React Notebook

~ BY SAGAR BISWAS

Part 12: Class Components in React

🎯 What is a Class Component?

A **class component** is a React component defined using **JavaScript classes** instead of functions.

Before the introduction of **Hooks**, class components were the primary way to handle **state** and **lifecycle methods** in React.

Key Difference: Function vs Class Component

Feature	Function Component	Class Component
Syntax	Simple JS function	ES6 class syntax extending React.Component
State Management	Via useState hook	With this.state
Lifecycle Methods	Not available without hooks	Built-in like componentDidMount
Render Method	❌ Not required	✅ Required (render())
Simplicity	✅ Simpler	❌ More verbose
Preferred in 2024+	✅ Yes (with Hooks)	⚠️ Only if you need advanced patterns

★ Step-by-Step Example

📂 **File: src/components/Card1.js (Functional Component with props)**

```
// A simple functional component
function Card1(props) {
  return (
    <div>
      <h3>{props.name}</h3>
      <p>{props.desc}</p>
      <p>Card1 Component</p>
    </div>
  );
}

export default Card1;
```

📂 **File: src/components/Card2.js (Class Component with props)**

```
import React, { Component } from "react";

// A class-based component
class Card2 extends Component {
  // render() is a required method in class components. It returns the JSX to be displayed
  render() {
    return (
      <div>
        <h3>{this.props.name}</h3> {/* Access props via this.props */}
        <p>{this.props.desc}</p>
        <p>Card2 Component</p>
      </div>
    );
  }
}

export default Card2;
```

📂 **File: src/App.js**

```
import React from "react";
import Card1 from "./components/Card1"; // Functional component
import Card2 from "./components/Card2"; // Class component

function App() {
  return (
    <div>
      <h1>Inside the App Component</h1>

      {/* Passing props to both components */}
      <Card1 name="Card 1" desc="This is a functional component." />
      <Card2 name="Card 2" desc="This is a class-based component." />
    </div>
  );
}

export default App;
```

Summary Notes

- Functional components are now **the standard** in React with the help of **Hooks** (useState, useEffect, etc.).
- Class components are still useful for:
  - Legacy projects
  - Understanding how React evolved
  - Learning state and lifecycle the traditional way

Part 13: React Component Under the Hood

🎯 **Purpose**

To understand how **JSX** works behind the scenes and how it is converted to **React.createElement()**.

React doesn't actually understand JSX directly — it's just **syntactic sugar**. JSX gets **compiled to React.createElement()** calls before the browser sees it.

Example Overview

We'll build the same UI in **two different ways**:

Component	**Technique**	**Description**
Welcome1	Using JSX	Clean and readable syntax
Welcome2	Using React.createElement()	What JSX compiles into behind the scenes

📂 **File: src/App.js**

```
import React from "react";

// ✅ JSX version (what we usually write)
function Welcome1() {
  return (
    <div className="welcome1">
      <h2>Welcome to the App!</h2>
      {/* JSX syntax allows us to write HTML-like code in JavaScript */}
      <p>This is a functional component using JSX.</p>
    </div>
  );
}

// ⚙️ React.createElement version (under the hood)
function Welcome2() {
  return React.createElement(
    "div",                             // Element type: <div>
    { className: "welcome2" },         // Props/attributes for <div>
    React.createElement("h2", {}, "Welcome to the App!"), // Child: <h2>
    React.createElement("p", {}, "This is a functional component using
React.createElement().")
    // {} is for props/attributes like className, id, style, title, onClick, etc.
  );
}
function App() {
  return (
    <div>
      {/* Renders both components side by side */}
      <Welcome1 />
      <Welcome2 />
    </div>
  );
}

export default App;
```

☠ **Key Takeaways**

Concept	JSX Version	Under-the-Hood Equivalent
HTML-like Syntax	<div className="box">Hi</div>	React.createElement('div', {className: 'box'}, 'Hi')
Multiple Children	Nested inside JSX	Passed as arguments in createElement()
Cleaner to Read/Write	✅	❌ More verbose
Required for React?	❌ JSX is optional	✅ createElement() is what React understands

☠ **Summary**

- JSX makes it easier to build UI by **looking like HTML**.
- React uses **React.createElement()** internally to build its **Virtual DOM**.
- Tools like **Babel** convert JSX into React.createElement() during the build process.
    - Link: https://babeljs.io/docs/

🏳 Part 14: Developer Tools and Extensions (VS Code + Chrome)

1. VS Code Extension: ES7+ React/Redux/GraphQL/React-Native Snippets

This extension provides **handy code snippets** that help React developers write code faster and cleaner.

Common Shortcuts

Shortcut	Description
rafce	React **Arrow Function Component** with export
rafc	React **Arrow Function Component** with default export
rafcp	React **Arrow Function Component** with PropTypes
rfc	React Functional Component (regular function syntax)
rcc	React Class Component

📌 To use these: **Type the shortcut in a .js file and hit Enter.**

🧭 **To explore more** snippets:

Visit the extension page on VS Code Marketplace and search for: **“ES7+ React/Redux/GraphQL/React-Native snippets”**

2. Chrome Extension: React Developer Tools

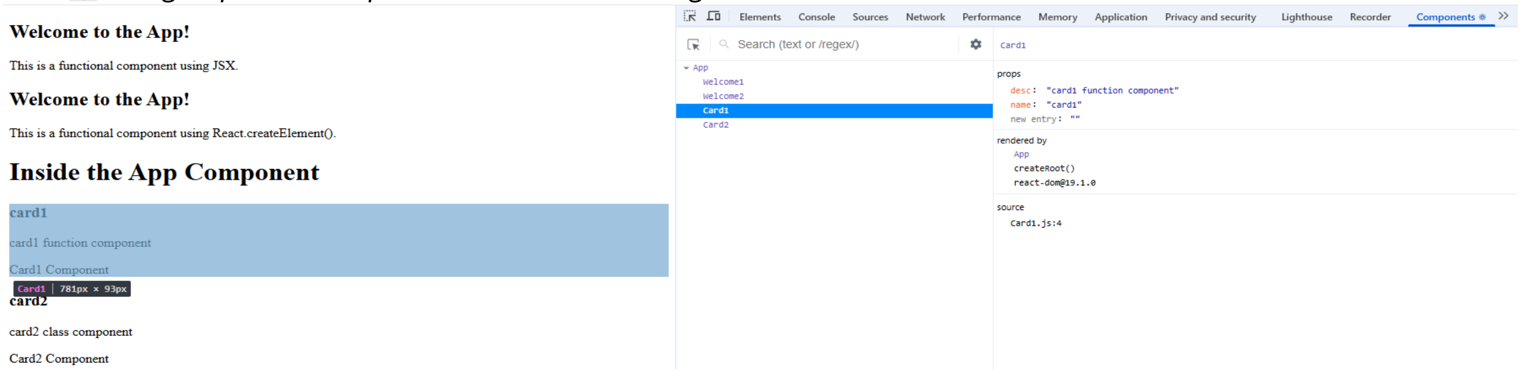
📦 Install from: https://chromewebstore.google.com/

Search: **React Developer Tools** and click **"Add to Chrome"**

Benefits of React Developer Tools Extension

Benefit	Description
Inspect React Components	View and inspect the component hierarchy directly in Chrome DevTools.
See Props and State	Easily view props and state values of each component.
Debug Faster	Helps identify what’s causing re-renders and allows time-travel debugging in development.
Virtual DOM Tree	View the actual structure of your React app, including nested components.
Developer-Friendly	Shows component names, hooks (like useState, useEffect), and even performance optimizations.

📷 *using Inspect → Components tab after installing the extension.*



Welcome to the App!

This is a functional component using JSX.

Welcome to the App!

This is a functional component using React.createElement().

Inside the App Component

card1

card1 function component

Card1 Component

Card1 | 781px × 93px

card2

card2 class component

Card2 Component

```
▾ App
    Welcome1
    Welcome2
    Card1
    Card2
```

```
Card1

props
  desc: "card1 function component"
  name: "card1"
  new entry: ""

rendered by
  App
  createRoot()
  react-dom@19.1.0

source
  Card1.js:4
```

Part 15: How to Use React Icons in Your Project

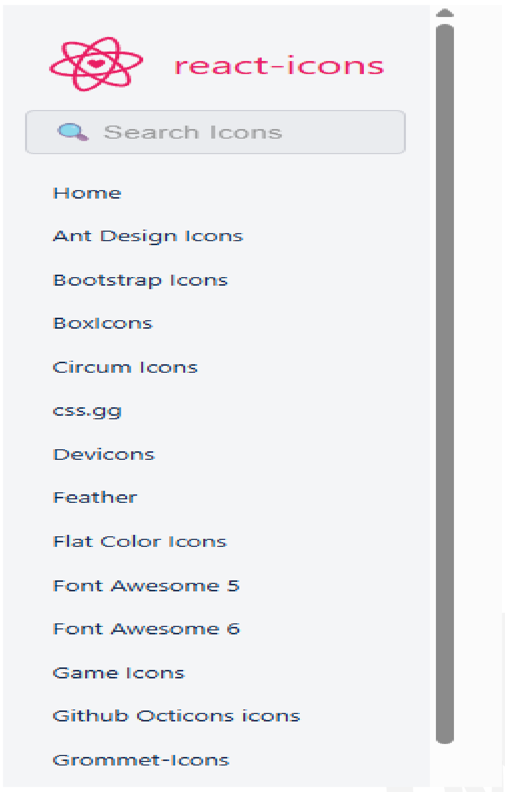
What are React Icons?

React Icons is a library that lets you easily include **popular icon packs** (like Font Awesome, Material Icons, etc.) as **React components**.

✅ No need for external <link> tags — just import and use!

Step-by-Step Instructions

🔗 **Visit the official docs:** https://react-icons.github.io/react-icons



Installation

Open your terminal and run:

```
npm install react-icons -save
```

Confirm Installation

Check your package.json file. You should see :

```
"react-icons": "^5.5.0"
```

This confirms that the icon library has been installed successfully.

Tips

- Use **Ctrl + F** on the icon directory page to search for icons by name.
- Icons are **scalable SVGs**, meaning you can size them using width, height, or fontSize in CSS.

Basic Usage Example (fa6 Font Awesome 6)

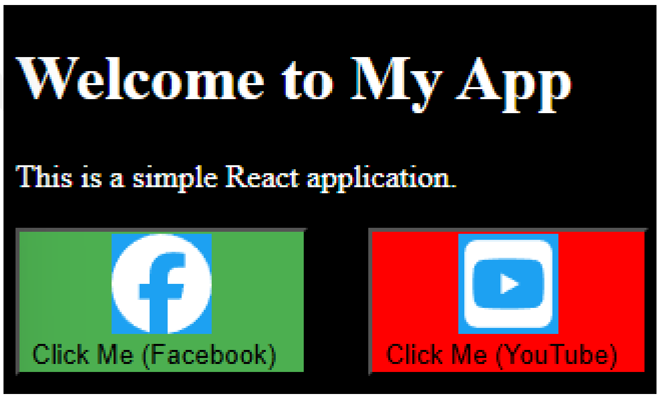
File: App.js

```
import React from "react";
import "./components/index.css"; // Importing custom styles
import { FaFacebook, FaSquareYoutube } from "react-icons/fa6"; // Importing icons

// Functional component rendering two icon buttons
const App = () => {
  return (
    <div>
      <h1>Welcome to My App</h1>
      <p>This is a simple React application.</p>

      <div className="btnContainer">
        {/* Facebook Button */}
        <button style={{ backgroundColor: "#4CAF50" }}>
          <FaFacebook className="FBicon" />
          <br />
          Click Me (Facebook)
          <span style={{ marginLeft: "8px" }}></span>
        {/* span used to add spacing between text and icon */}
        </button>

        {/* YouTube Button */}
        <button style={{ backgroundColor: "#FF0000" }}>
          <FaSquareYoutube className="YTicon" />
          <br />
          Click Me (YouTube)
          <span style={{ marginLeft: "8px" }}></span>
        </button>
      </div>
    </div>
  );
};

export default App;
```

📂 **File: index.css**

```
body {
  background-color: black;
  color: white;
}

.btnContainer {
  display: flex; /* Flex makes layout responsive with auto alignment, spacing, and resizing of elements. */
  gap: 30px;     /* Adds spacing between buttons */
}

.FBicon {
  width: 50px;
  height: 50px;
  background-color: #1DA1F2;
  color: white;
}

.YTicon {
  width: 50px;
  height: 50px;
  background-color: #1DA1F2;
  color: white;
}
```

🖼️ **Output:**


Welcome to My App

This is a simple React application.

☠️ **Benefits of Using React Icons**

Benefit	Description
🔌 Easy Integration	Import just what you need — no huge icon bundles.
🧱 Component-Based	Icons are React components — easy to style and control like any other component.
🎯 Popular Icon Packs	Includes Font Awesome, Material Icons, Bootstrap Icons, Remix Icons, and more.
💡 Better Performance	Tree-shakable: Only used icons are bundled in your app.
🎨 Fully Customizable	Use inline styles or class-based CSS to style icons.

🏳 Part 16: How to Use Bootstrap with React

☠ Official Docs

Visit: https://react-bootstrap.github.io/docs/getting-started/introduction/

☠ Installation Steps

1. **Open your terminal** in your project folder.
2. Run the following command to install Bootstrap and React-Bootstrap:

```
npm install react-bootstrap bootstrap
```

3. **Verify installation** in package.json:

```
"dependencies": {
  ...
  "bootstrap": "^5.3.7",
  "react-bootstrap": "^2.10.10",
  ...
}
```

If you see "react-bootstrap": "^2.10.10" — ✅ Bootstrap is successfully installed.

☠ How to Use Bootstrap Components

React Bootstrap components can be **imported individually** or using **destructuring**.

```
// Option 1: Named imports
import { Button, Card } from 'react-bootstrap';

// Option 2: Direct imports
import Button from 'react-bootstrap/Button';
import Card from 'react-bootstrap/Card';
```

☠ Add Bootstrap CSS

Include this line in your src/index.js or App.js:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

This line ensures Bootstrap's default styles are applied globally.

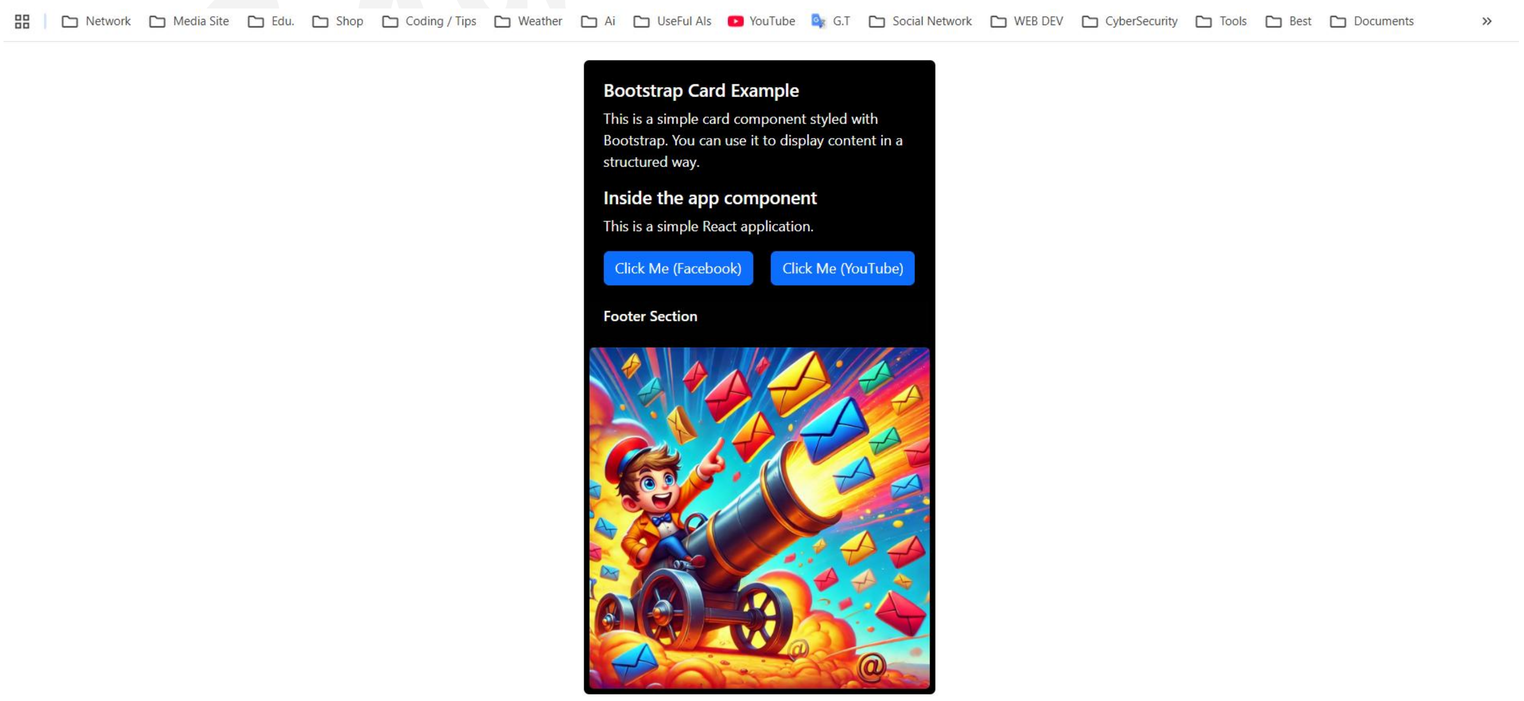
★ Example: Bootstrap Card + Button + Custom Styling

📂 src/App.js

```
import React from "react";
import Button from "react-bootstrap/Button";
import Card from "react-bootstrap/Card";
import "bootstrap/dist/css/bootstrap.min.css"; // Importing global Bootstrap styles

const App = () => {
  return (
    <Card
      style={{
        backgroundColor: "black",
        color: "white",
        padding: "5px",
        maxWidth: "393px",
        margin: "20px auto",
      }}
    >
      {/*
```

```
        padding      → adds space inside the card
        maxWidth     → limits the card's width
        margin       → centers the card horizontally with space around it
      */}

      <Card.Body>
        <Card.Title>Bootstrap Card Example</Card.Title>

        <Card.Text>
          This is a simple card component styled with Bootstrap. You can use it
          to display content in a structured way.
        </Card.Text>

        <div>
          <h5>Inside the App Component</h5>
          <p>This is a simple React application.</p>

          {/* Button group with spacing using Flexbox */}
          <div style={{ display: "flex", gap: "20px" }}>
            <Button variant="primary">Click Me (Facebook)</Button>
            <Button variant="danger">Click Me (YouTube)</Button>
          </div>
        </div>
      </Card.Body>

      <Card.Footer>
        <h6>Footer Section</h6>
      </Card.Footer>

      {/* Bootstrap image added to the bottom */}
      <Card.Img
        src="https://imgur.com/9WiGoHL.png"
        alt="Card visual"
        style={{ width: "100%", height: "auto", marginTop: "10px" }}
        // marginTop: "10px" adds space above the image
      />
    </Card>
  );
};

export default App;
```

Output Preview



Bootstrap Card Example

This is a simple card component styled with Bootstrap. You can use it to display content in a structured way.

Inside the app component

This is a simple React application.

Click Me (Facebook) Click Me (YouTube)

Footer Section

☠ Benefits of Using Bootstrap in React

Benefit	Description
Prebuilt Styles	Use components like Button, Card, Navbar, etc., out of the box
Fast Prototyping	Quickly build beautiful layouts with minimal custom CSS
Component-Based	Perfect match for React's component architecture
Responsive	All Bootstrap components are mobile-friendly by default

---

Part 17: state, setState() and Event Handler -- (Using Class Component)

💡 Concept: Props vs State

Feature	props	state
Mutability	❌ Immutable (read-only)	✅ Mutable (can be updated)
Purpose	Used to **pass data from parent**	Used to **store and manage component data**
Controlled By	Parent component	Component itself
Usage	this.props.someValue	this.state.someValue

❓ Why is state Important?

- state lets your component **remember and manage data** over time — even as users interact with it.
- When the state changes, **React re-renders the component automatically**, updating the UI.
- Useful for interactive UIs like counters, forms, messages, to-do lists, etc.

📂 File: src\App.js

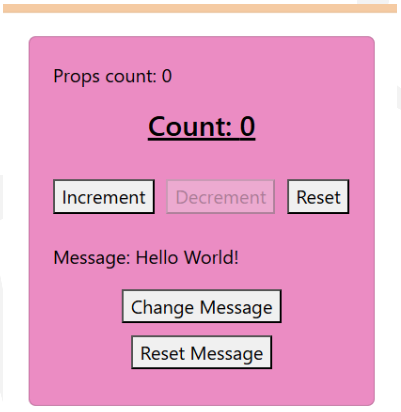
```
import React from 'react';
import State from './State';

function App() {
  return (
    <div className="App">
      {/* Passing props to the State component */}
      <State count="0" />
    </div>
  );
}

export default App;
```

📂 File: src\State.js

```
import React, { Component } from "react";
import Card from "react-bootstrap/Card"; // Bootstrap Card; npm install react-bootstrap
bootstrap
import "bootstrap/dist/css/bootstrap.min.css"; /* The following line can be included in your
src/index.js or App.js file */
```

```
export default class State extends Component {
  // Constructor method for initializing state
  constructor(props) {
    super(props); // Pass props to the parent class (Component)
    // The "parent component" refers to whichever component renders the State component-it
could be App.js, but it could also be any other component that uses <State />.

    // Initialize the component's internal state
    this.state = {
      count: 0,                  // State variable for counter
      message: "Hello World!"    // State variable for message
    };
  }

  // render() is a required method in class components.
  // It returns the JSX to be displayed
  render() {
    // Destructuring state
    const { count, message } = this.state;

    return (
      <Card
        style={{
          backgroundColor: "hsl(325, 70.40%, 73.50%)",
          color: "rgb(0, 0, 0)",
          padding: "20px",
          maxWidth: "295px",
          margin: "20px auto"
        }}
      >
        <div>
          {/* Display prop passed from App.js */}
          <p>Props count: {this.props.count}</p>
          {/* this used to access the props of the component; App component in this case
because it called in App.js */}

          <h4 style={{ textAlign: "center", textDecoration: "underline" }}>
            Count: {count}
          </h4>
          <br />

          {/* Event handlers to update state */}
          <div style={{ display: "flex", gap: "10px" }}>
            {/* Increment button with disable limit */}
            <button
              onClick={() => this.setState({ count: count + 1 })}
              disabled={count === 10}
            >
              Increment
            </button>

            {/* Decrement button with disable limit */}
            <button
              onClick={() => this.setState({ count: count - 1 })}
              disabled={count === 0}
            >
              Decrement
            </button>

            {/* Reset button */}
            <button onClick={() => this.setState({ count: 0 })}>
              Reset
            </button>
          </div>

          <br />
          {/* Message section */}
          <p>Message: {message}</p>

          {/* Change message button */}
          <button
            style={{ display: "block", margin: "10px auto" }}
            onClick={() => this.setState({ message: "Hello React!" })}
          >
            Change Message
          </button>

          {/* Reset message button */}
          <button
            style={{ display: "block", margin: "10px auto" }}
            onClick={() => this.setState({ message: "Hello World!" })}
          >
            Reset Message
          </button>
        </div>
      </Card>
    );
  }
}
```

Output:



☠ Summary

Concept	Description
this.state	Holds component's local, changeable data
this.setState()	Used to update state values and trigger a re-render
render()	Returns the JSX layout (must be present in class components)
this.props	Access props passed from parent component
Buttons	Trigger events to modify state

🏳 Part 18: Conditional Rendering in React

Showing different components based on user state

📌 Scenario:

You want to render:

- <HomePage /> if the user is **logged in**
- <LogInPage /> if the user is **not logged in**

☠ File Structure:

```
src/
├── App.js
└── ConditionalRendering/
    ├── HomePage.js
    ├── LonInPage.js
    └── ConditionalRendering.js
```

★ Examples:

📂 **src/ConditionalRendering/HomePage.js**

```
import React from 'react';

export default function HomePage() {
  return (
    <div>
      <h1>Home Page</h1>
    </div>
  );
}
```

📂 **src/ConditionalRendering/LonInPage.js**

```
import React from 'react';

export default function LonInPage() {
  return (
    <div>
      <h1>Login Page</h1>
    </div>
  );
}
```

📂 **src/ConditionalRendering/ConditionalRendering.js**

```
import React, { Component } from "react";
import HomePage from "./HomePage";
import LogInPage from "./LonInPage";

export default class ConditionalRendering extends Component {
  constructor(props) {
    super(props);

    this.state = {
      isLoggedIn: false, // Change this to true to see the Home Page
    };
  }

  render() {
    const { isLoggedIn } = this.state; // destructuring state

    // rendering components based on condition

    // -------- Method 1: using if-else statement --------
    // if (isLoggedIn) {
    //   return <HomePage />;
    // } else {
    //   return <LogInPage />;
    // }

    // -------- Method 2: using element variable --------
    // let element;
    // if (isLoggedIn) {
    //   // storing the component in a variable
    //   element = <HomePage />;
    // } else {
    //   element = <LogInPage />;
    // }
    // return <div>{element}</div>;

    // -------- Method 3: using ternary operator --------
    // return (
    //   <div>
    //     {isLoggedIn ? <HomePage /> : <LogInPage />}
    //   </div>
    // );

    // -------- Method 4: using short-circuit evaluation --------
    return (
      <div>
        {isLoggedIn && <HomePage />}
        {/* if isLoggedIn is true then check the next condition; return null if false */}

        {!isLoggedIn && <LogInPage />}
        {/* if isLoggedIn is false then render LogInPage */}
      </div>
    );
  }
}
```

📁 **src/App.js**

```
import React from 'react';
import ConditionalRendering from './ConditionalRendering/ConditionalRendering.js';

function App() {
  return (
    <div>
      <ConditionalRendering />
    </div>
  );
}

export default App;
```

☠ **Summary of Methods:**

Method	Code Style	Description
if-else	if (isLoggedIn) {}	Basic condition handling
Variable	let page = ...	Store component in a variable and render
Ternary	{isLoggedIn ? A : B}	Common way to toggle UI
Short-circuit	{isLoggedIn && A}	Render component only if the condition is true

---

🏳 Part 19: Event Handler – onChange & onClick (Class Component)

Purpose:

- Learn how to **handle user input** with onChange
- Respond to user actions like button clicks with onClick
- Understand how **React state updates asynchronously**

📂 **File: src\EventHandlerClass\EventHandlerClass.js**

```
import React, { Component } from "react";

export default class EventHandlerClass extends Component {
  // we always need constructor to create state for class
  constructor(props) {
    super(props);
    this.state = {
      changedValue: "", // state to store input value
    };
  }

  // Event handler for button click
  handleClick = () => {
    console.log("Button Clicked");
  };

  // Event handler for input change
  handelOnChange = (event) => {
    // console.log("Input Value:", event.target.value);

    // updating the state with the input value
    this.setState(
      { changedValue: event.target.value }, // new state value
      () => {
        // ✅ This callback runs after the state has been updated
        console.log("✅ Updated State Value:", this.state.changedValue);
      }
    );

    // ❗ This log will not show the updated value immediately
    console.log("⏳ State Value (before update):", this.state.changedValue);

    // 🧠 Note: setState is asynchronous — it doesn't update the state instantly.
    // It schedules the update to happen after the current function completes.
  };

  render() {
    return (
      <div>
        {/* Input field with onChange handler */}
        <input
          onChange={this.handelOnChange}
          type="text"
          placeholder="Enter text here"
        />

        {/* Button with onClick handler */}
        <button style={{ margin: "10px" }} onClick={this.handleClick}>
          Click
        </button>

        {/* Displaying the input value from the state */}
        <p>Input Value: {this.state.changedValue}</p>
      </div>
    );
  }
}
```

📂 **File: src\App.js**

```
import React from 'react';
import EventHandlerClass from './EventHandlerClass/EventHandlerClass';

function App() {
  return (
    <div>
      <EventHandlerClass />
    </div>
  );
}
```

```
    export default App;
```

★ **Summary:**

Feature	Explanation
onChange	Captures user input from form fields (like <input>)
onClick	Handles button click events
setState()	Used to update state and re-render component
state	Stores and tracks input data dynamically
setState(callback)	The callback runs **after** the state has been updated
asynchronous	State updates don't happen instantly; they're queued by React internally

---

🏳 Part 20: Event Handler Binding in React

📌 **Purpose:**

Understand **how this works in class components** and why **event handler binding** is necessary.

❓ **Why Binding Is Needed:**

In JavaScript, when using **class methods as event handlers**, this is not bound by default. That means:

- Inside the method, this might be undefined
- You need to bind this manually OR use arrow functions

📁 **File: src\EventBinding\index.js**

```
import React, { Component } from "react";

export default class EventBinding extends Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 0,
    };

    // ☑ Binding the method to the class instance
    // this.Decrease = this.Decrease.bind(this);
    // ensures that 'this' inside the Decrease() method refers to this component
  }

  // ☑ Method to decrease the count (standard ES6 method, works perfectly when bound)
  Decrease() {
    this.setState(() => ({
      count: this.state.count - 1,
    }));
  }

  // ☑ Alternative: define the method as an arrow function
  // Arrow functions automatically bind 'this' to the component
  // Decrease = () => {
  //   this.setState(() => ({
```

```
  //     count: this.state.count - 1,
  //   }));
  // };

  render() {
    return (
      <div>
        <h1>Count: {this.state.count}</h1>

        {/* Button to increase the count */}
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Increase
        </button>

        {/* ✅ No error for the ES6 arrow function method */}
        <button onClick={() => this.Decrease()}>
          Decrease
        </button>

        {/* ❌ This will cause an error if the method is not bound in the constructor */}
        <button onClick={this.Decrease}>
          Decrease (Error)
        </button>

        {/* ✅ Using .bind() on call to fix binding issue with normal function */}
        <button onClick={this.Decrease.bind(this)}>
          Decrease (Bind)
        </button>

        {/*
          .bind(this) is used to explicitly set the value of 'this'
          inside the method, ensuring it refers to the current class instance.
        */}
      </div>
    );
  }
}
```

File: src\App.js

```
import React from 'react';
import EventBinding from './EventBinding';

function App() {
  return (
    <div>
      <EventBinding />
    </div>
  );
}

export default App;
```

★ Summary of Methods

Method	Works Without Error?	Notes
Decrease = () => {} (arrow fn)	✅ Yes	Automatically binds this
Decrease() (regular fn)	❌ No	Requires manual binding
this.Decrease.bind(this)	✅ Yes	Binds manually on every render — **not performant for large apps**

this.Decrease = this.Decrease.bind(this) in constructor	✅ Yes	Best practice if using regular methods

---

📍 Part 21: React Hooks – useState Hook

❓ What are Hooks?

- **Hooks** are functions introduced in React v16.8 that allow **functional components to use state and other React features** (like lifecycle methods).
- Before hooks, only **class components** could manage state using this.state and setState().
- useState is the **most commonly used hook** to manage local state in functional components.

★ Class Component with State

📁 **src\HooksUseState\IndexClassCmp.js**

```
import React, { Component } from "react"; // Importing Component to create a class component

export default class IndexClassCmp extends Component {
  constructor(props) {
    super(props);
    // Initializing the component's state
    this.state = {
      count: 0,
    };
  }

  // Arrow function to increment count, avoids the need to bind 'this'
  increment = () => {
    this.setState({
      count: this.state.count + 1,
    });
  };

  render() {
    const { count } = this.state;
    return (
      <div>
        <h2>Count: {count}</h2>
        <button onClick={this.increment}>increment (Class Component)</button>
      </div>
    );
  }
}
```

★ Functional Component with useState

📁 **src\HooksUseState\IndexFunctionalCmp.js**

```
import { useState } from "react"; // useState hook lets you manage state in function
components

export default function IndexFunctionalCmp() {
  // count = current value of the state
  // setCount = function to update the state
  // 0 = initial state value
  const [count, setCount] = useState(0);
  // Function to decrement the count
  const Decrement = () => {
    setCount(count - 1); // updating the state using setCount
  };

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={Decrement}>decrement (Functional Component)</button>
    </div>
  );
}
```

Main App Component

📁 **src\App.js**

```
import React from 'react';
import IndexClassCmp from './HooksUseState/IndexClassCmp.js';
import IndexFunctionalCmp from './HooksUseState/IndexFunctionalCmp.js';

function App() {
  return (
    <div>
      <IndexClassCmp />
      <IndexFunctionalCmp />
    </div>
  );
}

export default App;
```

☠️ **props vs state** ⚖️

Feature	**props**	**state**
Read-only?	✅ Yes	❌ No
Passed from parent?	✅ Yes	❌ No
Can be updated?	❌ No	✅ Yes
Where used?	Functional & Class components	Functional & Class components
Purpose	Configuration from parent	Internal data that changes over time

🎯 **Why is useState Important?**

- It **removes the need for class components** just to use state.
- Makes your code **cleaner, shorter, and easier to maintain**.
- Functional components are now the **standard in modern React development**.
- Enables **declarative UI**: your components respond automatically to state changes.

🏳 Part 22: useState → Update State Based on Previous State

❓ Why do we need to use the previous state?

React **batches** multiple state updates together for performance.
If you update the state **multiple times in a row**, React might still reference the **initial (old) value**, unless you **explicitly use the previous state.**

★ Example Component: **StateUpdate.js**

📁 **src\Components\StateUpdate.js**

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

export default function StateUpdate() {
  const [count, setCount] = useState(0); // 0 is the initial value

  // ❌ WRONG way: this doesn't behave as expected
  // Each update uses the same stale value (count = 0)
  // const Increment = () => {
  //   setCount(count + 1); // 0 + 1 = 1
  //   setCount(count + 1); // still 0 + 1 = 1
  //   setCount(count + 1); // still 0 + 1 = 1
  // };

  // ☑ CORRECT way: using previous state
  const Increment = () => {
    // Method 1: Arrow function inside setCount
    setCount((PrevCount) => PrevCount + 1); // 0 → 1
    setCount((PrevCount) => PrevCount + 1); // 1 → 2

    // Method 2: Explicit return
    setCount((PrevCount) => {
      return PrevCount + 1; // 2 → 3
    });
  };

  return (
    <div>
      <h2>Count: {count}</h2>
      {/* We are passing the function reference — not calling it immediately */}
      <button onClick={Increment}>
        {/* ⚠ Not calling the function here (no parentheses) */}
        {/* ☑ This tells React: "Call this function when the button is clicked" */}
        {/*
       ❓ What are the different ways to define and call a function?

      ☑ Example 1: Pass function reference (RECOMMENDED for most cases)
      <button onClick={Increment}>Click Me</button>

      ❌ Example 2: Call function immediately (DO NOT DO THIS in JSX)
      <button onClick={Increment()}>Click Me</button>
      -- This will call the function immediately when the component renders

      ☑ Example 3: Use inline arrow function (when you need to pass arguments or control
logic)
      <button onClick={() => Increment()}>Click Me</button>
      <button onClick={() => console.log("clicked")}>Log</button>

      ☑ Example 4: Define inline function with logic
      <button onClick={() => {
        if (count < 5) {
          Increment();

        }
      }}>Safe Increment</button>
    */}
      Increment
    </button>
  </div>
  );
}
```

📁 **src\App.js**

```
import StateUpdate from "./Components/StateUpdate";

function App() {
  return (
    <div>
      <StateUpdate />
    </div>
  );
}

export default App;
```

🏳 Part 23: **Event Bubbling | stopPropagation()**

✅ **src\App.js**

```
import "./index.css";

function App() {
  // 💡 Parent element event handler
  const HandelParentEvent = (event) => {
    console.log("parent event: ", event);
  };

  // 💡 Child button event handler
  const HandelChildEvent = (event) => {
    console.log("child event: ", event);

    // 🚫 Stop the event from bubbling up to the parent
    event.stopPropagation();
    // This prevents the parent div's onClick from being triggered
  };

  return (
    <div className="parent" onClick={HandelParentEvent}>
      <h2>Welcome to my app</h2>

      <button onClick={HandelChildEvent}>Click Me</button>
    </div>
  );
}

export default App;

// After clicking on the div, the only parent event will be triggered.
// But when you click the button, the child event will be called first, then the parent event
will be called. This process is known as event bubbling. To prevent event bubbling, you can
utilize with `event.stopPropagation()`.
```

☠ **What is Event Bubbling?**

- When an event is triggered on a child element, it **bubbles up** to its parent elements.

- For example, clicking the <button> triggers **both the button and its parent's** onClick unless stopped.

❌ **Problem:**

- Clicking the <button> logs both **"child event"** and **"parent event"**.

✅ **Solution:**

- Use event.stopPropagation() inside the child event handler to **stop bubbling**.

☠ **Output in Console:**

If event.stopPropagation() is used:

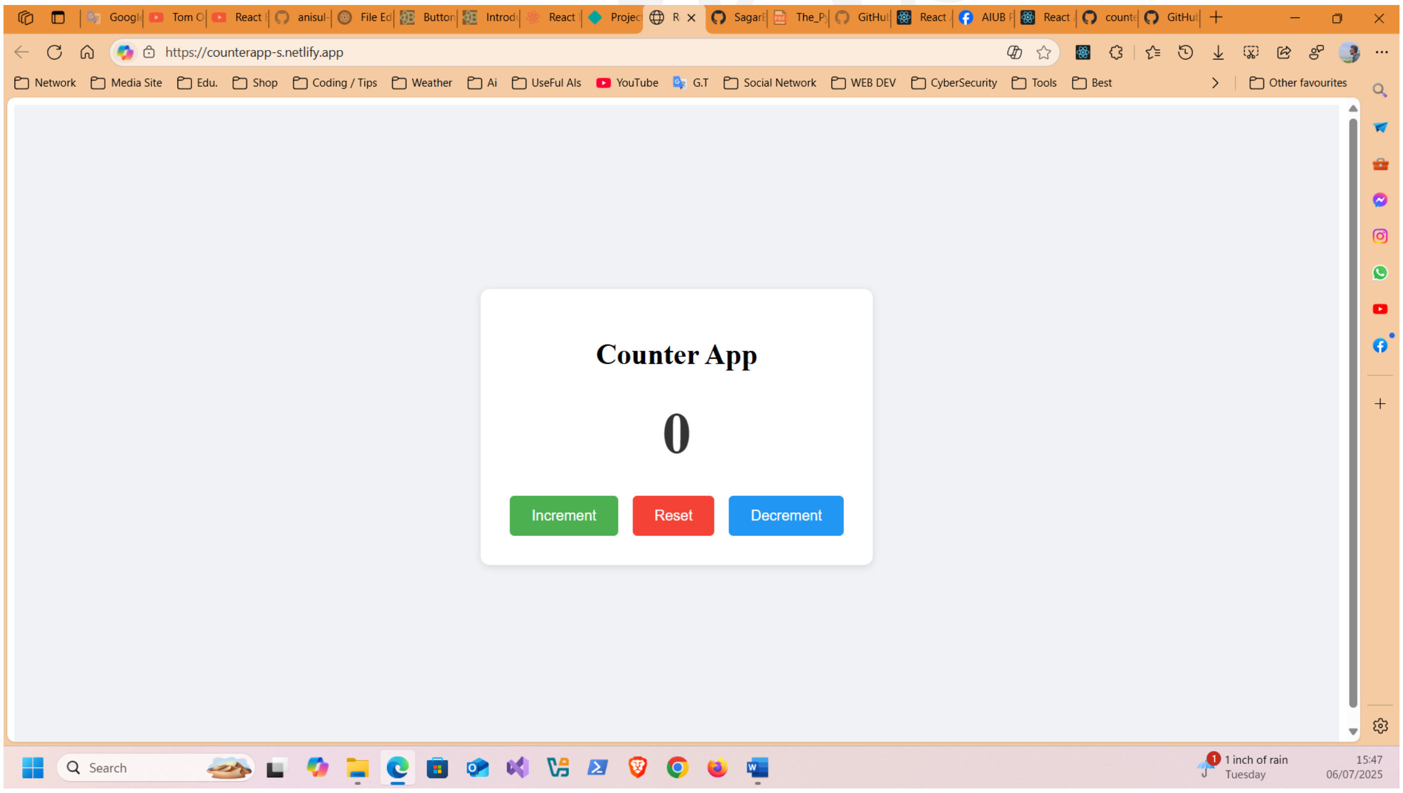
child event: [object PointerEvent]

If **NOT used**:

child event: [object PointerEvent]

parent event: [object PointerEvent]

🏳 Part 24 Assignment-2 | Counter App=



☠ Assignment Requirements Recap

Purpose:

- Use useState() hook
- Handle events (onClick)
- Implement conditional rendering (disable buttons)

Marks Breakdown:

- Increment, Decrement, Reset = **6 points**
- Disable Increment at count 5 and Decrement at -5 = **4 points**

☠ **Project Demo Reference**

Your app behavior should match this: 🌐 https://counterapp-s.netlify.app

---

----------------------------- X -----------------------------