Expression
2. Class Members
    - Property
    - Accessor
    - Method
    - Constructor
3. Class static and non-static members
4. Class private members [ES6 - # for private] [WeakMap]
5. Class Inheritance & Polymorphism

FAQ: What is issue with OOP?
Ans:
    - It will not support low level features.
    - It can't directly interact with hardware services.
    - It uses more memory.
    - It is tedious.

Creating Class Component:

1. You can configure class using declaration or expression.

2. Class name must start with uppercase letter.

3. A component class must inherit "React.Component" or "React.PureComponent" base class.

Syntax:
```
        export class Login extends React.Component | React.PureComponent
        {

        }
```

FAQ: What is Pure Component?
Ans: Pure component mounts only when there are changes in content.
      Impure Component mounts every time even when there are no changes in content.

4. The component class is a derived class hence it must call the super constructor.

Syntax:
```
      export class Login extends  React.Component
      {
```

```
               constructor()
           {
              super();
           }
       }
```

5. A component must return JSX element. React class component returns JSX element by using "render()" method.

Syntax:
```
    export class Login extends React.Component
    {
          constructor(){
          super();
          }
          render() {
         return(
               <div>
                // JSX
               </div>
              );
           }
    }
```

Ex:
admin-login.jsx

```
import React from "react";

export class AdminLogin extends React.Component
{
      constructor(){
         super();
      }
      render(){
        return(
            <div className="container-fluid">
                <h3 className="mt-3">Admin Login</h3>
            </div>
        )
      }
}
```

State in Class Component:
- Class will not support hooks.
- Class is stateful.
- It have a state implicitly.
- You can configure state inside constructor using "this" keyword.
- State is an object type with key and value reference.

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

- State comprises of keys that can handle any type of value.
- Data Binding is same as in function component.

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

Ex:
admin-login.jsx

```
import React from "react";

export class AdminLogin extends React.Component
{
      constructor(){
         super();
         this.state = {
            title: 'Admin Login',
            categories: ['All', 'Electronics', 'Fashion']
         }
      }
      render(){
        return(
            <div className="container-fluid">
                <h3 className="mt-3">{this.state.title}</h3>
                <select>
                    {
                        this.state.categories.map(category=><option
key={category}>{category}</option>)
                    }
                </select>
            </div>
        )
      }
}
```

Component Mount:
- Class component have "componentDidMount()" method.
- It can define actions to perform at time of mounting component.
- You can't use "useEffect()" hook.
- You can defined unmount actions by using the method "componentWillUnmount()".
- You can set a new value into state while mounting or on any event by using "setState()".

Syntax:

```
    componentDidMount()
    {
        this.setState({ key : newValue });
    }
```

Ex:

```
import axios from "axios";
import React from "react";


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

      LoadCategories(){
         axios.get('https://fakestoreapi.com/products/categories&#39;)
         .then(response=>{
             this.setState({categories: response.data});
         })
      }

      LoadProducts(){
         axios.get(`https://fakestoreapi.com/products`)
         .then(response=>{
             this.setState({products: response.data});
         })
      }

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


      render(){
        return(
            <div className="container-fluid">
                <h3>Categories</h3>
                <select>
                    {
                        this.state.categories.map(category=> <option
key={category}>{category}</option>)
                    }
                </select>
                <div className="mt-3 d-flex flex-wrap">
                    {
                        this.state.products.map(product=><img key={product.id}
src={product.image} width="100"  height="100"/>)
                    }
                </div>
            </div>
        )
      }
}
```

```
---------------------------------------
Class Components
----------------------------

- Pure Component
- React Component
- Render Method
- State in Class Component
    this.state = { }
    this.setState({ })
- Component Did Mount
- Style Binding
    <div  style={ {key:value } }>
- Class Binding
    <div className={ }>
- Data Binding
    <p> { } </p>
                     Events in Class Component
- Events are subscribed to methods in class.
- All Synthetic Events are same as you have used in Function Component.

Syntax:
        class  Login extends  React.Component
        {
              ....
              handleLoginClick()
              {
              }
              render() {
               return(
                 <div>
                    <button onClick={this.handleLoginClick}> </button>
                 </div>
                )
               }
        }

- Event arguments are defined using "event" parameter, which provides access to
element and event details.

        handleLoginClick(e)
        {
            e.clientX;
            e.target.id;
            e.target.name;
            ....
        }

Ex:
 admin-login.jsx

import axios from "axios";
import React from "react";
```

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

      handleLoginClick(e){
         alert('Login Success');
         console.log(`Id=${e.target.id}\nName=${e.target.name}\nX=${e.clientX}`);
      }

      render(){
        return(
            <div className="container-fluid">
                <h2>Events</h2>
                <button onClick={this.handleLoginClick} id="btnLogin"
name="Login-Button">Login</button>
            </div>
        )
      }
}
```

Note: Events configured in class are subscribed to methods in class.
      If methods are using state then it is mandatory to register method and bind
to
      class while creating instance of component class.

Syntax:

```
       constructor() {
        super();
        this.state = { }
        this.handleLoginClick = this.handleLoginClick.bind(this);
        }
```

FAQ: Why method can't use state?
Ans: State is configured before method is registered. Hence state can't
recognize
       methods that are configured after creating component object.

FAQ: Can we use event subscriber without registering inside constructor?
Ans: Yes. You can bind and register a method only when event is trigged.

```
      <button onClick={this.handleLoginClick.bind(this)}>
```

FAQ: Can we register and use subscriber methods without bind()?
Ans: Yes. By keeping the memory allocated for method alive event after the
method
      end. You can keep memory alive by returning method as a result of event.

```
Syntax:
    <button onClick={()=> this.handleLoginClick() }>

Ex:
admin-login.jsx

import axios from "axios";
import React from "react";


export class AdminLogin extends React.Component
{
      constructor(){
         super();
         this.state = {
            msg:"Click Login"
         }
         this.handleLoginClick = this.handleLoginClick.bind(this);

      }

      handleLoginClick(){
        this.setState({msg: 'Login Success..'});
      }

      handleCancelClick(){
         this.setState({msg: 'Canceled..'});
      }

      handleContinueClick(){
         this.setState({msg: 'Continue with OTP'});
      }

      render(){
        return(
            <div className="container-fluid">
                <h2>Events</h2>
                <button onClick={this.handleLoginClick} id="btnLogin"
name="Login-Button">Login</button>
                <button
onClick={this.handleCancelClick.bind(this)}>Cancel</button>
                <button onClick={()=> this.handleContinueClick()}
>Continue</button>
                <p>{this.state.msg}</p>
            </div>
        )
      }
}
```

Forms in Class Component

- You can't use any library that requires and uses hooks.
- You can use any library that provides pre-defined components for handling forms.
- Formik Library provides components like

```
        <Formik>
        <Form>
        <Field>
        <ErrorMessage>

- You can use "Yup" for validation in formik component.

Ex:
admin-login.jsx

import axios from "axios";
import { ErrorMessage, Field, Form, Formik } from "formik";
import React from "react";
import * as yup from "yup";


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


      render(){
        return(
            <div className="container-fluid">
                <h2>Admin Login</h2>
                <Formik initialValues={{UserName:'', Password:''}}
onSubmit={(values)=>{console.log(values)}}
validationSchema={yup.object({UserName: yup.string().required('User Name
Required'), Password: yup.string().required('Password Required')})} >
                    <Form>
                        <dl>
                            <dt>User Name</dt>
                            <dd> <Field type="text" name="UserName" /> </dd>
                            <dd className="text-danger"> <ErrorMessage
name="UserName" /> </dd>
                            <dt>Password</dt>
                            <dd> <Field type="password" name="Password" /> </dd>
                            <dd className="text-danger"> <ErrorMessage
name="Password" /> </dd>
                        </dl>
                        <button type="submit">Login</button>
                    </Form>
                </Formik>
            </div>
        )
      }
}

- Controlled Components using Class
- Component Life Cycle
-------
  Controlled Class Components
```

```
- Class comprises of built-in "Props".
- These properties can control the component behavior.
- Props are object type with key and value collection.
- All keys of props are dynamic.

Syntax: function component

        function Name(props)
        {
             return(
                <div> { props.key } </div>
                  )
        }

Syntax: class component

        class Name extends React.Component
        {
              constructor() {
             super();
            }
              render() {
             return(
                  <div> { this.props.key } </div>
                   )
            }
        }

Note: You can implement conditional rendering in class components by using
      various decision making statements in render() method.

Ex:
1. Go to controlled-components folder

2. Add a new file  "toolbar.jsx"

import React from "react";

export class Toolbar extends React.Component
{
     constructor(){
        super();
     }
     render(){
        return(
            <div className="btn-toolbar my-2 bg-danger d-flex justify-content-
between">
                <button className="btn btn-danger"> {this.props.title} </button>
                <div className="btn-group">
                    {
                        this.props.menuItems.map(item=><button className="btn
btn-danger" key={item}>{item}</button>)
                    }
                </div>
                <div className="btn-group">
                    <button className="btn btn-danger bi bi-person-
fill"></button>
                    <button className="btn btn-danger bi bi-heart-fill"></button>
                    <button className="btn btn-danger bi bi-cart4"></button>
                </div>
            </div>
        )
     }
}
```

3. Go to any component and import toolbar

admin-login.jsx

```
import axios from "axios";
import { ErrorMessage, Field, Form, Formik } from "formik";
import React from "react";
import * as yup from "yup";
import { Toolbar } from "../controlled-components/toolbar";


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


      render(){
        return(
            <div className="container-fluid">
                <header>
                    <Toolbar title='Shopper.' menuItems={['Home', 'Shop',
'Pages', 'Blog', 'Docs']} />
                    <Toolbar  title='Amazon' menuItems={['Amazon Home',
'Electronics', 'Fashion']} />
                </header>
                <h2>Admin Login</h2>
                <Formik initialValues={{UserName:'', Password:''}}
onSubmit={(values)=>{console.log(values)}}
validationSchema={yup.object({UserName: yup.string().required('User Name
Required'), Password: yup.string().required('Password Required')})} >
                    <Form>
                        <dl>
                            <dt>User Name</dt>
                            <dd> <Field type="text" name="UserName" /> </dd>
                            <dd className="text-danger"> <ErrorMessage
name="UserName" /> </dd>
                            <dt>Password</dt>
                            <dd> <Field type="password" name="Password" /> </dd>
                            <dd className="text-danger"> <ErrorMessage
name="Password" /> </dd>
                        </dl>
                        <button type="submit">Login</button>
                    </Form>
                </Formik>
            </div>
        )
      }
}
----------------------
```

TypeScript

FAQ: What are the issues with JavaScript?
Ans:
- It is not strongly typed.
- It is not implicitly strictly typed.
- It is not an OOP language.
- Limited extensibility and code level security.

- Typescript is strictly typed version of JavaScript.
- It is strongly typed.
- It is an OOP language.
- Typescript is built with typescript.
- It supports low level features.
- It can directly interact with hardware services.
- It uses less memory and faster.
- Typescript can build large scale applications.
- "Anders Hejlberg" of Microsoft is the architect of TypeScript language.
  [also known for his contribution towards a language C#]
- Typescript is trans compiled into JavaScript.

Typescript Architecture:
1. Core Compiler
    - core.ts   It sets up the environment for running typescript program.
    - program.ts   It checks the program structure.
    - scanner.ts   It is responsible for handling input.
    - emitter.ts   It is responsible for handling output.
    - parser.ts   It is responsible for type conversion.
    - checker.ts   It is responsible for verifying the types. [data types]

2. Standalone Compiler
    - tsc.ts   It is responsible for trans compiling the Typescript code into
                         JavaScript.

3. Language Service
    - services.ts   It is responsible for managing a business layer for typescript
                         language. It provides a library of functions and values for
                         typescript.

4. Typescript Server
    - server.ts   It is responsible for hosting your application and handling

```
                   request & response.
                   Typescript programs are compiled and processed on server.

5. VS Shim
    - shims.ts           It is responsible for making typescript cross platform.
                   It generates manage code.

6. Managed            It refers to the managed library.
    Language           A managed library is cross platform library.
    Service
```

Setup Environment for Typescript:

1. Install typescript for your PC

```
    C:\>npm install -g typescript
```

2. Check the version of types

```
    C:\> tsc  -v        [ latest is 5.8.2 ]
```

3. Create a new folder for typescript

```
    D:\typescript
```

4. Open in folder in Visual studio code

5. Open terminal and run the following commands

```
        > npm  init  -y            [ package.json ]
        > tsc  -init             [ tsconfig.json ]   [tslint.json - obsolete]
```

6. Add a new typescript file into project

```
         index.ts

    console.log("Welcome to TypeScript");

    > tsc  index.ts             [ generates  index.js ]
    > node index.js            [ runs index.js program]
```

TypeScript Language

1. Variables & Data Types

- Variables are declared and initialized by using var, let, const.
- Variable naming rules are same as in JavaScript.
- TypeScript is a strongly typed language, hence every variable requires a data type to configure.

Syntax:

```
        let  variableName : datatype;
```

```
- If data is not defined then the default type is "any".
- The type "any" is root for all data types.

Syntax:
        let  age;        age:any;

- The data type for variable can be JavaScript
    a) Primitive Type
    b) Non Primitive Type

- JavaScript Primitive Data Types

    a) number
    b) boolean
    c) string
    d) null
    e) undefined
    f) Symbol
    g) bigint

Syntax:
        let  username:string = "John";
        let  age:number = 23;
        let  subscribed:boolean = true;
        etc..

- TypeScript supports "Type Inference". The data type of variable can be
configured according to the value initialized.

Syntax:
        let age = 22;           // age:number
        age = "A";        // invalid

- TypeScript supports union of types. It allows to configure multiple types for a
variable.

Syntax:
          let  x : number | string;
        x = 10;
        x = "A";
        x = true;        // invalid

Ex:
    let username: string | null = prompt("Enter Name");        // prompt returns
string
                                                         or null.

TypeScript Non Primitive Types:

1. Array Type
- Typescript array can be configured for similar data types.
- It can also handle various types like JavaScript.

Syntax:
        let  sales:number[]  = new Array();
        let  names:string[]  =  [ ];
        let  values:any[]   = [ ];
```

- Array() constructor will not allow to initialize various data types even when the type is configured as "any". It uses the first value type as data type.
- Array() constructor will allow to assign various data types when data type any.

Syntax:

```
        let  values:any[]  = new Array(10, "A");        // invalid only numbers
allowed

        let values:any[] = new Array();
        values[0] = 10;        // valid
        values[1] = "A";        // valid
        values[2] = true;        // valid
```

- Array meta character "[ ]" allows initialization and assignment of various types when data type is configured as "any".
- If a collection allows initialization and assignment of various types then its known as
  "Tuple".

```
          let  values:any[] = [ ];
```

- Array supports union of types only for assignment not for initialization.

Syntax:

```
          let  values:string[] | number[] = [ ];
          values[0] = 10;
          values[1] = "A";
          values[2] = 20;
```

Syntax:

```
          let  values:string[] | number[] = [10, "A" ];        // invalid
```

- All array methods are same as in JavaScript.

Ex:
index.ts

```
let values:string[]|number[] = [];

values[0] = 10;
values[1] = "A";
values[2] = 20;
values[3] = "B";

values.map(value=>{
    console.log(value);
})
```

```
> tsc index.ts
> node index.js
```

2. Object Type
- JavaScript object is schema less.
- TypeScript object is a structured object.
- It can set rules for configuring values in the reference of keys.
- Rules are defined as an object.

Syntax:
```
    let  obj : { rules } = {  data } ;
```

- Rules refer to key and value.
- Data refer to key and value.

Syntax:
```
     let  obj : {key:datatype}  = { key:value }
```

- Rules are considered as a contract, and you have to implement exactly the same that is defined in contract.
- You can't add new keys or you can't miss existing keys.
- Contract allows to configure optional keys by using null reference character "?".

Syntax:
```
     let obj : { key?:datatype } = { } ;
```

Ex:

index.ts

```
let product:{Name:string, Price:number, Rating?:number} = {
    Name: "TV",
    Price: 50000.33,
}
product.Rating = 4.5;
if(product.Rating){

console.log(`Name=${product.Name}\nPrice=${product.Price}\nRating=${product.Ratin
g}`);
} else {
    console.log(`Name=${product.Name}\nPrice=${product.Price}`);
}
```

Ex:
index.ts

```
let product:{Name:string, Price:number, Qty:number, Total():number,
Print?():void} = {

    Name: "TV",
    Price: 56000.44,
    Qty: 2,
    Total:function(){
        return this.Qty * this.Price
    },
    Print:function(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${thi
s.Total()}`);
    }
}
product.Print();
```

- All object manipulations are same as in JavaScript.

TypeScript Language
- Variables
- Data Types
- Union of Types
- Type Inference
- Primitive & Non Primitive
- Array
- Tuple
- Object
- Optional Keys [?]

Array of Objects:
- You can configure strongly typed array with schema based objects.
- The data type is defined as object that handles array.

Syntax:

```
     let  values : { } [ ]  = [ ] ;

     {  }        => It sets rules for object
     [  ]        => It handles multiple values
```

Ex:
index.ts

```
let products:{Name:string, Price?:number}[] = [];

products = [
    {Name: 'TV', Price:5000.33},
    {Name:'Mobile', Price:1200.33}
];

products.map(product=>{
    console.log(`${product.Name} - ${product.Price}`);
})
```

Map Type:
- The class "Map" is used to configure a Map type.
- It is a generic type in typescript.
- You can make keys strongly typed for specific data type.
  [JavaScript map keys handle any type]
- You can configure any or restrict to specific.

```
Syntax:
    let  data : Map<keyType, valueType> =  new Map();

    data.get()
    data.set()
    data.has()
    data.keys()
    data.values()
    data.entries()
    data.delete()
    data.clear() etc..

Ex:

let data:Map<string, number> = new Map();
let list:Map<number, any> = new Map();
let values:Map<any, any> = new Map();

data.set("A", 1200);

console.log(data.get("A"));
data.keys()
data.entries()
data.values()


Date Type:
- You can configure strongly typed date values in typescript.
- All date and time functions are same as in JavaScript.

Syntax:
     let  now : Date   = new Date();        // loads current date & time
     let  mfd  : Date  = new Date("yy-mm-dd hrs:min:sec.milliSec");

    now.get..()
    now.set..()

Regular Expression Type:
- Typescript provides "RegExp" as data type for regular expression.
- It allows meta characters and quantifiers enclosed in "/  /".

Syntax:
     let pattern: RegExp = /\+91\d{10}/;

- Value is verified with regular expression by using  "match()" method.
- The value type must be any type to verify with regular expression.

Note: Statements and Operators are same as in JavaScript.

1. Operators
    - Arithmetic
    - Logical
    - Comparison
    - Assignment
    - Bitwise
    - Special  etc.

2. Statements
    Selection
        if, else, switch, case, default
    Iteration
        for..in, for..of
    Looping
        for, while, do while
    Jump
        break, continue, return
    Exception handling
        try, catch, throw, finally
```

Typescript Functions

- You can configure a function with declaration or expression.
- Typescript function declaration comprises of
    a) parameter data type
    b) function with return data type or void

Syntax:
```
       function  Name(param : datatype) : datatype | void
       {
       }
```

Ex:
index.js

```
function Addition(a:number, b:number):number
{
     return a + b;
}

function Print():void
{
    console.log(`Addition=${Addition(50,20)}`);
}
Print();
```

- Typescript function can have optional parameters.
- Optional parameter must be last parameter.
- A required parameter can't follow an optional parameter.

Syntax:
```
     function Name(param1:type,  param2?:type)        // valid
     {
     }

     function Name(param1?type: param2:type)            // invalid
     {
     }
```

```
- Optional parameters are verified by using "undefined" type.

Ex:

index.ts
function Details(id:number, name:string, price?:number):void
{
     if(price) {
         console.log(`id=${id}\nname=${name}\nprice=${price}`);
     } else {
        console.log(`id=${id}\nname=${name}\n`);
     }
}
Details(1, "TV");
Details(2, "Mobile", 5000.44);

- A function can configure rest parameters.
- Rest parameter must be defined as array type.
- It can be strongly typed array with specific type of arguments or any type.

Syntax:
      function  Name(...params: number[])
      {
      }
      Name(10, 20, 50);

      function Name(...params: any[])
      {
      }
      Name (1, "A", true);

Ex:
index.ts

function Details(...product:any[]):void
{
     let [id, name, price] = product;

     if(price) {
         console.log(`id=${id}\nname=${name}\nprice=${price}`);
     } else {
        console.log(`id=${id}\nname=${name}\n`);
     }
}
Details(1, "TV");
Details(2, "Mobile", 5000.44);

Type Script Promise:
- Promise function uses a generic type.
- It allows to restrict specific data typed on resolve.
- However the reject function is always any type.

Syntax:
    let  fetch : Promise<datatype> = new Promise(function(resolve, reject){

        resolve(datatype);
        reject(any_type);

    })

    fetch.then((data)=>{ }).catch((error)=>{ }).finally(()=>{ })
```

Ex:
index.ts

```
let FetchData:Promise<{Name:string, Price:number}> = new
Promise(function(resolve, reject){

        let url = "https://server.com&quot;;

        if(url==="https://fakestore.com&quot;){
             resolve({Name:'TV', Price:50000});
        } else {
             reject('Invalid URL - API Not found');
        }
})

FetchData.then(function(data){
    console.log(data);
})
.catch(function(error){
    console.log(error);
})
```

Typescript OOP

Contracts in OOP:
- A contract defines rules for designing any component.
- It enables reusability, separation and easy extensibility.
- In OOP contract as designed as "interface".

Syntax:

```
        interface  IName
        {
              // rules
        }
```

- Contract must contain only declaration not implementation.
- A contract can define rules for both property and method.

Syntax:

```
        interface  IName
        {
            property: datatype;
            method(): datatype | void;
        }
```

- A contract can have optional rules defined using null reference character [?].

Syntax:
```
        interface IName
        {
            property?: datatype;
            method?(): datatype | void;
        }
```

- Optional rules are required to define goals for any contract.

Note: Objective is time bound and mandatory to implement.
      Goal is not time bound and optional to implement.

- A contract can have read-only rules.
- Every rule can be assigned with a new value or functionality after initialization.
- You can prevent assignment by configuring the rule as "readonly".

Syntax:
```
       interface  IName
       {
          readonly  property: datatype;
       }
```

- You can extend a contract by adding new rules.
- Extensibility is achieved by using inheritance.
- A contract "extends" another contract.
- It allow to implement multiple contracts. [multiple inheritance]
- It is possible as interface will not have a constructor.
- Interface will not allow to create an instance.

Syntax:
```
        interface   IName extends  Contract1, Contract2
        {
        }
```

Ex:
```
interface ICategory
{
    CategoryName?:string;
}

interface IRating {
    Rating:number;
}

interface IProduct extends ICategory, IRating
{
    Name:string;
    readonly Price:number;
    Qty:number;
    Total():number;
    Print?():void;
}


let tv:IProduct = {
    Name : "Samsung TV",
    Price: 40000.44,
    Qty: 2,
    CategoryName: "Electronics",
    Rating: 4.2,
    Total() {
        return this.Qty * this.Price
    },
    Print(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nTotal=${this.Total()}\nCateg
ory=${this.CategoryName}\nRating=${this.Rating}`);
    }
};
tv.Print();
```

- A contract is used as type for any memory reference.
- The default type is object, you can configure as array.

Syntax:

```
    let obj : IProduct = {  };

    let obj : IProduct[] = [ { }, { } ];
```

Classes:
- Class declaration and expression is same as in JavaScript.
- It supports ES6+ static members.
- Class members are same
    a) Property
    b) Accessor
    c) Method
    d) Constructor
- Class supports access modifiers like
    a) public
    b) private
    c) protected

FAQ: What is difference between static and non-static?
Ans:
 a) Static
     - It refers to continuous memory.
     - Memory allocated for first object will continue for next.
     - It uses more memory.
     - It is good for continuous operations.
     - A static member is accessible with in or outside class by using class name.

 b) Non Static | Dynamic

```
    - It refers to discreet memory.
    - Memory is newly allocated for object every time.
    - It is safe and uses less memory.
    - It is good for disconnected actions.
    - It is accessible with in class by using "this" keyword and outside class by
using
      and instance of class.

POC:

class Demo
{
    static s = 0;
    n = 0;
    constructor(){
        Demo.s = Demo.s + 1;
        this.n = this.n + 1;
    }
    Print(){
        console.log(`s=${Demo.s} n=${this.n}`);
    }
}

let obj1 = new Demo();
obj1.Print();

let obj2 = new Demo();
obj2.Print();

let obj3 = new Demo();
obj3.Print();
----------------------
TypeScript OOP
---------------------

- Contracts / Interface
- Classes
    - Members are same as in JS
    - Static and Non Static Members

Access Modifiers:
1. public
2. private
3. protected

- public allows access from any location and by using any instance.
- private allows access with in class.
- protected allows access with in class and from derived class only by using
derived class reference. [or instance]

Ex:
class Super
{
    public Name:string = "TV";
    private Price:number = 40000;
    protected Stock:boolean = true;
    public Print():void {
        console.log(`${this.Stock}\n${this.Name}\n${this.Price}`);
    }
}
class Derived extends Super
{
    public Print():void {
        let obj = new Derived();
        obj.Stock;  // protected
        obj.Name;   // public
    }
}

let obj = new Derived();
obj.Name;       // public
```

- Inheritance and rules are same as in JavaScript. [extends]
- A class can implement the contract.
- Contract is used to defined rules for a class.
- Class implements contract and allows to customize by adding new properties and methods.
- Class is a program template, hence it allows customization.
- Class can implement multiple contracts.

Syntax:
```
        class Name implements Contract1, Contract2
        {
        }
```

Ex:
```
interface IProduct
{
    Name:string;
    Price:number;
}
interface ICategory
{
    CategoryName:string;
}

class Product implements IProduct, ICategory
{
    public Name = "TV";
    public Price = 50000;
    public CategoryName = "Electronics";
}
```

Templates in OOP:
- A template provides pre-defined structure.
- It comprises of some features implemented and some of them need to be implemented.
- Client implements templates by customizing according to the requirements.

- Templates are mostly used for Rollouts and Implementation of secured modules in application for any business.
- Template hides its structure and provides only implementation.
- The process of hiding structure and providing only implantation to developer is known as "Abstraction".
- Typescript can create template as "Abstract Class".

Syntax:

```
    abstract class Name
    {

    }
```

- Abstract class can have members implemented and yet to implement.
- The members that require implementation are marked as "abstract".

```
    {
      public abstract property;
      public abstract method();
    }
```

- If a class have at least on abstract member then it must be marked as abstract.
- You have extend abstract class to implement the abstract member.
- You can't create instance for abstract class. But it can have a constructor.

Ex:

```
interface IProduct
{
    Name:string;
    Price:number;
    Qty:number;
    Total():number;
    Print():void;
}
abstract class ProductTemplate implements IProduct
{
    constructor(){
        console.log("Abstract class Constructor");
    }
    public Name:string = "";
    public Price:number = 0;
    public Qty:number = 1;
    public abstract Total():number;
    public abstract Print():void;
}
class Component extends ProductTemplate
{
        Name = "Samsung TV";
        Price = 50000;
        Qty = 2;
        Total(){
            return this.Qty * this.Price;
        }
        Print(){
```

```
console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${thi
s.Total()}`);
        }
}

let tv = new Component();
tv.Print();
------------
  Generics
 -------------
```

- Generic is used to configure "type safe" component or property.
- It is open to handle any type of data, and makes it strongly typed once it knows the data type of value.
- Typescript allows generic
    a) method
    b) function
    c) parameter
    d) class
    e) property etc..

- Generic type is defined by using "<T>" type reference.

Syntax:
```
       function Name<T>(param: T): T
       {
           return  param | expression;
       }
```

Ex:
```
function Print<T>(param:T):void{
    console.log(`Param=${param}`);
}

Print<number>(30);
Print<string>("A");
Print<string[]>(["A","B"]);
```

Ex: Generic Class & Property

```
interface IOracle
{
    UserName:string;
    Password:string;
    Database:string;
}
interface IMongoDB
{
    URL:string;
}

class Database<T>
{
    public connectionString:T|undefined;
    public Print():void{
        for(var property in this.connectionString){
            console.log(`${property} : ${this.connectionString[property]}`);
        }
    }
}

let oracle = new Database<IOracle>();
oracle.connectionString = {UserName:'scott', Password:'tiger', Database:'stu'};
oracle.Print();

let mongodb = new Database<IMongoDB>();
mongodb.connectionString = {URL: 'mongodb://127.0.0.1:27017'};
mongodb.Print();
```

Ex:

```
interface IProduct
{
    Name:string;
    Price:number;
}
interface IEmployee
{
    FirstName:string;
    LastName:string;
    Designation:string;
}

class FetchData<T>
{
     public Response<T>(data:T){
         console.log(data);
     }
}

let product = new FetchData<IProduct>();
product.Response<IProduct>({Name: 'TV', Price:55000});
product.Response<IProduct[]>([{Name:'Mobile', Price:30000}, {Name:'Watch', Price:
13000}]);
```

- You can't use operators on generic types.
- Generics are always handled by using methods or functions.

Syntax:

```
        function Add<T>(a:T, b:T) : T
        {
           return  a + b;    // invalid
        }
```

Ex:

```
function Sum(a:any,b:any){
    return a + b;
}

function Addition<T>(a:T, b:T):T
{
    return Sum(a,b);
}

console.log(`Addition=${Addition<number>(40,20)}`);
```

Enumeration [Enum]

- It is a collection of constants.
- Enum can have numeric, string or expression as constant.
- It can't have Boolean value or expression as constant.
- Enum allows auto implementation of values if the type is "number".

Syntax:

```
    enum Values
    {
        A,                // A=0
        B=20,
        C                // C=21
    }
    Values.A;
    Values.A = 20;    // invalid
```

- Auto implementation is allows only when the previous value type is a number.
- Enum allows reverse mapping.
- It is a technique of accessing key with reference of value.

Ex:

```
enum StatusCodes
{
     Client,
     OK = 200,
     Found,
     NotFound = 404,
     Server = "Inernal Server Error",
}
console.log(`${StatusCodes.NotFound}: ${StatusCodes[404]}`);
```

Ex:

```
enum StatusCodes
{
    A = 10,
    B = 20,
    C = A + B
}
console.log(`Addition=${StatusCodes.C}`);
```

Module System

- A module system enables features like

```
    a) Code Separation
    b) Code Reusability
    c) Extensibility
    d) Maintainability
    e) Testability
- TypeScript uses CommonJS module system.
- A Typescript module comprises of
    a) Contracts
    b) Templates
    c) Components
    d) Functions
    e) Values

Ex:
1. Add folders
    - contracts
    - templates
    - components
    - app

2. contracts
    ProductContract.ts

export interface ProductContract
{
    Name:string;
    Price:number;
    Qty:number;
    Total():number;
    Print():void;
}

3. templates
    ProductTemplate.ts
import { ProductContract } from "../contracts/ProductContract";

export abstract class ProductTemplate implements ProductContract
{
      public Name:string = "";
      public Price:number = 0;
      public Qty: number = 0;
      public abstract Total():number;
      public abstract Print():void;
}

4. components
    ProductComponent.ts

import { ProductTemplate } from "../templates/ProductTemplate";

export class ProductComponent extends ProductTemplate
{
     Name = "Samsung TV";
     Price = 50000;
     Qty = 2;
     Total(){
        return this.Qty * this.Price;
     }
     Print(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${thi
s.Total()}`);
     }
}
```

```
5. app
    Index.ts

import { ProductComponent } from "../components/ProductComponent";

let tv = new ProductComponent();
tv.Print();


    D:\..\app> tsc index.ts
             > node index.js
```

Namespace

- A namespace is a collection of sub-namespaces and OOP components.
- It can have a set of contracts, templates or components.
- It is used to build and organize libraries at large scale.

Syntax:

```
         namespace  Project
         {
          namespace  Module
          {
                // contracts
            // templates
            // components
          }
         }
```

- A namespace is imported by using "///<reference />" directive.

Syntax:

```
        ///<reference  path="./folder/module" />
```

- You can alias namespace of access using fully qualified name.

Syntax: Fully Qualified

```
        Project.Module.ClassName

        let obj = new Project.Module.ClassName();
```

Syntax: Aliasing Namespace

```
        import   className = Project.Module.ClassName;

        let obj = new className();
```

- To compile the typescript library with namespace you have to use the command

```
        > tsc -outFile  index.js  index.ts

        > node index.js
```

Note: In typescript latest release the parent name space doesn't require export.
      All other members must be marked as export.

Ex:
1. Add folders
        -contracts
        -templates
        -components
        -app

2. contract/ProductContract.ts

```
namespace Project
{
     export namespace Contracts
      {
            export interface IProduct
            {
                Name:string;
                Price:number;
                Qty:number;
                Total():number;
                Print():void;
            }
      }
}
```

3. template/ProductTemplate.ts

```
///<reference path="../contracts/ProductContract.ts" />

import ProductContract = Project.Contracts.IProduct;

namespace Project
{
     export namespace Templates
     {
          export abstract class ProductTemplate implements ProductContract
          {
               public Name:string = "";
               public Price:number = 0;
               public Qty:number = 0;
               public abstract Total():number;
               public abstract Print():void;
          }
     }
}
```

4. components/ProductComponent.ts

```
///<reference path="../templates/ProductTemplate.ts" />

import ProductTemplate =  Project.Templates.ProductTemplate;

namespace Project
{
      export namespace Components
      {
           export class ProductComponent extends ProductTemplate
           {
                Name = "Samsung TV";
                Price = 45000;
                Qty = 2;
                Total(){
                    return this.Qty * this.Price;
                }
                Print(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${thi
s.Total()}`);
                }
           }
      }
}
```

5. app/index.ts

```
///<reference path="../components/ProductComponent.ts" />

import ProductComponent = Project.Components.ProductComponent;

let tv = new ProductComponent();
tv.Print();
```

6. Compile and Run

```
    D:..\app> tsc  -outFile  index.js  index.ts

             > node index.js
```

React Application with TypeScript

- You can use various bundling tools for creating react app.
    a) Webpack
    b) Vite

c) Parcel

- Vite is a build tool that provides various features for developer to
    a) build
    b) debug
    c) test
    d) deploy

Creating a new Project using Vite and TypeScript for React:

1. Make sure that your device is installed with Node 18+ version

2. Open your PC location in command prompt

```
      D:\>npm  create  vite@latest   app-name  -- --template  react-ts
[Typescript]
      D:\>npm  create  vite@latest   app-name  -- --template  react
[JavaScript]
```

3. A project folder is created at specified location

4. Change into the folder and run the command

```
      D:\app-name> npm  install
```

   - It will install react, react-dom, typescript dependencies.

5. Run your project

```
    D:\app-name> npm run dev

    http://localhost:5173
```

React Vite App File System:

node_modules	: comprises of library files installed using npm.
public	: comprises of static resources
src	: comprises of dynamic resources
.gitignore	: configuration to ignore folders while publishing to GIT
eslint.config.js	: It is JavaScript language analysis tool
index.html	: startup page
package.json	
package.lock.json	
README.md	
tsconfig.json	: typescript configuration file
tsconfig.app.json	: typescript configuration for current app
tsconfig.node.json	: node JS migrations
vite.config.ts	: vite configuration for app [It configures react for your app]

Note: The index.js is now replaced with "main.tsx" in "src" folder.

```
      index.html  is importing  main.tsx
-----------------------------------
What's new with React TypeScript:

1. Component is a function that returns JSX.Element

     - TypeScript supports type inference, hence the return type is not mandatory
to
    define.

     Syntax:
     export function Login() : JSX.Element
     {
         return (
            <>
                JSX
            </>
            );
      }


2. JSX Rules are same.

3. Component file must have extension .tsx.

          Login.tsx

Note: All library files must have extension ".ts".
        - Contracts
        - Templates
      Component & Hooks are defined as ".tsx".


4. Component state is generic type.

    const [get, set] = useState<T>();

    const [categories, setCategories] = useState<string[]>([ '  ', '  ']);

    You can use a contract for configuring data type.

    const [product, setProduct] = useState<IProduct>();

    interface IProduct
    {
      Id:number;
      Name:string;
    }

5. Data Binding, Style Binding, Class Binding & Event Binding all are same.

6. All hooks are same.

7. Axios, Formik, Yup, Routing, cookies etc. all are same.
```

8. Controlled components uses Props type is an any type object.

Syntax:
```
        export function Navbar(props:any)
        {
        }
```

9. Setup Bootstrap or MUI for project is same. MUI components are same.

10. Import bootstrap, cookie provider and other global provides in "main.tsx".

Note: The state configure for handling data must be strongly typed.
      It is default nullable, if value is not initialized.

Syntax: If value is not initialized
```
         const [categories, setCategories] = useState<string[]>();

         categories?.map(category=> <li> </li>)
```

Syntax: If value is initialized
```
        const [categories, setCategories] = useState<string[]>([ ' ' ]);

        categories.map(category=> <li></li>)
```

Ex: Fakestore API

1. Add contracts folder into "src"

2. Add a new file "fakestore-contract.ts"

```
export interface FakestoreContract
{
    id:number;
    title:string;
    description:string;
    image:string;
    price:number;
    rating: {rate:number, count:number}
}
```

3. demo.tsx

```
import { useEffect, useState } from "react";
import { FakestoreContract } from "../contracts/fakestore-contract";
import axios from "axios";


export function Demo(){


    const [categories, setCategories] = useState<string[]>();
    const [products, setProducts] = useState<FakestoreContract[]>();
    function LoadCategories(){
        axios.get(`https://fakestoreapi.com/products/categories`)
        .then(response => {
            setCategories(response.data);
        })
    }
    function LoadProducts(){
        axios.get(`https://fakestoreapi.com/products`)
        .then(response=>{
            setProducts(response.data);
        })
    }

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

    return(
        <div className="container-fluid">
            <header>
                <h3 className="text-center">Fakestore</h3>
            </header>
            <section className="row mt-4">
                <nav className="col-2">
                    <label className="form-label">Select Category</label>
                    <select className="form-select">
                        {
                            categories?.map(category=><option
key={category}>{category}</option>)
                        }
                    </select>
                </nav>
                <main className="col-10 d-flex flex-wrap overflow-auto"
style={{height:'400px'}}>
                     {
                        products?.map(product=>
                            <div key={product.id} className="card p-1 m-2"
style={{width:'200px'}}>
                                <img src={product.image} className="card-img-top"
height='120' />
                                <div className="card-header"
style={{height:'100px'}}>
                                    {product.title}
                                </div>
                            </div>
                        )
                     }
                </main>
            </section>
        </div>
    )
}
```

MERN Stack Application
[Project]

- Video Library Application

    Admin Module

    * Admin can login
    * Admin can add videos
    * Edit videos
    * Delete videos

    User Module

    * User can register
    * User can login
    * User can browse and view videos
    * User can search videos by title, category etc.
    * User can save videos to watch later. [My List]

- Libraries and Frameworks

MongoDB	: for database
Express JS	: for middleware
Node JS	: for server side app
React	: for UI - front end
React Router	: for routing
Axios	: for API communication
Formik	: for form
Yup	: for validation
React Cookies	: for user state
Redux Toolkit	: for Videos watch later [application state]

Setup Database for Application

- MongoDB database
- It is an Non-SQL database [no-SQL]
- It uses BSON type data types.
- It is similar to JSON.
- It provides simple methods for handling CRUD operations.

1. Install MongoDB community server on your device.

2. Select "MongoDB compass" tool while installing MongoDB server.

3. Compass is a GUI tool that provides an UI to handle database on server.

    https://www.mongodb.com/try/download/community

4. After installing MongoDB start its server

    - Go to "services.msc" from your programs
    - Right Click on "MongoDB Server" and select start.

5. Open MongoDB compass from programs

6. Connect to Server using following connection string

```
      mongodb://127.0.0.1:27017
```

7. After connecting you will find default databases

    a) admin
    b) config
    c) local

MongoDB Terminology:

Oracle, MySQL	MongoDB
Database	Database
Table	Collection
Record / Row	Document
Field / Column	Field / Key
Join	Embedded Document

```
    find()
    insertOne()
    insertMany()
    updateOne()
    updateMany()
    deleteOne()
    deleteMany()
```

Video Library Project - MERN Stack
Setup Database - MongoDB

CRUD Operations in MongoDB:
- Database
- Collection [Table]
- Document [Record]
- Field

1. Open MongoDB Compass
2. Connect with MongoDB server
```
        mongodb://127.0.0.1:27017
```
3. Select connection
4. Open MongoDB Shell. It is a CLI tool for handling database.

Commands:
1. To create a new database or to start using existing database

```
        > use  databaseName
        > use  demodb         // creates a new database if doesn't exist
```

Note: Database will be displayed in list only when it is having collections.

2. To create a new collection [table]

```
        >db.createCollection("name", { attributes })

        >db.createCollection("products")
```

3. To view databases and collections

```
        > show dbs            // to view database
        > show collections        // to view collections [tables]
```

4. Add data into collection [table]

```
        a) insertOne()
        b) insertMany()

    > db.collectionName.insertOne({key:value})
    > db.collectionName.insertMany([{key:value}, {key:value},..])
```

All JavaScript data types are used for MongoDB

Ex:

```
 > db.products.insertMany([{Id:1, Name:"TV", Price:23000, Rating:{Rate:4.2,
Count:500}}, {Id:2, Name:"Mobile", Price:12000, Rating:{Rate:4.1, Count:230}}])
```

5. Read data from collection

```
        a) find()
        b) findOne()
```

Syntax:

```
        > db.collectionName.find({query})

        > db.products.find({})            // returns all documents
        > db.products.find({id:3})
```

Operators:

```
            $gt        greater than
            $gte        greater than or equal
            $lt        less than
            $lte        less than or equal
            $eq        equal
            $ne        not equal
            $or        OR
            $and    AND

        > db.products.find({Price:{$gte:5000}})
        > db.products.find({'Rating.Rate': {$gt:4}})
        > db.products.find({$and:[{Category:"Electronics"},
{Price:{$gte:5000}}]})
```

6. Update Document

```
        a) updateOne()
        b) updateMany()

    > db.collectionName.updateOne({findQuery}, {updateQuery})

    Operators:

        $set                update data into specified field
        $unset              removes a field
        $rename             changes field name

    > db.products.updateOne({Id:1}, {$set:{Price:25000, Name:"Samsung TV"}})
```

7. Delete Document

```
    a) deleteOne()
    b) deleteMany()

    > db.products.deleteOne({findQuery})

    > db.products.deleteMany({Category:"Electronics"})
    > db.products.deleteOne({Id:3})
```

Zoom Meeting ID: 91690775166
PassCode   : 112233

Timing   9:30 AM to 6:30 PM

Server Side Application
[Node & Express JS]

- Node is a JavaScript runtime used to build web applications, scripts, command line tools etc.
- Express JS is a middleware framework for Node JS.
- Middleware enables communication between client-server-database.
- You can create API using node & expression.

1. Create a new folder on your PC

```
        D:\server-app
```

2. Setup package JSON

```
        >npm init -y
```

3. Install the following libraries

```
        > npm  install  express  --save
        > npm  install  cors  --save

4. Add a new file  "server.js"

const express = require("express");

const app = express();

app.get("/", (req, res)=>{
     res.send("Welcome to API");
     res.end();
});

app.get("/products", (req, res)=>{
     res.send([{Name:"TV"},{Name:"Mobile"}]);
     res.end();
});

app.listen(5050);
console.log(`Server Started http://127.0.0.1:5050`);

5. Run
    > node server.js
-------------------------------------------
Create Database and Collection for Project
1. Create a new database on MongoDB
        "video-tutorial"

2. Add collections

        1. admin [collection]

        {
          admin_id : string,
          password : string
        }

        2. users [collection]

        {
          userid: string,
          username: string,
          password: string,
          email:string
        }

        3. videos [collection]

        {
          video_id : number,
          title: string,
          description: string,
          url: string,
          likes: number,
          views: number,
          dislikes: number,
          category_id: number [FK]
        }

        4. categories [collection]

         {
          category_id: number, [PK]
          category_name: string
        }
```

Create API to handle request [End Points]

```
    GET        /admin                     get admin users
    GET        /users                     get all users
    GET        /videos                    get all videos
    GET        /videos/1                  get specific video by id
    GET        /categories                get all categories

    POST    /register-user                add new user into database
    POST    /add-video                    add new video into database
    PUT        /edit-video/1              update and save video details
    DELETE    /delete-video/1             delete video by ID.
```

1. Go to your server app [add server folder into react-typescript-app]

2. Install following libraries

```
        > npm install  express  mongodb  cors  --save
```

```
        express        : It is a middleware for configuring API end points
        mongodb     : It is a drivers library to connect with MongoDB database
        cors            : Cross Origin Resource Sharing for handling request like
                      GET, POST, PUT, DELETE. [restrictions]
```

3. Add a new file "api.cjs" into server folder

```
// import libraries

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

// Create connection string and app

const conString = "mongodb://127.0.0.1:27017";

const app = express();
app.use(cors());
app.use(express.urlencoded({extended:true}));
app.use(express.json());
```

```
// Create API end points

app.get('/admin',(req, res)=>{

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");

        database.collection("admin").find({}).toArray().then(documents=>{
              res.send(documents);
              res.end();
        });
    });
});

app.get('/users',(req, res)=>{

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");

        database.collection("users").find({}).toArray().then(documents=>{
              res.send(documents);
              res.end();
        });
    });
});

app.get('/videos',(req, res)=>{

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");

        database.collection("videos").find({}).toArray().then(documents=>{
              res.send(documents);
              res.end();
        });
    });
});

app.get('/videos/:id',(req, res)=>{

    var id = parseInt(req.params.id);

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");

        database.collection("videos").findOne({video_id:id}).then(document=>{
              res.send(document);
              res.end();
        });
    });
});

app.get('/categories',(req, res)=>{

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");

        database.collection("categories").find({}).toArray().then(documents=>{
              res.send(documents);
              res.end();
        });
    });
});

app.post('/register-user', (req, res)=>{

        var user = {
            userid: req.body.userid,
            username: req.body.username,
            password: req.body.password,
            email: req.body.email
        };

        mongoClient.connect(conString).then(clientObject=>{

            var database = clientObject.db("video-tutorial");

            database.collection("users").insertOne(user).then(()=>{
                 console.log('User Registered');
                 res.send();
            });
        });
});


app.post('/add-video', (req, res)=>{

    var video = {
        video_id : parseInt(req.body.video_id),
        title: req.body.title,
        description: req.body.description,
        url: req.body.url,
        likes: parseInt(req.body.likes),
        dislikes: parseInt(req.body.dislikes),
        views: parseInt(req.body.views),
        category_id: parseInt(req.body.category_id)
    };

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");
        database.collection("videos").insertOne(video).then(()=>{
             console.log('Video Added');
             res.send();
        });
    });
});

app.put('/edit-video/:id', (req, res)=>{

    var id = parseInt(req.params.id);

    var video = {
        video_id : parseInt(req.body.video_id),
        title: req.body.title,
        description: req.body.description,
        url: req.body.url,
        likes: parseInt(req.body.likes),
        dislikes: parseInt(req.body.dislikes),
        views: parseInt(req.body.views),
        category_id: parseInt(req.body.category_id)
    };

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");

        database.collection("videos").updateOne({video_id:id},{$set:
video}).then(()=>{
             console.log('Video Updated');
             res.send();
        });
    });
});

app.delete('/delete-video/:id', (req, res)=>{

    var id = parseInt(req.params.id);

    mongoClient.connect(conString).then(clientObject=>{

        var database = clientObject.db("video-tutorial");

        database.collection("videos").deleteOne({video_id:id}).then(()=>{
             console.log('Video Deleted');
             res.send();
        });
    });
});



app.listen(4040);
console.log(`Server Started http://127.0.0.1:4040`);
```

4. Go to package.json

```
    scripts: {

        "api"  :  "node  ./server/api.cjs"

    }
```

 5. From terminal you can run command

```
    >npm run api
```

Creating Database for Project
    - MongoDB
Creating API and Configure End Points
    - Node & Express JS

Creating UI with React:

1. Install required libraries for React app

```
    > npm install  bootstrap bootstrap-icons  formik yup axios  react-cookie --save
    > npm install @mui/material @emotion/react @emotion/styled  --save
    > npm install  react-router-dom --save
```

2. Go to "main.tsx" and import Cookies Provider

3. Set "App" component as startup component.

```
      import {  CookiesProvider }  from  'react-cookie';

      <CookiesProvider>
           <App />
      </CookiesProvider>
```

4. Import bootstrap & icons CSS into main.tsx

5. Add contracts folder into "src" and setup all contracts required to connect with database in backend.

```
        contracts/admin-contract.ts
        contracts/user-contract.ts
        contracts/video-contract.ts
        contracts/categories-contract.ts
```

6. Add components folder into src with following component files

```
        user-login.tsx
        user-dash.tsx
        admin-login.tsx
        admin-dash.tsx
        admin-add-video.tsx
        admin-edit-video.tsx
        admin-delete-video.tsx
        register-user.tsx
        video-home.tsx
----------------------------------------------------
```

Functionalities to Implement
- Likes, dislikes counter must increase and store in database
- User must able to view the video separately in a new route.
- It must increase the views count.
- User must able to search by category or title.
- User can register a new account.
- Setup validation while adding, editing videos.

React Redux

- Redux is a JavaScript library.
- It is used to configure and maintain global application memory for JavaScript based applications.
- It is a large scale version of application memory when compared to useReducer in React.
- It is predictable and debuggable.
- It provides a complete toolkit for developers to manage application state.
- Redux can be used with angular, react, vue and all JavaScript apps.

Redux Components:
- a) Store
- b) State
- c) Reducer

- Store is the location where data is kept.
- State can access data from store and update to UI.
- Reducer comprises of actions required to update data in store.

Setup Redux for Project

1. Install Redux toolkit with React support

```
    > npm install @reduxjs/toolkit   react-redux  --save
```

2. Redux toolkit provides
    - a) Slicer
    - b) Store
    - c) Reducer
    - d) Initial State

3. Create a new slicer

    - Slicer configure the initial data to store in global memory.
    - It initializes the global memory.
    - It uses initial state.
    - You can create by using "createSlice()" method.
    - It also defines the actions to perform
    - Actions are required to update the data in global memory.

```
Syntax:
        video-slicer.tsx

      let initialState = {
          videos : [ ],
          videosCount: 0
       }

     const  videoSlice = createSlice({
        name: 'video',
            initialState,
        reducers: {
            addToList: (payload)=>{ videos.push(payload) }
        }
      })

    export  videoSlice.actions;
```

4. Configure a store from Redux toolkit

    - It requires configureStore() method
    - It can create a store at application level
    - It can get data from your reducer and update into store.
    - Store is provided Global so that you can access from any component.

```
Syntax:
     store.tsx

     import { configureStore }  from  "@redux/toolkit";

    export function configureStore(){
        // specify the reducers.
    }
```

5. Go to main.tsx and set provider for store.

```
Syntax:
        <Provider  store={ store } >
            <App />
        </Provider>
```

Redux in Video Library Project

1. Install Redux with React support into project

```
    > npm install  @reduxjs/toolkit   react-redux   --save
```

2. Go to "src" folder and add a new folder by name "slicers".

3. Add a new file "video-slicer.tsx"

4. Import "createSlice" and configure the slice with initial state and reducer actions.

```
5. Create Slice method comprises various properties
    a) actions
    b) reducer

   You have to export the actions and reducers

          video-slicer.tsx

import { createSlice } from "@reduxjs/toolkit";

const initialState = {
    videos : [],
    videosCount: 0
}

const videoSlice = createSlice({
    name: 'video',
    initialState,
    reducers: {
        addToSaveList : (state:any, action)=>{
            state.videos.push(action.payload);
            state.videosCount = state.videos.length;
        }
    }
});

export const {addToSaveList} = videoSlice.actions;
export default videoSlice.reducer;

6. Go to "src" and add a new folder by name "store"

7. Add a new file into store folder by name

            "store.tsx"

8. Configure store by using redux toolkit "configureStore" function.
    Store uses your video slicer and implements the reducers defined in slicer.

Note: The slicer is configured as default export, you have to import default
      and implement.

            store.tsx

import { configureStore } from "@reduxjs/toolkit";
import videoSlicer from "../slicers/video-slicer";

export default configureStore({
    reducer: videoSlicer
});

9. Go to "main.tsx" and configure the provider by importing store from react-
redux
    [ Provider locates value in memory and injects into component ]
```

    - Import provider & store [from your local store]

                main.tsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.tsx'
import '../node_modules/bootstrap/dist/css/bootstrap.css';
import '../node_modules/bootstrap-icons/font/bootstrap-icons.css';
import { CookiesProvider } from 'react-cookie';
import store from './store/store.tsx';
import { Provider } from 'react-redux';

createRoot(document.getElementById('root')!).render(
  <StrictMode>
     <CookiesProvider>
         <Provider store={store} >
            <App />
         </Provider>
     </CookiesProvider>
  </StrictMode>,
)
```

10. Go to your project user-dashboard component.

12. Import useDispatch(), which is responsible for dispatching the actions configured in reducer and update data into store.

13. Dispatch will carry the payload to store from component.

                user-dash.tsx

```
import { useEffect, useState } from "react";
import { useCookies } from "react-cookie"
import { Link, useNavigate } from "react-router-dom";
import { VideoContract } from "../contracts/video-contract";
import axios from "axios";
import { addToSaveList } from "../slicers/video-slicer";
import { useDispatch} from "react-redux";

export function UserDash(){

    const [cookies, setCookie, removeCookie] = useCookies(['user_id']);
    const [videos, setVideos] = useState<VideoContract[]>();

    let navigate = useNavigate();
    const dispatch = useDispatch();

    useEffect(()=>{
        axios.get(`http://127.0.0.1:4040/videos`)
        .then(response=>{
             setVideos(response.data);
        });
    },[])

    function SignoutClick(){
        removeCookie('user_id');
        navigate('/');
    }

    function AddToWatchLaterClick(video:VideoContract){
        dispatch(addToSaveList(video));
    }


    return(
        <div>
            <h3 className="d-flex mt-4 justify-content-
between"><span>{cookies['user_id']}  <button className="bi bi-plus btn">My
List</button> </span> <span>User Dash</span> <button onClick={SignoutClick}
className="btn btn-link">  Signout</button> </h3>
            <div className="my-3 w-50">
                <div className="input-group">
                    <input type="text" className="form-control"
placeholder="Search videos: Java, Aws, React" /> <button className="bi bi-search
btn btn-warning"></button>
                 </div>

            </div>
            <section className="d-flex flex-wrap">
                {
                    videos?.map(video=>
                        <div className="card m-2 p-2" style={{width:'300px'}}
key={video.video_id}>
                            <div className="card-header">
                                <iframe width="100%" height="200"
src={video.url}></iframe>
                            </div>
                            <div className="card-body">
                                <div className="fw-bold">{video.title}</div>
                                <p>{video.description}</p>
                            </div>
                            <div className="card-footer">
                                <button className="btn bi bi-hand-thumbs-up">
{video.likes} </button>
                                <button className="btn bi bi-hand-thumbs-down">
{video.dislikes} </button>
                                <button className="btn bi bi-eye-fill">
{video.views} </button>
                                <button className="btn bi bi-plus " onClick={()=>
{ AddToWatchLaterClick(video) } } > Watch Later</button>
                            </div>
                        </div>
                    )
                }
            </section>
        </div>
    )
}
```

Note : Download "redux-dev-tools" extension in your browser.

14. To access data from store you can import store in any component

```
    import store from "../store/store";

    store.getState().store.videos
    store.getState().store.videosCount
```

Implementing Callback & Memo in video library Project:
- Callback and Memo are used to save round trips.
- You can cache the data and use across multiple requests.
- Data is fetched from server only when there are changes identified on server.
- useMemo can store your data in memory.
- useCallback can store a function in memory.

Syntax:
```
   const ref = useMemo(()=>{
         // gets value
    },[dependency])
```

Testing & Deploying

- Testing is the process of verifying AS-IS & TO-BE.
- AS-IS refers to developer design & TO-BE is client requirement.

```
        AS-IS === TO-BE            => Test Pass
        AS-IS !== TO-BE            => Test Fail
```

- JavaScript based testing frameworks are required for testing React application
    - JEST
    - Jasmine Karma
    - VITest

- React application designed using JavaScript is enabled with JEST framework.

Testing a Component:
- Testing component comprises of 3 phases
    a) Arrange
    b) Act
    c) Assert

- Arrange is the process of configuring the component to test.
- Act defines the design and functionality to test.
- Assert is to verify test results and report the results.
- JEST framework provides various mock functions for arrange, act and assert

test()	It configures a case
render()	It renders the component to test
screen	It provides methods to access UI content

```
     getBy..()            These are reference methods
     toBe..()            These are assert methods

Ex:
weather.jsx
Weather Component
Client Requirements:
    - Title must be "Weatherman"
    - It must have a link to navigate to Google Weather
        <a> requires href with value "developers.google.com"

1. Add a new test file

        weather.test.js  [weather.spec.js]

2. Import the component and test functions

    import  { screen , render }  from  "@testing-library/react";
    import  { component   from "./component";

3. Configure the test case using "test()"

    test("title", ()=>{

    });

4. Render the component

    render(<Component/>);

5. Initialize the references for elements to test

    var  ref = screen.getBy..();

6. Configure the act

    expect(ref).toBe..();
    expect(ref).toHave..();

    weather.test.js

import { screen, render } from "@testing-library/react";
import { Weather } from "./weather";

// Title Test

test("Title Test",()=>{

    render(<Weather />);

    let title = screen.getByTestId("title");

    expect(title).toHaveTextContent(/Weatherman/);
});
```

```
// Link to Google test

test("Google Link Test", ()=>{

    render(<Weather />);

    let link = screen.getByText(/Google Weather/);

    expect(link).toBeInTheDocument();
    expect(link).toHaveAttribute("href", "https://developers.google.com&quot;);
});
```

7. Start testing

```
    > npm run test
```

Deploying
- It is the process of building application for production so that it can Go-Live.
- You can deploy on local server or cloud server.
- Local Servers
    a) IIS
    b) XAMP
    c) WAMP
    d) Tomcat etc..
- Cloud Servers
    a) Firebase
    b) AWS
    c) Azure
    d) Netlify
    e) GIT Hub Pages etc.

Ex: Deploying on Firebase [Google Cloud]

1. Login into your Firebase account with Google ID

    https://firebase.com

2. Go to "Console" and create a new project

     Name : weatherman-react-app

3. Install firebase tools on your PC

```
    C:\>npm install  firebase-tools  -g
```

4. Go to your project terminal

    - Build your application for production

```
        > npm run build
```

    - Login into firebase

```
        > firebase login

    - Initialize firebase

        > firebase init

       ?Firebase features do you want to set up :   Hosting
       ? Project : Use existing project
       ? Select Project : weatherman-react-app
       ? production folder : build
       ? re-write index.html : No

    - Deploy

        > firebase deploy

Note: After making changes

        > npm run build
        > firebase deploy
----------------------------------------------------
```