REACT-18

BY

-SUDHAKAR SHARMA

REACT INTRODUCTION

- Overview of frameworks, libraries for client side Web applications
- React version history
- What's New in React 18
- Understanding "what" and "why" React
- Environment Setup for React Application
- Understanding NPM commands
- Using VS Code
- VS Code extensions for ES6, React

REACT ESSENTIAL FEATURES AND SYNTAX

- React App Project Directory Structure
- Overview of Webpack, Babel
- React Component Basic
- Create React Component
- Understanding JSX
- Limitations of JSX
- Working with Components and Reusing Components
- Helloworld app in React

REACT COMPONENTS , PROPS AND STATE

- Functional/Class/Pure Components
- Understanding and using Props and State
- Handling Events with methods
- Manipulating the State
- Two way data-binding
- Functional (Stateless) VS Class (Stateful) Components
- Between component child communication
- Dynamically rendering contents
- Showing Lists, List and keys
- Refs

STYLING COMPONENTS

- CSS Styling
- Scoping Styles using Inline Styles
- Limitations of inline styes
- Inline Styles with Radium
- Using Psuedo classes/media quries with inline styles
- CSS Modules, importing css classes
- Adding Bootstrap, Semantic UI to React apps
- Using react-bootstrap, reactstrap packages

DEBUGGING REACT APPS

- Understanding React Error Messages
- Handling Logical Errors,
- Debugging React apps using google developer tools and React DevTool
- Understanding Error Boundaries

REACT COMPONENT LIFE CYCLE

- Updating life cycle hooks

- PureComponents
- React’s DOM Updating Strategy
- Returning adjacent elements
- Fragments

REACT HOOKS

- Introducing Hooks
- Hooks at a Glance
- Using the State Hook
- Using the Effect Hook
- Rules of Hooks
- Building Your Own Hooks
- Hooks API Reference
- Hooks FAQ

REACT COMPONENT IND ETAILS

- Higher Order Components
- Passing unknown Props
- Validating Props
- Using References
- React Context API
- Best practices for React Projects
- Demo apps

HTTP REQUESTS/AJAX C ALLS

- HTTP Requests in React
- Introduction of Axios package
- HTTP GET Request, fetching & transforming data
- HTTP POST, DELETE, UPDATE
- Handing Errors
- Creating/Using Axios intances

REACT ROUTING V5 and V6

- Routing and SPAs
- Setting Up the Router Package
- react-router vs react-router-dom
- Preparing the Project For Routing
- Switching Between Pages, Routing-Related Props
- The "withRouter" HOC & Route Props
- Passing & extracting route/query parameters
- Using Switch to Load a Single Route
- Navigating Programmatically

REACT FORMS AND FORM VALIDATION

- Creating a Custom Dynamic Input Component
- Setting Up a JS Config for the Form
- Dynamically Create Inputs based on JS Config
- Adding a Dropdown Component
- Handling User Input

- Handling Form Submission

- Adding Custom Form Validation
- Fixing a Common Validation
- Adding Validation Feedback
- Showing Error Messages
- Handling Overall Form Validity

DEPLOYING REACT APP TO THE WEB REACT REDUX

- Redux principles
- Install and setup redux
- Creating actions, reducer and store
- What is React Redux
- Why React Redux
- Install and setup
- Presentational vs Container components
- Understand high order component
- Understanding mapStateToProps and mapDispatchtToProps usage

REACT MATERIAL UI
ERROR HANDLING

NEW FEATURES OF REACT 18

- Automatic Batching
- Transitions
- Suspense Features
- New Strict Mode Behaviors

UNIT TESTING IN REACT

- Understand the significance of unit testing
- Understand unit testing jargon and tools
- Unit testing react components with Jest
- Unit testing react components with enzyme

WEBPACK PRIMER

- What is webpack
- Why webpack
- Install and setup webpack
- Working with webpack configuration file
- Working with loaders
- Quick word on code splitting, lazy loading, tree shaking
- Setting up Hot Module Replacement

SERVER -SIDE RENDERING WITH REACT

- What is server-side rendering (SSR)?
- Why SSR
- Working with render To String and render To Static Markup methods

DEPLOYING ON CLOUD

- Firebase Deployment

<u>26-10-2023</u>

What is React?
 What is React JS?
- React & React JS both a same.
- React is a java script library for building interactive UI.
- React is used for building Web & Native UI.
- Web is "Web Application"
- Native is OS native applications. [Android, iOS, Windows]

What is difference between React and Angular?
- React is a library.
- Angular is a Framework.

Where Angular is used?
- It is used in projects where lot of interactions and flow need to be controlled client side. Where React is used?
- It is used in project where lot of framework is already in use backend, we just need a good front end UI.

Why we need React & Angular?
- Modern web development have lot of challenges.
- Fluid UX
- Unified UX
- Loosely Coupled
- Simplified Deployment

What is Solution?
- Better build SPA [Single Page Applications]
- Can we build SPA with JS and JQ? Yes
- JS & JQ need lot of DOM manipulations
- Lot of coding
- Heavy
- Slow

What is solution?
- knockout js
- backbone js
- ember js
- vue js
- react js
- angular js &

angular 27-10-2023

Features of React:
1. It is component based.
    - Easy to reuse
    - Easy to extend
    - Easy to upgrade
    - Easy to build

2. It uses Virtual DOM

FAQ: **What is DOM**?
Ans :
- It is a hierarchy of elements in browser.
- DOM uses parsing phases

markup => bytes => chars => token => node => DOM => layout => render => paint

FAQ: **Where a button occurs in DOM hierarchy**?
Ans:
    window.document.forms[].elements[]

FAQ: What is Shadow DOM?
Ans:
- It is a hierarchy of elements in a component.
- Every component have a shadow
- Shadow comprises of root and child nodes.

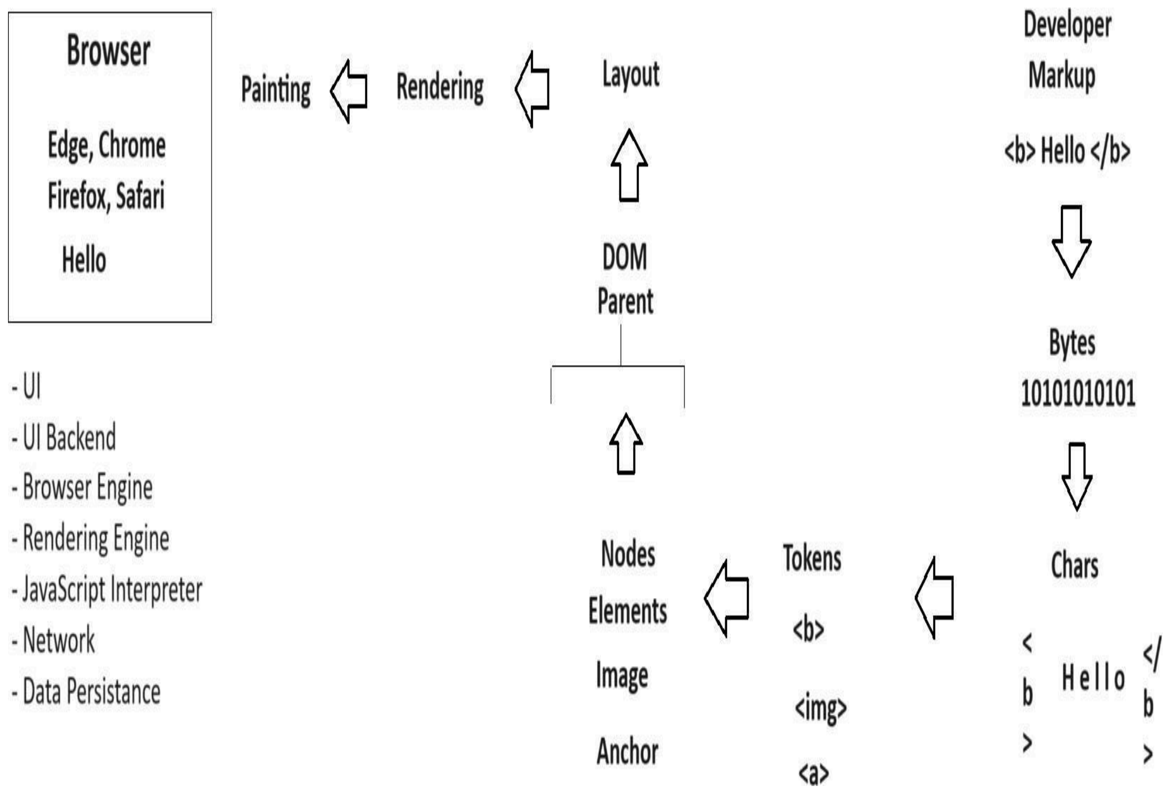
FAQ: **What is Virtual DOM**?
Ans:
- It is a copy of acutal DOM in memory.
- React uses virtual DOM that updates into actual DOM.
- It returns result before it updates to actual DOM.
- Hence it looks faster in rendering.

3. It is faster

4. It is modular

5. It uses less memory

6. It is light weight and easy to extend.



28/10/2023 , 29/10/2023 NO CLASSES
<u>30-10-2023</u>

What is React?
What are the Features of React?
1. Component Based
2. Virtual DOM
3. Modular
4. Faster
5. Light weight
What is DOM, Shadow DOM and Virtual DOM?
What are the issues with React?
- It is not designed for what you are using, hence lot of GAPs.
- You need lot of 3rd party libraries and frameworks support.
- Faster pace of development leads to poor documentation.

Where we can use React?
- You can use react in existing web application.
  [.net, php, python, jsp, node-express, HTML web app..]
- You can build a react application and integrate with existing
  application. [Distributed Applications]
- You can use react in Mobile Navite applications.
  [Android, ios, windows] [ionic, nativescript, cordova, react native]

React in Existing Web Application:

1. Setup Environment for React and Web

a) **Download and Install Node JS on your PC**. nodejs.org
It will get the package manager "NPM"

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

b) **Download and Install "Visual Studio Code" editor**

c) **Install following extentions for visual studio code**

a) Live Server
b) VsCode Icons
c) IntelliSense for CSS class names in HTML

2. **Create a new Web Application**

a) Create a new folder for your project on PC

D:\react-site

b) Open folder in VS Code

c) Open Terminal and Run the following command

```
>npm init -y
[package.json]
```

d) Add following folders into project

a) public : To keep static resources like html, images, videos, text documents, pdf, ppt, etc..

b)

.css, .scss

s
r
c

:
T
o
k
e
e
p
a
l
l
d
y
n
a
m
i
c
r
e
s
o
u

r	s
c	,
e	.
s	t
l	s
i	,
k	e
e	t
,	c
.	.
j	.

c) Add following files into public folder

index.html
home.html

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

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

Ex:
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>
   <h1>Welcome to Our Website</h1>
   <p>This page is not using React.</p>
   <p>React is in our <a href="home.html">Home</a></p>
</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 17", document.getElementById("root"));
   </script>
</head>
<body>
```

```
    <noscript>Please enable JavaScript.</noscript>
    <h2>Home</h2>
    <div id="root"></div>
</body>
</html>
```

31-10-2023

Creating React 18 Application

- React 18+ version re-written the library.
- React 18 introduces new features that improves the library with functionality and approach.
- You can create a complete React 18 application using various bundling tools.
- There are various bundling tools like
    a) Webpack
    b) Vite
    c) Parcel etc..
- If you have installed NPM as package manager then Webpack is in-built available.
- To create a new React 18 application with webpack you have to run the following command from your command prompt.

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

Ex:

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

- Open project folder in Visual Studio Code

React Project Infrastructure

File / Folder	Description
node_modules	It comprises of library resources. It contains the files related to various libraries that are installed in your project using NPM.
public	It comprises of static resources [html, images, text , pdf, videos etc..]
src -	It comprises of dynamic resources [.css, .js, .ts, .scss, .tsx, .jsx..]
.gitignore	It comprises of configuration of resources which are to

be ignored while collabrating with GIT repository.

package.json	It comprises of project meta data. [name, version, license, dependencies..]
package-lock.json	It comprises of dependencies meta data.
Readme.md	It is help documentation for project.

- To start react application run the following command in terminal

```
> npm start
```

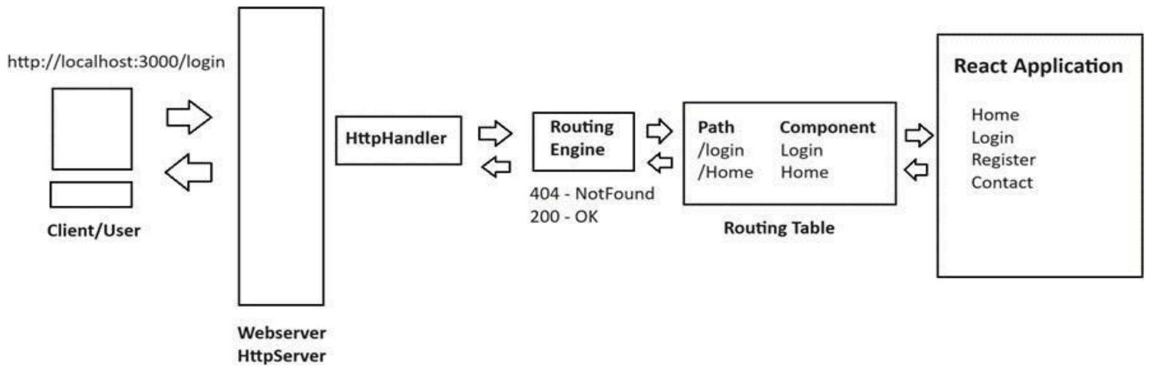
- Open any browser and request the following

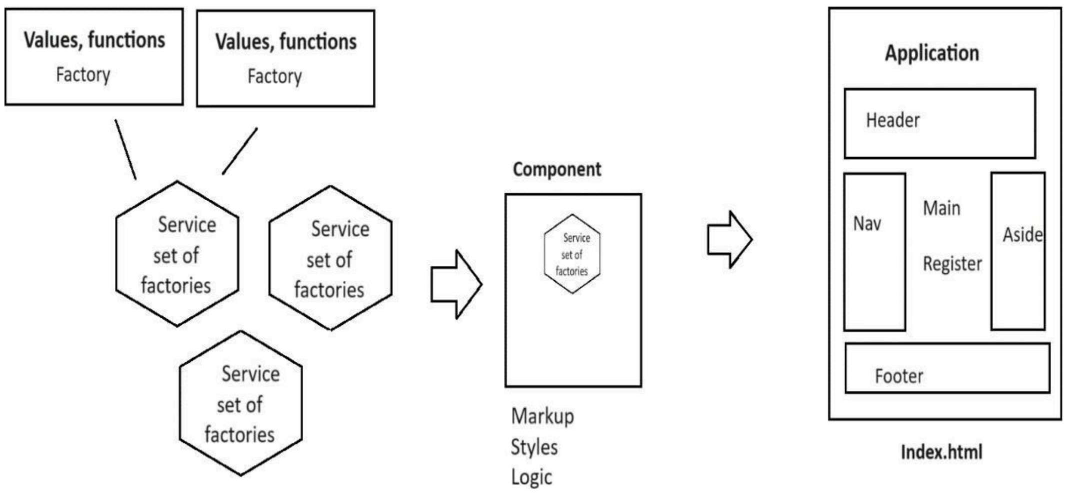
http://localhost:3000

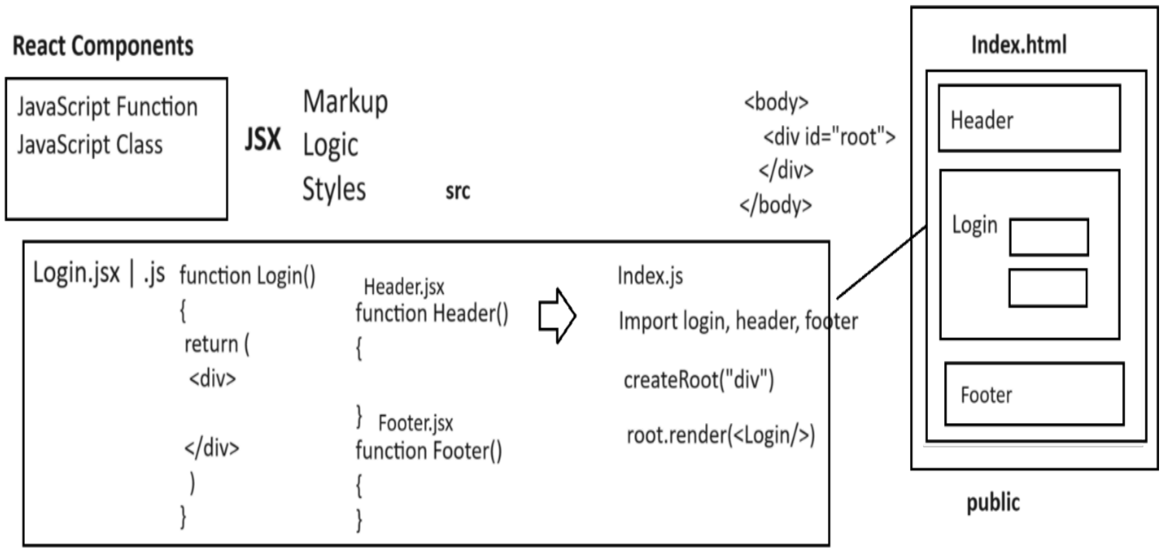
- To change the initial output you have to modify the function "App()" src / app.js

```
function App()
{
  return (      <div>
                                      <h1> Welcome to React </h1>
                </div>
                )

}
```

1-11-2023







Summary:

```
>npx create-react-app    shopping-react
```

1. Go to src folder and add a new file

"login.jsx"

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

2. Go to index.js

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

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

3. Run app

```
                >       npm start

                 http://localhost:3000
```

JavaScript Topics
- Modules
- Functions

2-11-2023

JavaScript Functions
1. Function Declaration
2. Function Expression
3. Function Closure
4. Function Parameters
5. Function Callbacks
6. Function Promises
7. Function Return
8. Function Recursion
9. Arrow Functions

FAQ: Why we need a function?
Ans: Function is used to refactor the code.

FAQ: What is Refactor?
Ans: Refactoring is the process of encapsulating a set of statements and storing under the reference of a function or file, so that you can reuse the statements.

FAQ: How a function is configured?
Ans: There are 2 ways of configuring a function in JavaScript.

a) Function Declaration
b) Function

Expression Syntax: Declaration

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

Syntax: Expression

```
const  Name = function() {

}
```

Note: A function expression allows to change the functionality according to state and situation. It uses IIFE pattern. [Immediately Invoked Function Expression]

IIFE is used for anonymous functions and function expressions.

Syntax:

```
(function() {

 })();
```

c

o

n

s

t

h

e

l

l

o

=

f

u

n

c

t

i

o

n

(

)

{

}

;

h

e

l

l

o

(

)

;

FAQ: What is function declaration, signature and definition?

Ans : Every function configuration comprises of 3 phases

a) declaration
b) signature
c) definition

```
function Hello(params)
{
}

function Hello(params)                    => Declaration
```

```
            Hello(params)                                  => Signature
            { }                                                  => Definition
```

Ex:

```
<script>
    var result;
    var password = prompt("Enter Password");
    if(password==="admin"){
      result = function(){
         document.write("Success..");
      }
    } else {
      result = function(){
         document.write("invalid..");
      }
    }
    result();
</script>
```

FAQ: What is role of parameters in a function?
Ans : A parameter is used to modify the function.

FAQ: What type of parameters are allowed in a function?
Ans : Any type. Primitive, Non Primitive, function.

```
                        Primitive => number, string, boolean, null, undefined, symbol, bigint
                        Non => array, object, map, set
```

Ex:

```
<script>
   function Demo(id, name, stock, cities, rating, print){
      print();
      document.write(`
         id=${id} <br>
         Name=${name}<br>
         Stock=${stock} <br>
         Cities=${cities.join("-->")} <br>
         Rating: ${rating.rate} [${rating.count}] <br>
      `);

   }
   Demo(1, "TV", true, ["Delhi", "Hyd"], {rate:4.2, count:3000},
function(){document.write("Hello<br>")});
</script>
```

FAQ: What is the max limit of parameters?
Ans: 1024

FAQ: What is rest parameter?
Ans: A single rest parameter can allows multiple arguments.
Every function can have only one rest parameter.
Rest parameter must be the last parameter in formal list.
It is defined by using "...paramName".

Ex:
```
<script>
   function Demo(title,...details)
   {
       var [id, name, price, stock] = details;
       document.write(`
           <h2>${title}</h2>
           Id=${id} <br>
           Name=${name} <br>
           Price=${price} <br>
           Stock=${stock}
       `);
   }
   Demo("Product Details",1, "TV", 54600.44, true);
</script>
```

FAQ: What is "spread" syntax for parameters?
Ans: It allows to spread single actual parameter values into multiple formal parameters.

FAQ: What is difference between spread and rest?
Ans: Rest is about formal parameters.
Spread is about actual parameters.

Ex:
```
<script>
  function Demo(id, name, price)
  {
      document.write(`
         ID = ${id} <br>
         Name = ${name} <br>
         Price = ${price}
      `);
  }
  var details = [1, "Mobile", 45000.33];
  Demo(...details);
</script>
```

FAQ: Why a function requires "return"?
Ans: A function with return is used to build expressions.
Expression comprises of logic and data storage.
It allows to use the memory of function before it is destroyed.

Ex:
```
<script>

   function Addition(a, b)
   {
      return a + b;
   }
   function Result(){
      document.write("Addition=" + Addition(20,30));
   }
   Result();
</script>
```

FAQ: What is a function can return?
Ans : A function can return any type, primitive, non-primitive or function.

Ex:
```
<script>
   function Hello(){
       return
       function(){
         return "Welcome to React";
       }
   }
   document.write(Hello()());
</script>
```

4-11-2023 5-11-2023 NO CLASSES

6-11-2023

Creating a new React Application:

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

React Components

- A component is a template with markup, styles and functionality.
- Markup is defined with HTML.
- Styles with CSS.
- Functionality with JavaScript or TypeScript.
- A component can be designed in React using

a) JavaScript Function
b) JavaScript Class

Designing a Component with JavaScript Function:
1. A component function can be configured using
a) Expression
b) Declaratio

n const Login = function() {

}
(or)
export function Login() {

}
2. Every function component must return mark. It can't be void. export function Login() {
```
    return (
        <markup> </markup>
    )
}
```

3. Function uses JSX as language hence there are certain rules to follow for JSX.

a) A function must return markup only as one fragment.

```
return(
      <h1>React</h1>
invalid
      =>

        <p> JavaScript Library </p>
)

return(                        <div>
                                  <h1>React</h1>

                                  =>
valid
                                  <p> JavaScript Library </p>
)                              </div>

return(

                               <>
                                 <h1> React </h1>

)                                => valid
                                 <p> JavaScript Library </p>
return(                        </>
<React.Fragment>
        <h1> React </h1>
                                        => valid
        <p> JavaScript Library </p>
</React.Fragment>
)
```

b) It can't have void elements, every element must have and end token.

```
return(  <div>
           <img src="pic.jpg">            => invalid
           <input type="text">            => invalid
         </div>
         )
```

```
return(  <div>
           <img src="pic.jpg"> </img>
                                              => valid
           <input type="text"> </input>
         </div>
         )
                  (or)
return(  <div>
           <img src="pic.jpg" />
           <input type="text" />
                                              => valid
         </div>
         )
```

c) You can't use attributes for elements, only properties are allowed.

```
<img src="" />                 => valid
<img class="" />               => invalid
<img className="" />           => valid
```

4. All components must be inside "src" folder.

5. Typically every component comprises 3 files in real-world application

a) login.js
b) login.css
c)
  login.test.js
  [login.spec.js]

- ".js" file comprises of markup and logic
- ".css" file comprises of styles

- ".test.js" comprises of test cases.

Note: Markup and logic can be defined in ".jsx" or ".js"

Ex: Design a register component for React Application and Start with Register.

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

2. Add "register" folder into components

3. Add following files into register folder

register.jsx
register.css

```
  register.jsx
import "./register.css";
export function Register()
{
   return(
      <>
        <main>
          <form>
             <h2>Register User</h2>
             <dl>
                <dt>User Name</dt>
                <dd><input type="text" /></dd>
                <dt>Password</dt>
                <dd><input type="password"></input></dd>
                <dt>Age</dt>
                <dd><input type="number" /></dd>
                <dt>Email</dt>
                <dd><input type="email" /></dd>
             </dl>
             <button className="btn-register">Register</button>
          </form>
        </main>
      </>
   )
}
register.css
form {
   border:1px solid gray;
   box-shadow: 2px
   black; padding: 20px;
```

```
    border-radius: 20px;
    width: 20%;
}
main {
    display: flex;
    justify-content:
    center; align-items:
    center; height: 100vh;
}
.btn-register {
    background-color:
    black; color:white;
    font-size: 20px;
    padding: 5px;
    width: 100%;
}
input {
    width: 100%;
    font-size:
    20px;
}
```

4. Go to index.js in src folder

```
import { Register } from './components/register/register';

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

7-11-2023

Setup Bootstrap for React Project:

1. Install Bootstrap and Bootstrap Icons from terminal

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

2. To use bootstrap and icons in all components, you have to import in "index.js"

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

3. You can use bootstrap and its icons in any component by refering to their class names.

Ex:

register.jsx

```
export function Register()
{
   return(
      <>
        <main className="d-flex justify-content-center mt-4">
          <form className="border border-2 border-dark rounded p-4">
             <h2 className="bi bi-person-fill">Register User</h2>
             <dl>
                <dt>User Name</dt>
                <dd><input type="text" className="form-control" /></dd>
                <dt>Password</dt>
                <dd><input type="password" className="form-control"></input></dd>
                <dt>Age</dt>
                <dd><input type="number" className="form-control" /></dd>
                <dt>Email</dt>
                <dd><input type="email" className="form-control" /></dd>
             </dl>
             <button className="btn btn-primary w-100">Register</button>
          </form>
        </main>
      </>
   )
}
```

Netflix Design:
1. Add a new folder into project "src" by name "Netflix"

2. Add following components hierarchy

```
                    -netflix-index
                              netflix-index.jsx
                              netflix-index.css
                    -netflix-header
                              netflix-header.jsx
                              netflix-header.css
                    -netflix-main
                              netflix-main.jsx
                              netflix-main.css
                    -netflix-register
                              netflix-register.jsx
                              netflix-register.css
```

Note: You can access and use any component in another component by following 2 steps
Step-1: Import the component library

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

Step-2: You can access by using component selector

```
<NetflixHeader> </NetflixHeader>
                    (or)
<NetflixHeader />
```

NETFLIX REGISTER:-

NETFLIX.REGISTER.JSX:-

```
export  function NetflixRegister()
{
   return(
      <form className="mt-4">
         <p>Ready to watch? Enter your email to create or restart your membership.</p>
         <div className="input-group input-group-lg">
            <input type="email" placeholder="Your email address" className="form-
control"/>
            <button className="btn btn-danger ms-2">
               Get Started <span className="bi bi-chevron-right"></span>
            </button>
         </div>
      </form>
   )
}
```

NETFLIX-HEADER:-

NETFLIX HEADER CSS:-

```
.brand-title
   { font-size: 30px;
   color:red;
   font-weight: bold;
   font-family: Arial;
}
```

NETFLIX-HEADER.JSX:

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

export function NextflixHeader()
{
     return(
       <header className="p-4 text-white d-flex justify-content-between">
          <div>
             <span className="brand-title">NETFLIX</span>
          </div>
          <div className="d-flex">
             <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>
             <button className="btn btn-danger ms-3">Signin</button>
          </div>
       </header>
     )
}
```

NETFLIX-INDEX:-

NETFLIX-INDEX.CSS

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

NETFLIX-INDEX.JSX:

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

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

export function NetflixIndex()
{
   return(
      <div className="bg-image">
         <div className="bg-shade">
             <NextflixHeader />
             <NetflixMain />
```

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

NETFLIX MAIN

NETFLIX MAIN.CSS

```
main {
    display: flex;
    flex-direction:
    column;
    justify-content: center;
    align-items: center;
    height: 400px;
    color:white;
}
.main-title {
    font-size: 40px;
    font-weight: bold;
    font-family: Arial;
}
.main-subtitle {
    font-size:
    30px;
    font-family: Arial;
}
```

NETFLIX.INDEX.JSX

```
      import { NetflixRegister } from '../neflix-register/netflix-register';
import './netflix-main.css';

export function NetflixMain()
{
    return(
        <main>
            <div className="main-title">The biggest Indian hits. Ready to watch here from ₹
149.</div>
            <div className="main-subtitle">Join today. Cancel anytime.</div>
            <NetflixRegister />
        </main>
    )
}
```

8-11-2023

Data Binding in React

- Data Binding is the process of accessing data from source and render in UI, identifying the changes in UI and updating back into data source.

- Data Binding is classified into 2 types

  a) One Way Binding
  b) Two Way Binding

- Trygve introduced the concept of data binding in applications, which are separated into 3 components called
  a) Model
  b) View
  c) Controller

- Model represents data.
- View represents UI.
- Controller represents application logic.
- Controller handles communication between model and view.
- One Way Binding allows application to access data implicitly and binding UI, but will not allow to update the changes back to model.
- React supports only "one way binding".
  * It is secured
  * It uses less memory
  * It is faster in rendering
  * It improves the performance of application
  * However it is not good for updating the data source.
  * It requires explicit actions.
- React can handle one way data binding without using DOM manipulations or DOM methods.
- React uses a binding expression "{ }" to bind dynamic

  data to UI. JavaScript Data Binding expression "${ }"

- React uses JavaScript data types
  a) Primitive
    number, string, boolean, null, undefined, symbol, bigint
  b) Non Primitives
    array, object, map
  c) Date type
  d) Regular Expression Type

Syntax:

```
var userName = "John";

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

Ex: Primitive Types

data-binding.jsx

```
export function DataBinding()
{
   var userName =
   "John"; var age = 22;
   var subscribe = true;
   return(
      <>
        <h2>Data Binding</h2>
        <p>Hello ! {userName} you will be {age+1} next year. </p>
        <p> {(subscribe===true)?"Thank you for subscribing":"Please subscribe to our
channel"} </p>
      </>
   )
}
```

Ex: Array
- React uses Array methods to read elements from array and present in UI.
- Array methods used for reading all elements
  a) forEach()
  b) map()
  c) toString()
  d) join()

data-binding.jsx

```
export function DataBinding()
{
   var categories = ["All", "Electronics", "Footwear", "Fashion"];

   return(
      <>
        <nav className="bg-dark text-white p-2 d-flex justify-content-between">
          {
             categories.map(category=>
                <span key={category} className="me-4">{category}</span>
                )
          }
        </nav>
        <div className="btn-group-vertical">
          {
             categories.map(category=>
                <button key={category} className="btn btn-danger mb-1 mt-
1">{category}</button>
                )
          }
        </div>
        <ol>
          {
            categories.map((category)=>
                <li key={category}>{category}</li>
            )
          }
        </ol>
        <select>
          {
            categories.map(category=> <option key={category}>{category}</option> )
          }
        </select>
        <ul className="list-unstyled">
          {
            categories.map(category =>
                <li key={category}>
                    <input type="checkbox"/> <label>{category}</label>
                </li>
                )
          }
        </ul>
        <table className="table table-hover w-25">
          <thead>
            <tr>
                <th>Categories</th>
            </tr>
          </thead>
          <tbody>
            {
                categories.map(category =>
                    <tr key={category}>
                        <td className="d-flex justify-content-between"><span>{category}</span>
<button className="btn btn-danger bi bi-trash"></button> </td>
                    </tr>
                    )
            }
          </tbody>
        </table>
       </>
   )
}
```

Ex: Object, Array of Objects, Date, Regular Expression

9-11-23

Object Type [JSON]

- It is a key | value collection.
- Key is string type
- Value can be any type.

```
    var product = { Id:1, Name:"TV", InStock:true, Cities:["Delhi", "Hyd"], {Rate:4.3,
Count:3400} };

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

Ex:
data-binding.jsx

```
export function DataBinding()
{
    var product = {
        Name: "Samsung TV",
        Price: 45000.33,
        Stock: true,
        Cities: ["Delhi", "Hyd"],
        Rating: {Rate:3.5, Count:5000}
    }

    return(
        <>
          <h2>Product Details</h2>
          <dl>
            <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>Shipped To</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 text-success"></span>
[{product.Rating.Count}]
            </dd>
          </dl>
        </>
    )
}
```

FAQ's
:
1. What is difference between Object and Map
type? A.

Object	Map
Key and Value collection	Key and Value collection
Key can be only string type	Key can be any type
Value can any type	Value can be any type
It requires explicit operators, properties and methods to manipulate.	It provides several implicit properties and method to manipulate.
It is slow.	It is fast.
It is structured.	It is schema less.

2. How to remove a key from object?
A. By using delete operator

delete product.Price;

3. How to verify a key?
A. By using "in" operator.

4. How to access all keys?
A. By using "Object.keys()", or "for..in" iterator

5. What are the various Map
members? A.

set()	Add a new key and value
get()	To access a value with reference of key
keys()	Returns all keys
values()	Returns all values
has()	Returns true if specified key exits
delete()	Removes a value with reference of key
clear()	Removes all keys and values
entries()	Return all keys and values
size	Returns the total count of keys.

Ex:
data-binding.jsx

```
export function DataBinding()
{
    var product = {
        Name: "Samsung TV",
        Price: 45000.33,
    Stock: "true"
};
return(
    <>
       <h2>Product Details</h2>
```

```
{Object.keys(product).map(key=><p>{key}:{product[key]}
}
</>
    )
}
```

```
<                                                                   >
/                                                                   )
p
```

Array of Objects:

data-binding.jsx

```
export function DataBinding()
{
    var menu = [
        {Category: "Electronics", Products:["Televisions", "Mobiles", "Watches"]},
        {Category: "Fashion", Products: ["Men Fashion", "Kids Fashion", "Women Fashion"]}
    ];

    return(
        <>
          <ol>
              {                                                   menu.map(item=>
                                                                    <li key={item.Category}>{item.Category}
                                                                      <ul>
                                                                          {
                                item.Products.map(product =>
                                  <li key={product}>{product}</li>
                                 )
                              }
                                                                       </ul>
                                                                     </li>
                                                                     )
                                            }
                                          </ol>
                                        </>
                                     )
                                  }
```

Ex: Data List

data-binding.js

x

```
import React from "react";

export function DataBinding()
{
    var topics = [
         {Title: "HTML", Description: "It is a markup language."},
         {Title: "CSS", Description: "It configure styles for HTML."}
    ];

    return(
        <>
```

<dl>

```
            {
               topics.map(topic=>
                     <React.Fragment key={topic.Title} >
                     <dt key={topic.Title}>{topic.Title}</dt>
                     <dd key={topic.Description}>{topic.Description}</dd>
                     </React.Fragment>
                   )
            }
          </dl>
        </>
    )
}
```

Ex: Tabular Data data-binding.jsx

```
import React from "react";

export function DataBinding()
{
    var iccTable = [
        {Team: "India", Matches: 8, Won: 8, Lost: 0, PTS: 16, NRR: "+2.456", Flag:"india.jpg"},
        {Team: "South Africa", Matches: 8, Won:6, Lost:2, PTS: 12, NRR: "+1.376",
Flag:"sf.jpg"},
        {Team: "Australia", Matches: 8, Won: 6, Lost: 2, PTS: 12, NRR: "+0.861",
Flag:"aust.jpg"},
    ]

    return(
        <>
         <h1>ICC Worlcup 2023 Table</h1>
         <table className="table w-50 table-hover table-striped">
             <thead>
                <tr>
                    <th>Team</th>
                    <th>Matches</th>
                    <th>Won</th>
                    <th>Lost</th>
                    <th>PTS</th>
                    <th>NRR</th>
                </tr>
             </thead>
             <tbody>
                {
                    iccTable.map(country =>
                        <tr key={country.Team}>
                            <td> <img src={country.Flag} width="40" /> {country.Team}</td>
                            <td>{country.Matches}</td>
                            <td>{country.Won}</td>
                            <td>{country.Lost}</td>
                            <td>{country.PTS}</td>
                            <td>{country.NRR}</td>
                        </tr>
                )
        }
     </tbody>
 </table>
</>
```

```
    )
}
```

Ex: Flipkart

data-binding.js

x

```
import React from "react";

export function DataBinding()
{
    var mobiles = [
        {Title: "APPLE iPhone 14 (Blue, 128 GB)", Price: 57999, Rating: 4.6, Features: ["128 GB ROM", "15.49 cm (6.1 inch) Super Retina XDR Display", "12MP + 12MP | 12MP Front Camera", "A15 Bionic Chip, 6 Core Processor Processor", "1 Year Warranty for Phone and 6 Months Warranty for In-Box Accessories"], Photo:"iblue.jpg"},
        {Title: "APPLE iPhone 14 (Starlight, 128 GB)", Price: 57999, Rating: 4.6, Features: ["128 GB ROM", "15.49 cm (6.1 inch) Super Retina XDR Display", "12MP + 12MP | 12MP Front Camera", "A15 Bionic Chip, 6 Core Processor Processor", "1 Year Warranty for Phone and 6 Months Warranty for In-Box Accessories"], Photo:"starlight.jpg"}
    ]

    return(
        <>
            {
                mobiles.map(item =>
                    <div key={item.Title} className="row p-2 m-2">
                        <div className="col-2">
                            <img src={item.Photo} width="110%" />
                        </div>
                        <div className="col-7">
                            <div className="h4 text-primary">{item.Title}</div>
                            <div className="bg-success rounded rounded-pill p-1 text-center text-white" style={{width:'70px'}}>{item.Rating} <span className="bi bi-star-fill text-white"></span> </div>
                            <div className="mt-3">
                                <ul>
                                    {
                                        item.Features.map(feature =>
                                            <li key={feature}>{feature}</li>
                                        )
                                    }
                                </ul>
                            </div>
                        </div>
                        <div className="col-3">
                            <div className="h1">{item.Price.toLocaleString('en-in', {style: 'currency', currency: 'INR'})}</div>
                        </div>
                    </div>
                )
            }
        </>
    )
}
```

Data binding and State

Date Type:
- Date is stored in memory using "Date()".
- The default date format is "Year-Month-Day Hrs:Min:Sec.MilliSec"

Syntax:

```
var departure = new Date();                              => Load current date and time.
var departure = new Date("2023-11-22 18:30:35.67");  => Specific date and time
```

- To present date and time values JavaScript provides following methods

getHours()	0 to 23
getMinutes()	0 to 59
getSeconds()	0 to 59
getMilliSeconds()	0 to 99
getDate()	1 to 28, 29, 30, 31
getDay()	0 = Sunday
getMonth()	0 = January
getFullYear()	2023
toLocaleDateString()	
toLocaleTimeString()	

- You can dynamically change date and time values in JavaScript using
  setHours()
  setMinutes()
  setSeconds()
  getMilliSeconds()
  getDate()
  getMonth()
  setFullYear()

Ex:
data-binding.jsx

```
import React from "react";
export function DataBinding()
{
    var departure = new Date();
    var months = ["Jan", "Feb", "March", "April",
"May","June","July","Aug","Sep","Oct","November", "Dec"];
    var weekdays = ["Sunday","Mon","Tue","Wed","Thu","Friday","Sat"];
    return(
        <>
          Departure : {weekdays[departure.getDay()]}, {departure.getDate()}
{months[departure.getMonth()]}  - {departure.getFullYear()}
        </>
    )
}
```

Note: Never use variables to store dynamic data in React.

Variables are immutable in React.
Component requires mutable data. Hence variables are not recommended.

FAQ: Where to store data for a component?
Ans : Always use a "State" for storing data.

State in Component

- Web applications use "http" as protocol, which is a state less protocol.
- State less uses the mechanism "Go-Get-Forget".

```
GO        - Establish connection with server
GET        - Get response from server
FORGET - Remove all traces about the request from server.
```

- State less nature is good for web applications, as they can manage memory very well.
- It is a draw back to continues of operations, as it can't remembers the data or previous request and provide to next request.
- Hence web application requires to implement various state management techniques both client side and server side.
- Some of client side state management techniques in JavaScript
  - a) Query String
  - b) Local Storage
  - c) Session Storage
  - d) Cookies
- React components require state to store the data and use across request.
- State is mutable.
- React Class Components are statefull, they provides state implicitly.
- React 18+ versions support state for function components.
- State in function components is defined by using the Hook called "useState()".

FAQ: Can we use state hook in class component?
Ans: No. Hooks are allowed only in a function component.

Syntax:

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

Note: useState hook is a member of "react" module

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

```
const [userName] = useState('John');
<p> Hello ! { userName } </p>
```

FAQ's:
1. Why const is used for configuring state? Can we use var or let for state?
A. State should not allow to assign values, hence it is configured as "const".

State must be initialized can't be assigned.
You can't use var or let as they allow assignment.

2. If you use const for state then how to store value into state?
A. Every time to store value into state it must be initialized.

Syntax:

```
        const [userName, setName] = useState(' ');
        setName("John");          // valid => Initializing memory
        setName = "John";          // invalid => Assigning
```

3. Why it is "useState()" not "new useState()" ?

```
        const [userName] = new useState(); [memory allocation operator]
        const [userName] = useState();
```

A. The "new" operator is dynamic memory allocation operator, which uses
"Single Call" mechansim.
useState() uses "Single ton" mechanism, object is created for first request and the same object is used across any number of requests.

Ex:
data-binding.jsx

```
import { useState } from "react";
export function DataBinding()
{
    const [title] = useState('Product Details');
    const [product] = useState({Name:'Samsung TV', Price:456000.44}); const
    [categories] = useState(["All", "Electronics", "Footwear", "Fashion"]);

    return(
        <>
          <h1>{title}</h1>
          <dl>
           <dt>Name</dt>
           <dd>{product.Name}</dd>
           <dt>Price</dt>
           <dd>{product.Price}</dd>
           <dt>Select Category</dt>
           <dd>
               <select>
                   {
                       categories.map(category=>
                           <option key={category}>{category}</option>
                           )
                   }
               </select>
           </dd>
          </dl>
        </>
    )
}
```

React Two Way Binding

Two Way Data Binding:
- It is the process of accessing data from source and binding to UI.
- Identifying the changes in UI and updating back into source.
- JavaScript handles 2-way binding using various events.
- React can use all JavaScript browser events.
- But changes in value can be identified only with "onChange" event.
- Hence two way data binding can be managed only with "onChange" event.

Syntax:
```
    <input type="text"  onChange={handleNameChange} />
    <select onChange={handleCityChange} />

    function handleNameChange()
    {
    }
    function handleCityChange()
    {
    }
```

- You have to use "event" argument to access information about element and event.
  [JavaScript uses "this" for object details & "event" for event details]

- React can use only one argument that is "event", which provides access to both object and event details.

```
    function handleNameChange(event)
    {
       event.clientX;        ]
       event.clientY;        ]    Event Details
       event.keyCode;        ]
       event.which; etc..    ]

       event.target.id         ]
       event.target.value      ]    Object details
       event.target.className]
       event.target.name       ]
    }
```

- You can access the value from element and initialize into state reference by using setter.

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

    function handleNameChange(e)
    {
      setUserName(e.target.value);
    }
    <input type="text" value={userName} onChange={handleNameChange} />
```
Ex:

data-binding.jsx

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

```
export function DataBinding()
{
    const [userName, setUserName] = useState('John');

    function handleNameChange(e){

        setUserName(e.target.value);
    }

    return(
        <>
          <h2>Edit</h2>
           User Name : <input type="text" value={userName} onChange={handleNameChange}
/>
           <p>Hello ! {userName}</p>
        </>
    )
}
```

Ex: Two Way Binding with various elements - Directly onChange

data-binding.jsx

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

export function DataBinding()

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

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


    return(
        <>
          <div className="container-fluid">
          <h2>Product Details</h2>
           <section className="row">
              <nav className="col-3">
                <dl>
                <dt>Name</dt>
                <dd><input type="text" onChange={NameChange} value={product.Name}
className="form-control" /></dd>
                <dt>Price</dt>
                <dd><input type="number" onChange={PriceChange} value={product.Price}
className="form-control" /></dd>
                <dt>City</dt>
                <dd>
                    <select onChange={CityChange} value={product.City} className="form-select">
                        <option>Select City</option>
                        <option>Delhi</option>
                        <option>Hyd</option>
                    </select>
                </dd>
                <dt>Stock</dt>
                <dd className="form-switch">
                    <input type="checkbox" onChange={StockChange} checked={product.Stock}
className="form-check-input" /> <label className="form-check-lable">Available</label>
                </dd>
                </dl>
              </nav>
              <main className="col-9 text-dark">
                 <dl>
                    <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>Shipped To</dt>
                    <dd>{product.City}</dd>
                 </dl>
              </main>
           </section>
          </div>
        </>
    )
```

```
}
```

Ex: Two Way Binding on Button Click

data-binding.jsx

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

export function DataBinding()
{
    const [product, setProduct] = useState({Name:'TV', Price:0, City:'Select City', Stock:false});
    const [updatedProduct, setUpdatedProduct] = useState({Name:'', Price:0, City:'',
Stock:false});


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

    function handleUpdateClick(){
         setUpdatedProduct(product);
    }

    return(
        <>
          <div className="container-fluid">
          <h2>Product Details</h2>
           <section className="row">
        <nav className="col-3">
          <dl>
          <dt>Name</dt>
          <dd><input type="text" onChange={NameChange} value={product.Name}
className="form-control" /></dd>
```

```
                <dt>Price</dt>
                <dd><input type="number" onChange={PriceChange} value={product.Price}
className="form-control" /></dd>
                <dt>City</dt>
                <dd>
                    <select onChange={CityChange} value={product.City} className="form-select">
                        <option>Select City</option>
                        <option>Delhi</option>
                        <option>Hyd</option>
                    </select>
                </dd>
                <dt>Stock</dt>
                <dd className="form-switch">
                    <input type="checkbox" onChange={StockChange} checked={product.Stock}
className="form-check-input" /> <label className="form-check-lable">Available</label>
                </dd>
                </dl>
                <button onClick={handleUpdateClick} className="btn btn-primary w-
100">Update</button>
              </nav>
              <main className="col-9 text-dark">
                 <dl>
                    <dt>Name</dt>
                    <dd>{updatedProduct.Name}</dd>
                    <dt>Price</dt>
                    <dd>{updatedProduct.Price}</dd>
                    <dt>Stock</dt>
                    <dd>{(updatedProduct.Stock==true)?"Available":"Out of Stock"}</dd>
                    <dt>Shipped To</dt>
                    <dd>{updatedProduct.City}</dd>
                 </dl>
              </main>
           </section>
          </div>
        </>
    )
}
```

Data Binding
a) One Way Binding
b) Two Way Binding

Style Binding
Class Binding
Event Binding

React Style and Class binding

Summary - Databinding

- One Way Binding
- Two Way

Binding Style

Binding

- Style Binding is the process of binding CSS attributes to any JSX element dynamically .
- It is required to change appearence of element

dynamically. JavaScript Style Binding:

```
    <input type="text" id="UserName">
    document.getElementById("UserName").style.cssAttribute = "value";
    CSS => background-color => backgroundColor
        => font-style => fontStyle
```

React Style Binding:

```
    <input type="text" style={ {attributeName:value, attributeName:value} } />

    <input type="text" style="color:red" />              => invalid
    <input type="text" style={ {color: 'red'} } />           => valid
```

Ex:
style-binding.jsx

```
export function StyleBinding()
{
    return(
        <div className="container-fluid">
            <h2>Style Binding</h2>
           <input type="text" style={{border:'2px solid red', boxShadow:'2px 2px 2px red'}} />
        </div>
    )
}
```

Note: Style attribute value must be always string
type. Style attributes are defined as JavaScript
object.

Ex:
style-binding.jsx

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

```
export function StyleBinding()

{
    const [validationStyle, setValidationStyle] = useState({border:'', boxShadow:''});
    const [displayStyle, setDisplayStyle] = useState({display:'none'});
    function handleNameChange(e){
        if(e.target.value.length<4) {
            setValidationStyle({
                border : '2px solid
                red',
                boxShadow: '2px 2px 2px red'
            })
        } else {
            setValidationStyle({  border:
                '2px solid green',
                boxShadow: '2px 2px 2px green'
            })
        }
    }

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

    return(
        <div className="container-fluid">
            <h2>Style Binding</h2>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" style={validationStyle} onChange={handleNameChange}
placeholder="Name Min 4 Chars" /></dd>
                <dt>Terms of Service</dt>
                <dd>
                    <textarea disabled>Our terms of service</textarea>
                </dd>
                <dd>
                    <input type="checkbox" onChange={handleCheckboxChange} /> <label>I
Accept</label>
                </dd>
                <dd>
                    <button style={displayStyle}>Submit</button>
                </dd>
            </dl>
        </div>
    )
}
```

FAQ: What is issue with style binding?
Ans : Style is inline and it is good for individual elements.

It is not good for reusability across elements.
It is not good for configuring multiple effects for element.

Class Binding

- It is the process of binding a CSS class dynamically to any JSX element.

- It allows to configure multiple styles simultaneously at the same time.
- A class is configure to JSX element by using "className".

Syntax:
```
       <div className={ dynamicValue } />

         dynamicValue = "class1 class2 class3.."
```

Ex:
class-binding.css

```
.dark-theme {
    background-color: black;
    color:white;
    padding: 20px;
}


form {
    border:1px solid gray;
    padding: 10px;
    border-radius: 20px;
}
```

class-binding.jsx

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

export function ClassBinding()
{
    const [theme, setTheme] = useState('w-25');

    function handleThemeChange(e){
        if(e.target.checked) {
            setTheme('dark-theme w-25');
        } else {
            setTheme('w-25');
        }
    }

    return (
        <div className="container-fluid d-flex justify-content-center
align-items-center" style={{height:'100vh'}}>
            <form className={theme}>
                <div className="form-switch">
                 <input type="checkbox" onChange={handleThemeChange} className="form-
check-input" /> <label className="form-check-label">Dark Mode</label>
                </div>
                <h2 className="bi bi-person-fill"> User Login </h2>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" className="form-control" /></dd>
                    <dt>Password</dt>
                    <dd><input type="password" className="form-control" /></dd>
                </dl>
                <button className="btn btn-dark w-100">Login</button>
```

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

Ex: Bootstrap Classes

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


export function ClassBinding()
{
    const [theme, setTheme] = useState('w-25');
    const [buttonStyle, setButtonStyle] = useState('btn btn-dark w-100');

    function handleThemeChange(e){
        if(e.target.checked) {
            setTheme('bg-dark text-white p-2 w-25 border border-2 rounded');
            setButtonStyle('btn btn-light w-100');
        } else {
            setTheme('w-25');
            setButtonStyle('btn btn-dark
            w-100');
        }
    }

    return (
        <div className="container-fluid d-flex justify-content-center
align-items-center" style={{height:'100vh'}}>
            <form className={theme}>
                <div className="form-switch">
                 <input type="checkbox" onChange={handleThemeChange} className="form-
check-input" /> <label className="form-check-label">Dark Mode</label>
                </div>
                <h2 className="bi bi-person-fill"> User Login </h2>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="text" className="form-control" /></dd>
                    <dt>Password</dt>
                    <dd><input type="password" className="form-control" /></dd>
                </dl>
                <button className={buttonStyle}>Login</button>
            </form>
        </div>
    )
}
```

Ex: Changing Bootstrap Icon using class binding

class-binding.jsx

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


export function ClassBinding()
{
    const [sortClass, setSortClass] = useState('bi bi-sort-alpha-down btn btn-danger');
    function handleSortClick(){
        setSortClass((sortClass=='bi bi-sort-alpha-down btn btn-danger')?'bi bi-sort-alpha-up btn
btn-danger':'bi bi-sort-alpha-down btn btn-danger');
    }

    return (
        <div className="container-fluid d-flex justify-content-center
align-items-center" style={{height:'100vh'}}>
            <button onClick={handleSortClick} className={sortClass}></button>
        </div>
    )
}
```

Task on Style Binding

style-binding.jsx

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


export function ClassBinding()
{
   const [fontFamily, setFontFamily] = useState('Arial');
   const [fontSize, setFontSize] = useState('20px'); const
   [fontColor, setFontColor] = useState('#000000');
   const [textStyle, setTextStyle] = useState({color:fontColor, fontSize:fontSize,
fontFamily:fontFamily});

   function handleFontSizeChange(e){
      setFontSize(e.target.value + 'px');
   }
   function handleColorchange(e){
      setFontColor(e.target.value);
   }
   function handleFontFamilyChange(e){
      setFontFamily(e.target.value);
   }


   function handleApplyClick(){
      setTextStyle({
         fontSize : fontSize,
         fontFamily: fontFamily,
         color: fontColor
      })
   }

   return (
      <div className="container-fluid">
        <fieldset className="w-25">
         <legend>Text Formatting</legend>
         <dl>
            <dt>Font Family</dt>
            <dd>
               <select onChange={handleFontFamilyChange} className="form-select">
                  <option>Select Font</option>
                  <option>Arial</option>
                  <option>Algerian</option>
               </select>
            </dd>
            <dt>Font Color</dt>
            <dd>
               <input onChange={handleColorchange} type="color" className="form-control-
```

```
color" />
```

```
</dd>
<dt>Font Size</dt>
<dd>
```

```
                    1 <input onChange={handleFontSizeChange} type="range" min="10"
max="100" className="form-range" /> 100
                </dd>
            </dl>
            <button onClick={handleApplyClick} className="btn btn-primary">Apply</button>
           </fieldset>
           <div align="center">
              <p style={textStyle}>
                 Sample Text
              </p>
           </div>
        </div>
    )
}
```

Event Binding

Event Binding in React

Event Binding in React

- Event is a message sent by sender to its subscriber in order to notify the change.
- Event follows a software design pattern called "Observer".
- Observer is a communication pattern. It is under "Behavioural Patterns".
- Event uses "Delegate" mechanism [Function Pointer].

Syntax:

```
function InsertClick()                 => Subscriber
{
}
<button onclick="InsertClick()">        => Sender
```

- Subscriber defines the actions to perform.
- Sender sends a notification and specifies when to perform the action. [Sender triggers the actions].

Synthetic Events:

- Every event is a browser event.
- In JavaScript HTML elements can use the browser events.
- React can't use "browser" events. [Browser events are related BOM]
- React uses a virtual DOM library called "SyntheticEvents", which map to browser events.

Actual DOM Event	Synthetic Event
onclick	onClick
onblur	onBlur
onchange	onChange etc..

- Every event extends "SyntheticEvent"

[interface] Event Handler:
- Events are configured for elements using an EventListener or EventHandler.
- React uses EventHandler from SyntheticEvents.

Syntax:

```
<button onClick={InsertClick}>

onClick={InsertClick}                 =>
EventHandler onClick        => Event

onChange     : ChangeEventHandler      : EventHandler
onClick      : ClickEventHandler      : EventHandler
onFocus      : FocusEventHandler     : EventHandler
```

- Event handler can be configured in ways

  a) Inline functionality
  b) Embedded functionality

Syntax: Inline

```
    <button onClick={function(){ alert("Insert Clicked"); }}>  Insert </button>
    <button onClick={ ()=> {alert("Insert Clicked"); }}> Insert </button>
```

Syntax: Embedded

```
    function InsertClick()
    {
    }

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

Event Arguments
- Every event in React have a default argument, which maps to "Event" base.
- Event base can provide access to both element related properties and event properties.
- Element Properties are accessible using "target"

```
        e.target.Id
        e.target.name
        e.target.className
        e.target.value etc...
```

- Event Properties are accessible

```
        directly e.clientX

        e.clientY
        e.shiftKey
        e.which
        e.ctrlKey etc...
```

Ex:
event-binding.jsx

```
export function EventBinding(){
    function Database(e){
        console.log(`
            X Position : ${e.clientX} \n
            Button Name: ${e.target.name}
        `);
    }

    return(
        <div className="container-fluid">
            <h2>Event Binding</h2>
            <button name="Insert" onClick={Database}>Insert</button>
            <button name="Update" onClick={Database} >Update</button>
            <button name="Delete" onClick={Database} >Delete</button>
        </div>
    )
}
```

Ex:
event-binding.jsx

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

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

    function Database(e){
      switch(e.target.name){
        case "Insert":
        setMsg('Record Inserted');
        break;
        case "Update":
        setMsg('Record Updated');
        break;
        case "Delete":
        setMsg('Record Deleted');
        break;
      }
    }

    return(
       <div className="container-fluid">
          <h2>Event Binding</h2>
          <button name="Insert" onClick={Database}>Insert</button>
          <button name="Update" onClick={Database} >Update</button>
          <button name="Delete" onClick={Database} >Delete</button>
          <p>{msg}</p>
       </div>
    )
}
```

Events in React

Summary
- Events
- What is Event?
- Which design pattern event uses?
- Delegate Mechanism
- SyntheticEvent
- EventHandler
- Inline, Embedded
- EventArgs

Event Propagation
- It is a mechanism followed for events configured for element with parent and child hierarchy.
- The child element event triggers and continue the propagation to its parent.
- This leads to the execution of all events configured for child and parent level hierarchy.
- You can use "stopPropagation()" of event to prevent execution of parent events automatically.

Syntax:
```
    function handleButtonClick(e)
    {
      e.stopPropagation();
    }
```

Ex:
event-binding.jsx

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

export function EventBinding(){

    function handleNavClick(){
        alert("Navbar Clicked");
    }
    function handleHomeClick(e){
        alert("Home Button Clicked");
        e.stopPropagation();
    }

    return(
        <div className="container-fluid">
            <nav onClick={handleNavClick} className="bg-dark text-white m-4 p-4">
                <h3>Navbar</h3>
                <button onClick={handleHomeClick} className="btn btn-light">Home</button>
            </nav>
        </div>
    )
}
```

Prevent Default

- HTML form provides generic buttons
    a) submit
    b) reset

```
    <button type="submit">        => It can submit form data to server
    <button type="reset">         => It can reset the form data
    <button>                      => default is submit
```

- HTML form have non-generic button
    a) button

```
    <button type="button">        => without any pre-defined functionality.
```

- Generic elements have pre-defined functionality, which firesup after the custom functions that you defined for element.

- You can prevent the default functionality of any element by using the method "preventDefault()"

Syntax:

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

Ex:

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

export function EventBinding(){

    function handleSubmitClick(e){
        alert("Form will submit its data to Server - API");
        e.preventDefault();
    }

    return(
        <div className="container-fluid">
            <h2>Register</h2>
            <form onSubmit={handleSubmitClick}>
              User Name <input type="text" name="UserName" /> <button
type="submit">Submit</button>
            </form>
        </div>
    )
}
```

FAQ's:
1. Can we have multiple forms in a component?
A. Yes.

2. Can we configure a form within another form? Can we have nested forms?
A. No.

3. Can a form have multiple submit buttons?   => .NET React
A. Yes.

Ex:

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

export function EventBinding(){
    function handleSubmit(e){
        e.preventDefault();
        for(var i=1; i<e.target.length; i++){
            console.log(e.target[i].name);
        }
    }

    return(
        <div className="container-fluid">
            <h2>Register</h2>
            <form onSubmit={handleSubmit}>
                <input type="text" name="UserName" />
                <button name="Insert">Insert</button>
                <button name="Update">Update</button>
                <button name="Delete">Delete</button>
            </form>
        </div>
    )
}

 console.log(document.getElementsByTagName("button"));
```

4. How to handle onload action in React?
A. React will not have load event for component.
   React component have "Mount" event.
   Mount event is defined for handling actions when component is mounted on to page.
   Function component mount event is defined by using a Hook called "useEffect".

Syntax:

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

Ex:

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

export function EventBinding(){

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

    useEffect(()=>{
        setUserName('John');
    },[]);

    return(
        <div className="container-fluid">
            <h2>Register</h2>
            <p>Hello ! {userName}</p>
        </div>

    )
}
```

Various Synthetic Events

- Keyboard Events
- Mouse Events
- Button Events
- Clipboard Events

- Timer Events
- Touch Events
- Form Events
- Element State Events etc.

React Events - Examples

Summary
- Event
- Event Handler
- Event Args
- SyntheticEvent
- StopPropagation
- PreventDefault

Various SyntheticEvents in React
- -

1. Synthetic Mouse Events

    onMouseOver
    onMouseOut
    onMouseUp
    onMouseDown
    onMouseMove

Ex:
mouse-event.css

```
nav img {
    border:none;
}
nav img:hover {
    border:2px solid blue;
}
main img:hover {
    transform: scale(1.5);
    transition: 2s;
}
main img {
    transform: scale(1);
    transition: 2s;
}
```

mouse-event.jsx

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

export function MouseEvent(){

    const [images] = useState(["m1.jpg", "m2.jpg", "m3.jpg", "m4.jpg"]);
    const [preview, setPreview] = useState('m1.jpg');
```

```
function handleMouseOver(e){
    setPreview(e.target.src);
}

return(
        <div className="container-fluid">
            <section className="mt-4 row">
                <nav className="col-2">
                   {
                     images.map(imagePath=>
                        <div className="mb-4 p-2" style={{width:'120px'}}>
                            <img style={{cursor:'grab'}} onMouseOver={handleMouseOver}
src={imagePath} width="100" height="100" />
                        </div>
                        )
                   }
                </nav>
                <main className="col-10">
                   <div style={{height:'400px', width:'400px', overflow:'hidden'}}>
                    <img src={preview} width="400" height="400" />
                   </div>
                </main>
            </section>
        </div>
    )
}
```

Ex: Mouse Move

mouse-move.jsx

```
import { useState } from "react"


export function MouseMove(){

    const [styleObject, setStyleObject] = useState({position:'', top:'',
    left:''}); function handleMousemove(e){
        setStyleObject({
            position: 'fixed',
            top: e.clientY +
            'px', left: e.clientX +
            'px'
        })
    }

     return(
        <div onMouseMove={handleMousemove}>
            <div style={{height:'1000px'}}>
            <h1>Mouse mouse pointer to test</h1>
            </div>
            <img src="flag.gif" style={styleObject}  width="50" height="50"/>
        </div>
     )
}
```

2. Synthetic Keyboard Events

```
    onKeyUp
    onKeyDown                    JOHN
    onKeyPress

    KeyEvent Properties
    - keyCode
    - charCode
    - which
    - shiftKey
    - ctrlKey
    - altKey
```

Ex:

```
import { useState } from "react"

export function KeyDemo(){

    const [users] = useState([{UserId:'john'}, {UserId:'john12'}, {UserId:'john_nit'}]);
    const [userError, setUserError] = useState('');
    const [isUserValid, setIsUserValid] = useState(false);
    const [capsWarning, setCapsWarning] = useState({display:'none'});

    function VerifyUserId(e){
        for(var user of users){
            if(user.UserId===e.target.value){
                setUserError('User Id Taken - Try Another');
                setIsUserValid(false);
                break;
            } else {
                setUserError('User Id Available');
                setIsUserValid(true);
            }
        }
    }


    function VerifyCaps(e){
        if(e.which>=65 && e.which<=90){
            setCapsWarning({display: 'block'});
        } else {
            setCapsWarning({display:'none'});
        }
    }

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

```
<dl>
    <dt>UserId</dt>
    <dd><input type="text" onKeyUp={VerifyUserId} /></dd>
    <dd className={(isUserValid)?'text-success':'text-danger'}>{userError}</dd>
```

```
                <dt>Password</dt>
                <dd>
                    <input type="password" onKeyPress={VerifyCaps} />
                </dd>
                <dd className="text-warning" style={capsWarning}>
                    <div className="bi bi-exclamation-triangle"> Warning - CAPS ON</div>
                </dd>
            </dl>
        </div>
    )
}
```

Task: For Password Display Password strength strong, weak, poor.

Condition:

If password is 4 to 15 chars alpha numeric with atleast one uppercase letter : strong If password is 4 to 15 chars alpha numeric without uppercase letter : weak
if password is < 4 chars : poor

You can show strength using <meter>

Write Regular Expression : 4 to 15 chars with atleat 1 uppercase letter
value.match() string function.

```
if(value.match(regExp)) {

  }
```

React Events Continued

Synthetic Events

1. Mouse Events
    - onMouseOver
    - onMouseOut
    - onMouseDown
    - onMouseUp
    - onMouseMove
2. Keyboard Events
    - onKeyUp
    - onKeyDown
    - onKeyPress

Task: Password Strength

```
import { useState } from "react"

export function KeyDemo(){

    const [errorMsg, setErrorMsg] = useState('');
    const [errorClass, setErrorClass] = useState('');
    const [grade, setGrade] = useState(1);

    function VerifyPassword(e){
        if(e.target.value.match(/(?=.*[A-Z])\w{4,15}/)) {
            setErrorMsg('Strong Password');
            setErrorClass('text-success');
            setGrade(100);
        } else {
            if(e.target.value.length<4){
                setErrorMsg('Poor Password');
                setErrorClass('text-danger');
                setGrade(30);
            } else {
                setErrorMsg('Weak Password');
                setErrorClass('text-warning');
                setGrade(70);
            }
        }
    }

    return(
        <div className="container-fluid">
            <h2>Register</h2>
            <dl className="w-25">
                <dt>Password</dt>
            <dd>
                <input className="form-control" type="password" onKeyUp={VerifyPassword}
/>
                <div>
                <meter min="1" max="100" value={grade} className="w-100"></meter>
                </div>
            </dd>
            <dd className={errorClass}>
                {errorMsg}
            </dd>
        </dl>
    </div>
)
}
```

Task: With bootstrap progress

```
import { useState } from "react"

export function KeyDemo(){

    const [errorMsg, setErrorMsg] = useState('');
    const [errorClass, setErrorClass] = useState('');
    const [progress, setProgress] = useState({width:''});


    function VerifyPassword(e){
        if(e.target.value.match(/(?=.*[A-Z])\w{4,15}/)) {
            setErrorMsg('Strong Password');
            setErrorClass('bg-success');
            setProgress({width: '100%'});

        } else {
            if(e.target.value.length<4){
                setErrorMsg('Poor Password');
                setErrorClass('bg-danger');
                setProgress({width: '30%'});

            } else {
                setErrorMsg('Weak Password');
                setErrorClass('bg-warning');
                setProgress({width:'70%'});
            }
        }
    }

    return(
        <div className="container-fluid">
            <h2>Register</h2>
            <dl className="w-25">
                <dt>Password</dt>
                <dd>
                    <input className="form-control" type="password" onKeyUp={VerifyPassword}
/>
                    <div className="progress mt-1">
```

```
                        <div className={`progress-bar progress-bar-animated progress-bar-striped
mt-1 ${errorClass}`} style={progress}>
                            {errorMsg}
                        </div>
```

```
                    </div>
                </dd>
            </dl>
        </div>
    )
}
```
3. Element State Events
    onBlur
    onFocus
    onChange
    onSelectStart

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

export function KeyDemo(){

    const [codeError, setCodeError] = useState('');

    function VerifyCode(e){
        if(e.target.value=="") {
            setCodeError("IFSC Code is required");
        } else {
            setCodeError("");
        }
    }

    return(
        <div className="container-fluid">
            <h2>Register</h2>
            <dl className="w-25">
                <dt>IFSC Code</dt>
                <dd><input type="text" onBlur={VerifyCode} className="form-control"
placeholder="Block Letters"/></dd>
                <dd className="text-danger">{codeError}</dd>
            </dl>
        </div>
    )
}
```
Task : Try changing the case for IFSC code to uppercase on blur.

Ex:

```
import { useState } from "react"

 export function KeyDemo(){

    const [codeError, setCodeError] = useState('');
    const [cityError, setCityError] = useState('');
```

```
function VerifyCode(e){
        if(e.target.value=="") {
            setCodeError("IFSC Code is required");
        } else {
            setCodeError("");
        }
    }

    function CityChange(e){
        if(e.target.value=="-1"){
            setCityError("Pleas select your city");
        } else {
            setCityError("");
        }
    }

    return(
        <div className="container-fluid">
            <h2>Register</h2>
            <dl className="w-25">
                <dt>IFSC Code</dt>
                <dd><input type="text" onBlur={VerifyCode} className="form-control"
placeholder="Block Letters"/></dd>
                <dd className="text-danger">{codeError}</dd>
                <dt>Select City</dt>
                <dd>
                    <select onChange={CityChange} className="form-select">
                        <option value="-1">Choose City</option>
                        <option value="Delhi">Delhi</option>
                        <option value="Hyderabad">Hyd</option>
                    </select>
                </dd>
                <dd className="text-danger"> {cityError} </dd>
            </dl>
        </div>
    )
}
```

4. Button Events

    onClick
    onDoubleClick
    onContextMenu

Ex:
```
import { useEffect } from
"react"; import { useState } from
"react"
export function KeyDemo(){
```

```
function OpenImage(){

    window.open("m1.jpg","Mobile","width=300 height=400");
}
```

```
    useEffect(()=> {
        document.oncontextmenu =
        function(){
            alert("Right Click Disabled");
            return false;
        }
    },[]);

    return(
        <div className="container-fluid">
            <h2>Button Events</h2>
            <img src="m1.jpg" onDoubleClick={OpenImage} width="100" height="100" />
            <p>Double Click to View large</p>
            <h2>Our Terms and Conditions</h2>
            <textarea rows="4" cols="40">Read our terms of service.</textarea>
        </div>
    )
}
```

5. Clipboard
   Events onCut
   onCopy
   onPaste

Ex:

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

export function KeyDemo(){
    const [msg, setMsg] = useState('');
    function OpenImage(){
        window.open("m1.jpg","Mobile","width=300 height=400");
    }

    useEffect(()=> {
       document.onpaste = function(){
         return false;
       }
    },[]);

    return(
        <div className="container-fluid">
            <h2>Button Events</h2>
            <img src="m1.jpg" onDoubleClick={OpenImage} width="100" height="100" />
            <p>Double Click to View large</p>
            <h2>Our Terms and Conditions</h2>
            <textarea onCut={()=> { setMsg('Removed and Copied to Clipboard'); }}
onCopy={()=>{setMsg('Copied to Clipboard')}} rows="4" cols="40">Read our terms
of service.</textarea>
            <p>{msg}</p>
        </div>
    )
```

```
}
```

6. Touch Events

```
    onTouchStart
    onTouchEnd
    onTouchMove
```

Ex:

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

export function KeyDemo(){
    const [msg, setMsg] = useState('');
    function GetDetails(e){
        setMsg(e.target.title);
    }

    return(
        <div className="container-fluid">
            <h2>Touch Events</h2>
            <img src="m1.jpg" title="I Phone 14 - 30% OFF"
onTouchStart={GetDetails} width="200" height="300" />
            <img src="m2.jpg" title="IPhone 14 With Ceramic Back" onTouchStart={GetDetails}
width="200" height="300" />
            <p className="h2">{msg}</p>
        </div>
    )
}
```

7. Timer Events

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

Timer Events

Summary
- Mouse
- Keyboard
- Clipboard
- Button
- Touch

Timer Events

- JavaScript date and time values are configured using Date().

```
const [now] = useState(new Date());          // loads current date & time
const [dept] = useState(new Date("yy-mm-dd hr:min:sec.millisec");
```

- To read date and time values you can use the functions

getHours()	0-23
getMinutes()	0-59
getSeconds()	0-59
getMilliSeconds()	0-99
getDate()	1, 28, 29, 30, 31
getDay()	0 = Sunday
getMonth()	0 = January
getFullYear()	2023
toLocaleDateString()	
toLocaleTimeString()	

- To set date and time values setHours()
  setMinutes()
  setSeconds()
  setMilliSeconds()
  setDate()
  setMonth()
  setYear()

- React can use JavaScript timer events

  a) setTimeout()
  b) clearTimeout()
  c) setInterval()
  d) clearInterval()

setTimeout():
- It is used to control debouncing in application.
- Debouncing is a technique where a task can be locked in the memory and keep inactive [sleep] for specific duration and then later release into process.
- Task is removed from memory and released into process. Hence task execute only once.

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

clearTimeout():

- It can remove the task from memory.
- You can remove before it is released into process.

Syntax:
```
 clearTimeout(referenceName)
```

setInterval():
- It is a timer event that loads the given task into memory and release into process a regular time intervals.
- It is not removed from memory, a copy of function is released into process. Hence it executes repeatedly until removed from memory.

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

clearInterval():
- It removes the task from memory.

Syntax:
```
clearInterval(referenceName)
```

Ex: Interval

timer-demo.jsx

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

export function TimerDemo(){


    const [now] = useState(new Date());
    const [status, setStatus] =
    useState(''); const [time, setTime] =
    useState('');

    function GetSalutation(){
        var date = new Date();
        var hrs = date.getHours();
        if(hrs>=0 && hrs<=12) {
            setStatus("Good Morning !");
        } else if (hrs>12 && hrs<=16) {
            setStatus("Good Afternoon !");
        } else if (hrs>16 && hrs<=23) {
            setStatus("Good Evening !!!");
        }
    }
    function Clock(){
```

```
    var now = new Date();
    var time = now.toLocaleTimeString();
    setTime(time);
}
```

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

    return(
        <div className="container-fluid">
            <header className="bg-dark text-white p-2 d-flex justify-content-between">
                <div>
                    <span>Shopper.</span>
                </div>
                <div>
                    <span>{status}</span>
                </div>
                <div>
                    <span className="bi bi-calendar"> {now.toDateString()} </span>
                    <span className="bi bi-clock"> {time} </span>
                </div>
            </header>
        </div>
    )
}
```

Ex: Timeout

```
import { useState } from "react"
export function TimeoutDemo()
{
    const [msg, setMsg] = useState('');

    function msg1(){
        setMsg('Volume Set : 1');
    }
    function msg2(){
        setMsg('Volume Set : 2');
    }
    function msg3(){
        setMsg('Volume Set : 3');
    }
    function msg4(){
        setMsg('Volume Set : 4');
    }
    function IncreaseVolume(){
       setTimeout(msg1, 2000);
       setTimeout(msg2, 4000);
       setTimeout(msg3, 6000);
       setTimeout(msg4, 8000);
    }
```

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

```
            <div className="mt-4">
                <button  onClick={IncreaseVolume} className="btn btn-primary me-2"> +
</button>
            </div>
            <h3>{msg}</h3>
        </div>
    )
}
```

Form Events

- These are the events defined for <form> element.
- These events will fireup only with generic buttons, submit & reset.

```
onSubmit
onReset
```

Syntax:

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

    <button type="submit"> Submit </button>
    <button type="reset"> Cancel </button>
</form>
```

Ex:

```
import { useState } from "react"

export function TimeoutDemo()
{
    function handleSubmit(e){
        e.preventDefault();  alert('Form
        submit to Server');
    }

    return(
        <div className="container-fluid">
          <form onSubmit={handleSubmit}>
            <h2>Register User</h2>
            <dl>
              <dt>User Name</dt>
              <dd><input type="text" name="UserName"/></dd>
            </dl>
            <button>Submit</button>
          </form>
        </div>
    )
}
```

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

Data from API

Distributed Computing

- Distributed computing allows 2 different applications running on 2 different mechines to share information.
- It also allows 2 different objects running in 2 different process on same mechine to share information.
- There are various distributed computing technologies used by different languages

CORBA	Common Object Request Broken Architecture	14 languages, C++, Java
DCOM	Distribute Component Object Model	Visual Basic
RMI	Remote Method Invocation	J2EE
EJB	Enterprise Java Beans	Java
Web Service		All technologies
Remoting		.NET

Web Service Specifications:

a) SOAP
b) REST
c) JSON

SOAP
- Service Oriented Architecture Protocol
- Consumer Sends XML request
- Provider Sends XML response

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

REST
- Representational State Transfer
- Consumer Sends a query request
- Provider Sends XML response optionally JSON.

```
    ?category=electronics
```

JSON
- JavaScript Object Notation
- Consumer sends JSON request
- Providers sends JSON response

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

- API is Application Programming Interface, that allows data to reach every device.

- JavaScript based technologies can use various techniques for handling communication with API requests.

XMLHttpRequest object

        fetch() promise
        jQuery Ajax methods
        3rd Party libraries, axios, whatwgFetch etc..

Ex:

1. Go to public folder and

   add product.json

```
{
    "title": "APPLE iPhone 14 (Blue, 128 GB)",
    "price": 60999,
    "rating": {"rate":4.6, "count": 3500, "reviews":
    600}, "image": "iblue.jpg",
    "features": [
        "Bank Offer10% off on Axis Bank Credit Card EMI Transactions, up to ₹1,000 on orders
of ₹5,000 and aboveT&C",
        "Bank Offer10% off on Flipkart Axis Bank Credit Card EMI Transactions, up to ₹1000 on
orders of ₹5000 and aboveT&C",
        "Bank Offer10% off on Citi Credit Card EMI Transactions, up to ₹1,000 on orders of
₹5,000 and aboveT&C",
        "Special PriceGet extra ₹8901 off (price inclusive of cashback/coupon)T&C"
    ]
}
```

2. Add a component

   flipkart.jsx

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

export function Flipkart(){

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

    useEffect(()=>{

        var http = new XMLHttpRequest();
        http.open("get", "product.json",true);
        http.send();
        http.onreadystatechange = function(){
            if(http.readyState==4){
                setProduct(JSON.parse(http.responseText));
            }
        }

    },[]);
```

```
return(
    <div className="container-fluid">
        <div className="mt-4 row">
            <div className="col-4">
```

```
                    <img src={product.image} />
                </div>
                <div className="col-8">
                    <h3 className="text-primary">{product.title}</h3>
                    <div>
                        <span className="badge bg-success">{product.rating.rate}
<span className="bi bi-star-fill"></span> </span>
                        <span className="fw-bold ms-3 text-secondary">{product.rating.count}
Ratings & {product.rating.reviews} Reviews</span>
                    </div>
                    <div className="mt-4">
                       <ul className="list-unstyled">
                            {
                                product.features.map(feature=>
                                    <li key={feature} className="bi mb-3 bi-tag-fill">
<span>{feature}</span> </li>
                                    )
                            }
                       </ul>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

XmlHttp, Fetch and jQuery Methods

XMLHttpRequest

- It is a JavaScript object used to commnuicate with a end point.
- XMLHttpRequest object provides various properties and methods to handle communication

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

Member	Description
open()	It is used to configure the request. It comprises a) Request type [GET, POST..] b) Request end point location [source] c) Boolean Async [true, false] Syntax: http.open("GET", "URL", true);
send()	It sends the request to end point. It will process the configured request. Syntax: http.send();
onreadystatechange	It is a function of AJAX life cycle in JavaScript, that defines actions to perform when response is ready. It depends on "readystate" status, which includes 0 : Initial, 1 : Pending, 2 : Processing, 3 : Completed, 4 : Ready [Response Ready]
responseText	It returns string response
responseXML	It returns XML response
responseHTML	It returns markup response

Ex:

1. Public folder => products.json

```
[
    {
        "title": "APPLE iPhone 14 (Blue, 128 GB)",
        "price": 60999,
        "rating": {"rate":4.6, "count": 3500, "reviews": 600},
        "image": "iblue.jpg",
        "features": [
            "Bank Offer10% off on Axis Bank Credit Card EMI Transactions, up to ₹1,000 on
orders of ₹5,000 and aboveT&C",
            "Bank Offer10% off on Flipkart Axis Bank Credit Card EMI Transactions, up to ₹1000
on orders of ₹5000 and aboveT&C",
            "Bank Offer10% off on Citi Credit Card EMI Transactions, up to ₹1,000 on orders of
₹5,000 and aboveT&C",
            "Special PriceGet extra ₹8901 off (price inclusive of cashback/coupon)T&C"
        ]
    },
    {
        "title": "APPLE iPhone 14 (Starlight, 128 GB)",
        "price": 60999,
        "rating": {"rate":4.5, "count": 1500, "reviews": 30},
        "image": "starlight.jpg",
        "features": [
            "Bank Offer10% off on Axis Bank Credit Card EMI Transactions, up to ₹1,000 on
orders of ₹5,000 and aboveT&C",
            "Bank Offer10% off on Flipkart Axis Bank Credit Card EMI Transactions, up to ₹1000
on orders of ₹5000 and aboveT&C",
            "Bank Offer10% off on Citi Credit Card EMI Transactions, up to ₹1,000 on orders of
₹5,000 and aboveT&C",
            "Special PriceGet extra ₹8901 off (price inclusive of cashback/coupon)T&C"
        ]
    },
    {
        "title": "APPLE iPhone 14 (Red, 128 GB)",
        "price": 60999,
        "rating": {"rate":4.5, "count": 7500, "reviews": 2430},
        "image": "m2.jpg",
        "features": [
            "Bank Offer10% off on Axis Bank Credit Card EMI Transactions, up to ₹1,000 on
orders of ₹5,000 and aboveT&C",
            "Bank Offer10% off on Flipkart Axis Bank Credit Card EMI Transactions, up to ₹1000
on orders of ₹5000 and aboveT&C",
            "Bank Offer10% off on Citi Credit Card EMI Transactions, up to ₹1,000 on orders of
₹5,000 and aboveT&C",
            "Special PriceGet extra ₹8901 off (price inclusive of cashback/coupon)T&C"
        ]
    }
]
```

2. Comp

onent

flipkart.jsx

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

export function Flipkart(){
    const [products, setProducts] = useState([{title:'', price:0, rating:{rate:0, count:0,
reviews:0}, image:'', features:[]}]);

    useEffect(()=>{

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

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

    },[]);

    return(
        <div className="container-fluid">
            {
                products.map(product=>
                    <div className="row mt-3 mb-3 p-2 border border-2 border-dark">
                        <div className="col-2">
                            <img src={product.image} width="100%" />
                        </div>
                        <div className="col-8">
                            <div className="h5 text-primary">{product.title}</div>
                            <div>
                                <span className="badge bg-success text-white">{product.rating.rate}
<span className="bi bi-star-fill"></span> </span>
                                 <b className="text-secondary ms-3">{product.rating.count} Ratings &
{product.rating.reviews} Reviews</b>
                            </div>
                            <div className="mt-2">
                               <ul className="list-unstyled">
                                 {
                                   product.features.map(feature=>
                                       <li className="mb-2"> <span className="bi bi-tag-fill text-
success"></span> <span> {feature} </span> </li>
                                       )
                                 }
                               </ul>
                            </div>
                        </div>
                        <div className="col-2">
                                <div className="h3"> {product.price.toLocaleString('en-in',{style:
'currency', currency:'INR'})} </div>
                        </div>
                    </div>
                )
            }
        </div>
    )
```

}

Issues with XMLHttpRequest Object:
- It is a request to end point which requires explicit Async configuration.
- The response is not directly in JSON format if end point is sending JSON.
- It requires explicit parsing methods [JSON.parse()].
- It is poor in error handling.
- It can't track the client and server side errors implicitly.

JavaScript Fetch Promise

- Fetch is JavaScript window object method.
- Techically it is a promise.
- It is implicitly Async.
- It provides implicit catch() method, however it can't catch the issues.
- The catch method requires an explicit "throw".
- It returns data in binary format.
- Explicit parsing for data is required.

Syntax:
```
    fetch("url").then().catch().finally()
```

Ex:
flipkart.jsx

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

export function Flipkart(){

    const [products, setProducts] = useState([{title:'', price:0, rating:{rate:0, count:0,
reviews:0}, image:'', features:[]}]);
    useEffect(()=>{

       fetch("products.json")

       .then(response=> response.json())
       .then(products=>{
            setProducts(products)
       })

    },[]);

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

```
dark">
```

```
products.map(product=>
    <div key={product.title} className="row mt-3
    mb-3 p-2 border border-2 border-

        <div className="col-2">
            <img src={product.image} width="100%" />
        </div>
```

```
                        <div className="col-8">
                            <div className="h5 text-primary">{product.title}</div>
                            <div>
                                <span className="badge bg-success text-white">{product.rating.rate}
<span className="bi bi-star-fill"></span> </span>
                                 <b className="text-secondary ms-3">{product.rating.count} Ratings &
{product.rating.reviews} Reviews</b>
                            </div>
                            <div className="mt-2">
                              <ul className="list-unstyled">
                                {
                                  product.features.map(feature=>
                                      <li key={feature} className="mb-2"> <span className="bi
bi-tag- fill text-success"></span> <span> {feature} </span> </li>
                                      )
                                }
                              </ul>
                            </div>
                        </div>
                        <div className="col-2">
                                <div className="h3"> {product.price.toLocaleString('en-in',{style:
'currency', currency:'INR'})} </div>
                        </div>
                    </div>
                    )
            }
        </div>
    )
}
```

jQuery Ajax

- jQuery is a JavaScript library
- Returns data in JSON
- jQuery Ajax methods are implicitly typed.
- Explicit parsing is not required.
- It provides better error handling techniques, as it can use the life cycle methods to track the status of Ajax request.

```
        $.ajax()
        $.ajaxStart()
        $.ajaxStop()
        $.ajaxComplete()
        $.ajaxSuccess()
        $.ajaxError() etc...
```

Syntax:

```
    $.ajax({
```

```
method: " ",
url: "",
data: { },
success: (),
error:()
```

```
        })

        $.getJSON()
```

Ex:

```
>       npm install jquery
```

--save flipkart.jsx

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

export function Flipkart(){

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

    useEffect(()=>{

         $.ajax({
            method: 'GET',
            url: 'products.json',
            success: products => {
                setProducts(products);
            }
         })

    },[]);

    return(
        <div className="container-fluid">
            {
                                                    products.map(product=>
                                                        <div key={product.title} className="row mt-3
dark">                                                  mb-3 p-2 border border-2 border-

                                                            <div className="col-2">
                                                                <img src={product.image} width="100%" />
                                                            </div>
                                                            <div className="col-8">
                                                                <div className="h5
                                                                text-primary">{product.title}</div>
                                                                <div>
                                                                    <span className="badge bg-success
                                                                    text-white">{product.rating.rate}
<span className="bi bi-star-fill"></span> </span>
                                    <b className="text-secondary ms-3">{product.rating.count} Ratings &
{product.rating.reviews} Reviews</b>
```

```
                    </div>
                    <div className="mt-2">
                      <ul className="list-unstyled">
                        {
                          product.features.map(feature=>
                              <li key={feature} className="mb-2"> <span className="bi
bi-tag- fill text-success"></span> <span> {feature} </span> </li>
```

```
                                        )
                                    }
                                </ul>
                            </div>
                        </div>
                        <div className="col-2">
                                <div className="h3"> {product.price.toLocaleString('en-in',{style:
'currency', currency:'INR'})} </div>
                        </div>
                    </div>
                    )
            }
        </div>
    )
}
```

- jQuery Ajax have CORS issues [Cross Origin Resource Sharing]
- Browser block the data from Cross Origin
- It is not a secured way of transporting data.

Axios

Axios

Summary
- XMLHttpRequest
- fetch() promise
- jQuery Ajax

Axios

- It is a 3rd party library that provides a middleware to communicate with end point.
- It supports CORS.
- It provides better error handling.
- It can handle multiple requests simultaneously at the same time.
- It is more secured.
- It is modular and async as it used promises.

1. Install axios

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

2. Import axios base library into component

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

3. Configure axios request

```
 axios(
    {
      method: "",
      url: "",
      data:" "
    }
    )

    (or)

axios.get("url").then().catch().finally()

     (or)

axios([{ method:"", url:"" }, { method:"", url:""}])
```

Note: The axios request on success will returns a response object that comprises of various details like

> data	: The data returned from end-point as response
> statusText	: Returns the status text
> status	: Returns the status code

Status Codes:

2xx	: success, ok
3xx	: methods, redirections
4xx	: client errors
5xx	: server errors

Ex:
flipkart.jsx

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

export function Flipkart(){

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

    useEffect(()=>{
         axios.get("products.json")
         .then((response)=>{
             setProducts(response.data);
         })
         .catch((reason)=>{
            console.log(reason.message);
         })
     },[]);

    return(
        <div className="container-fluid">
            {
                                                                          products.map(product=>
                                                                              <div key={product.title} className="row mt-3
dark">                                                                        mb-3 p-2 border border-2 border-

                                                                                  <div className="col-2">
                                                                                      <img src={product.image} width="100%" />
                                                                                  </div>
                                                                                  <div className="col-8">
                                                                                      <div className="h5
                                                                                      text-primary">{product.title}</div>
                                                                                      <div>
                                                                                          <span className="badge bg-success
                                                                                          text-white">{product.rating.rate}
<span className="bi bi-star-fill"></span> </span>
                                  <b className="text-secondary ms-3">{product.rating.count} Ratings &
{product.rating.reviews} Reviews</b>
                             </div>
                             <div className="mt-2">
                                <ul className="list-unstyled">
                                   {
                                     product.features.map(feature=>
                                         <li key={feature} className="mb-2"> <span className="bi
bi-tag- fill text-success"></span> <span> {feature} </span> </li>
                                         )
                                   }
                                </ul>
                              </div>
                          </div>
                          <div className="col-2">
                                 <div className="h3"> {product.price.toLocaleString('en-in',{style:
'currency', currency:'INR'})} </div>
                          </div>
```

```
                                                  </div>
                                                  )

                             }
                          </div>
                        )
                     }
```

Fetch Data from Remote API

Ex: Nasa API   [api.nasa.gov] => Register and generate API key

nasa-api.jsx

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

export function NasaAPI(){
    const [marsObj, setMarsObj] = useState({});
    useEffect(()=>{
        axios.get("https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=6ffzhNjjV1XA2HkP9u2ghEmZVMK8Rb2M2ZG4n6Fl&quot;)
        .then(response=>{
            setMarsObj(response.data);
        })
        .catch(reason => {
            console.log(reason.message);
        })


    },[])

    return(
        <div className="container-fluid">
            <h2>Mars Rover Photos</h2>
            <table className="table table-hover">
                <thead>
                    <tr>
                        <th>Photo Id</th>
                        <th>Preview</th>
                        <th>Camera</th>
                        <th>Rover</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        marsObj.photos.map(photo=>
<tr key={photo.id}>
    <td>{photo.id}</td>
    <td>
```

```
                                    <div>
                                        <a href={photo.img_src} download className="btn mb-3"> <span
className="bi bi-download"></span> </a>
                                    </div>
                                    <img src={photo.img_src} width="150" height="150"/>
                                </td>
                                <td>
                                    {photo.camera.full_name}
                                </td>
                                <td>
                                    {photo.rover.name}
                                </td>
                            </tr>
                            )
                    }
                </tbody>
            </table>
        </div>
    )
}
```

Fakestoreapi.com

Fakestore

XMLHttpRequest
jquery ajax
fetch
axios

fakestoreapi.com

Request Method	Route	Description
GET	/products	It returns an array of products. [ ]
GET	/products/1	It returns specific product that matches given id. { }
GET	/products/categories	It returns all categories. [ ]
GET	/products/category/electronics	It returns an array of all products belong to specific category.

API Data in Fakestore:

```
  {
    id:number,
    title:string,
    image:string,
    price:number,
    category:string,
    description:string,
    rating: {rate:number, count:number}
  }
```

FAQ: How you can know about the keys in any unknown object?
Ans:

a) Object.keys() => returns the array of all keys in object

   Syntax: []
   ```
   Object.keys(objectName).map(function(key){
        typeof objectName[key]
   })
   ```
b) for..in iterator

   Syntax:

   ```
   for(var property in object)
   ```

```
{
}
```

Note: You can use Web Debuggers to test and explore API requests.

a) postman

        b) fiddler
        c) swagger etc..

Ex:
```
<script>
    fetch("http://fakestoreapi.com/products/1&quot;)
    .then(res=> res.json())
    .then(product => {
        Object.keys(product).map(key=>{
             document.write(`${key} : ${typeof product[key]}<br>`);

        })
        Object.keys(product.rating).map(key=> {
                document.write(key + "<br>");
             })
    })
</script>
```

Ex:
Fakestore.jsx

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

export function Fakestore(){

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

    function LoadCategories(){
        axios.get("http://fakestoreapi.com/products/categories&quot;)
        .then(response=> {
            response.data.unshift("all");
            setCategories(response.data);
        })
    }

    function LoadProducts(url){
        axios.get(url)
        .then(response=> {
            setProducts(response.data);
        })
    }

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

    },[]);
```

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


    function handleAddClick(e){
        axios.get(`http://fakestoreapi.com/products/${e.target.name}`)
        .then(response => {
            cartItems.push(response.data);
            alert(`${response.data.title}\nAdded To Cart`);


        })
    }

    return(
        <div className="container-fluid">
            <header className="d-flex bg-dark text-white justify-content-between p-3">
                <div className="h3">Fakestore.</div>
                <div className="fs-4">
                    <span className="me-4"><a>Home</a></span>
                    <span className="me-4"><a>Jewelery</a></span>
                    <span className="me-4"><a>Electronics</a></span>
                </div>
                <div className="fs-4">
                    <button className="bi btn btn-light bi-cart4 position-relative">
                     <span className="badge position-absolute rounded rounded-circle bg-danger
text-white">0</span>
                    </button>
                </div>
            </header>
            <section className="mt-3 row">
                <nav className="col-2">
                  <div className="mt-4">
                    <label className="fw-bold form-label">Select Category</label>
                    <div>
                        <select onChange={handleCategoryChange} className="form-select">
                           {
                                                                        categories.map(category=>
                                                                            <option value={category} key={category}>
</option>                                                                   {category.toUpperCase()}

                                                                            )
                                                                    }
                        </select>
                    </div>
                  </div>
                </nav>
                <main className="col-10 overflow-auto" style={{height:'500px' , display:'flex',
flexWrap:'wrap'}}>
```

```
                    {
                        products.map(product=>
                             <div className="card p-2 m-2" key={product.id} style={{width:'200px'}}>
                                <img src={product.image} className="card-img-top" height="120" />
                                <div className="card-header" style={{height:'120px'}}>
                                    <p>
                                        {product.title}
                                    </p>
                                </div>
                                <div className="card-body">
                                    <dl>
                                        <dt>Price</dt>
                                        <dd>{product.price}</dd>
                                        <dt>Rating</dt>
                                        <dd>
                                            {product.rating.rate} <span className="bi bi-star-fill text-
success"></span>                                                    </dd>
                                                                    </dl>
                                                                </div>
                                                                <div className="card-footer">
                                                                    <button name={product.id}
                                                                    onClick={handleAddClick} className="btn
btn-dark w-100 bi bi-cart3"> Add to Cart</button>
                                </div>
                             </div>
                            )
                    }
                </main>
            </section>
        </div>
    )
}
```

React CRUD with Inline Data

Configuring State
- useState
- useReducer
- Redux Store

Ex: useState to keep api data and the data required across requests.

fakestore.jsx

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

export function Fakestore(){

    const [categories, setCategories] = useState([]);
    const [products, setProducts] = useState([{id:0, title:'', description:'', price:0, image:'',
category:'', rating:{rate:0, count:0}}]);
    const [cartItems] = useState([]);
    const [cartCount, setCartCount] = useState(0);
    const [toggleTable, setToggleTable] = useState({display:'none'});

    function LoadCategories(){
        axios.get("http://fakestoreapi.com/products/categories&quot;)
        .then(response=> {
            response.data.unshift("all");
            setCategories(response.data);
        })
    }

    function LoadProducts(url){
        axios.get(url)
        .then(response=> {
            setProducts(response.data);
        })
    }

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

    },[]);

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


    function handleAddClick(e){
        axios.get(`http://fakestoreapi.com/products/${e.target.name}`)
        .then(response => {
            cartItems.push(response.data);
            alert(`${response.data.title}\nAdded To Cart`);
            setCartCount(cartItems.length);
        })
    }
    function handleCartClick(){
        setToggleTable({display: (toggleTable.display==="none")?"block":"none"});
    }

    return(
        <div className="container-fluid">
            <header className="d-flex bg-dark text-white justify-content-between p-3">
                <div className="h3">Fakestore.</div>
                <div className="fs-4">
                    <span className="me-4"><a>Home</a></span>
                    <span className="me-4"><a>Jewelery</a></span>
                    <span className="me-4"><a>Electronics</a></span>
                </div>
                <div className="fs-4">
                    <button onClick={handleCartClick} className="bi btn btn-light bi-cart4 position-
relative">
                     <span className="badge position-absolute rounded rounded-circle bg-danger
text-white">{cartCount}</span>
                    </button>
                </div>
            </header>
            <section className="mt-3 row">
                <nav className="col-2">
                  <div className="mt-4">
                    <label className="fw-bold form-label">Select Category</label>
                    <div>
                        <select onChange={handleCategoryChange} className="form-select">
                            {
                                                                    categories.map(category=>
                                                                        <option value={category} key={category}>
</option>                                                                {category.toUpperCase()}

                                                                        )
                                                                }
                        </select>
                    </div>
                  </div>
                  <div>
                    <table style={toggleTable} className="table table-hover caption-top">
                        <caption>Your Cart Items</caption>
                        <thead>
                            <tr>
        <th>Title</th>
        <th>Price</th>
        <th>Preview</th>
    </tr>
</thead>
```

```
                        <tbody>
                            {
                                cartItems.map(item=>
                                    <tr key={item.id}>
                                        <td>{item.title}</td>
                                        <td>{item.price}</td>
                                        <td><img width="50" src={item.image} height="50"/></td>
                                    </tr>
                                    )
                            }
                        </tbody>
                    </table>
                  </div>
                </nav>
                <main className="col-10 overflow-auto" style={{height:'500px' , display:'flex',
flexWrap:'wrap'}}>
                    {
                        products.map(product=>
                             <div className="card p-2 m-2" key={product.id} style={{width:'200px'}}>
                                <img src={product.image} className="card-img-top" height="120" />
                                 <div className="card-header" style={{height:'120px'}}>
                                    <p>
                                        {product.title}
                                    </p>
                                </div>
                                <div className="card-body">
                                    <dl>
                                        <dt>Price</dt>
                                        <dd>{product.price}</dd>
                                        <dt>Rating</dt>
                                        <dd>
                                            {product.rating.rate} <span className="bi bi-star-fill text-
success"></span>
                                                                                    </dd>
                                                                                </dl>
                                                                            </div>
                                                                            <div className="card-footer">
                                                                                <button name={product.id}
                                                                                onClick={handleAddClick} className="btn
btn-dark w-100 bi bi-cart3"> Add to Cart</button>
                                </div>
                             </div>
                            )
                    }
                </main>
            </section>
        </div>
    )
}
```

CRUD Operations on API

C	- Create	POST

R   - Read   GET
U   - Update   PUT
D   - Delete   DELETE

- CRUD operations are handled on the data source provided by API.
- API must provide the request methods to handle all CRUD operations.
- You can manage the data temporarily using various techniques.

Ex:

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


export function InmemoryCRUD()
{
    const [products, setProducts] = useState([{Id:1, Name: 'TV'}, {Id:2, Name: 'Mobile'}]);
    const [newProduct, setNewProduct] = useState({Id:0, Name: ''});

    function handleIdChange(e){
        setNewProduct({
            Id: parseInt(e.target.value),
            Name: newProduct.Name
        })
    }
    function handleNameChange(e){
        setNewProduct({
            Id: newProduct.Id,
            Name: e.target.value
        })
    }

    function handleAddClick(){
        products.push(newProduct);
        setProducts(products.filter(product => product.Id
        !==0)); alert(`Product Successfully Added..`);
    }

    function handleDeleteClick(id){
            setProducts(products.filter(product => product.Id !== id));
            alert('Record Deleted');
    }

    return(
        <div className="container-fluid">
            <h2>Testing CRUD</h2>
            <div>
                <label className="fw-bold">Add New Product</label>
                <div>
                    <dl>
    <dt>Product Id</dt>
    <dd><input type="number" onChange={handleIdChange} /></dd>
    <dt>Name</dt>
    <dd><input type="text" onChange={handleNameChange} /></dd>
</dl>
```

```
                <button onClick={handleAddClick}>Add Product</button>
            </div>
        </div>
        <table className="table table-hover">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody>
                {
                    products.map(product=>
                        <tr key={product.Id}>
                          <td>{product.Name}</td>
                          <td>
                            <button className="bi bi-pen-fill me-2 btn btn-warning"></button>
                            <button onClick={ ()=> { handleDeleteClick(product.Id) }} className="bi
bi-trash btn btn-danger"></button>
                          </td>
                        </tr>
                        )
                }
            </tbody>
        </table>
    </div>
  )
}
```

React Forms

Complete CRUD Example - Inline Edit

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


export function InmemoryCRUD()
{
    const [products, setProducts] = useState([{Id:1, Name: 'TV'}, {Id:2, Name: 'Mobile'}]);
    const [newProduct, setNewProduct] = useState({Id:0, Name: ''});

    function handleIdChange(e){
        setNewProduct({
            Id: parseInt(e.target.value),
            Name: newProduct.Name
        })
    }
    function handleNameChange(e){
        setNewProduct({
            Id: newProduct.Id,
            Name: e.target.value
        })
    }

    function handleAddClick(){
        setProducts([...products, newProduct]);
        setNewProduct({
            Id:'',
            Name:''
        })
        alert(`Product Successfully Added..`);
    }

    function handleDeleteClick(id){
            var flag = window.confirm("Sure");
            if(flag==true) {
                products.splice(id,1);
            setProducts([...products]);
            alert('Record Deleted');
            }

    }

    return(
        <div className="container-fluid">
            <h2>Testing CRUD</h2>
            <div>
                <label className="fw-bold">Add New Product</label>
                <div>
                    <dl>
                    <dt>Product Id</dt>
                    <dd><input type="number" value={newProduct.Id}
onChange={handleIdChange} /></dd>
                    <dt>Name</dt>
                    <dd><input type="text" value={newProduct.Name}
```

```
onChange={handleNameChange} /></dd>
                    </dl>
                    <button onClick={handleAddClick}>Add Product</button>
                </div>
            </div>
            <table className="table table-hover">
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        products.map((product,index)=>
                            <tr key={index}>
                              <td>{product.Name}</td>
                              <td>
                                <button className="bi bi-pen-fill me-2 btn btn-warning"></button>
                                <button onClick={ ()=> { handleDeleteClick(index) }} className="bi
bi- trash btn btn-danger"></button>
                              </td>
                            </tr>
                            )
                    }
                </tbody>
            </table>
        </div>
    )
}
```

- Rest Parameters   ...param
- Spread Operator   ...[ ]

Note: You can use the spread operator to read values upto end from a collection.
      You can concat new values into collection by using destructuring technique

Ex:
```
<script>
    var one = ["A", "B"];
    var two = "C";
    var three = "D";
    var collection = [...one, two, three];
    console.log(collection);             => [ "A", "B", "C",
    "D"]
```

```
</script>
```

- You can use "map()" with index to identify the index of given value in an iterator.
- You can remove any item at specified index by using splice().

```
          product.map((product, index)=>
              <li key={index}> {product.Name} </li>
          )
```

Ex:
inmemory-crud.jsx

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


export function InmemoryCRUD()
{
    const [products, setProducts] = useState([{Id:1, Name: 'TV'}, {Id:2, Name: 'Mobile'}]);
    const [newProduct, setNewProduct] = useState({Id:0, Name: ''});

    function handleIdChange(e){
        setNewProduct({
            Id: parseInt(e.target.value),
            Name: newProduct.Name
        })
    }
    function handleNameChange(e){
        setNewProduct({
            Id: newProduct.Id,
            Name: e.target.value
        })
    }

    function handleAddClick(){
        products.push(newProduct);
        setProducts([...products]);
        alert('Product Added..');
        setNewProduct({Id:0, Name:''});
    }

    function handleDeleteClick(index){
          var flag = window.confirm("Are you sure\nWant to Delete?");
          if(flag==true){
              products.splice(index,1);
              setProducts([...products]);
          }
    }

    return(
        <div className="container-fluid">
            <h2>Testing CRUD</h2>
            <div>
                <label className="fw-bold">Add New Product</label>
                <div>
```

```
                    <dl>
                        <dt>Product Id</dt>
                        <dd><input type="number"
value={newProduct.Id} onChange={handleIdChange} /></dd>
```

```
                        <dt>Name</dt>
                        <dd><input type="text" value={newProduct.Name}
onChange={handleNameChange} /></dd>
                    </dl>
                    <button onClick={handleAddClick}>Add Product</button>
                </div>
            </div>
            <table className="table table-hover">
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        products.map((product,index)=>
                            <tr key={index}>
                              <td>{product.Name}</td>
                              <td>
                                <button className="bi bi-pen-fill me-2 btn btn-warning"></button>
                                <button onClick={ ()=> { handleDeleteClick(index) }} className="bi
bi- trash btn btn-danger"></button>
                              </td>
                            </tr>
                            )
                    }
                </tbody>
            </table>
        </div>
    )
}
```

Complete CRUD :

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


export function InmemoryCRUD()
{
    const [products, setProducts] = useState([{Id:1, Name: 'TV'}, {Id:2, Name: 'Mobile'}]);
    const [newProduct, setNewProduct] = useState({Id:0, Name: ''});
    const [activeProductId, setActiveProductId] = useState(0);
    const [newName, setNewName] = useState();

    function handleIdChange(e){
        setNewProduct({
            Id: parseInt(e.target.value),
            Name: newProduct.Name
        })
    }
    function handleNameChange(e){
        setNewProduct({
            Id: newProduct.Id,
            Name: e.target.value
        })
    }

    function handleAddClick(){
        products.push(newProduct);
        setProducts([...products]);
        alert('Product Added..');
        setNewProduct({Id:0, Name:''});
    }

    function handleDeleteClick(index){
          var flag = window.confirm("Are you sure\nWant to Delete?");
          if(flag==true){
              products.splice(index,1);
              setProducts([...products]);
          }
    }

    function handleEditClick(id){
        setActiveProductId(id);
        var prod = products.find(product=> product.Id===id);
        setNewName(prod.Name);
    }

    function handleSaveClick(id) {
        setActiveProductId(0);
        var editName = products.find(product=> product.Id===id);
        editName.Name = newName;
    }

    function handleNameChangeOnEdit(e){
        setNewName(e.target.value);
    }

    return(
        <div className="container-fluid">
            <h2>Testing CRUD</h2>
            <div>
                <label className="fw-bold">Add New Product</label>
                <div>
                    <dl>
                        <dt>Product Id</dt>
                        <dd><input type="number"
value={newProduct.Id} onChange={handleIdChange} /></dd>
                        <dt>Name</dt>
                        <dd><input type="text" value={newProduct.Name}
```

```
onChange={handleNameChange} /></dd>
                </dl>
                <button onClick={handleAddClick}>Add Product</button>
            </div>
        </div>
        <table className="table table-hover">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody>
                {
                    products.map((product,index)=>
                        <tr key={index}>
                          <td> {(product.Id==activeProductId)?<input type="text"
onChange={handleNameChangeOnEdit}
value={newName} />:<label>{product.Name}</label> }  </td>
                          <td>
                            {(product.Id==activeProductId)?<button onClick={()=>
handleSaveClick(product.Id)} className="btn btn-success bi bi-floppy me-2">
</button>:<button onClick={ ()=> handleEditClick(product.Id) } className="bi bi-pen-fill
me-2 btn btn-warning"></button>}
                            <button onClick={ ()=> { handleDeleteClick(index) }} className="bi
bi- trash btn btn-danger"></button>
                          </td>
                        </tr>
                        )
                }
            </tbody>
        </table>
    </div>
)
}
```

React Forms

React Forms

- Form provides an UI for application from where user can interact with the data.
- It enables all CRUD operations on data.
- React can use HTML forms or any 3rd party form components.
- React HTML form requires lot of Event binding techniques and references.
- React HTML form requires several JavaScript methods and Synthetic Events for validation.
- Validation is required to ensure that contractionary and unauthorized data is not get stored into database.

Ex: React Form

form-demo.jsx

```
import { useState } from "react"

export function FormDemo()

{
    const [userDetails, setUserDetails] = useState({UserName:'', Password:'', Mobile:'', City:'', Gender:''});
    const [errors, setErrors] = useState({UserName:'', Password:'', Mobile:'', City:'', Gender:''})

    function handleNameChange(e){
        setUserDetails({
            UserName: e.target.value,
            Password: userDetails.Password,
            Mobile: userDetails.Mobile,
            City: userDetails.City,
            Gender: userDetails.Gender
        })
    }
    function handlePasswordChange(e){
        setUserDetails({
            UserName: userDetails.UserName,
            Password: e.target.value,
            Mobile: userDetails.Mobile,
            City: userDetails.City,
            Gender: userDetails.Gender
        })
    }
    function handleMobileChange(e){
        setUserDetails({
            UserName: userDetails.UserName,
            Password: userDetails.Password,
            Mobile: e.target.value,
            City: userDetails.City,
            Gender: userDetails.Gender
        })
    }
    function handleCityChange(e){
        setUserDetails({
            UserName: userDetails.UserName,
            Password: userDetails.Password,
            Mobile: userDetails.Mobile,
            City: e.target.value, Gender:
            userDetails.Gender
        })
    }

    function handleGenderChange(e){
        setUserDetails({
            UserName: userDetails.UserName,
            Password: userDetails.Password,
            Mobile: userDetails.Mobile,
            City: userDetails.City,
            Gender: e.target.value
        })
    }

    function handleNameBlur(e){
        if(e.target.value==""){
            setErrors({UserName: 'User Name Required'});
        } else {
            setErrors({UserName: ''});
        }
    }

    function handleFormSubmit(e){
        e.preventDefault();
        if(userDetails.UserName==""){
            setErrors({
                UserName: 'User Name Required'
            })
        } else {
             alert(JSON.stringify(userDetails));
        }
    }

    return (
        <div className="container-fluid">
           <form className="w-25" onSubmit={handleFormSubmit}>
           <h3>Register User</h3>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" onKeyUp={handleNameBlur} onBlur={handleNameBlur}
onChange={handleNameChange} className="form-control" /></dd>
                <dd className="text-danger">{errors.UserName}</dd>
                <dt>Password</dt>
                <dd><input type="password" onChange={handlePasswordChange}
className="form-control" /></dd>
```

```
                <dt>Mobile</dt>
                <dd><input type="text" onChange={handleMobileChange} className="form -
control" /></dd>
                <dt>City</dt>
                <dd>
                    <select onChange={handleCityChange} className="form-select">
                        <option>Select City</option>
                        <option>Delhi</option>
                        <option>Hyd</option>
                    </select>
                </dd>
                <dt>Gender</dt>
                <dd>
                    <input type="radio" onChange={handleGenderChange} name="gender"
value="Male" /> <label>Male</label>
                    <input type="radio" onChange={handleGenderChange} name="gender"
value="Female" /> <label>Female</label>
                </dd>
            </dl>
            <button type="submit" className="btn btn-primary w-100">Register</button>
           </form>
        </div>
    )
}
```

- React can use various 3rd party form libraries to simplify the CRUD operations and validations.
  a) Formik
  b) Reack Hook Form
  c) Telerik Forms etc..

Formik

- It is a 3rd party library for React form.
- It provides pre-defined event binding techniques that allows to access and update form data without manually writing change events.

1. Install formik library for react application

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

2. Formik provides "useFormik()" hook to configure a form element so that you can bind the values for handling CRUD.

```
const formik = useFormik({
    initialValues: { },
    validation: function(){},
    validationSchema: {},
    onSubmit: function(){ } etc..
})
```

3. You have to bind formik change event with form elements

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

4.	InitialValues refer to the values that a form must handle. They are binded with form elements by using "name" attribute.

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

        initialValues: { UserName: ' ' }
```

5. Formik uses onSubmit to collect data from form and submit to server

```
        {
          onSubmit : (values) => { //post to server }
        }

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

Ex:
 form-demo.jsx

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

export function FormDemo()
{

    const formik = useFormik({
        initialValues : {
            UserName: '',
            Password: '',
            Mobile: '',
            City: '',
            Gender:''
        },
        onSubmit: (values) => {
            alert(JSON.stringify(values));
        }
    })

    return (
        <div className="container-fluid">
           <form className="w-25" onSubmit={formik.handleSubmit}>
           <h3>Register User</h3>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" name="UserName" onChange={formik.handleChange}
className="form-control" /></dd>
                <dd className="text-danger"></dd>
                <dt>Password</dt>
                <dd><input type="password" name="Password"
onChange={formik.handleChange} className="form-control" /></dd>
```

```
            <dt>Mobile</dt>
            <dd><input type="text" name="Mobile" onChange={formik.handleChange}
className="form-control" /></dd>
            <dt>City</dt>
```

```
                                                       <dd>
                                                          <select name="City"
                                                          onChange={formik.handleChange}
                                                          className="form-
select">
                                                             <option>Select City</option>
                                                             <option>Delhi</option>
                                                             <option>Hyd</option>
                                                          </select>
                                                       </dd>
                                                       <dt>Gender</dt>
                                                       <dd>
                                                          <input type="radio"
                                                          onChange={formik.handleChange}
                                                          name="Gender"
value="Male" /> <label>Male</label>
               <input type="radio" onChange={formik.handleChange} name="Gender"
value="Female" /> <label>Female</label>
            </dd>
         </dl>
         <button type="submit" className="btn btn-primary w-100">Register</button>
        </form>
      </div>
   )
}
```

Formik Validations and Components

```
Formik Library
    useFormik()
        {
          initialValues,
          onSubmit,
          onChange,
          onBlur,
          validate,
          validationSchema
        }
    formik.handleChange     => Form Input, Select, ..Elements
    formik.handleSubmit     => <form> element
```

Formik Validation

- Formik validation comprises of 2 techniques
    a) Validation
    b) ValidationSchema

- Validation uses a function that comprises of validation logic and error log.
- Developer have to configure the logic for validating elements and log errors explicitly.

Syntax:

```
         useFormik({
            initialValues: { }
            validate: function(){} => returns error object
        })
```

Syntax:

```
        function ValidateUser(userDetails)
        {
            var error = { };

            return error;
        }

        const formik = useFormik({
            initialValues : { }
            validate: ValidateUser
        })
```

- Formik will return the errors object configured with "validate" attribute.

```
        { formik.errors.UserName }
        { formik.errors.Password }
```

Ex: Formik with Validate function

form-demo.jsx

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

```
export function FormDemo()
{
    function ValidateUser(userDetails){
         var error = {UserName:'', Password:'', Mobile:'', City:'', Gender:''};

          if(userDetails.UserName==""){
              error.UserName = "User Name Required";
          } else {
              if(userDetails.UserName.length<4){
                error.UserName = "Name too short";
              } else {
                error.UserName = "";
              }
          }

          if(userDetails.Password==""){
             error.Password = "Password Required";
          }

          if(userDetails.Mobile==""){
             error.Mobile = "Mobile Required";
          } else {
              if(userDetails.Mobile.match(/\+91\d{10}/)) {
                  error.Mobile = "";
              } else {
                  error.Mobile = "Invalid Mobile";
              }
          }

          if(userDetails.City=="-1") {
             error.City = "Please Select Your City";
          }

          if(userDetails.Gender==""){
             error.Gender = "Please Choose Your Gender";
          }

         return error;
    }

    const formik = useFormik({
        initialValues : {
            UserName: '',
            Password: '',
            Mobile: '',
            City: '',
            Gender:''
        },
        validate : ValidateUser,
        onSubmit: (values) => {
            alert(JSON.stringify(values));
        }
    })
```

```
    return (
        <div className="container-fluid">
           <form className="w-25" onSubmit={formik.handleSubmit}>
           <h3>Register User</h3>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" name="UserName" onChange={formik.handleChange}
className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.UserName}</dd>
                <dt>Password</dt>
                <dd><input type="password" name="Password"
onChange={formik.handleChange} className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.Password}</dd>
                <dt>Mobile</dt>
                <dd><input type="text" name="Mobile" onChange={formik.handleChange}
className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.Mobile}</dd>
                <dt>City</dt>
                <dd>
                    <select name="City" onChange={formik.handleChange}  className="form-
select">                                                                <option value="-1">Select City</option>
                                                                        <option>Delhi</option>
                                                                        <option>Hyd</option>
                                                                    </select>
                                                                </dd>
                                                                <dd
                                                                className="text-danger">{formik.errors.City}</dd>
                                                                <dt>Gender</dt>
                                                                <dd>
                                                                    <input type="radio"
                                                                    onChange={formik.handleChange}
                                                                    name="Gender"
value="Male" /> <label>Male</label>
                    <input type="radio" onChange={formik.handleChange} name="Gender"
value="Female" /> <label>Female</label>
                </dd>
                <dd className="text-danger">{formik.errors.Gender}</dd>
            </dl>
            <button type="submit" className="btn btn-primary w-100">Register</button>
           </form>
        </div>
    )
}
```

- React can use Validation Schema, which is a pre-defined validation structure for validating forms.
- React doesn't provide any pre-defined validation services.
- You have to depend on 3rd party validation schema services.

- Formik is mostly used with validation schema service called "Yup".

1. Install Yup library

```
> npm i  yup --save
```

2. Import the compelete yup schema library or your can configure a modular library.

```
import * as yup from "yup";
import required, min, max as yup from "yup"
```

3. You can access the validation schema by using "yup.object()"

```
const formik = useFormik({
    initialValues: { },
    validationSchema: yup.object({
        UserName: yup.required().string().number().min() etc..
    })
})
```

4. You can access the validation errors by using the error object returned by formik validation schema

```
            {formik.errors.UserName}
```

Ex:
form-demo.jsx

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

export function FormDemo()
{

    const formik = useFormik({
        initialValues : {
            UserName: '',
            Password: '',
            Mobile: '',
            City: '',
            Gender:''
        },
        validationSchema : yup.object({
            UserName: yup.string().required("User Name Required").min(4, "Name too Short.."),
            Password: yup.string().required("Password Required"),
            Mobile: yup.string().required("Mobile Required").matches(/\+91\d{10}/,"Invalid Mobile")
        }),
        onSubmit: (values) => {
            alert(JSON.stringify(values));
        }
    })

    return (
        <div className="container-fluid">
          <form className="w-25" onSubmit={formik.handleSubmit}>
          <h3>Register User</h3>
```

```
<dl>
    <dt>User Name</dt>
```

```
                <dd><input type="text" name="UserName" onChange={formik.handleChange}
className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.UserName}</dd>
                <dt>Password</dt>
                <dd><input type="password" name="Password"
onChange={formik.handleChange} className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.Password}</dd>
                <dt>Mobile</dt>
                <dd><input type="text" name="Mobile" onChange={formik.handleChange}
className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.Mobile}</dd>
                <dt>City</dt>
                <dd>
                    <select name="City" onChange={formik.handleChange}  className="form-
select">                                                                    <option value="-1">Select City</option>
                                                                            <option>Delhi</option>
                                                                            <option>Hyd</option>
                                                                        </select>
                                                                    </dd>
                                                                    <dd
                                                                    className="text-danger">{formik.errors.City}</dd>
                                                                    <dt>Gender</dt>
                                                                    <dd>
                                                                        <input type="radio"
                                                                        onChange={formik.handleChange}
                                                                        name="Gender"
value="Male" /> <label>Male</label>
                    <input type="radio" onChange={formik.handleChange} name="Gender"
value="Female" /> <label>Female</label>
                </dd>
                <dd className="text-danger">{formik.errors.Gender}</dd>
            </dl>
            <button type="submit" className="btn btn-primary w-100">Register</button>
           </form>
        </div>
    )
}
```

- Formik provides a "spread" technique for configure all events to any field with single statement.

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

Ex:

form-demo.jsx

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

```
export function FormDemo()
{

    const formik = useFormik({
        initialValues : {
            UserName: '',
            Password: '',
            Mobile: '',
            City: '',
            Gender:''
        },
        validationSchema : yup.object({
            UserName: yup.string().required("User Name Required").min(4, "Name too Short.."),
            Password: yup.string().required("Password Required"),
            Mobile: yup.string().required("Mobile Required").matches(/\+91\d{10}/,"Invalid Mobile")
        }),
        onSubmit: (values) => {
            alert(JSON.stringify(values));
        }
    })

    return (
        <div className="container-fluid">
           <form className="w-25" onSubmit={formik.handleSubmit}>
           <h3>Register User</h3>
            <dl>
                <dt>User Name</dt>
                <dd><input type="text" name="UserName" {...formik.getFieldProps("UserName")}
className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.UserName}</dd>
                <dt>Password</dt>
                <dd><input type="password" name="Password"
 {...formik.getFieldProps("Password")} className="form-control" /></dd>
                 <dd className="text-danger">{formik.errors.Password}</dd>
                <dt>Mobile</dt>
                <dd><input type="text" name="Mobile" {...formik.getFieldProps("Mobile")}
className="form-control" /></dd>
                <dd className="text-danger">{formik.errors.Mobile}</dd>
                <dt>City</dt>
                <dd>
                    <select name="City" {...formik.getFieldProps("City")}  className="form-select">
                        <option value="-1">Select City</option>
                        <option>Delhi</option>
                        <option>Hyd</option>
                    </select>
                </dd>
                <dd className="text-danger">{formik.errors.City}</dd>
                <dt>Gender</dt>
                <dd>
                    <input type="radio" {...formik.getFieldProps("Gender")} name="Gender"
value="Male" /> <label>Male</label>
                    <input type="radio" {...formik.getFieldProps("Gender")} name="Gender"
value="Female" /> <label>Female</label>
                </dd>
                <dd className="text-danger">{formik.errors.Gender}</dd>
            </dl>
            <button type="submit" className="btn btn-primary w-100">Register</button>
           </form>
```

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

- Formik provides components, which are predefined with funcitonality and styles.
- Formik components will simplify the UI.

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

Syntax:

```
    <Formik initialValues={}       validationSchema={}     onSubmit={}>
        <Form>
            <Field> </Field>     text, number, password, email, radio, select..
            <ErrorMessage />
        </Form>
    </Formik>
```

Ex:
form-demo.jsx

```
import { useState } from "react";
import { ErrorMessage, Field, Form, Formik, useFormik } from "formik";
import * as yup from "yup";

export function FormDemo()
{

    return (
        <div className="container-fluid">
            <Formik
               initialValues={{ProductId:0, Name:'', Price:0}}
               validationSchema={yup.object({
                ProductId: yup.number().required("Product Required"),
                Name: yup.string().required("Name Required").min(4, "Name too short"),
                Price: yup.number().required("Price Required")
               })}

               onSubmit={(values)=> alert(JSON.stringify(values)) }
            >

             <Form>
                <h1>Register Product</h1>
                 <dl>
                    <dt>Product Id</dt>
                    <dd> <Field type="number" name="ProductId" ></Field> </dd>
                    <dd className="text-danger"> <ErrorMessage name="ProductId" /> </dd>
```

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

```
                    <dd> <Field type="number" name="Price"></Field> </dd>
                    <dd className="text-danger"> <ErrorMessage name="Price" /> </dd>
                 </dl>
                 <button>Submit</button>
             </Form>

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

React Hook Form library

Accessing Data from Form
Validating Form Data
Formik Components
- Formik
- Form
- Field
- ErrorMessage

- Formik components provides form state validation.
- Form state validation is used to verify all fields in a form simultaneously at the same time.
- Form state provides services like

a) isValid	: returns true if all fields are valid
b) dirty	: returns true if any one field modified
c) touched	: returns true if any one field in form get focus.
d) values	: returns specific field value
e) errors	: returns error object that comprises of all errors

Syntax:
```
    <Formik>
        {
        form => <Form> </Form>
        }
    </Formik>

        form.isValid
        form.dirty
        form.touched
```

Ex:
form-demo.jsx

```
import { useState } from "react";
import { ErrorMessage, Field, Form, Formik, useFormik } from "formik";
import * as yup from "yup";

export function FormDemo()
{

    return (
        <div className="container-fluid">
            <Formik
              initialValues={{ProductId:0, Name:'', Price:0}}
              validationSchema={yup.object({
               ProductId: yup.number().required("Product Required"),
               Name: yup.string().required("Name Required").min(4, "Name too short"),
               Price: yup.number().required("Price Required")
```

```
  }))}

  onSubmit={(values)=> alert(JSON.stringify(values)) }
>
 {
```

```
                 form =>
                 <Form>
                 <h1>Register Product</h1>
                  <dl>
                     <dt>Product Id</dt>
                     <dd> <Field type="number" name="ProductId" ></Field> </dd>
                     <dd className="text-danger"> <ErrorMessage name="ProductId" /> </dd>
                     <dt>Name</dt>
                     <dd> <Field type="text" name="Name"></Field> </dd>
                     <dd className="text-danger"> <ErrorMessage name="Name" /> </dd>
                     <dt>Price</dt>
                     <dd> <Field type="number" name="Price"></Field> </dd>
                     <dd className="text-danger"> <ErrorMessage name="Price" /> </dd>
                  </dl>
                  <button disabled={(form.isValid)?false:true} >Submit</button>
                  <button style={{display:(form.dirty)?'inline':'none'}} className="ms-
2">Save</button>
                 </Form>
             }


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

React Hook Form

1. Install

   >npm i react-hook-form --save

2. import the library into component

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

3. Configure useForm hook

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

4. Bind with fields

   <input type="text" {...register("Name", { required:true, minlength:4 }) } />

5. Bind submit

   <form onSubmit={handleSubmit(onSubmit)}>

6. Collect the form data

```
const onSubmit = (values) => console.log(values);
```

7. Handle Errors

```
    { (errors.Name?.type === 'required')?<span> Required </span>: <span></span> }
```

Ex:
hook-form-demo.jsx

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

export function HookFormDemo(){

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

    const onSubmit = (values) => { alert(JSON.stringify(values)) }

    return(
        <div className="container-fluid">
           <form onSubmit={handleSubmit(onSubmit)}>
                <h2>Register Product</h2>
                <dl>
                    <dt>Product Name</dt>
                    <dd><input type="text" {...register("Name", { required: true, minLength:4 } )}
/></dd>
                    <dd className="text-danger"> {(errors.Name?.type ===
'required')?<span>Name Required</span>:<span></span> && (errors.Name?.type ===
"minLength")?<span>Name too short</span>:<span></span>} </dd>
                    <dt>Price</dt>
                    <dd><input type="number" {...register("Price", { required: true,
min:1000, max:10000})} /></dd>
                    <dd className="text-danger"> {(errors.Price?.type === 'min')?<span>Price Min
1000 required</span>:<span></span> && (errors.Price?.type === 'max')?<span>Price max
10000 only</span>:<span></span>} </dd>
                </dl>
                <button>Submit</button>
           </form>
        </div>
    )
}
```

MongoDB and API

MERN Stack Example

M	- MongoDB	Database
E	- Express JS	Middleware
R	- React JS	UI Library
N	- Node JS	Server Side [API]

MongoDB

- It an no-sql database.
- It is document oriented.
- It supports ORM.
- It supports Indexing.
- It supports Ad-hoc queries.

General Database	MongoDB Terminology
Database	Database
Table	Collection
Row/Record	Document
Field/Column	Field/Key
Joins	Embedded Documents

Setup MongoDB on your PC:

1. Download and Install MongoDB community server

   https://www.mongodb.com/try/download/community

2. While installing mongodb make sure that you selected "MongoDB Compass"

3. Start MongoDB Server

   run => services.msc => mongodb server [start]

4. Open MongoDB Compass and connect to server

   Connection String : mongodb://127.0.0.1:27017

5. Create a new Database in MongoDB for "Appointments, Reminders" [To-DO]

Database Name : todo

Collection : appointments

6. Insert a sample document

```
{
  "title": "test title",
    "date": {"$date": "2023-12-10T10:00:00.000Z"},
      "description": "something about appointment"
    }
```

CRUD Commands:

1. How to read data from
   collection find()

Syntax:

```
    db.collection.find({query})
    db.appointments.find({})          => return all
    db.appointments.find({ field:value })          exact match
```

Operators:

$gt	greater than
$lt	less than
$gte	greater than or equal
$lte	less than or equal
$ne	not equal
$or	OR
$and	AND

Syntax:

```
    db.appointments.find({ date: {$gte:15} })          0 to 31
    db.appointments.find({ })                          all
```

Ex:

```
 db.appointments.find({$and:[{date:{$gt:15}}, {date:{$lt:25}}]});
```

2. Inserting Documents into
   collection insertOne({ })
   insertMany([{ }, { }])

Syntax:

```
  db.collection.insertOne({ field:value });
  db.appointments.insertOne({"Id":1, "Title":"Test",
  "Date":"2023-12-20",
"Description":"something about"})
```

3. Updating
   Document
   updateOne()
   updateMany()

Operator:

```
    $set
    $unset
    $rename
```

Syntax:

```
    db.collection.updateOne({find}, {update})

    db.appointments.updateOne({Id:1}, {$set:{Title: "Meeting"}})
```

4. Delete
   Document
   deleteOne()
   deleteMany()

Syntax:

```
db.collection.deleteOne({findQuery})

db.appointments.deleteOne({Id:1})
```

Node & Express
[Create API]

1. Install the following libraries in your React application

```
> npm i express --save
> npm i mongodb ---save
> npm i cors --save
```

express	: It is required to configure API with server side routing.
mongodb	: It is a driver library that allows communication between API and mongodb database.
cors	: It is used to manage Cross Origin Resouce Sharing in network applications.

2. Add a folder into your project "Server"

3. Add a new JavaScript by name "api.js"

   [Node JS] api.js

```
var express = require("express");

var app = express();

app.get("/", (req, res)=>{
  res.send("<h1>To DO - API</h1>");
})

app.listen(4000);
console.log(`Server Started : http://127.0.0.1:4000`);
```

4. Open Terminal

```
> node api.js
```

5. You can request following in

   browser http://127.0.0.1:4000

End to End API

MongoDB Database
Server Side API

Database : todo
Collection :
appointments

Creating API requests for ToDo App:

Method	Route	Purpose
GET	/appointments	It returns all appointments [ ]
GET	/appointments/1	It returns specifiec appointment by ID
POST	/addtask	It allows adding an appointment into collection
PUT	/edittask/1	It allows updating appointment details
DELETE	/deletetask/1	It allows removing appointment from collection

API Debuggers: Fiddler, Postman, Swagger etc..

1. Install following libraries into project

```
>npm i express --save
>npm i mongodb --save
>npm i cors --save
```

2. Create a server folder in project
3. Add a new file "api.js" into server

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

var connectionString =
"mongodb://127.0.0.1:27017"; var app = express();
app.use(cors());
app.use(express.urlencoded({extended:true}));
app.use(express.json());

app.get("/", (req, res)=>{
  res.send("<h1>To DO - API</h1>");
});

app.get("/appointments", (req, res)=> {
   mongoClient.connect(connectionString).then(clientObject=>{
       var database = clientObject.db("todo");
       database.collection("appointments").find({}).toArray().then(documents=>{
           res.send(documents);
           res.end();
})
```

```
    })
});

app.get("/appointments/:id", (req,
     res)=>{ var id =
     parseInt(req.params.id);
     mongoClient.connect(connectionString).then(clientObject=>{
          var database = clientObject.db("todo");
          database.collection("appointments").find({Id:id}).toArray().then(documents=>
              {  res.send(documents);
              res.end();
          })
     })
});

app.post("/addtask",(req, res)=>{
      var task = {
          Id: parseInt(req.body.Id),
          Title: req.body.Title,
          Date: new Date(req.body.Date),
          Description: req.body.Description
      };
      mongoClient.connect(connectionString).then(clientObject=>{
          var database = clientObject.db("todo");
          database.collection("appointments").insertOne(task).then(()=>{
             console.log(`Task Added Successfully..`);
             res.end();
          })
      })
});

app.put("/edittask/:id",(req, res)=>{
   var id = parseInt(req.params.id);
   mongoClient.connect(connectionString).then(clientObject=
        >{ var database = clientObject.db("todo");
        database.collection("appointments").updateOne({Id:id},{$set:{Id:parseInt(req.body.Id),
Title: req.body.Title, Date: new Date(req.body.Date), Description:
req.body.Description}}).then(()=>{
            console.log("Task Updated Successfully..");
            res.end();
        })
   })
});

app.delete("/deletetask/:id", (req, res)=>{
    var id = parseInt(req.params.id);
    mongoClient.connect(connectionString).then(clientObject=
        >{ var database = clientObject.db("todo");
        database.collection("appointments").deleteOne({Id:id}).then(()=>{
            console.log("Task Deleted Successfully..");
            res.end();
        })
    })
});
```

```
app.listen(4000);
console.log(`Server Started : http://127.0.0.1:4000`);
```

4. Run the API

```
> node api.js
  http://127.0.0.1:4000
```

Complete To-DO App with End to end

Designing UI for To-DO App

Backend API Requests: [http://127.0.0.1:4000]

```
GET           /appointments
GET           /appointments/id
POST        /addtask
PUT           /edittask/id
DELETE
/deletetask/id
```

Front End React Component:

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

export function ToDoApp()
{
    const [appointments, setAppointments] = useState([]);
    const [toggleAdd, setToggleAdd] = useState({display:'block'});
    const [toggleEdit, setToggleEdit] = useState({display:'none'});
    const [editAppoint, setEditAppointment] = useState([{Id:0, Title:'', Date:'',

    Description:''}]); const formik = useFormik({

        initialValues : {
            Id: 0,
            Title: '',
            Description: '',
            Date: new Date()
        },

        onSubmit: (appointment) => {
            axios.post('http://127.0.0.1:4000/addtask&#39;, appointment);
            alert('Appointment Added Successfully..');
            window.location.reload();
        }

    })

    function LoadAppointments(){
        axios.get('http://127.0.0.1:4000/appointments&#39;)
        .then(response=> {
            setAppointments(response.data);
        })
    }

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

```
    function handleDeleteClick(e){
        var id = parseInt(e.target.value);
        var flag = window.confirm(`Are you sure\n Want to Delete?`);
        if(flag===true) {
            axios.delete(`http://127.0.0.1:4000/deletetask/${id}`);
            window.location.reload();
        }
    }

    function handleEditClick(id){
        setToggleAdd({display:'none'});
        setToggleEdit({display:'block'});
        axios.get(`http://127.0.0.1:4000/appointments/${id}`)
        .then(response=>{
            setEditAppointment(response.data);
        })


    }
    function handleCancelClick(){
        setToggleAdd({display:'block'});
        setToggleEdit({display:'none'});
    }

    function handleSaveClick(){
        // PUT method axios
    }

    return(
        <div className="container-fluid">
            <h1 className="text-center">To Do App</h1>
            <header>
                <div aria-label="AddAppointment" style={toggleAdd} >
                    <label className="form-label fw-bold">Add New Appointment</label>
                    <div>
                        <form onSubmit={formik.handleSubmit} className="w-50">
                         <div className="d-flex">
                         <input type="number" name="Id" className="form-control"
onChange={formik.handleChange} />
                         <input type="text" name="Title" onChange={formik.handleChange}
className="form-control" placeholder="Title"/>
                         <input type="date" name="Date" onChange={formik.handleChange}
className="form-control" />
                         </div>
                         <div className="mt-2">
                            <label className="form-label fw-bold">Description</label>
                            <textarea name="Description" onChange={formik.handleChange}
className="form-control">
```

```
</textarea>
<div className="mt-3">
    <button className="btn btn-primary">Add</button>
```

```
                        </div>
                      </div>

                    </form>
                 </div>
              </div>



              <div aria-label="EditAppointment" style={toggleEdit} >
                 <label className="form-label fw-bold">Edit Appointment</label>
                 <div>
                    <form onSubmit={formik.handleSubmit} className="w-50">
                     <div className="d-flex">
                     <input type="number" name="Id" value={editAppoint[0].Id}
className="form - control" onChange={formik.handleChange} />
                     <input type="text" name="Title" value={editAppoint[0].Title}
onChange={formik.handleChange} className="form-control" placeholder="Title"/>
                     <input type="date" name="Date" value={editAppoint[0].Date}
onChange={formik.handleChange} className="form-control" />
                     </div>
                     <div className="mt-2">
                        <label className="form-label fw-bold">Description</label>
                        <textarea name="Description" value={editAppoint[0].Description}
onChange={formik.handleChange} className="form-control">

                        </textarea>
                        <div className="mt-3">
                           <button className="btn btn-success">Save</button>
                           <button type="button" onClick={handleCancelClick}
className="ms-2 btn btn-danger">Cancel</button>
                        </div>
                     </div>

                    </form>
                 </div>
              </div>
           </header>
           <main className="mt-4">
              <div>
                 <label className="form-label fw-bold">Your Appointments</label>
                 <div className="d-flex flex-wrap">
                    {
                        appointments.map(appointment =>
                            <div className="alert alert-dismissible alert-success m-2
w-25" key={appointment.Id}>
                              <button className="btn btn-close" value={appointment.Id}
onClick={handleDeleteClick}></button>
                              <div className="h5 alert-title">{appointment.Title}</div>
                              <p>{appointment.Description}</p>
```

```
                                        <span className="bi bi-calendar"></span> {appointment.Date}
                                        <div className="mt-3">
                                            <button onClick={()=>{handleEditClick(appointment.Id)}}
className="bi bi-pen-fill btn btn-warning"> Edit </button>
```

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

To-Do App with GET, POST, PUT and DELETE - Component Properties

Complete To-DO App

api.js

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

var connectionString =
"mongodb://127.0.0.1:27017"; var app = express();
app.use(cors());
app.use(express.urlencoded({extended:true}));
app.use(express.json());

app.get("/", (req, res)=>{
  res.send("<h1>To DO - API</h1>");
});

app.get("/appointments", (req, res)=> {
    mongoClient.connect(connectionString).then(clientObject=>{
        var database = clientObject.db("todo");
        database.collection("appointments").find({}).toArray().then(documents=>{
            res.send(documents);
            res.end();
        })
    })
});

app.get("/appointments/:id", (req,
     res)=>{ var id =
     parseInt(req.params.id);
     mongoClient.connect(connectionString).then(clientObject=>{
         var database = clientObject.db("todo");
         database.collection("appointments").find({Id:id}).toArray().then(documents=>
             {  res.send(documents);
             res.end();
         })
     })
});

app.post("/addtask",(req, res)=>{
      var task = {
          Id: parseInt(req.body.Id),
          Title: req.body.Title,
          Date: new Date(req.body.Date),
    Description: req.body.Description
};
mongoClient.connect(connectionString).then(clientObject=>{
```

```
          var database = clientObject.db("todo");
          database.collection("appointments").insertOne(task).then(()=>{
             console.log(`Task Added Successfully..`);
             res.end();
          })
      })
});

app.put("/edittask/:id",(req, res)=>{
   var id = parseInt(req.params.id);
   mongoClient.connect(connectionString).then(clientObject=
        >{ var database = clientObject.db("todo");
        database.collection("appointments").updateOne({Id:id},{$set:{Id:parseInt(req.body.Id),
Title: req.body.Title, Date: new Date(req.body.Date), Description:
req.body.Description}}).then(()=>{
            console.log("Task Updated Successfully..");
            res.end();
        })
   })
});

app.delete("/deletetask/:id", (req, res)=>{
    var id = parseInt(req.params.id);
    mongoClient.connect(connectionString).then(clientObject=
        >{ var database = clientObject.db("todo");
        database.collection("appointments").deleteOne({Id:id}).then(()=>{
            console.log("Task Deleted Successfully..");
            res.end();
        })
    })
});

app.listen(4000);
console.log(`Server Started : http://127.0.0.1:4000`);


todo.jsx
import axios from "axios";
import { useEffect, useState } from "react";
import { useFormik } from "formik";

export function ToDoApp()
{
    const [appointments, setAppointments] = useState([]);
    const [toggleAdd, setToggleAdd] = useState({display:'block'});
    const [toggleEdit, setToggleEdit] = useState({display:'none'});
    const [editAppoint, setEditAppointment] = useState([{Id:0, Title:'', Date:'',

    Description:''}]); const formik = useFormik({
```

```
initialValues : {
    Id: appointments.length + 1,
    Title: '',
        Description: '',
        Date: new Date()
    },

    onSubmit: (appointment) => {
        axios.post('http://127.0.0.1:4000/addtask&#39;, appointment);
        alert('Appointment Added Successfully..');
        window.location.reload();
    }

})

const editFormik = useFormik({
    initialValues : {
        Id: editAppoint[0].Id,
        Title: editAppoint[0].Title,
        Date: `${editAppoint[0].Date.slice(0,editAppoint[0].Date.indexOf("T"))}`,
        Description: editAppoint[0].Description
    },
    enableReinitialize: true,
    onSubmit : (appointment) => {
        axios.put(`http://127.0.0.1:4000/edittask/${editAppoint[0].Id}`,
        appointment); alert('Appointment Modified Successfully..');
        window.location.reload();
    }
})

function LoadAppointments(){
    axios.get('http://127.0.0.1:4000/appointments&#39;)
    .then(response=> {
        setAppointments(response.data);
    })
}

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

function handleDeleteClick(e){
    var id = parseInt(e.target.value);
    var flag = window.confirm(`Are you sure\n Want to Delete?`);
    if(flag===true) {
        axios.delete(`http://127.0.0.1:4000/deletetask/${id}`);
        window.location.reload();
    }
}

function handleEditClick(id){
    setToggleAdd({display:'none'});
```

```
setToggleEdit({display:'block'});
axios.get(`http://127.0.0.1:4000/appointments/${id}`)
.then(response=>{
    setEditAppointment(response.data);
```

```
            console.log(response.data);
        })


    }
    function handleCancelClick(){
        setToggleAdd({display:'block'});
        setToggleEdit({display:'none'});
    }


    return(
        <div className="container-fluid">
            <h1 className="text-center">To Do App</h1>
            <header>
                <div aria-label="AddAppointment" style={toggleAdd} >
                    <label className="form-label fw-bold">Add New Appointment</label>
                    <div>
                        <form onSubmit={formik.handleSubmit} className="w-50">
                         <div className="d-flex">
                         <input type="hidden" name="Id" value={formik.values.Id} className="form
- control" onChange={formik.handleChange} />
                         <input type="text" name="Title" onChange={formik.handleChange}
className="form-control" placeholder="Title"/>
                         <input type="date" name="Date" onChange={formik.handleChange}
className="form-control" />
                         </div>
                         <div className="mt-2">
                            <label className="form-label fw-bold">Description</label>
                            <textarea name="Description" onChange={formik.handleChange}
className="form-control">

                            </textarea>
                            <div className="mt-3">
                                <button className="btn btn-primary">Add</button>
                            </div>
                         </div>

                        </form>
                    </div>
                </div>



                <div aria-label="EditAppointment" style={toggleEdit} >
                    <label className="form-label fw-bold">Edit Appointment</label>
                    <div>
                        <form onSubmit={editFormik.handleSubmit} className="w-50">
                         <div className="d-flex">
                         <input type="number" name="Id" value={editFormik.values.Id}
```

```
className="form-control" onChange={editFormik.handleChange} />
                    <input type="text" name="Title" value={editFormik.values.Title}
onChange={editFormik.handleChange} className="form-control" placeholder="Title"/>
                    <input type="date" name="Date" value={editFormik.values.Date}
```

```
onChange={editFormik.handleChange} className="form-control" />
                        </div>
                        <div className="mt-2">
                           <label className="form-label fw-bold">Description</label>
                           <textarea name="Description" value={editFormik.values.Description}
onChange={editFormik.handleChange} className="form-control">

                           </textarea>
                           <div className="mt-3">
                               <button className="btn btn-success">Save</button>
                               <button type="button" onClick={handleCancelClick}
className="ms-2 btn btn-danger">Cancel</button>
                           </div>
                        </div>

                       </form>
                   </div>
               </div>
           </header>
           <main className="mt-4">
               <div>
                   <label className="form-label fw-bold">Your Appointments</label>
                   <div className="d-flex flex-wrap">
                       {
                           appointments.map(appointment =>
                                <div className="alert alert-dismissible alert-success m-2
w-25" key={appointment.Id}>
                                   <button className="btn btn-close" value={appointment.Id}
onClick={handleDeleteClick}></button>
                                   <div className="h5 alert-title">{appointment.Title}</div>
                                   <p>{appointment.Description}</p>
                                   <span className="bi bi-calendar"></span>
{appointment.Date.slice(0,appointment.Date.indexOf("T"))}
                                   <div className="mt-3">
                                       <button onClick={()=>{handleEditClick(appointment.Id)}}
className="bi bi-pen-fill btn btn-warning"> Edit </button>
                                   </div>
                                </div>
                               )
                       }
                   </div>
               </div>
           </main>
       </div>
    )
}
```

ToDo:
- Hide ID in edit and new forms.

- Auto generate ID in new appointment form.
- Set validation for form

Title	: required, max 25 chars.
Date	: required, must be above current date

Description : required, max 150 chars
- Enable Save button in edit form only when any value is modified.
- Handle validation using Yup.

Summary
- Function Components
- Data Binding
- Style Binding
- Event Binding
- Class Binding
- State
- UseEffect
- API Interactions
  a) XMLHttpRequest, fetch
  b) jQuery Ajax
  c) Axios
- Forms
- Form Validations
- End to End integration [MERN Stack]
- To-Do App

Component Properties

- A component requires properties to modify according state and situation.
- You can create a reusable component, which can be customized according to the state and situation.
- Function component parameters are considered as its properties.

Syntax:
```
function Component(props)
{
}
```
- The component props is an object with key and value. Syntax:
```
props = {
    Key : value,
}
```

- You can access the component using token syntax

```
<Component> </Component>
```

- The props are passed in the token.

```
<Component property=value> </Component>
```

Ex:
1. Add a new folder
   component-library

2. Add a new component folder user-login

3. Add a new file
    user-login.jsx

```
export function UserLogin(props)
{
  return(
    <div className="container-fluid">

      <dl className={props.Theme}>
         <h3>{props.Title}</h3>
        <dt>{props.UserLabel}</dt>
        <dd><input type={props.UserType} className="form-control" /></dd>
        <dt>{props.Verify}</dt>
        <dd><input type={props.VerifyType} className="form-control" /></dd>
        <button className={props.ButtonType}>Login</button>
      </dl>

    </div>
  )
}
```

4. Add a new

    component

    prop-demo.jsx

```
import { UserLogin } from "../../components-libary/userlogin";

export function PropsDemo(){
  return(
    <div className="container-fluid">
      <h1>Shopping Home</h1>
      <UserLogin ButtonType="btn btn-light w-100" Theme="w-25 bg-primary text-white
p-2" Title="User Login" UserLabel="Your Email" UserType="email" Verify="Confirmation Code"
VerifyType="number" />
      <hr/>
      <UserLogin ButtonType="btn btn-warning w-100" Theme="w-25 bg-dark text-white
p- 2" Title="Admin Login" UserLabel="Mobile" UserType="text" Verify="Your OTP"
VerifyType="number" />
    </div>
  )
}
```

Component Properties and Classes

Ex:

1. Add a new component into library data-grid.jsx

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

                                }
                            </tr>
                            )
                    }

                </tbody>
            </table>
        </div>
    )
}
```

2. Implement and customize grid in your component with custom

data props-demo.jsx

```
import { DataGrid } from "../../components-libary/data-grid";

export function PropsDemo(){
    return(
        <div className="container-fluid">
            <DataGrid theme={'table-primary table-striped'} caption="Employee Table - Updated
Nov 2023" fields={["First Name", "Last Name", "Designation"]} data={[{FirstName:"Raj",
LastName:"Kumar", Designation:"Manager"}, {FirstName:"Kiran", LastName:"Rao",
Designation:"Developer"}]} />
            <DataGrid theme={'table-warning table-bordered'} caption="Product Details"
fields={["Name", "Price"]} data={[{Name:"TV", Price:34000}, {Name:"Mobile", Price:12300.33},
{Name:"Watch", Price:4600.33}]} />
        </div>
    )
}
```

Class Components in React

JavaScript Classes:

- Class is a program template. It comprises of data and logic, which you can implement and customize according to your requirements.
- Class is reffered as a "Model" when it is representing the data.
- Class is reffered as an "Entity" when it is representing the bussiness.
- It is used to keep all related data and logic together.
- "Alan Kay" introduced object.
- "Johan Olay" & "Kristian Nygaard" introduced OOP with SIMULA in early 1960's.
- OOP introduced code reusability.
- In early 1970's "Trygve" introduced code separation with a framework called "MVC".
  [Model-View-Controller] => language [Small Talk]
- In early 1975 C++
- In early 1990 Java
- In early 2003   C#

Note: JavaScript is not an OOP language, it supports only few features of OOP.
          JavaScript is OBPS [Object Based Programming System].

Configuring a Class:

1. Class Declaration

```
        class Product
        {
        }
```

2. Class Expression let

```
    Product = class {

    }
```

Class Members:

- JavaScript class member can be

    1. Property
    2. Accessor
    3. Method
    4. Constructor

FAQ's:

1. Can we define a variable as class member?

A. No.

2. Can we define a variable in class?

A. Yes.

```
class Demo
{
   var x = 10;         // invalid
}

class Demo
{
   Print(){
    var x = 10;         // valid
    }
}
```

3. Why a variable is not allowed as class member?
A. Variables immutable types, and class member can be only mutable type.

4. Why a class requires only mutable members?
A. It is template and immutable members will not allow to customize template.

Property

- The data in class is stored with reference of a Property.

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

- A property can handle any type of data, primitive or non-primitive.

- A property is mutable.

Accessors

- Accessors are used to provide a fine grained control over property.
- Accessors are 2 types

a)	get()	Getter
b)	set()	Setter

- Getter can read and return a value.
- Setter can assign a new value.

Syntax:

```
get AliasName()
{
   return propertyValue;
}

set  AliasName(newValue)
{
        property = newValue;
    }
```

Ex:

```
<script>
    var username = prompt("Enter Your Name");
    var role = prompt("Enter Your Role", "admin|customer");
    class Product
    {
        _productName;
        get ProductName(){
            return this._productName;
        }
        set ProductName(newName){
            if(role==="admin"){
                this._productName = newName;
            } else {
                document.write(`Hello ! ${username} your are not authorized to set product
 name.`);
            }
        }
    }
    let tv = new Product();
    tv.ProductName = prompt("Enter Product Name");
    if(tv.ProductName)
    {
        document.write(`Product Name = ${tv.ProductName}`);
    }
</script>
```

Class Members

JavaScript Class
- Property
- Accessor
- Method
- Constructor

Ex:
```
<script>
    class Product
    {
        Name = "Samsung TV";
        Rating = {
            Customer : {
                Rate : {Count:4.5}
            },
            Vendor : {
                Rate : {Count:3.6}
            }
        }
        get CustomerRating(){
            return this.Rating.Customer.Rate.Count;
        }
        set CustomerRating(newRating){
            this.Rating.Customer.Rate.Count = newRating;
        }
    }
    let obj = new Product();
    obj.CustomerRating = prompt("Enter New Rating for Customer");
    document.write(`Name=${obj.Name}<br>Customer Rating : ${obj.CustomerRating}`);
</script>
```

Method
- The logic is defined using a method.
- Class can't have a function as member.
- Class can have a function as member of any method.

Syntax:
```
    class Name
    {
       function(){}         // invalid
    }

    class Name
    {
      Method() {              //
         valid function f1() {
         }
    }
  }
```

Note: In technical terms, computer programming languages can use

function => to return a value

        method   => to configure void
        procedure => to define hybrid behaviour [function / method]

- All function behaviours are similar for methods in JavaScript.
    - Parameters
    - Return
    - Rest, Spread etc..

Ex:

```
<script>
    class Product
    {
        Name = "Samsung TV";
        Price = 45000.44;
        Qty = 1;
        Total(){
            return this.Qty * this.Price;
        }
        Print(){
            document.write(`Name=${this.Name}<br>Price=${this.Price}<br>Total=${this.Total()}`)
;
        }
    }
    let tv = new Product();
    tv.Qty = parseInt(prompt("Enter Quantity"));
    tv.Print();
</script>
```

FAQ: Why a function can't be class member?
Ans: functions are immutable, hence they can't be class members.

Constructor

- A constructor is used for instantiation.
- Instantiation is the process of creating an object for class.
- Every class implicitly have a constructor, which is know as "Default Constructor".
- Constructor is a special type of "subroutine", which executes automatically for every object.
- Constructor can execute only once per-object.
- JavaScript constructor is anonymous and defined using the keyword "constructor".

Syntax:

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

    let obj = new Name;          // valid => calls constructor
    let obj = new Name();          // valid => calls constructor and can pass params if
                                    required.
```

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        class Database
        {
           constructor(dbName){
               document.write(`Connected with ${dbName} Database <br>`);
           }
           Insert(){
               document.write("Record Inserted..");
           }
           Delete(){
               document.write("Record Deleted..");
           }
        }
        function DBActions(action){
           switch(action){
               case "Insert":
                let obj1 = new Database(document.querySelector("select").value);
                obj1.Insert();
                break;
               case "Delete":
                let obj2 = new Database(document.querySelector("select").value);
                obj2.Delete();
                break;

           }
        }
    </script>
</head>
<body>
    <select>
        <option>Choose Database</option>
        <option>Oracle</option>
        <option>MySql</option>
        <option>MongoDB</option>
    </select>
    <button onclick="DBActions('Insert')">Insert</button>
    <button onclick="DBActions('Delete')">Delete</button>

</body>
</html>
```

Summary:
- Property
- Accessor
- Method

- Constructor

Code Extensibiltiy & Reusability

- Code extensibility and reusability can be achived by using

  a) Aggregation
  b) Inheritance

Inheritance and Polymorphism

Code Reusability and Extensibility
1. Aggregration
2. Inheritance

Aggregation
- It is the process of accessing the members of one class in another class without configuring any relation between classes.
- Technically is Object to Object communication and often reffered as "Has-A-Relation".

Syntax:
```
    class A
    {
       Print(){
       }
    }
    class B
    {
      Print() {
         let obj = new A();
         obj.Print();
       }
    }
```

Ex:
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        class HDFC_Bank_Version1
        {
            PersonalBanking = "Personal Banking Service - General Customers <br>";
            NRIBanking = "NRI Banking Service - NRI Customers <br>";
            Print(){
                document.write(`${this.PersonalBanking}<br>${this.NRIBanking}<br>`);
            }
        }
        class HDFC_Bank_Version2
        {
            CreditCards = "Credit Card Payments<br>";
            Print(){
                let obj = new HDFC_Bank_Version1();
                obj.Print();
                document.write(`${this.CreditCards}<br>`);
            }
```

```
}
class HDFC_Bank_Version3
{
    Loans = "Personal and Car Loans <br>";
    Print(){
```

```
                let obj = new HDFC_Bank_Version2();
                obj.Print();
                document.write(`${this.Loans}<br>`);
            }
        }
        function InstallClick(){
            var ver = document.querySelector("select").value;
            switch(ver)
            {
                case "ver1":
                let obj1 = new HDFC_Bank_Version1();
                obj1.Print();
                break;
                case "ver2":
                let obj2 = new HDFC_Bank_Version2();
                obj2.Print();
                break;
                case "ver3":
                let obj3 = new HDFC_Bank_Version3();
                obj3.Print();
                break;
                default:
                document.write("Please Select a Version");
                break;
            }
        }
    </script>
</head>
<body>
    <fieldset>
        <legend>Install Bank App</legend>
        <select>
            <option value="-1">Select Version</option>
            <option value="ver1">Version-1 [2022]</option>
            <option value="ver2">Version-2 [2023-March]</option>
            <option value="ver3">Version-3 [2023-Nov] </option>
        </select>
        <button onclick="InstallClick()">Install</button>
    </fieldset>
</body>
</html>
```

Inheritance:
- It allows to access the members of one class in another by configuring relationship between classes.
- It is often reffered as "Is-A-Relation".
- A new class can "extend" existing class, which is known as inheritance.
- New class is known as "Derived Class" and existing class is known as "Super" class.

Syntax:

```
class Super
{
}
```

```
    class Derived extends Super
    {
    }
```

- You can access the members of super class in derived by using "super" keyword. Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        class HDFC_Bank_Version1
        {
            PersonalBanking = "Personal Banking Service - General Customers <br>";
            NRIBanking = "NRI Banking Service - NRI Customers <br>";
            Print(){
                document.write(`${this.PersonalBanking}<br>${this.NRIBanking}<br>`);
            }
        }
        class HDFC_Bank_Version2 extends HDFC_Bank_Version1
        {
            CreditCards = "Credit Card Payments<br>";
            Print(){
                super.Print();
                document.write(`${this.CreditCards}<br>`);
            }
        }
        class HDFC_Bank_Version3 extends HDFC_Bank_Version2
        {
            Loans = "Personal and Car Loans <br>";
            Print(){
                super.Print();
                document.write(`${this.Loans}<br>`);
            }
        }
        function InstallClick(){
            var ver = document.querySelector("select").value;
            switch(ver)
            {
                case "ver1":
                let obj1 = new HDFC_Bank_Version1();
                obj1.Print();
                break;
                case "ver2":
                let obj2 = new HDFC_Bank_Version2();
                obj2.Print();
                break;
                case "ver3":
```

```
let obj3 = new HDFC_Bank_Version3();
obj3.Print();
```

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

Note: The basic inheritance requires base class contructor to execute before the derived class constructor. In JavaScript you have to call the super class constructor in derived class to complete the inheriance mechanism.

Syntax:

```
    class Super
    {
       constructor(){ }
    }
    class Derived extends Super
    {
       constructor(){
         super();
       }
    }
```

Ex:

```
<script>
    class Super
    {
        constructor(){
            document.write("Super Class Constructor<br>");
        }
    }
    class Derived extends Super
    {
        constructor(){
```

```
        super();
        document.write("Dervied Class Constructor");
    }
}
```

```
    let obj = new Derived();
</script>
```

Polymorphism

- Poly means "many" and morphos means "forms".
- Polymorphism allows a component to work for various situations.

Ex:

```
<script>
    class Employee
    {
        FirstName;
        LastName;
        Designation;
        Print(){
            document.write(`${this.FirstName} ${this.LastName} - ${this.Designation}<br>`);
        }
    }
    class Developer extends Employee
    {
         FirstName = "Raj";
         LastName = "Kumar";
         Designation = "Developer";
         Role = "Developer Role : Build, Debug, Test";
         Print(){
            super.Print();
            document.write(`${this.Role}`);
         }
    }
    class Admin extends Employee
    {
        FirstName = "Kiran";
         LastName = "Kumar";
         Designation = "Admin";
         Role = "Admin Role : Authorizations";
         Print(){
            super.Print();
            document.write(`${this.Role}`);
         }
    }
    class Manager extends Employee
    {
        FirstName = "Tom";
         LastName = "Hanks";
         Designation = "Manager";
         Role = "Manager Role : Approvals";
         Print(){
```

```
        super.Print();
        document.write(`${this.Role}`);
    }
}
```

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

Class Components in React

React Class Components

Issues with OOP:
- OOP will not support low level features.
- Low level features are required to interact with hardware services directly.
- OOP can't interact with hardware services directly.
- It is complex in configuration.
- It uses more memory.
- It is slow.

Features of OOP:
- Easy to reuse
- Easy to extend
- Secured Code
- Clean separation
- Dynamic Polymorphism etc..

Creating a Class Component:
1. Class is configured using declaration

```
class Name
{
}
```

2. A class gets component behaviour by extending

   a) React.Component
   b) React.PureComponent

Note: A "PureComponent" is modular, it updates only the changes in component without reloading entire component.

An "ImpureComponent" is legacy type, which loads entire component to update any change.

Syntax:

```
class  Name  extends React.Component
{
}

class  Name  extends  React.PureComponent
{
}
```

- Every class component is known as "StateFull" component.
- React.Component is the super class that configures "State" at the time of

constructing an object for class.
- If your component have to use state then it must have a constructor that calls super constructor.

Syntax:

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

- Every class component returns a markup by using "render()" method, which is a super class method.

Syntax:

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

    render() {
       return(
                <React.Fragment>

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

Ex:
admin-login.jsx

```
import React from "react";

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

     render(){
        return(
            <React.Fragment>
                <div className="container-fluid">
                <h2>Admin Login</h2>
                <dl>
                    <dt>UserId</dt>
                    <dd><input type="text"/></dd>
                </dl>
                <button>Login</button>
                </div>
            </React.Fragment>
        )
    }
}
```

State in Class Component:
- Class components can't use react hooks.
- Hence you can't use "useState()" hook in class component.
- Class component is a state full component. It comprises of default state.
- You have to configure state in constructor by using

```
        this.state = {
            key : value,
            key : value
        }
```

- You can access the state by using "this.state.key" Syntax:

```
        constructor() {
             super();
             this.state = {
              title : "Welcome"
            }
         }

        <h1> { this.state.title } </h1>
```

- To set a value into state, react class component provides setState()

    method. this.setState({ })

Ex:

```
import React from "react";


export class AdminLogin extends React.Component
{
     constructor(){
        super();
        this.state = {
            title : "Product Details",
            product: {Name:"TV", Price: 34000.44},
            categories: ["All", "Electronics", "Fashion"]
        }
     }

     render(){
        return(
            <React.Fragment>
                <div className="container-fluid">
                    <h1>{this.state.title}</h1>
                    <dl>
                        <dt>Name</dt>
                        <dd>{this.state.product.Name}</dd>
```

```
<dt>Price</dt>
<dd>{this.state.product.Price}</dd>
```

```
                    <dt>Category</dt>
                    <dd>
                        <select>
                            {
                                this.state.categories.map(category=>
                                    <option key={category}>{category}</option>
                                    )
                            }
                        </select>
                    </dd>
                </dl>

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

- Class component configures actions to perform at the time of mounting by using the

  method componentDidMount()

  componentWillMount()

Note: You can't use "useEffects()" hook.

Ex:

```
import React from "react";

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

    }

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

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

```
            this.setState({
                products: products
            })
        })
    }

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

    render(){
        return(
            <React.Fragment>
                <div className="container-fluid">
                    <h2>Select Category</h2>
                    <select>
                        {
                            this.state.categories.map(category=>
                                <option key={category}>{category}</option>
                                )
                        }
                    </select>
                    <div className="mt-3">
                        {
                            this.state.products.map(product=>
                                <img key={product.id} src={product.image} width="100" height="100"
className="m-2" />
                                )
                        }
                    </div>
                </div>
            </React.Fragment>
        )
    }
}
```

Component Cycle

- Creating Class Components
- State in Class Component
- SetState
- Methods
- Component mount phase

Events in Class Component

- Class component uses the same SyntheticEvent base.
- All Synthetic Events you used in function component are same for class component.
- The events in class component map to class methods.

Syntax:
```
        InsertClick()
        {
        }

        <button onClick={this.InsertClick}>
```

- The event methods in class component can use and execute general DOM functions.
- Class event related method can't use class state, you have to bind the event method with current class.
- You can bind any method with class in constructor.

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

Ex:
class-event-demo.jsx

```
import React from

"react";

export class ClassEventDemo extends React.Component
{
     constructor(){
        super();
        this.state = {
            Msg: ""
        }
        this.InsertClick = this.InsertClick.bind(this);
     }

     InsertClick(){
        this.setState({
            Msg: "Record Inserted"
        })
     }
```

```
render(){
  return(
      <div className="container-fluid">
          <button onClick={this.InsertClick}>Insert</button>
          <p>{this.state.Msg}</p>
      </div>
  )
```

```
    }
}
```

- You can bind event to class directly while configuring the event handler.

```
   <button onClick={ this.InsertClick.bind(this) }>
```

- You can define event to use the state without bind() method.
- You have to return the memory of specified event method, which uses the current the class memory.

```
   <button onClick={ () => this.InsertClick() }>
```

Ex:

```
import React from "react";

export class ClassEventDemo extends React.Component
{
    constructor(){
      super();
      this.state = {
          Msg: ""
      }

    }

    InsertClick(){
      this.setState({
          Msg: "Record Inserted"
      })
    }

    render(){
      return(
          <div className="container-fluid">
              <button onClick={()=> this.InsertClick()}>Insert</button>
              <p>{this.state.Msg}</p>
          </div>
      )
    }
}
```

- Class event args are same as you defined for function components. e.target.id
  e.clientX

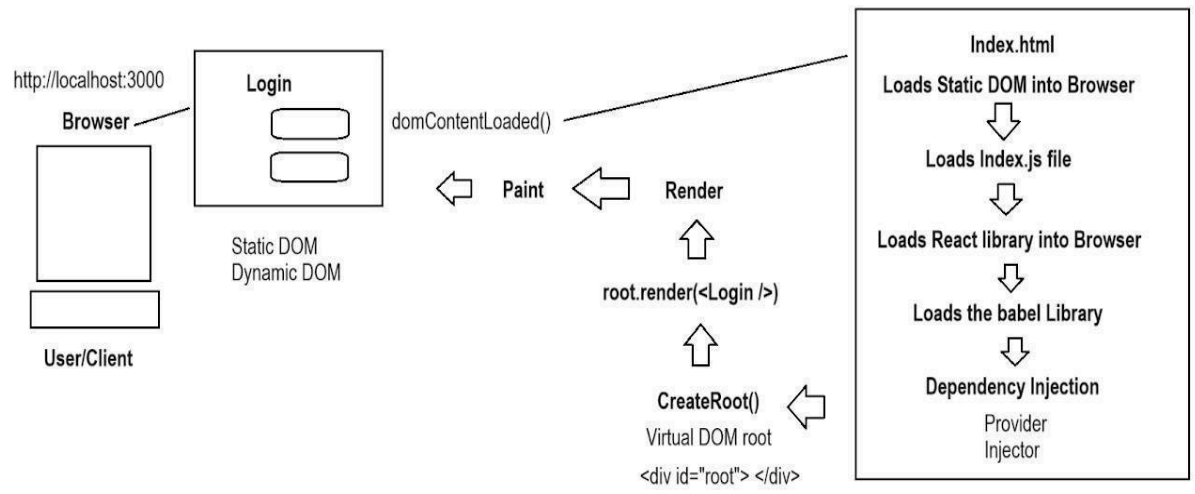
Ex:

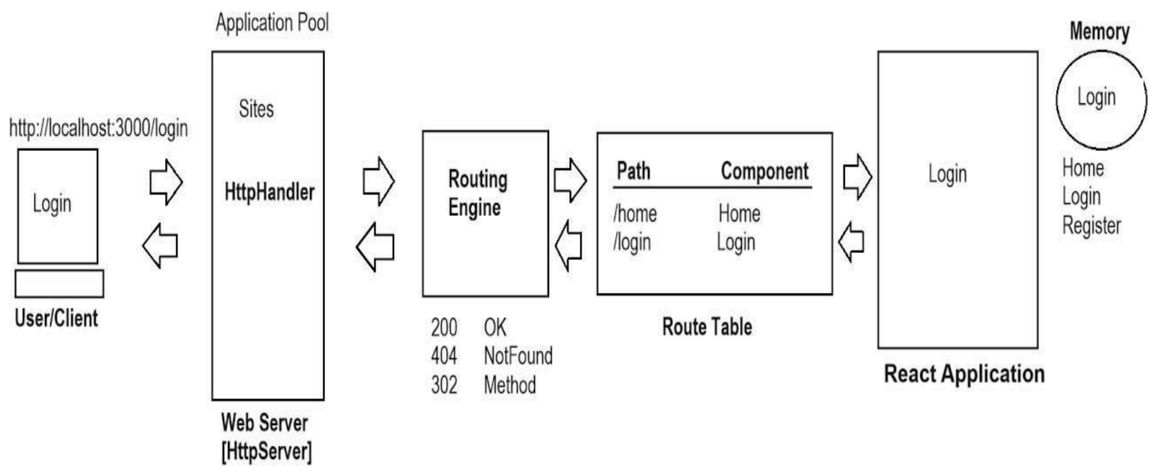
```
import axios from "axios";
import React from "react";

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

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


    LoadCategories(){
        axios.get("http://fakestoreapi.com/products/categories&quot;)
        .then(response => {
            response.data.unshift("all");
            this.setState(
              {
                  categories: response.data
              }
            )
        })
    }

    LoadProducts(url){
      axios.get(url)
      .then(response => {
          this.setState(
            {
                products: response.data
            }
          )
      })
    }

    componentDidMount(){
      this.LoadCategories();
      this.LoadProducts("http://fakestoreapi.com/products&quot;);
    }

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

    render(){
      return(
          <div className="container-fluid">
              <header className="bg-dark h2 text-white p-2 text-center mt-2">Shopper</header>
              <section className="row">
                  <nav className="col-2">
                    <label className="form-label fw-bold">Select Category</label>
                    <div>
                     <select onChange={this.handleCategoryChange} className="form-select">
                         {
                                                                                    this.state.categories.map(category =>
                                                                                        <option key={category} value={category}>
  </option>                                                                             {category.toUpperCase()}
```

```
)                                                                               }
                 </select>
                </div>
             </nav>
             <main className="col-10 d-flex flex-wrap">
                 {
                    this.state.products.map(product=>
                        <div key={product.id} className="card p-2 m-2" style={{width:'200px'}}>
                            <img src={product.image} className="card-img-top" height="140" />
                            <div className="card-header">
                                <p>{product.title}</p>
                            </div>
                        </div>
                        )
                    }
                </main>
            </section>
        </div>
    )
  }
}
```

Component Life Cycle Hooks [Methods]





Component Life Cycle Methods

FAQ's:

1. What are the activities performed on component creation? A.
   - State is configured
   - Properties are defined
   - Methods are bound to current class

2. What are the actions on Mount? A.
   - Subscribe Events
   - Data Binding
   - Class Binding
   - Style Binding
   - Event Binding
   - Values are assigned to state

3. What are the actions on Update? A.
   - Events will fireup
   - Expressions are evaluated
   - Results are computed
   - Response is ready [render]

4. When render occurs? A.
   - After mount
   - After update

5. When unmount occurs?
A. When user navigates from one component to another.
The active component will unmount and requested component will mount.

6. What are the actions on Unmount? A.
   - Unsubscribe to events
   - Destroy the state
   - Clean up the memory allocated for component

Ex:

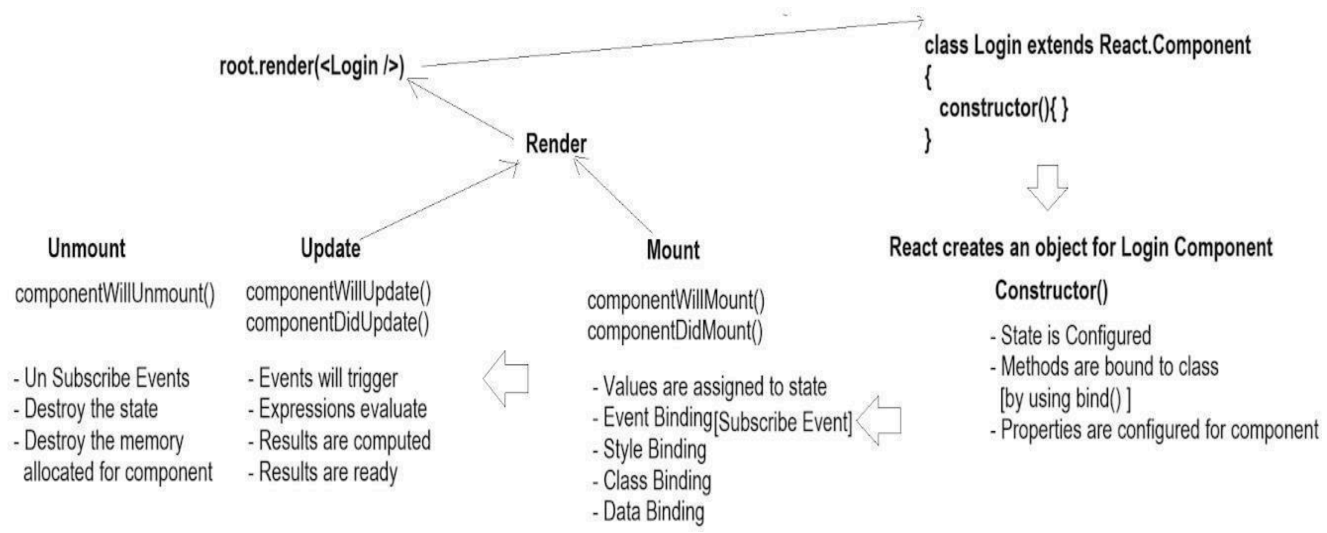
```
import React from "react";

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

```
componentDidMount(){
    alert(`Login Component Mounted Successfully..`);
}

componentWillUnmount(){
        alert(`Login component will unmount`);
    }


    render(){
        return(
            <div>
                <h3>Login</h3>
                <dl>
                    <dt>User Name</dt>
                    <dd><input type="email"></input></dd>
                </dl>
                <button>Login</button>
            </div>
        )
    }
}
export class Register extends React.Component
{
    constructor(){
        super();
    }

    componentDidMount(){
        alert(`Register Component Mounted Successfully..`);
    }

    componentWillUnmount(){
        alert(`Register component will unmount`);
    }
    render(){
        return(
            <div>
                <h3>Register</h3>
            </div>
        )
    }
}

export class CycleDemo extends React.Component
{
     constructor(){
        super();
        this.state = {
             component: ""
        }
     }

     LoginClick(){
```

```
    this.setState({
        component: <Login />
    })
}
```

```
    RegisterClick(){
      this.setState({
         component: <Register />
      })
    }


    render(){
       return(
          <div className="container-fluid">
             <div className="mt-3">
             <button onClick={this.LoginClick.bind(this)} className="btn btn-primary me-
2">Login</button> <button onClick={this.RegisterClick.bind(this)} className="btn btn-danger
ms-1">Register</button>
             </div>
             <div className="mt-4">
                 {this.state.component}
             </div>
          </div>
       )
    }
}
```

React Material



Class Component Properties

- Class component properties are defined in constructor.
- Properties are object type, which you can access using "this" keyword.

Syntax:

```
    class Name extends React.Component
    {
      constructor(props)
      {
        super();
      }
      render() {
         return(
            <div>
                {this.props.key}
            </div>
        )
    }
   }
    <Name property="value"> </Name>
```

Ex:
 Components-Library
 navbar.jsx

```
import React from "react";

export class Navbar extends React.Component
{
    constructor(props){
        super();
    }
    render(){
        return(
            <nav className={`p-3 d-flex justify-content-between ${this.props.theme}`}>
                <div className="h4">{this.props.BrandTitle}</div>
                <div>
                    {
                        this.props.MenuItems.map(item=>
                            <button className="btn btn-link" key={item}>{item}</button>
                            )
                    }
                </div>
                <div>
                    <span className="bi bi-person-fill me-2"></span>
                    <span className="bi bi-search me-2"></span>
                    <span className="bi bi-bell-fill"></span>
                </div>
            </nav>
        )
```

```
    }
}
```

shop.jsx

```
import React from "react";
import { Navbar } from "../../components-libary/navbar";


export class Shop extends React.Component
{
     constructor(){
        super();
     }
     render(){
        return(
            <div className="container-fluid">
                <Navbar theme={'bg-warning'} BrandTitle="Shopper." MenuItems={["Home",
"Shop", "Kids", "Men"]} />
                <Navbar BrandTitle="Training Online" MenuItems={["Home", "Faculty",
"Courses"]} />
            </div>
        )
     }
}
```

React Material Component Library

- It is a component library built by "React" community for building interactive UI and native UI for react application.
- It is good for rapidly building an UI for react apps.
- It provides responsive components and native components.

Setup React Material UI component library for Project: [mui.com]

1. Install MUI

```
  > npm install @mui/material @emotion/react @emotion/styled
```

2. Import the library required for component

```
       import { TextField } from @mui/material;              => legacy
                      (or)
       import TextField from @mui/material/TextField;  => Modular
```

3. Expore the component "API" documentation to know about its properties and styles.

```
    <TextField variant="" label="" color="" />
```

Ex:

material-demo.jsx

```
import { TextField } from "@mui/material";
import { useState } from "react";

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

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

    return(
        <div className="container-fluid">
            <div className="w-25">
                <h3>Bootstrap Style</h3>
                <label className="form-label">User Name</label>
                <div>
                    <input type="text" className="form-control" placeholder="Enter Your Name" / >
                </div>
            </div>
            <div className="w-25">
                <h3>React MUI</h3>
                <TextField color="success" onChange={handleNameChange} label="User Name"
variant="standard" />
            </div>
            <p>Hello ! {userName} </p>
        </div>
    )
}
```

Ex: Date Picker

https://mui.com/x/react-date-pickers/getting-started/

```
import { TextField } from "@mui/material";
import { useState } from "react";
import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs';
import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider';
import { DatePicker } from '@mui/x-date-pickers/DatePicker';


export function MaterialDemo()
{

    return(
        <LocalizationProvider dateAdapter={AdapterDayjs}>
            <div className="container-fluid">
                <h2>Departure</h2>
            <div className="w-25">
                <DatePicker />
            </div>
```

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

React Routing

Routing

- Web development is all about Objects and Techniques.
- Web Objects include both client side and server side.
  - Document Objects
  - Browser Objects
  - CSS Objects
  - Session Object
  - Application Object
  - Cookie Object
  - Server Object
  - Path Object
  - Memory Object etc..
- Web Techniques include both server side & client
  side Style Binding
  Class Binding
  Data Binding
  Model Binding
  Caching
  State Management
  Routing
  Event Binding etc..

- Routing is a technique used in web applications to configure user and SEO friendly URL.
- It allows to query any content directly from URL.
- It enables Ajax requests, which allows to load new details into page without reloading the page.
- It enable SPA [Single Page Application] approach, where user can stay on one page and can access everything into page.
- It can enable partial postbacks and asynchronous rendering.
- Routing can be both server side or client side.
- React JS routing library is not a part of its core library.
- React routing library have 2 major versions
  - a) V5 [React Router DOM 5] upto 17x of react
  - b) V6 [React Router DOM 6] from 18x

of react Note: V6 is a complete re-write of routing

library.

Without Routing:
http://www.amazon.in/electronics.aspx?category=mobiles&model=samsung

With Route:
http://www.amazon.in/electronics/mobiles/samsun
g

Setup Routing:
1. Install react-router-dom library

    >npm i react-router-dom

    https://reactrouter.com/en/main

2. Router DOM components you can import and directly implements in application

```
<BrowserRouter>
<Routes>
<Route>
<Link>
<Outlet>
etc..
```

3. Routing is configured in startup component and it allows to access other components and render into startup component.

Router Components:

1. BrowserRouter
- It is a provider for routes in virtual DOM.
- Provider is a component of DI [Dependency Injection] which is used to location your resource in memory.
- BrowserRouter can translate the virtual DOM routes into actual DOM location object.

2. Routes
- It is a component for configuring collection of routes.
- It is used to construct a "Route Table".

3. Route
- It is used to configure a route inside routes collection.
- Route defines the request path and element to render as response.

Syntax:

```
<BrowserRouter>
    <Routes>
      <Route path="" element=""> </Route>
      <Route path="" element=""> </Route>
    </Routes>
</BrowserRouter>
```

- Route path comprises meta characters and request

  name. path="/" It is used configure default to render

  path="*" It is used to configure the content to render when requested component not found.

  path="name" It is used to request specific

4. Link
- It is used to configure a Hyperlink that navigates to specific path in routing.

```
<Link to="path"> Text | Image </Link>
```

Ex:

1. Create a new react app

```
> npx create-react-app  react-routing-demo
```

2. Install router dom

```
> npm i react-router-dom  --save
```

3. Go to "app.js"

```
import logo from './logo.svg';
import './App.css';
import { BrowserRouter, Link, Route, Routes } from 'react-router-dom';
import { Login } from './components/login/login';

function App() {
  return (
    <div className="App">

        <BrowserRouter>
            <header>
              <h1>Shopper.</h1>
              <nav>
                 <Link to='/'>Home</Link> <span></span>
                 <Link to='kids'>Kids</Link> <span></span>
                 <Link to='men'>Men's Fashion</Link> <span> </span>
                 <Link to='login'> Login </Link>
              </nav>
            </header>
            <hr />
            <Routes>
                <Route path='/' element={<><h2>Home</h2><p>Year end sale 40%
OFF</p></>}></Route>
                <Route path='kids' element={<><h2>Kids Fashion</h2><p>30% OFF
on kidswear</p></>} ></Route>
                <Route path='men' element={<><h2>Men's Fashion</h2><p>Winter wear, Shoes,
Jeans..</p></>} ></Route>
                <Route path='login' element={<Login />} />
                <Route path='*' element={<><code>Not Found: Path you requested not
found</code></>}></Route>
            </Routes>
        </BrowserRouter>
    </div>
  );
}

export default App;
```

React Route Parameters

<BrowserRouter>
<Routes>
<Route>
<Link>

Route Parameters

- A parameter allows to query any content directly from URL.
- In traditional web applications we use lot of query strings.

  http://www.amazon.in/products.aspx?category=electronics&subcategory=mobiles

- Route parameter provides a uniform and simple technique for queries

  http://www.amazon.in/products/electronics/mobiles

- Route with parameter is configure using the following syntax

  <Route path="name/:param1/:param2.." element={ <Component /> } />

- Route parameters are explicitly passed with values from URL or Link

  component. http://localhost:3000/name/value1/value2

  <Link to="/name/value1/value2"> Text </Link>

- React provides "useParams()" hook, from "react-router-dom" library.
- It can read route parameters and access in any

  component. let params = useParams();

- All parameters are returned as

  "object". params = { key : value,

  key: value }

  params = { param1: value1, param2: value2 }

  params.param1 = value1
  params.param2 = value2

Ex:
app.js

```
import logo from './logo.svg';
import './App.css';
import { BrowserRouter, Link, Route, Routes } from 'react-router-dom';
import { Login } from './components/login/login';
import { Mobiles } from
'./components/mobiles/mobiles'; import { Details }
from './components/details/details';

function App() {
  return (
    <div className="App">

        <BrowserRouter>
            <header>
              <h1>Shopper.</h1>
              <nav>
                 <Link to='/'>Home</Link> <span></span>
                 <Link to='kids'>Kids</Link> <span></span>
                 <Link to='men'>Men's Fashion</Link> <span> </span>
                 <Link to='login'> Login </Link> <span></span>
                 <Link to='mobiles/iphone'> Iphone </Link> <span></span>
                 <Link to='mobiles/realme'> Realme </Link>
              </nav>
            </header>
            <hr />
            <Routes>
                <Route path='/' element={<><h2>Home</h2><p>Year end sale 40%
OFF</p></>}></Route>
                <Route path='kids' element={<><h2>Kids Fashion</h2><p>30% OFF
on kidswear</p></>} ></Route>
                <Route path='men' element={<><h2>Men's Fashion</h2><p>Winter wear, Shoes,
Jeans..</p></>} ></Route>
                <Route path='login' element={<Login />} />
                <Route path='mobiles/:category' element={<Mobiles />} />
                <Route path='details/:id/:name/:price/:stock' element={<Details/>} />
                <Route path='*' element={<><code>Not Found: Path you requested not
found</code></>}></Route>
            </Routes>
        </BrowserRouter>
    </div>
  );
}

export default App;
```

mobiles.jsx

```
import { useEffect, useState } from "react"
import { useParams } from "react-router-dom"

export function Mobiles()
{
    const [mobiles, setMobiles] = useState([{Name:"Realme C16", Category:"realme"},
{Name:"Iphone 14 pro-max", Category:"iphone"}, {Name: "Iphone 13",

    Category:"iphone"}]) let params = useParams();

    return(
        <div>
```

```
<h4>Mobiles - Search Results for {params.category.toUpperCase()} </h4>
<table width="400" border="1">
    <thead>
        <tr>
```

```
                        <th>Name</th>
                        <th>Category</th>
                    </tr>
                </thead>
                <tbody>
                    {
                      mobiles.filter(mobile => mobile.Category===params.category).map(item=>
                         <tr key={item.Name}>
                            <td>{item.Name}</td>
                            <td>{item.Category}</td>
                         </tr>
                        )
                    }
                </tbody>
            </table>
        </div>
    )
}
```

details.jsx

```
import { useParams } from "react-router-dom"


export function Details()
{
    let params = useParams();

    return(
        <div>
            <h3>Details</h3>
            <p>
                Id : {params.id} <br />
                Name : {params.name} <br/>
                Price : {params.price} <br/>
                Stock : {(params.stock==="true")?"Available": "Out of Stock"}
            </p>
        </div>
    )
}
```

Nested Routes

Routes
Route Parameters

FAQ: What is relative and absolute path?
Ans: Relative path defines a path within the context of current path.

Existing Path: http://localhost:3000/products

```
<Link to="details/1"> Details </Link>
```

New Relative Path: http://localhost:3000/products/details/1

Absolute path configures a new path to set by removing the existing path.

```
<Link to="/details/1"> Details </Link>
```

New Absolute Path   :   http://localhost:3000/details/1

FAQ: How a relative path is configured ?
Ans : By using nested routes technique.

Syntax:

```
<Routes>
  <Route path="parent"  element={ <ParentElement /> }>
     <Route path="child" element={ <ChildElement />} </Route>
  </Route>
 </Routes>
```

"child" path is the relative path for "parent".

FAQ: Where the route path results are rendered?
Ans: If it is absolute path then It renders in a new screen, how ever in root container.
If it is relative path then it renders in outlet area. It is defined with <Outlet> component of "react-router-dom".

Note: A component have to mount again when ever the relative path changes.
You can define the route parameter as "dependency" for component mount phase. "useEffect()" is a hook that configures mount phase with dependency.

Syntax:

```
const params = useParams();

useEffect(()=>{

},[ params.paramerterName ])
```

"Mount Dependency" is an array type hence there can be multiple dependencies.

FAQ: Can we have multiple outlets ?
Ans: Yes.

FAQ: How to handle dynamic navigation?
Ans: By using "useNavigate()" hook. [V6]
"useHistory()" hook. [V5]

Syntax:

```
    let  navigate  = useNavigate();

    navigate("relativePath");
    navigate("/absolutePath");
```

FAQ: What are search Parameters?
Ans : A search parameter is the query string in URL.

FAQ: How to access querystirng?
Ans: JavaScript => location.search
React => useSearchParams(); => It returns a "map" with key/value

Syntax:

localhost:3
000/produc
ts?name1=
value1&na
me2=value
2 let
[search] =
useSearch
Params();
s
e
a
r
c
h
.
k
e
y
s
(
)
s
e
a
r
c
h
.
v
a
l

u
e
s
(
)
s
e
a
r
c
h
.
e
n
t
r
i
e
s
(
)
e
t
c
.
.

React Cookies

React Cookie Library

- Cookie is a simple text document where client credentials can be stored.
- It can be appended into client browser memory or permanent memory. [HDD]
- Cookies are classified into
  a) Inmemory [Temporary]
  b) Persistent [Permanent]
- Javascript DOM can manage cookies using "document.cookie" property.
- React requires a virtual DOM library for handling cookies.
- React doesn't provide any built-in library for cookies, you have to implement using 3rd party.

1. Install React 3rd Party Cookie Library

```
> npm i react-cookie --save
```

2. If cookie is provided as a service by any 3rd parth then it will have 2 components

```
> Provider    : Locates the cookie in memory
> Injector    : Injects into a component

Provider        <CookieProvider>
Injector        useCookies()
```

3. You have to configure all component within the provider scope in order to use cookies. Hence configure in "index.js"

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

    <CookiesProvider>
        <TutorialIndex />        => root.render()
      </CookiesProvider>
```

4. Creating a cookie in any component import

```
{ useCookies } from 'react-cookie';

const [cookies, setCookie, removeCookie] = useCookies(['name']);

setCookie("name", value, [ { options } ]);
```

5. Accessing Cookie

```
cookies.name
```

6. Verifying cookie

```
if(cookies.name==null)        => true if cookie exists
```

7. Removing cookie

   removeCookie("name")

FAQ: How to verify cookie status in browser? [enabled or not]

Ans: JavaScript "navigator.cookieEnabled"

```
if (window.navigator.cookieEnabled)
{
}
```

TypeScript

TypeScript

FAQ: What are the issues with JavaScript?

Ans:

- It is not a strongly typed language.
- It is not implicitly strictly typed.
- It is not an OOP language.
- No dynamic polymorphism.
- No code level security.
- Not easy to extend.

TypeScript

- It is strictly typed mode of JavaScript.
- Anders Hejlberg introduced "TypeScript" as language. He is the architect of language called "C#" [.net]. [Microsoft]
- Google started a language called "atscript" in early 2013, to provide an alternative for JavaScript.
- Google started using TypeScript as language for its framework "Angular".
- TypeScript is transcompiled into JavaScript.
- TypeScript is just an alternative for JavaScript, it is not replacement.

Features:
- It is strongly typed.
- It is strictly typed.
- It is an OOP language.
- It supports complete extensibility, reusability, sepration.
- It supports code level security etc..

TypeScript Architecture:

1. TypeScript Core Compiler
   - It verifies the code syntax.
   - It checkes the data type.
   - It manages input and output operations
   - It verifies code blocks and conversions of data.

   [core.ts, program.ts, scanner.ts, parser.ts, emitter.ts, checker.ts]

2. Standalone Compiler
   - It transcompiles the typescript code into javascript.
   - Browser can interpret only

   JavaScript. [tsc.ts]

3. Language Service
   - A service is pre-defined business logic used by compilers to verify the code and

compile the code.

- Service is a set of factories.
- Factory is a set of functions and

values. [services.ts]

4. TS Server
- It hosts the TS programs.
- It compiles them and manages request /

response. [tsserver.ts]

5. VS Shim
- It makes the language platform neutral.
- It can run on any enivronment and

device. [shims.ts]

```
code => translated => native code => platform dependent.
             [un managed]

code => translated => Managed Code => platform independent
              [shims.ts]
```

6. Managed Language Service
- It is the code suitable for every OS service.
- It comprises of set of libraries that can run on every device and in any environment.

Setup and Install TypeScript on your PC:

1. Open command prompt

2. Run the command

```
> npm  install typescript -g
```

3. Check the version

```
> tsc -v
```

4. Create a project for

   Typescript D:\ts-project

```
> npm init -y             [package.json]
> tsc -init               [tsconfig.json]
```

5. Add folders
   - public
   - src

6. Add a TS file

   index.ts

```
let uname:string = "John";
let age:number = 24;

function PrintDetails():string
{
    return `Hello ! ${uname} your age ${age}`;
}
```

7. Transcompile

```
    > tsc index.ts
```

8. Add Index.js to HTML page

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../src/index.js"></script>
    <script>
        function bodyload(){
            document.querySelector("p").innerHTML = PrintDetails();
        }
    </script>
</head>
<body onload="bodyload()">
    <p></p>
</body>
</html>
```

TypeScript Language

- Features of TypeScript
- Architecture
- Setup Environment for TypeScript
- Trans compiling

Typescript Language

1. Variables
 - Typescript variables are strongly typed.
 - You have to define the data type while configuring variable

Syntax:

```
        var name:dataType = value;
```

 - If data type is not defined by default it is "any" type.
 - You can assign various types of values.

```
         var x;          => x is any type
         x = 10;
         x = "A";
         x = true;
```

- Typescript supports "type inference", if data type is not define then it can configure data type according to value initialized.

```
        var x = 10;     => x is number
        x = "A";        => invalid

        var x:number;
        var x:string;
```

- The data types of typescript are all same as JavaScript

    a) Primitive Types
        number
        string
        boolean
        null
        undefined
        symbol
        bigint

    Syntax:

```
        var x : number;
        const x : string =
        "A"; let x:boolean;
```

- TypeScript supports "union" of types.

  Syntax:

```
    var x : number | string;
```

```
        var x : number | string | boolean;
```

Ex:

```
    var uname:string|null = prompt("Enter Name");

    if(uname==null) {
         document.write("Please enter name");
    } else {
         document.write("Hello ! " + uname);
    }
```

Ex:

```
var uname:string|null = prompt("Enter Name");
if(uname==""){
    document.write("Name Can't be empty");
} else if(uname==null) {
    document.write("You canceled..");
} else {
    document.write("Hello ! " + uname);
}
```

index.html

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

TypeScript Non-Primitive Types

1. Array
- Typescript array can handle various types of data same as JavaScript.
- It can also restrict to same type as other languages.

JavaScript:

```
        var values = [ ];                   => any type
        var values = new Array();
```

TypeScript:

```
        var values:number[] = [ ];
```

```
var values:string[] = new Array();
var values:any[] = [ ];
var values:any[] = new Array();
```

Ex:
```
var categories:string[] = [];
categories[0] = "Electronics";
categories[1] = "Footwear";
categories[2] = "Fashion";
console.log(categories.toString());
```

Q: Check the syntax and define valid or invalid

```
        var values:any[] = [ true, 10, "A" ];           => valid

        var values:string[] = [ true, 10, "A"];          => invalid

        var values:any[] = [true, 10, "A"];              => valid

        var values:any[] = new Array(10, true);          => invalid

        var values:any[] = new Array(10, 20);            => valid

        var values:any[] = new Array();                  => valid
        values[0] = 10;
        values[1] = "A";
```

Note: You can't initialize different data types into array constructor even when the type configured as "any". However you can assign various types.

Q: What is a "Tuple"?
A: It is a collection that allows various types of values to initialize or assign into memory.

```
          var values:any[ ] = [ ];               => tuple
          var values:any[ ] = new Array();       => not tuple
```

- Typescript supports union of types for Array.
- How ever you can't initialize various types configured in union. You can assign.

```
    var values:string[] | number[] = [10, 20, 30];         => valid
    var values:string[] | number[] = [10, "A"];            => invalid
    var values:string[] | number[] = ["A", "B"];           => valid

    var values:stirng[] | number[] = [ ];                  => valid
    values[0] = 10;
    values[1] = "A";
```

Note: All array methods are same as in JavaScript.

2. Object
- It is a key and value collection.
- Key is string type and value can be strongly typed in

Typescript. Syntax:

```
    var obj:{key:datatype, key:datatype} = { }
```

Ex:

```
var product:{Name:string, Price:number, Stock:boolean, Rating:{Rate:number}} = {
    Name: "Samsung TV",
    Price: 24000.44,
    Stock: true,
    Rating: {Rate: 4.2}
}
console.log(`
    Name : ${product.Name}\n
    Price: ${product.Price}\n
    Stock: ${product.Stock}\n
    Rating: ${product.Rating.Rate}
`);
```

- Every key configured in structure is mandatory to implement.
- You can configure optional key by using null reference character "?".

Syntax:

```
    var obj : {key:dataType, key?:dataType} = { }
```

Ex:

```
var product:{Name:string, Price:number, Stock:boolean, Rating:{Rate:number}, Qty?:number}
= {
    Name: "Samsung TV",
    Price: 24000.44,
    Stock: true, Rating:
    {Rate: 4.2},
    Qty: 2
}
if(product.Qty){
    console.log(`
    Name : ${product.Name}\n
    Price: ${product.Price}\n
    Stock: ${product.Stock}\n
    Rating: ${product.Rating.Rate}\n
    Qty: ${product.Qty}
`);
} else {
    console.log(`
    Name : ${product.Name}\n
    Price: ${product.Price}\n
    Stock: ${product.Stock}\n
    Rating: ${product.Rating.Rate}
`);
}
```

Ex: Array of Objects

```
var values: { }[ ] = [ { }, { } ];
var values: {key:type} [ ] = [ { }, { } ];
var products:{Name:string, Price:number}[] = [
    {Name: "TV", Price: 34000.44},
    {Name: "Mobile", Price: 24000.33}
];
for(var product of products) {
    console.log(`${product.Name} - ${[product.Price]}\n`);
}
```

3. Date Type

```
        var dob : Date = new Date();
```

4. Regular Expression Type

```
        var pattern: RegExp = / /;
```

Note:
- TypeScript statements and operators are same as defined in JavaScript

TypeScript Function

Functions and Interface

Typescript Functions

- Function is configured with return type. function Name() : dataType | void
  {
  }
- Function parameters are defined with type.
  function Name(param:type) : dataType | void
  {

  }

Ex:

```
function Total(qty:number, price:number):number
{
    return qty * price;
}

function Print(Name:string, Price:number, Qty:number):void {
    console.log(`
      Name : ${Name}\n
      Price: ${Price}\n
      Qty: ${Qty}\n
      Total: ${Total(Qty, Price)}
    `);
}
Print("Samsung TV", 45000.33, 2);
```

- Function parameter can be configured as "optional parameter".
- Optional parameters are defined using null-reference character "?"
- Optional parameter must be the last parameter.
- A required parameter can't be defined after optional

parameter. Ex:

```
function Total(qty:number, price:number):number
{
    return qty * price;
}

function Print(Name:string, Price:number, Qty:number, Rating?:number):void {
   if(Rating){
    console.log(`
    Name : ${Name}\n
    Price: ${Price}\n
    Qty: ${Qty}\n
```

Total: ${Total(Qty, Price)}\n
Rating: ${Rating}

```
  `);
   } else {
    console.log(`
    Name : ${Name}\n
    Price: ${Price}\n
    Qty: ${Qty}\n
    Total: ${Total(Qty, Price)}
  `);
   }
}
Print("Samsung TV", 45000.33, 2, 4.3);
```

Typescript OOP

Contracts in OOP:
- Contract defines rules for designing a component.
- Rules can be for data and logic.
- Developer can't ignore any rule until or unless it is configured as optional.
- Developer can't add any functionality or data that is not defined in contract.
- In OOP contracts are designed by using "Interface".

Syntax:
```
        interface Name
        {
            // rules for properties & methods
        }
```

- It must contain only rules not implementation.
```
        interface Name
        {
            Property:DataType;
            Property:DataType = value;          // invalid
        }
```

Ex:
```
interface IProduct
{
    Name:string;
    Price:number;
    Qty:number;
    Total():number;
    Print():void;
}
let product:IProduct = {
    Name : "Samsung TV",
    Price: 34000.33,
    Qty: 2,
    Total : function() {
       return this.Qty * this.Price
    },
    Print : function(){
        console.log(`Name: ${this.Name}\n Price: ${this.Price}\n Qty : ${this.Qty}\n Total
:${this.Total()}`);
    }
}
product.Print();
```

- A contract can have optional rules. [?]
- Optional rules are defined to configure Goals.

Syntax:
```
    interface IName
    {
          Property?:Type;
          Method?():Type | void;
    }
```

Ex:
```
interface IProduct
{
    Name:string;
    Price:number;
    Qty:number;
    Total?():number;
    Print():void;
    Rating?:number;
}
let product:IProduct = {
    Name : "Samsung TV",
    Price: 34000.33,
    Qty: 2,
    Total : function() {
       return this.Qty * this.Price
    },
    Print : function(){
        console.log(`Name: ${this.Name}\n Price: ${this.Price}\n Qty : ${this.Qty}\n Total
:${this.Total()}`);
    }
}
product.Print();
```

- A contract can have readonly rules.
- Readonly rules will allow to initialize but will not allow to assign.

Syntax:
```
    interface IName
    {
       readonly Property:Type;
    }
```

Ex:

```
interface IProduct
{
    Name:string;
    readonly Price:number;
    Qty:number;
    Total?():number;
    Print():void;
}
let product:IProduct = {
    Name : "Samsung TV",
    Price: 34000.33,
    Qty: 2,
    Total : function() {
       return this.Qty * this.Price
    },
    Print : function(){
        console.log(`Name: ${this.Name}\n Price: ${this.Price}\n Qty : ${this.Qty}\n Total
:${this.Total()}`);
    }
}
product.Price = 50000.55; // invalid
product.Print();
```

- A contract can be extended by another contract.
- It supports multiple inheritance of contracts.

Syntax:

```
    interface IName extends Name1, Name2, Name3
    {

    }
```

Ex:

```
interface IBasicDetails
{
    Name:string;
    readonly Price:number;
    Qty:number;
    Total?():number;
    Print():void;
}
interface ICategory
{
    CategoryName:string;
}
interface IProduct extends ICategory, IBasicDetails
{
    Title:string;
}

let product:IProduct = {
```

Title: 'TV Details',
Name : "Samsung TV",
Price: 34000.33,
Qty: 2,

```
    CategoryName: 'Electronics',
    Total : function() {
       return this.Qty * this.Price
    },
    Print : function(){
        console.log(`${this.Title}\nName: ${this.Name}\n Price: ${this.Price}\n Qty : ${this.Qty}\n
Total :${this.Total()} \n Category : ${this.CategoryName}`);
    }
}
product.Print();
```

Ex:

```
interface Version1
{
    Personal:string;
    NRI:string;
}
interface Version2 extends Version1
{
    CreditCard:string;
}
interface Version3 extends Version2
{
    Loans:string;
}

let BankApp2022:Version2 = {
    Personal : "Personal Banking",
    NRI : "NRI Banking",
    CreditCard: "Credit Card Services"
}
```

Ex:

```
interface Version1
{
    Personal:string;
    NRI:string;
}
interface Version2
{
    Personal:string;
    CreditCard:string;
}
interface Version3
{
    Loans:string;
}

interface BankApp extends Version1, Version2, Version3
```

```
{
    BankName:string;
}
```

```
let HDFC:BankApp = {
    BankName : "HDFC Bank",
    Personal : "Personal Banking",
    NRI : "NRI Banking",
    CreditCard: "Credit Card Services",
    Loans: "Car, Personal Loans"
}

for(var property in HDFC)
{
    console.log(`${property} : ${HDFC[property]}`);
}
```

Class

Classes and Templates

TypeScript OOP
- Contracts - Interface
- Rules for properties & methods
- Readonly rules
- Optional rules
- Extending Contracts

TypeScript Class

- Configuring a class is same as JS.

a) Declaration
b) Expression

```
class Name
{
}

let ref = class { }
```

- Class members are same

a) Property
b) Accessor
c) Method
d) Constructor

- TypeScript class provides static and non-static members. Static:
- It refers to contineous memory.
- The memory allocated for first object will continue for next object.
- Static members are defined by using "static" keyword.
- They are accessed within or outside class by using class name.
- It is connected in access.

Dynamic:
- It refers to discreet memory.
- The memory allocated for an object is destroyed when object completes its actions.
- Memory is newly allocated for every object.
- It is disconnected in access.
- By default every member is dynamic.
- They are accessed within class by using "this" keyword and outside class by using instance of class.

Ex:

```
class Demo
{
   static s = 0;
   n = 0;
```

```
constructor(){
    Demo.s = Demo.s + 1;
```

```
        this.n = this.n + 1;
    }
    Print():void {
        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 provides access modifiers

    a) public
    b) private
    c) protected

public:
 - It configures accessiblity for a member.
 - A public member is accessible from any location, within the class or outside the class.
 - You can access in derived class or from any another class.

private:
 - It configures a private access for member.
 - It is accessible only with in the specified class.
 - It is not accessible outside class.

protected:
 - It configure a protected access for member.
 - Protected refers to accessiblity within the class and in derived class only by using derived class object.

Ex:

```
class Super
{
    public Name = "TV";
    private Price = 45000;
    protected Stock = true;
}
class Derived extends Super
{
    public Print(obj:Derived):void {
        obj.Name;
        obj.Stock;
    }
```

}

- Properties are configured with access modifier and data type

```
    public Name:string;
    private Price:number;
```

- Methods are configure with access modifiers , return type and parameter typ

    e. public Total(quantity:number, price:number):number

```
    {
       return quantity * price;
    }

    public Print() : void
    {
    }
```

- Inheritance, Aggregation and Polymorphism are same.

Templates in TypeScript

Ex:

```
interface ProductContract
{
    Name:string;
    Price:number;
    Qty:number;
    Total():number;
    Print():void;
}
abstract class ProductTemplate implements ProductContract
{
        public Name: string = "";
        public Price: number = 0;
        public Qty: number = 0;
        abstract Total():number;
        abstract Print():void;
}
class ProductComponent extends ProductTemplate
{
    Name = "Samsung TV";
    Price = 46000.44;
    Qty = 2;
    Total(){
        return this.Qty * this.Price;
    }
    Print(){
        console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Tot
al()}`);
    }
}
```

```
let obj = new ProductComponent();
obj.Print();
```

TypeScript Templates

- A template comprises of pre-defined data and logic, which you can implement and customized according to requirements.
- Templates are mostly used in rollouts.
- A template is configured as "abstract" class.
- An abstract class comprises of methods and properties implemented with functionality and marked for implementation.
- You can mark using "abstract".

```
abstract PropertyName:type;
abstract MethodName():type | void;
```

- Abstract members are implemented in extended class.
- An abstract class hides the structure and provides only functionality for implementation, which is known as "Abstraction".
- If a class is having at least one abstract member then it must be marked as abstract.

Ex:

```
interface ProductContract
{
   Name:string;
   Price:number;
   Qty:number;
   Total():number;
   Print():void;
}
abstract class ProductTemplate implements ProductContract
{
      public Name: string = "";
      public Price: number = 0;
      public Qty: number = 0;
      abstract Total():number;
      abstract Print():void;
}
class ProductComponent extends ProductTemplate
{
   Name = "Samsung TV";
   Price = 46000.44;
   Qty = 2;
   Total(){
      return this.Qty * this.Price;
   }
   Print(){
      console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Tot
al()}`);
```

```
    }
}
let obj = new ProductComponent();
obj.Print();
```

Generics

- Generic refers to "Type Safe".
- A type safe component allows any type of data and also restricts to specific type once initialized with data type.
- You can use generic type for
    a) function
    b) method
    c) parameter
    d) property
    e) class

Ex:
```
interface IProduct
{
   Name:string;
   Price:number;
}

function Print<T>(data:T):T
{
   return data;
}
console.log(Print<string>("Welcome"));
console.log(Print<number>(20));
console.log(Print<IProduct>({Name: "TV", Price: 34000.44}));
console.log(Print<string[]>(["A", "B", "C"]));
```

- You can't use operators on generic types, every functionality you must handle using methods or functions.

Ex:
```
function Sum(a:any, b:any) {
   return a + b;
}

function AddNumbers<T>(a:T, b:T)
{
   return Sum(a,b);
}

console.log(AddNumbers<number>(30, 40));
```

Ex:
```
interface IOracle
{
   UserId:string;
   Password:string;
   Database:string;
}
```

```
interface IMongoDb
{
    Url:string;
}
```

```
interface IMySql
{
    User:string;
    Pwd:string;
    DbName:string;
}

class Database<T>
{
     constructor(database:T){  for(var
         property in database){
             console.log(`${property} : ${database[property]}`);
         }
     }
}

console.log("-----Connecting with Oracle        ");
let ora = new Database<IOracle>({UserId:"scott", Password:"tiger", Database:"ProductsDb"});
console.log("-----Connecting with MongoDb        ");
let mongoClient = new Database<IMongoDb>({Url:"mongodb://127.0.0.1:27017"});
```

Ex:

```
interface IOracle
{
    UserId:string;
    Password:string;
    Database:string;
}
interface IMongoDb
{
    Url:string;
}
interface IMySql
{
    User:string;
    Pwd:string;
    DbName:string;
}

class Database
{
    public Connect<T>(connectionString:T) {
        for(var property in connectionString){
            console.log(`${property} : ${connectionString[property]}`);
        }
    }
}

let ora = new Database();
```

```
ora.Connect<IOracle>({UserId:"scott", Password:"tiger", Database:"EmpDb"});
```

Ex:

```
interface IOracle
```

```
{
    UserId:string;
    Password:string;
    Database:string;
}
interface IMongoDb
{
    Url:string;
}
interface IMySql
{
    User:string;
    Pwd:string;
    DbName:string;
}

class Database<T>
{
   public ConnectionString:T|null = null;
}

let ora = new Database<IOracle>();
ora.ConnectionString = {UserId: "scott", Password: "tiger", Database: "empdb"};

console.log(ora.ConnectionString);
```

Enum
[Enumeration]

- Enum is a set of constants.
- If you configure Enum with numeric constants, then they have auto implementation for values based on existing values.

Syntax:

```
        enum Name
        {
          Key = value,
        }
```

Ex:

```
enum ErrorCodes
{
     Redirect = 302,
     NotFound = 404,
     ServerTimeOut,
}
console.log(`${ErrorCodes.NotFound} : ${ErrorCodes[404]}`);
```

- Enum can handle number, string or expression.
- Expression must return a number or string.
- It will not allow boolean expression.

Ex:

```
enum ErrorCodes
{
   A = 10,
   B = 20,
   C = A + B
}
console.log(`Addition=${ErrorCodes.C}`);
```

FAQ: What is reverse-mapping?
Ans : Accessing a key with reference of value.

```
      EnumName[value]        = > Key
      EnumName.Key         => Value
```

TypeScript Module, Namespace

- Generics & Enums

TypeScript Module System

- Module is a set of values, functions, classes, templates, contracts etc.
- Module requires a module system like
  a) Common JS
  b) Require JS
  c) UMD
  d) AMD
  e) ESNext

Ex:
1. Add following folder structure

```
        Library
           |_contracts
           |_templates
           |_components
```

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;
    abstract Total():number;
    abstract Print():void;
}
```

4. components/ProductComponent.ts

```
import { ProductTemplate } from "../templates/ProductTemplate";

export class ProductComponent extends ProductTemplate
```

```
{
    Name = "Samsung TV";
    Price = 45000.44;
    Qty = 2;
    Total() {
        return this.Qty * this.Price;
    }
    Print(){
        console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Tot
al()}`);
    }
}
```

4. app/index.ts

```
import { ProductComponent } from
"../library/components/ProductComponent"; let tv = new ProductComponent();
tv.Print();
```

5. compile index.ts

```
    > tsc index.ts
    > node index.js
```

Namespace

- A namespace is a collection of related type of classes, template or components.
- It allows to configure a multi level hierarchy for members in module system, so that it reduces ambiquity issues.

Syntax:

```
        namespace Name
        {
            // sub_namespace
            // classes
            // contracts
            // templates
        }
```

- You can import and implement the library of namespace using "///" directive

```
        /// <reference path="../folder/file.ts" />
```

- You can compile the namespace library using "--outFile" flag

```
        > tsc --outFile  index.js  index.ts
```

Ex:

1. Add a folder "Project"

```
            Project
              |_Team-A
              |_Team-B
```

2. Team-A/TeamAComponent.ts

```
namespace Project
{
    export namespace TeamA
    {
        export class Demo
        {
            public Print():void {
                console.log(`Team A Print Method`);
            }
        }
    }
}
```

2. Team-B/TeamBComponent.ts

```
namespace Project
{
    export namespace TeamB
    {
        export class Demo
        {
            public Print():void {
                console.log(`Team B - Print Method`);
            }
        }
    }
}
```

3. App/index.ts

```
/// <reference path="../Team-A/TeamAComponent.ts" />
/// <reference path="../Team-B/TeamBComponent.ts" />

let obj1 = new Project.TeamA.Demo();
let obj2 = new Project.TeamB.Demo();

obj1.Print();
obj2.Print();
```

4. Compile into one outfile

```
        > tsc --outFile index.js index.ts
        > node index.js
```

Create React TypeScript Application

```
> npx create-react-app     appname  --template  typescript
```

Note: TypeScript React application is added with "tsconfig", that specifies the rules for typescript in project.

```
>         npm i bootstrap bootstrap-icons axios formik yup
```

--save Naming in TS project:

```
.ts          => contracts, templates
.tsx          =>
```

components State in

TypeScript:

```
    const [categories, setCategories] = useState<string[]>([ ]);
```

Ex:
/contracts/FakestoreProduct.js

```
export interface FakestoreProduct
{
    id:number;
    title:string;
    price:number;
    description:string;
    image:string;
    category:string;
    rating:{rate:number, count:number}
}
```

Shopping.tsx

```
import axios from "axios";
import { useEffect, useState } from "react"
import { FakestoreProduct } from "../contracts/FakestoreProduct";


export function Shopping() {

    const [categories, setCategories] = useState<string[]>([]);
    const [products, setProducts] = useState<FakestoreProduct[]>([]);

    function LoadCategories():void {
        axios.get('http://fakestoreapi.com/products/categories&#39;)
        .then((response)=> {
            setCategories(response.data);
        })
    }
    function LoadProducts():void {
```

```
    axios.get('http://fakestoreapi.com/products&#3
    9;)
    .then((response)=> {
        setProducts(response.data);
    })
}
```

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

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

            <div>
                {
                    products.map(product=>
                        <img className="m-2" src={product.image} width="100" height="100"
key={product.id} />
                        )
                }
            </div>
        </div>
    )
}
```

Typescript To-do App

1. Backend Database in
   MongoDB database :
   todo
   collections : users, appointments

   user = { UserId, UserName, Password, Mobile }
   appointments = { Id, Title, Date, Description }

2. API with Node & Express

```
> npm i mongodb express cors

  --save api.js
```

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

var connectionString =
"mongodb://127.0.0.1:27017"; var app = express();
app.use(cors());
app.use(express.urlencoded({extended:true}));
app.use(express.json());

app.get("/", (req, res)=>{
  res.send("<h1>To DO - API</h1>");
});

app.get("/appointments", (req, res)=> {
   mongoClient.connect(connectionString).then(clientObject=>{
       var database = clientObject.db("todo");
       database.collection("appointments").find({}).toArray().then(documents=>{
          res.send(documents);
          res.end();
       })
   })
});

app.get("/appointments/:id", (req, res)=>{
    var id = parseInt(req.params.id);
    mongoClient.connect(connectionString).then(clientObject=>{
        var database = clientObject.db("todo");
        database.collection("appointments").find({Id:id}).toArray().then(documents=>{
           res.send(documents);
           res.end();
        })
    })
});

app.post("/addtask",(req, res)=>{
    var task = {
```

```
Id: parseInt(req.body.Id),
Title: req.body.Title,
Date: new Date(req.body.Date),
          Description: req.body.Description
      };
      mongoClient.connect(connectionString).then(clientObject=>
          { var database = clientObject.db("todo");
          database.collection("appointments").insertOne(task).then(()=>{
             console.log(`Task Added Successfully..`);
             res.end();
          })
      })
});

app.put("/edittask/:id",(req, res)=>{
   var id = parseInt(req.params.id);
   mongoClient.connect(connectionString).then(clientObject=>
        { var database = clientObject.db("todo");
        database.collection("appointments").updateOne({Id:id},{$set:{Id:parseInt(req.body.Id), Title:
req.body.Title, Date: new Date(req.body.Date), Description: req.body.Description}}).then(()=>{
            console.log("Task Updated Successfully..");
            res.end();
        })
   })
});

app.delete("/deletetask/:id", (req, res)=>{
    var id = parseInt(req.params.id);
    mongoClient.connect(connectionString).then(clientObject=>{
        var database = clientObject.db("todo");
        database.collection("appointments").deleteOne({Id:id}).then(()=>{
            console.log("Task Deleted Successfully..");
            res.end();
        })
    })
});

app.get("/users", (req, res)=> {
    mongoClient.connect(connectionString).then(clientObject=>{
         var database = clientObject.db("todo");
         database.collection("users").find({}).toArray().then(documents=>{
             res.send(documents);
             res.end();
         })
    })
});

app.post("/adduser",(req, res)=>{
    var user = {
        UserId: req.body.UserId,
        UserName: req.body.UserName,
        Password: req.body.Password,
        Mobile: req.body.Mobile
    };
    mongoClient.connect(connectionString).then(clientObject=>{
        var database = clientObject.db("todo");
        database.collection("users").insertOne(user).then(()=>{
           console.log(`User Registered Successfully..`);
```

```
            res.end();
        })
    })
});
```

```
app.listen(4000);
console.log(`Server Started : http://127.0.0.1:4000`);
```

3. Create React TypeScript application

```
> npx create-react-app  react-typescript-todo  --template typescript
```

4. Install following libraries

```
>           npm i mongodb express cors bootstrap bootstrap-icons axios

            formik --save https://www.freepik.com/

> npm i react-router-dom --save
```

TypeScript To-Do – Updated

https://drive.google.com/file/d/1_5Cmeaq8VG8zOzElmSUJ88Ou8TJo8s94/view?usp=classroom_web&authuser=0

React Hooks

React Hooks

- React 17+ versions introduced Hooks for "Function Components".
- Hook is a function that acts as a service for component.
- Service is a pre-defined business logic, which you can inject and implement from any application.
- Service comprises of 2 components
  a) Provider
  b) Injector
- Provider locates the value or function in memory.
- Injector is responsible for injecting the value or function into component.
- React hooks can be used only in function components, not in class components.
- React provides several pre-defined hooks and also allows to create custom hooks.

Custom Hook:
- Hook must be a function.
- Hook function must return a type.
- Hook must be used in a function component only.
- It can be parameterized or parameterless.
- It must be used always at higher hierarchy, don't use a hook inside another function.

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

2. Add a new file
   "captcha.js"

```
export function useCaptcha()
{
    var a = Math.random() * 10;
    var b = Math.random() * 10;
    var c = Math.random() * 10;
    var d = Math.random() * 10;
    var e = Math.random() * 10;
    var f = Math.random() * 10;
    var code = `${Math.round(a)} ${Math.round(b)} ${Math.round(c)} ${Math.round(d)}
${Math.round(e)}   ${Math.round(f)}`;
    return code;
}
```

3. login.jsx

```
import { useCaptcha } from '../../hooks/captcha';
export function UserLogin()
{
    const captcha = useCaptcha();
    return(
<div className="container-fluid">
    <h3>User Login</h3>
    <dl>
        <dt>UserId</dt>
```

```
                <dd><input type="text"/></dd>
                <dt>Password</dt>
                <dd><input type="password" /></dd>
                <dt>Verify Code</dt>
                <dd>{captcha}</dd>
            </dl>
            <button>Login</button>
        </div>
    )
}
```

Ex: Custom API hook

hooks/getapi.js

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

export function useApi(url)
{
    const [data, setdata] = useState()
    useEffect(()=>{
        fetch(url)
        .then(res=>res.json())
        .then(responseData=> {
            setdata(responseData);
        })
    },[url])
    return data;
}
```

login.jsx

```
import { useCaptcha } from '../../hooks/captcha';
import { useApi } from '../../hooks/getapi';

export function UserLogin()
{
    const captcha = useCaptcha();

    const categories = useApi('http://fakestoreapi.com/products&#39;);

    return(
        <div className="container-fluid">
            <ol>
                {
                    categories.map(category=><li key={category.id}> {category.title} </li>)
                }
            </ol>
            <h3>User Login</h3>
```

```
<dl>
    <dt>UserId</dt>
    <dd><input type="text"/></dd>
    <dt>Password</dt>
```

```
                <dd><input type="password" /></dd>
                <dt>Verify Code</dt>
                <dd>{captcha}</dd>
            </dl>
            <button>Login</button>
        </div>
    )
}
```

- React built-in Hooks

```
        useState()
        useEffect()
        useContext()
        useRef()
        useReducer() etc..
```

FAQ: How to configure unmount & mount using useEffect?
Ans:

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

Ex:

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

export function Register(){
    useEffect(()=>{
        console.log('Register Mounted..');
        return()=>{
            console.log(`Register Unmounted`);
        }
    })
    return(
        <div>
            <h4>Register</h4>
        </div>
    )
}

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

```
return(
    <div>
        <h4>Home</h4>
    </div>
    )
}

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

    function homeClick(){
        setComponent(<Home />);
    }
    function registerClick(){
        setComponent(<Register />);
    }

    return(
        <div className="container-fluid">
            <button onClick={homeClick} >Home</button> <button
onClick={registerClick}>Register</button>
            <div className="mt-4">
                {component}
            </div>
        </div>
    )
}
```

UseRef

Ex:

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

export function TimeoutDemo()
{
    const [message, setMessage] = useState('');
    let messageTrigger = useRef(null);

    function Msg1(){
        setMessage('Hello !');
    }
    function Msg2(){
        setMessage('How are you ?');
    }
    function Msg3(){
        setMessage('Welcome to React');
    }

    function StartClick(){
        setTimeout(Msg1, 3000);
        messageTrigger.current = setTimeout(Msg2, 6000);
        setTimeout(Msg3, 10000);
```

```
}

function CancelClick(){
    clearTimeout(messageTrigger.current);
    }

    return(
        <div className="container-fluid">
            <button onClick={StartClick}>Start</button>
            <button onClick={CancelClick}>Cancel Msg2</button>
            <h2 className="mt-4 text-center">{message}</h2>
        </div>
    )
}
```

React Hooks - continued

useRef()

- It is used to create a reference for function or values, which are used internally by the application but not rendered.

Syntax:

```
let ref = useRef(null);
```

- You can access the reference value by using the reference property

  "current". ref.current; => read

  ref.current = value; => write

Ex:

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

export function TimeoutDemo()
{
    const [message, setMessage] = useState('');
    let messageTrigger = useRef(null);

    function Msg1(){
        setMessage('Hello !');
    }
    function Msg2(){
        setMessage('How are you ?');
    }
    function Msg3(){
        setMessage('Welcome to React');
    }

    function StartClick(){
        setTimeout(Msg1, 3000);
        messageTrigger.current = setTimeout(Msg2, 6000);
        setTimeout(Msg3, 10000);
    }

    function CancelClick(){
        clearTimeout(messageTrigger.current);
    }

    return(
        <div className="container-fluid">
            <button onClick={StartClick}>Start</button>
            <button onClick={CancelClick}>Cancel Msg2</button>
            <h2 className="mt-4 text-center">{message}</h2>
        </div>
    )
}
```

Ex: useRef hook

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

export function TimeoutDemo()
{

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

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

    function LoadProductAuto(){
        ProductId.current = ProductId.current + 1;
        axios.get(`http://fakestoreapi.com/products/${ProductId.current}`)
        .then(response=>{
            setProduct(response.data);
        })
    }

    function NextClick(){
        ProductId.current = ProductId.current + 1;
        LoadProduct(ProductId.current);
    }
    function PreviousClick(){
        ProductId.current = ProductId.current - 1;
        LoadProduct(ProductId.current);
    }

    function PlayClick(){
        timeInterval.current = setInterval(LoadProductAuto, 5000);
        setStatus('Slide Show - Started');
    }

    function PauseClick(){
        clearInterval(timeInterval.current);
        setStatus('Slide Show - Paused');
    }


    useEffect(()=>{
```

```
    LoadProduct(1);
},[]);

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

```
            <div className="card p-2 mt-4 w-50">
                <div className="card-header text-center">
                    <div>{product.title}</div>
                    <div> <b>{status}</b> </div>
                </div>
                <div className="card-body">
                    <div className="row">
                        <div className="col-1">
                            <button onClick={PreviousClick} className="bi bi-chevron-left
btn"></button>
                                                        </div>
                                                        <div className="col-10">
                                                        <img width="100%" src={product.image}
                                                        height="300" />
                                                        </div>
                                                        <div className="col-1">
                                                        <button onClick={NextClick} className="bi
                                                        bi-chevron-right btn"></button>
                                                        </div>
                    </div>
                </div>
                <div className="card-footer text-center">
                    <button onClick={PlayClick} className="btn btn-danger me-2 bi bi-
play"></button>
                    <button onClick={PauseClick} className="btn btn-warning bi bi-
pause"></button>
                </div>
            </div>
        </div>
    )
}
```

useContext()

- Context refers to memory.
- It is the memory allocated for parent component and made availalbe to other child components that run within the context of parent.
- If memory configure with useState() then it is not accessible to child component.
- useContext() is used to configure a context memory that can be share to its child components.
- It is used as a service that comprises
    a) Provider
    b) Injector
- Context memory is created by using "createContext()".
- Context memory uses a context provider that locates value in memory.
- useContext() is a injector that injects the context values into a component.

Syntax:
1. Create a Context

```
        let UserDetails = createContext(null);
```

2. Configure the context provider with value in parent component

```
<UserDetails.Provider value={ anyValue }>
```

```
            // scope for child components

        </UserDetails.Provider>
```

3. You can access the context memory in child components by using the injector "useContext()"

```
        let data = useContext(UserDetails);

        { data }
```

Ex:
context-demo.jsx

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

let UserDetailsContext = createContext(null);
export function FirstLevelComponent(){
    const user = useContext(UserDetailsContext);
    return(
        <div className="bg-warning p-4 text-dark">
            <h4>First Level Component - {user} </h4>
            <SecondLevelComponent />
        </div>
    )
}

export function SecondLevelComponent(){
    let user = useContext(UserDetailsContext);
    return(
        <div className="bg-danger p-4 text-white">
            <h4>Second Level Component - {user}</h4>
        </div>
    )
}


export function ContextDemo(){
    const [userName, setUserName] = useState('John');
    function UserChange(e){
        setUserName(e.target.value);
    }
    return(
        <div className="container-fluid p-4 bg-dark text-white">
            <UserDetailsContext.Provider value={userName} >
               <div>
                    User Name : <input type="text" onChange={UserChange} />
               </div>
```

```
<h2>Context Parent - {userName} </h2>
<FirstLevelComponent />
```

```
            </UserDetailsContext.Provider>
        </div>
    )
}
```

useReducer()

- It is a predictable state container.
- It can store data for application, that is accessible across all component.
- React 18x versions implicitly provide useReducer(), how ever if you want to use the predictable state container at larger scale you can use a JavaScript library called "Redux".

use reducer and redux

useReducer

- It is used to configure a centralized and predictable state for component.
- Centralized allows to store data at application level so that it is accessible to all components.
- Predictable allows the developer to track the changes in values and know exactly what is the data stored and rendered into UI.
- It defines application state, which is maintained from application start to end.
- Reducer comprises of following components

a) store
b) state
c) reducer
d) actions
e) UI

- Store is the location where data is kept.
- State is the medium which can access data from store and render into UI.
- UI is the location where data is displayed.
- UI is the location where changes in data are detected.
- Reducer identifies the action performed and updates data into store.

Steps:

1. Configure initial state, which is the store. It specifies the data to

   store. let initialState = { count : 0 } => store contains count=0

2. Create a reducer, which specifies the actions and data to update.

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

3. Any component can use the reducer and state configured by reducer [store] with a hook called "useReducer()"

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


function JoinClick() {
  dispatch({type:"john"})
```

```
 }

function ExitClick(){
    displatch({type:"exit"})
}
```

```
    { state.count } Viewers
```

Ex:
Reducer-Demo.jsx

```
import { useReducer } from "react";
let initialState = {count: 0};          //
store function reducer(state, action){
    switch(action.type){
        case "join":
        return {count: state.count + 1};
        case "exit":
        return {count: state.count - 1};
    }
}

export function ReducerDemo(){
    const [state, dispatch] = useReducer(reducer, initialState);
    function JoinClick(){
        dispatch({type: "join"});
    }
    function ExitClick(){
        dispatch({type: "exit"});
    }

    return(
        <div className="container-fluid">
            <div className="mt-4">
                <button onClick={JoinClick} className="btn btn-success">Join</button>
                <button onClick={ExitClick} className="btn btn-danger ms-2">Exit</button>
            </div>
            <h2>Live Broadcast</h2>
            <iframe width="400" height="200"
src="https://www.youtube.com/embed/kiAK3FcwUYA"&gt;
            </iframe>
            <div>
                Live Viewers : {state.count}
            </div>
        </div>
    )
}
```

Redux

- It is a JavaScript library for mantaining state.
- It is used by Javascript application to configure predictable state.
- It is centralized.
- It is debuggable.
- It is predictable.
- It is cross platform.

https://redux.js.org/

Using Redux in your React App:

1. Install Redux Library [base for redux]

```
>npm i react-redux --save
```

2. Install Redux Tool Kit [provides ad-ons for redux]

```
> npm i @reduxjs/toolkit  --save
```

3. At higher level of application development you can use
   > Slicers
   > Reducers
   > Store

4. Create a slicer

   * It is required for configuring Initial State.
   * Initial state defines what value you want to store in local state.
   * It also defines the reducer
   * reducer identifies the changes in state and updates the state valu es
   * reducer can access state and action
   * collect payload and stores in state.
   * Exports the actions and Initial State

React Redux Example

react-redux
@redux/toolkit

Ex: Implementing Redux in Fakestore App

1. Create a cart slicer
 - It contains initial state
 - It contains reducer actions
 - It contains reference name

```
  const cartSlice = createSlice({name:' ', InitialState, reducers: actions(){ }})
```

2. Configure store by using redux toolkit "createStore()"

```
   export default configureStore({

      reducer : {
         store: CartSlice
       }
   })
```

3. Configure provider at application level, It is responsible for locating the store and injecting into components.

```
        index.js

     import Provider form "react-redux"; import
     store from "./components/                /store;


    <Provider store={store} >                  => cartItems[], cartCount
          // your startup component
     </Provider>
```

4. Dispatch actions from any component

- react redux provides "useDispatch()"
- It is used to dispatch the action configured in reducer
- It will deliver the payload[data] to store, which is stored in initialState

```
    const dispatch = useDispatch();

    const handleAddToCart = (product) => {
        dispatch(addToCart(product));
    }
```

=> Download Redux DevTools to test redux-actions

5. Access and use the store from any component in application scope.

   ```
   import store from "../store";
   ```

```
{store.getState().store.cartCount}
```

Ex:
1. Add a new file into "fakestore" component folder

    "cart-slicer.js"

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
    cartItems: [],
    cartCount : 0
}

const cartSlice = createSlice({
    name: 'cart',
    initialState,
    reducers: {
        addToCart(state, action){
             state.cartItems.push(action.payload);
             state.cartCount = state.cartItems.length;
        },
    }
});
export const { addToCart } = cartSlice.actions;
export default cartSlice.reducer;
```

2. Add another file for

    store "store.js"

```
import cartSlicer from "./cart-slicer";
import { configureStore } from "@reduxjs/toolkit";

export default configureStore({
    reducer: {
        store: cartSlicer
    }
})
```

3. Go to index.js configure provider import { Provider } from 'react-redux';

```
import store from './components/fakestore/store';

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

```
root.render(
  <React.StrictMode>
    <Provider store={store}>
```

```
        <Fakestore />
      </Provider>
    </React.StrictMode>
);
```

4. Goto fakestore.jsx

```
import { useEffect, useState } from "react"
import axios from "axios";
import { useDispatch } from "react-redux";
import { addToCart } from "./cart-slicer";
import store from "./store";


export function Fakestore(){

    const [categories, setCategories] = useState([]);
    const [products, setProducts] = useState([{id:0, title:'', description:'', price:0, image:'',
category:'', rating:{rate:0, count:0}}]);
    const [toggleTable, setToggleTable] = useState({display:'none'});
    const dispatch = useDispatch();

    function LoadCategories(){
        axios.get("http://fakestoreapi.com/products/categories&quot;)
        .then(response=> {
            response.data.unshift("all");
            setCategories(response.data);
        })
    }

    function LoadProducts(url){
        axios.get(url)
        .then(response=> {
            setProducts(response.data);
        })
    }

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

    },[]);

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

```
}

function handleAddClick(product){
        alert(`${product.title}\n Added to Cart`);
        dispatch(addToCart(product));
    }

    function handleCartClick(){
        setToggleTable({display: (toggleTable.display==="none")?"block":"none"}) ;
    }

    return(
        <div className="container-fluid">
            <header className="d-flex bg-dark text-white justify-content-between p-3">
                <div className="h3">Fakestore.</div>
                <div className="fs-4">
                    <span className="me-4"><a>Home</a></span>
                    <span className="me-4"><a>Jewelery</a></span>
                    <span className="me-4"><a>Electronics</a></span>
                </div>
                <div className="fs-4">
                    <button onClick={handleCartClick} className="bi btn btn-light bi-cart4 position-
relative">
                     <span className="badge position-absolute rounded rounded-circle bg-danger
text-white">{store.getState().store.cartCount}</span>
                    </button>

                </div>
            </header>
            <section className="mt-3 row">
                <nav className="col-2">
                  <div className="mt-4">
                    <label className="fw-bold form-label">Select Category</label>
                    <div>
                        <select onChange={handleCategoryChange} className="form-select">
                            {
                                                                    categories.map(category=>
                                                                        <option value={category} key={category}>
</option>                                                               {category.toUpperCase()}

                                                                        )
                                                                }
                        </select>
                    </div>
                  </div>
                  <div>
                    <table style={toggleTable} className="table table-hover caption-top">
                        <caption>Your Cart Items</caption>
                        <thead>
                            <tr>
                                <th>Title</th>
                                <th>Price</th>
        <th>Preview</th>
    </tr>
</thead>
<tbody>
    {
```

```
                                    cartItems.map(item=>
                                        <tr key={item.id}>
                                            <td>{item.title}</td>
                                            <td>{item.price}</td>
                                            <td><img width="50" src={item.image} height="50"/></td>
                                        </tr>
                                        )
                                }
                            </tbody>
                        </table>
                      </div>
                    </nav>
                    <main className="col-10 overflow-auto" style={{height:'500px' , display:'flex',
flexWrap:'wrap'}}>
                        {
                            products.map(product=>
                                <div className="card p-2 m-2" key={product.id} style={{width:'200px'}}>
                                    <img src={product.image} className="card-img-top" height="120" />
                                    <div className="card-header" style={{height:'120px'}}>
                                        <p>
                                            {product.title}
                                        </p>
                                    </div>
                                    <div className="card-body">
                                        <dl>
                                            <dt>Price</dt>
                                            <dd>{product.price}</dd>
                                            <dt>Rating</dt>
                                            <dd>
                                                {product.rating.rate} <span className="bi bi-star-fill text-
success"></span>                                                </dd>
                                                                                                </dl>
                                                                                            </div>
                                                                                            <div className="card-footer">
                                                                                                <button onClick={()=> handleAddClick(product)}
                                                                                                className="btn btn-
dark w-100 bi bi-cart3"> Add to Cart</button>
                                    </div>
                                </div>
                                )
                        }
                    </main>
                </section>
            </div>
    )
}
```

Create a new React Application using Webpack Bundler

----

a) Minification
b) Bundling

- Minification is the process of compressing CSS and JS for production.
- Bundling is the process of reducing the number of requests and configuring the resources to be used by application.
- There are various bundling tools
    a) Webpack
    b) Parcel
    c) Vite etc..

React Testing & Webpack

Creating a React App using Webpack Bundler:

1. Create a new folder for project on

   your PC F:\react-app

2. Open in VS code

```
> npm init -y                [package.json]
```

3. Create basic folder

   structure app

```
|_public
| |_index.html
|_src
  |_index.js
  |_app.js
```

4. Install the following libraries for project

```
> npm install webpack webpack-cli --save-dev
> npm install html-webpack-plugin --save-dev
> npm install --save-dev webpack-dev-server
> npm i @babel/core @babel/preset-env babel-loader --save-dev
> npm i react react-dom --save
> npm i @babel/preset-react --save-dev
```

webpack, webpack-cli	=> core webpack library
html-webpack-plugin	=> to access HTML files and bundle
webpack-dev-server	=> to configure a server to run your app
@babel..	=> for compiling JSX & JS file
react, react-dom	=> react core, virtual dom

5. Go to index.js in "src" import React from "react";

```
import { createRoot } from "react-dom/client";
import { App } from "./app";

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

6. Create a webpack configuration file   => refer to => https://webpack.js.org/

webpack.config.js

```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  entry: "./src/index.js",
  output: {
    filename: "main.js",
    path: path.resolve(_dirname, "build"),
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: path.join(_dirname, "public", "index.html"),
    }),
  ],
  devServer: {
    static: {
      directory: path.join(_dirname, "build"),
    },
    port: 8080,
  },
  module: {
      rules: [
        {
          test: /\.(js)$/,
          exclude: /node_modules/,
          use: ["babel-loader"],
        },
      ],
  },
  resolve: {
    extensions: ["*", ".js"],
  },
};
```

7. Configure compiler for project by adding a new file

.babelrc =>https://babeljs.io/

```
{
    "presets": [
      "@babel/preset-env",
      ["@babel/preset-react", {
      "runtime": "automatic"
     }]
    ]
}
```

8. Goto package.json and configure the batch operations using

```
script srcipts: {

  .....
```

```
        "build": "webpack --mode production",
           "start": "webpack serve --mode development"
        ....
    }
```

9. Start react app

```
> npm run start
```

React Unit Testing

- Unit Testing is the process of testing every component function or class.
- It is required to check if "As-Is" matching with "To-Be".
- AS-IS refers to what you designed.
- TO-BE refers to what client wants.

```
AS-IS == TO-BE        => Test Pass
AS-IS !== TO-BE        => Test Fail
```

- There are various testing frameworks used for JS based applications
    a) Jasmine - Karma
    b) JEST
    c) Protractor etc..
- React Template is configured with JEST framework.
- Testing every component have 3 phases

    a) Arrange
    b) Act
    c) Assert
- JEST Basic functions & properties
  include test()
  render()
  screen
  data-testi
  d
  toBeInDocument()
  toHaveAttribute()
  getByTestId()
  getByName() etc..
- Test document is defined by
  using "name.test.js"
  "name.spec.js"

t
e
s
t
(

Ex:

)

=>toconfigurerenderer()

=>toar

rangeexpect()

=>toactandassert

1. test-demo.jsx

export function TestDemo(){

```
    return(
        <div>
            <h1 data-testid="title">Testing Demo</h1>
            <a href="login.html">Login</a>
        </div>
    )
}
```

2. test-demo.test.js

```
import { render, screen } from "@testing-library/react";
import { TestDemo } from "./test-demo";

test('Title Test',()=>{
    render(<TestDemo />);



});
```

```
l
e
t
t
i
t
l
e
=
s
c
r
e
e
n
.
g
e
t
B
y
T
e
s
t
I
d
(
"
t
i
t
le");expect(title).toHaveTextContent("TestingDemo");
```

```
test('Login Link Test', ()=>{
    render(<TestDemo />);

    let login = screen.getByText(/Login/i);
    expect(login).toBeInTheDocument();
    expect(login).toHaveAttribute("href", "login.html");
})
```

3. Run test

```
>npm test
```

Build and Deployment

Build & Deploy

- React application is default in Development mode.
- You have to build for production & Deploy
- Building is the process of checking for syntax errors, identifying the issues in application and creating production ready folder.
- All the resources are copied into production folder.
- All components are maped to "index.html"
- Typically production folder is added into project by name "dist / build".
- The command for building react application

```
> react-scripts build
> npm run build
```

Note: React 18+ versions introduced default build mapped for production.

- Deployment is the process of publishing your application on to local static server or cloud servers. So that users can access and use your application.

Local Static Servers
- IIS
- WAMP
- Tomcat
- Node & Express

Cloud Servers
- Firebase
- Aws
- Azure
- Netlify
- Github pages etc..

Firebase Deployment

1. Login into your firebase

   account https://firebase.com

2. Go to console and create a new

   project Name : reactnetflix-app

3. After adding project on cloud

4. Build your react application

```
>npm run build
```

5. Install firebase tools on your PC

```
> npm install -g firebase-tools
```

6. Login into firebase from CLI

```
> firebase login
```

7. Start deployment by using the command

```
> firebase init

? Which Firebase features do you want to set up for this directory? Hosting

? Use existing project

 ? Select your project : reactnetflix-app

  ? What do you want to use as your public directory? build

  ? Configure as a single-page app (rewrite all urls to /index.html)? No

  ? File build/index.html already exists. Overwrite? No


> firebase deploy
```

https://reactnetflix-app.web.app

Video Library Project
[MERN]

React Native

- It is a UI library for Android and IOS apps.
- It provides various components, which are native for android & ios.

Video Library Project

Complete REST API

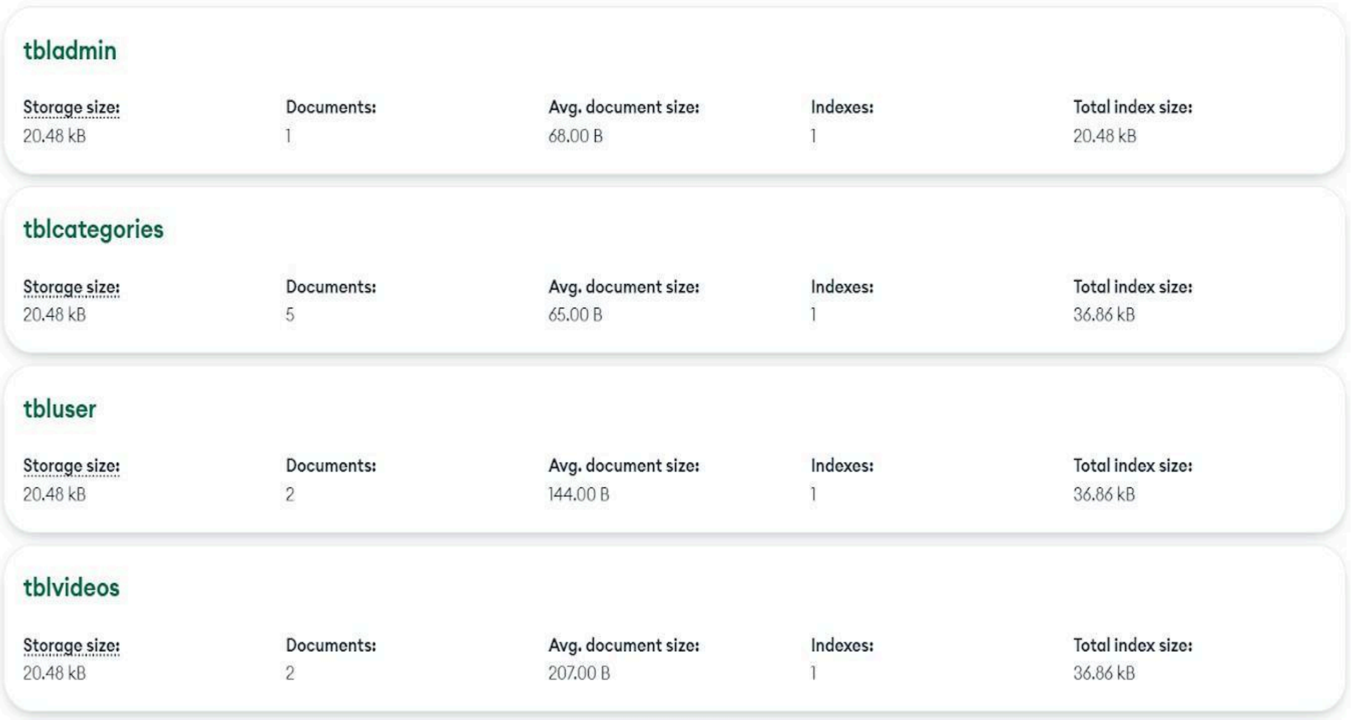
restapi.js

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

var conString = "mongodb://127.0.0.1:27017";

var app = express();
app.use(cors());
app.use(express.urlencoded({extended:true}));
app.use(express.json());

app.get("/admin", (req, res)=>{
   mongoClient.connect(conString).then((clientObj)=>
   {
      var database = clientObj.db("reactdb");
      database.collection("tbladmin").find({}).toArray().then((docs)=>{
         res.send(docs);
         res.end();
                                             });                                   });

});


app.get("/users", (req, res)=>{
   mongoClient.connect(conString).then((clientObj)=>
   {
      var database = clientObj.db("reactdb");
      database.collection("tbluser").find({}).toArray().then((docs)=>{
         res.send(docs);
         res.end();
                                             });                                   });

});


app.get("/categories", (req, res)=>{
   mongoClient.connect(conString).then((clientObj)=>
   {
      var database = clientObj.db("reactdb");
      database.collection("tblcategories").find({}).toArray().then((docs)=>{
         res.send(docs);
         res.end();
                                                                                     });
```

```
                                        });                                        });


app.get("/videos", (req, res)=>{
    mongoClient.connect(conString).then((clientObj)=>
    {
        var database = clientObj.db("reactdb");
        database.collection("tblvideos").find({}).toArray().then((docs)=>{
            res.send(docs);
            res.end();
        });
    });
});

app.get("/video/:id", (req, res)=>{

    var id = parseInt(req.params.id);

    mongoClient.connect(conString).then((clientObj)=>
        { var database = clientObj.db("reactdb");
        database.collection("tblvideos").find({VideoId:id}).toArray().then((docs)=> {
            res.send(docs);
            res.end();
                                                    });                                                    });

});

app.post("/adduser", (req, res)=>{
    var user = {
        UserId: req.body.UserId,
        UserName: req.body.UserName,
        Password: req.body.Password,
        Email: req.body.Email,
        Mobile: req.body.Mobile
    }
    mongoClient.connect(conString).then((clientObj)=>
        { var database = clientObj.db("reactdb");
        database.collection("tbluser").insertOne(user).then(()=>{
            console.log('User Added');
            res.redirect("/users");
            res.end();
                                                                            })
                                                                        });
});

app.post("/addcategory", (req,

    res)=>{ var category = {

        Category_Id: parseInt(req.body.Category_Id),
        CategoryName: req.body.CategoryName
    };

    mongoClient.connect(conString).then((clientObj)=>
        { var database = clientObj.db("reactdb");
        database.collection("tblcategories").insertOne(category).then(()=>{
            console.log('Category Added');
            res.redirect("/categories");
            res.end();
});                                                                             })
                                                                            });

app.post("/addvideo", (req, res)=>{
    var video = {
        VideoId: parseInt(req.body.VideoId),
        Title: req.body.Title,
        Url: req.body.Url,
        Comments: req.body.Comments,
        Likes : parseInt(req.body.Likes),
        Category_Id: parseInt(req.body.Category_Id)
    }

    mongoClient.connect(conString).then((clientObj)=>
        { var database = clientObj.db("reactdb");
        database.collection("tblvideos").insertOne(video).then(()=>{
            console.log('Video Added');
            res.redirect("/videos");
            res.end();
                                                                            })
                                                                        });
});


app.put("/editvideo/:id",(req, res)=>{
     var id = parseInt(req.params.id);
     mongoClient.connect(conString).then((clientObj)=>{
        var database = clientObj.db("reactdb");
        database.collection("tblvideos").updateOne({VideoId:id},{$set:
            { VideoId: parseInt(req.body.VideoId),
            Title: req.body.Title,
            Url: req.body.Url,
            Comments: req.body.Comments,
            Likes : parseInt(req.body.Likes),
            Category_Id: parseInt(req.body.Category_Id)
        }})
     }).then(()=>{
        console.log("Video Updated");
        res.end();
     })
});

app.delete("/deletevideo/:id", (req, res)=>{
    var id = parseInt(req.params.id);
    mongoClient.connect(conString).then((clientObj)=>
        { var database = clientObj.db("reactdb");
        database.collection("tblvideos").deleteOne({VideoId:id}).then(()=>{
            console.log("Video Deleted");
            res.end();
        })
    })
});
```

```
app.listen(2200);
console.log(`Server Started : http://127.0.0.1:2200`);
```

https://drive.google.com/file/d/1RQz7eP3sh5m-euYO699EicxbTuE7EwUG/view?usp=classroom_web&authuser=0



tbladmin

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	1	68.00 B	1	20.48 kB

tblcategories

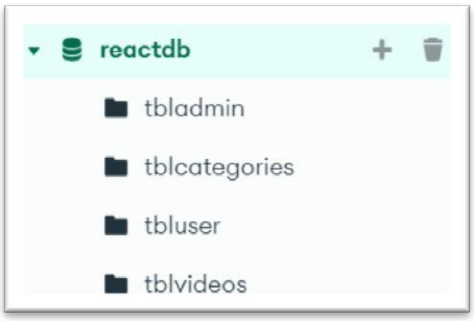
Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	5	65.00 B	1	36.86 kB

tbluser

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	2	144.00 B	1	36.86 kB

tblvideos

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	2	207.00 B	1	36.86 kB


reactdb
tbladmin
tblcategories
tbluser
tblvideos