**REACT**

**React** is a free and open-source JavaScript library for building front end client side user interfaces based on UI components. and it is maintained by **Meta (formerly Facebook)**.

Java Script is a interpreter based language.

Other UI libraries, VEI, AngularJS (framework).

It is widely used for creating fast, scalable, and interactive web applications.

React focuses on **component-based architecture**, making it easy to build reusable UI components.

In react any visual appearing things of UI are termed as components.

Parent of all components in react is App. All the remaining are child components like button, text fields, menu, grid, image, etc.

In react we can create components using java Script in two ways:

1. Functional components, ii) Class based components.

**It's used for handling the view layer for web and mobile apps.**

**JSX (JavaScript XML)**:

* JSX is a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript, enhancing readability.

**To create new react project**

**$create a dir and open with vscode**

**$npm create vite@latest**

**Do you want ot proceed? (Y/N): Y**

**Application name: react-application**

**Choose framework: react**

**Choose variant : JavaScript**

**Switch to the project : $cd react-application**

**$npm install**

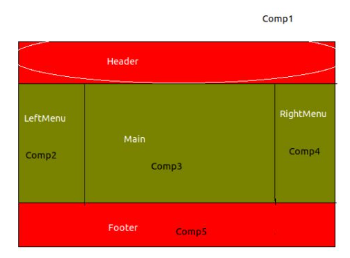
**$npm run dev**

**$**npm install react-router-dom@6

**$**npm install axios

**Components:**

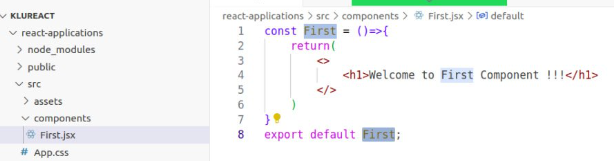
**Portion of the webpage.**



Inside “src” folder, create a new folder named : “components” -> First.jsx (F must be capital) and write following simple code.

At the end of this code: we must make this file as public to access by other components using

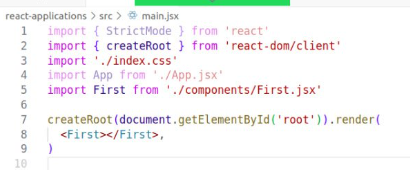
export default First;

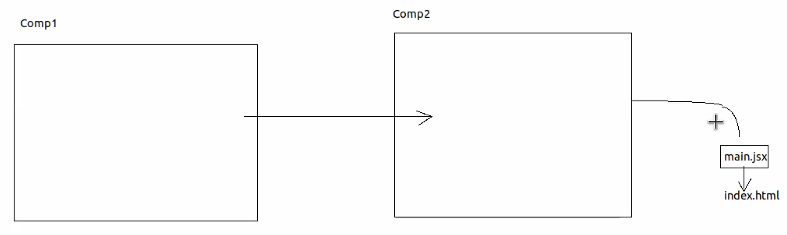


After this we must register this First.jsx file in the main.jsx file in src folder as follows:

Import this First file from ./components folder

Call this file within render() method by removing existing three lines.





Main.jsx will handover to index.html

In React, components need to return a single root element. (< > </> )

This requirement ensures that React can efficiently reconcile and update the DOM.

The <div> tag (or any other valid HTML element) is commonly used as a **container** to wrap multiple elements and satisfy this requirement

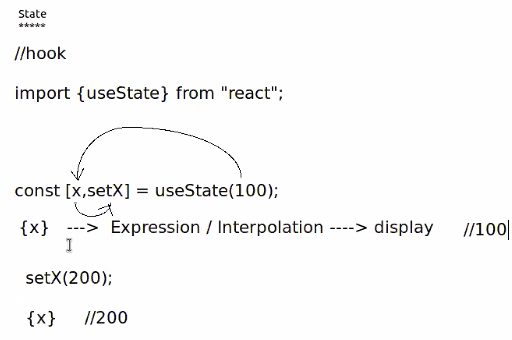
**STATE**

State is used to store the component data

State is immutable.

To define state, {useState} hook can be used.

{x} //Expression to print x value



Presentation Logic is present always in return(…..) statement.

**3 ways of receiving properties**

* **Props.Children**
* **Props**
* **destructuring**

**NextJS created on top of React.**

**ReactJS is depending on the third party libraries called react-router-dom@6 for Routes and Route.**

**Execution starts from index.js file. Delete existing content from it.**

**useEffect(()=>{},[]) hook will execute after presentation logic(return() statement).**

**useEffect hook used to make API calls.**

### **Types of Components in React**

#### ****1. Functional Components****

* **Definition**: JavaScript functions that return React elements (JSX).
  + **Advantages**:
* Simple and easy to understand.
* Lightweight since they do not have state or lifecycle methods (prior to hooks).
* With **React Hooks**, they can manage state and lifecycle..

#### ****2. Class Components****

* **Definition**: ES6 classes that extend React.Component and include a render method to return JSX.
* **Advantages**:
  + Can manage state and use lifecycle methods.
  + Ideal for complex logic (before hooks were introduced).

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

**Arrow Functions**

Syntax: const functionName = (parameters) => {

// Function body

};

**Note:** If the function body contains a single expression, you can omit the braces and the return keyword:

Example: const add = (a, b) => a + b;

**Advantages of Arrow Functions in React:**

 **Conciseness**:

* They reduce boilerplate code, especially for one-liner functions.

 **Inline Usage**:

* Ideal for small, one-time-use functions like event handlers or rendering logic.

Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

#### ****1. Inside Functional Components****

Arrow functions are often used to define event handlers and inline logic inside functional components.

Example:

**import React from "react";**

**const App = () => {**

**const handleClick = () => {**

**alert("Button clicked!");**

**};**

**return (**

**<div>**

**<h1>Hello, React!</h1>**

**<button onClick={handleClick}>Click Me</button>**

**</div>**

**);**

**};**

**export default App;**

**aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa**

**2. Inline Arrow Functions**

Arrow functions can be defined directly within JSX for short event handlers or rendering logic.

Example:

import React from "react";

const App = () => {

    return (

        <div>

            <h1>Inline Arrow Function Example</h1>

            <button onClick={() => alert("Inline arrow function clicked!")}>

                Click Me

            </button>

        </div>

    );

};

export default App;

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

#### ****3. Using Arrow Functions to Pass Parameters****

Arrow functions are useful when you need to pass arguments to event handlers.

Example:

import React from "react";

const App = () => {

    const handleClick = (name) => {

        alert(`Hello, ${name}!`);

    };

    return (

        <div>

            <h1>Passing Parameters Example</h1>

            <button onClick={() => handleClick("John")}>Click Me</button>

        </div>

    );

};

export default App;

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

### **Key Concepts in React Components**

#### ****1. Props (Properties)****

* Props are used to pass data from a parent component to a child component.
* Props are **read-only** and cannot be modified by the child.

**Example:**

//npm install prop-types

import PropTypes from "prop-types";

// Child Component: Greeting

const Greeting = (props) => {

  return (

    <div>

      <h1>Hello, {props.name}!</h1>

      <p>Welcome to {props.platform}.</p>

    </div>

  );

};

// Define PropTypes for Greeting Component

//Adding PropTypes helps prevent runtime errors and

// provides explicitly validated for their type and requirement //status.

Greeting.propTypes = {

  name: PropTypes.string.isRequired,

  platform: PropTypes.string.isRequired,

 };

// Parent Component: App

const PropertiesEx = () => {

  return (

    <div>

      <Greeting name="Seetha" platform="ReactJS" />

      <Greeting name="Rama" platform="ReactJS" />

    </div>

  );

};

export default PropertiesEx;

**Hook:**

* useState: Holding the data just like variable
* useEffect used to modify the component data

#### ****2. State****

* State is a mutable object used to store data that changes over time.
* In **class components**, state is managed using this.state.
* In **functional components**, state is managed using the useState hook.

**Example (Functional Component with useState):**

import { useState } from "react";

const StateEx = () => {

  // Initialize state using the useState hook

  const [count, setCount] = useState(0);

  const increment = () => setCount(count + 1);

  const decrement = () => setCount(count - 1);

  const reset = () => setCount(0);

  return (

    <div>

      <h1>Counter</h1>

      <p>Current Count: {count}</p>

      <button onClick={increment}>Increment</button>

      <button onClick={decrement}>Decrement</button>

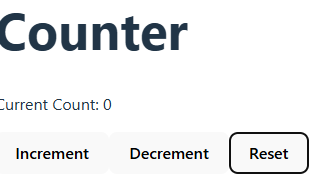
      <button onClick={reset}>Reset</button>

    </div>

  );

};

export default StateEx;



#### ****3. useEffect( )****

### Key Purposes of useEffect:

1. **Run code after render**: Perform actions like fetching data or updating the DOM after a component renders.
2. **Dependency-based updates**: Control when the effect runs based on state or props changes.
3. **Cleanup logic**: Execute cleanup tasks like unsubscribing from listeners or clearing timers when the component unmounts.

**Syntax:** **useEffect(**effectFunction, dependencyArray**);**

 **effectFunction**: contains the side-effect logic. It can optionally return a cleanup function.

 **dependencyArray** (optional): An array of values that the effect depends on. It controls when the effect should run.

**Example:**

import { useState, useEffect } from "react";

const UseEffect = () => {

  const [count, setCount] = useState(0);

  // useEffect to update the page title after every render

  useEffect(() => {

    document.title = `Count: ${count}`;

  }, [count]); // Dependency array ensures effect runs only when `count` changes

  return (

    <div>

      <h1>Counter: {count}</h1>

      <button onClick={() => setCount(count + 1)}>Increment</button>

      <button onClick={() => setCount(count - 1)}>Decrement</button>

    </div>

  );

};

export default UseEffect;

### **Best Practices**

1. Keep components small and focused.
2. Use functional components with hooks unless you need advanced features.
3. Avoid deeply nested components; break them into smaller ones.
4. Use meaningful names and organize components in folders.

**Navigation (Routes)**

**In react, we can navigate from one component to another component using a package called “react-router-dom”.**

**It will consists following components.**

1. **Browser-Router**
2. **Routes**
3. **Route**
4. **Link**

**In order to work in react-router-dom,**

**Install the react-router-dom 6.0 : $npm install react-router-dom@6**

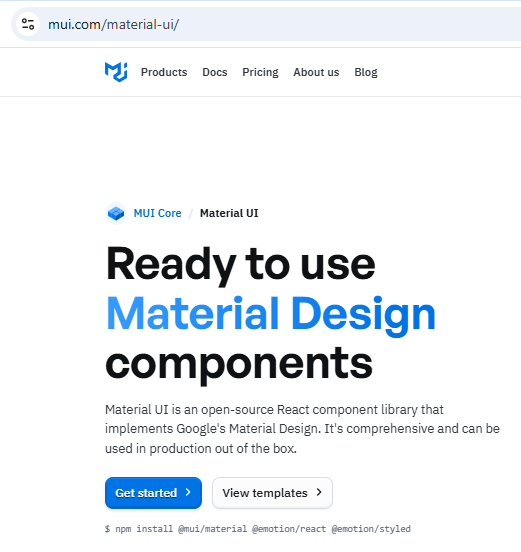
**Do not use switch component, which is React 5.0 version.**

**In App.js, first we need to import NavBar component, import Browser-Router, Routes, Route.**

**Each Route takes two attributes (path and Element)**

**Element: Mention the component name to which the route is to be navigate.**

**Path: Example: ( /Home )**

****

**Task: Display the products in a table format 5 products with 6 attributes in a table format**

**Install the packages of MUI using npm install command.**

**($**npm install @mui/material @emotion/react @emotion/styled )

**Inside MUI, display your picture in a card component**

**Display the sign-in or login component from the MUI. (Paper layout)**

**Fix one NavBar according to our project. Use one app bar according to our project from MUI (Responsive App Bar)**

**============@@@@@@@@@@@@@@@@2==============**

**Registration page using MUI**

**Generate data using JSON (Java Script Object Notation) format using https://www.mockaroo.com/**

**Using this data as a component.**

**Import the JSON data as components.**

**Render the data in a table.**

**Hint: 10 fields and 500 rows**

### **Virtual DOM vs Actual DOM**

The **Virtual DOM** and **Actual DOM** are core concepts in React that work together to manage and update the UI efficiently. Here's a detailed explanation:

### **1. What is the Actual DOM?**

* The **Document Object Model (DOM)** is a programming interface for HTML and XML documents.
* It represents the structure of a webpage as a tree of objects, where each node corresponds to an element on the page (e.g., <div>, <p>, <button>).

#### ****Key Characteristics of the Actual DOM****

* **Directly interacts with the browser**: It represents the real UI displayed on the screen.
* **Slow for updates**: Updating the Actual DOM is computationally expensive because:
  + It involves re-calculating styles, layouts, and repainting.
  + Even minor changes require the browser to reprocess the entire DOM tree.

**Key Characteristics of the Virtual DOM**

* **Exists in memory**: It’s not visible or directly tied to the browser.
* **Efficient updates**: React uses the Virtual DOM to calculate the minimum number of changes needed to update the Actual DOM.
* **Reconciliation**: React compares the Virtual DOM with its previous version (using a process called "diffing") to determine what has changed.

DOM => **Document Object Model (DOM) EXAMPLE**

****

**Promises**

JavaScript Promises make handling asynchronous operations like API calls, file loading, or time delays easier.

Think of a Promise as a placeholder for a value that will be available in the future.

It can be in one of three states

* **Pending:** The task is in the initial state.
* **Fulfilled:**The task was completed successfully, and the result is available.
* **Rejected:** The task failed, and an error is provided.