**WEEK – 6**

1. **ReactJS-HOL**

**🧠 Key Concepts**

**1. Single-Page Application (SPA) and Its Benefits**

A Single-Page Application (SPA) is a web application that loads a single HTML page and dynamically updates content as the user interacts with the app, without requiring a full page reload.

**Benefits:**

* **Enhanced User Experience:** Smooth and responsive interactions.
* **Reduced Server Load:** Only necessary data is fetched, not entire pages.
* **Faster Navigation:** Content updates without full page reloads.

**2. React and Its Working**

React is a JavaScript library developed by Facebook for building user interfaces, especially for SPAs. It allows developers to create reusable UI components. React uses a virtual DOM to efficiently update and render components when data changes.

**3. Differences Between SPA and MPA**

* **SPA (Single-Page Application):**
  + Loads a single HTML page.
  + Updates content dynamically.
  + Better suited for dynamic platforms.(
* **MPA (Multi-Page Application):**
  + Each page is a separate HTML file.
  + Full page reloads on navigation.
  + Better for SEO and static content.

**4. Pros & Cons of Single-Page Applications**

**Pros:**

* Fast and responsive user experience.
* Reduced server load.
* Efficient client-side rendering.([Homepage](https://www.spaceotechnologies.com/blog/single-page-application-vs-multi-page-application/?utm_source=chatgpt.com), [FreeCodeCamp](https://www.freecodecamp.org/news/what-is-the-virtual-dom-in-react/?utm_source=chatgpt.com))

**Cons:**

* SEO challenges due to dynamic content.
* Initial load time can be longer.
* Browser history management can be complex. ([Medium](https://medium.com/%40VAISHAK_CP/the-pros-and-cons-of-single-page-applications-spas-06d8a662a149?utm_source=chatgpt.com))

**5. React Overview**

React is a component-based library that allows developers to build encapsulated components that manage their own state, then compose them to make complex UIs. It promotes the creation of reusable UI components, which present data that changes over time.

**6. Virtual DOM**

The Virtual DOM is a lightweight JavaScript representation of the actual DOM. React uses it to optimize updates by comparing the virtual DOM with the real DOM and updating only the parts that have changed, enhancing performance.

**7. Features of React**

* **JSX (JavaScript XML):** Syntax extension that allows mixing HTML with JavaScript.
* **Components:** Encapsulated, reusable pieces of UI.
* **One-way Data Binding:** Data flows in a single direction, making the application more predictable.
* **Virtual DOM:** Efficiently updates and renders components.
* **Performance:** Optimized rendering and updates.

**🛠️ Hands-On Lab: Setting Up a React Environment**

**Prerequisites**

* Node.js and NPM: Install from [Node.js Official Website](https://nodejs.org/en/download/).
* Visual Studio Code: Download from [Visual Studio Code](https://code.visualstudio.com/).

**Steps to Create a New React Application**

1. **Install Create React App:**Open your command prompt and run:

npx create-react-app myfirstreact

1. **Navigate to the Project Directory**:

cd myfirstreact

1. **Open the Project in Visual Studio Code:**

code .

1. **Modify App.js:**
   * Navigate to the src folder.
   * Open App.js.
   * Replace its content with:

import React from 'react';

function App() {

return (

<div>

<h1>Welcome to the first session of React</h1>

</div>

);

}

export default App;

1. **Run the Application:**

npm start

1. **View in Browser:**Open your browser and navigate to http://localhost:3000 to see your React application in action.
2. **ReactJS-HOL**

**🧠 React Component Concepts**

**What Is a React Component?**

A **component** in React is an independent, reusable building block—similar to a JavaScript function—that encapsulates logic and UI structure. It accepts inputs (called **props**) and returns JSX to render UI elements.

**Differences Between React Components and JavaScript Functions**

* Conceptually, components act like JavaScript functions: they take inputs (props) and return output (JSX or React elements)
* However, **class components** are ES6 classes extending React.Component, offering features like internal state and lifecycle methods. **Function components** are plain JavaScript functions without internal state unless you use Hooks.

**⚙️ Types of React Components**

1. **Class Components**
   * Implemented as ES6 classes inheriting from React.Component.
   * Must define a render() method that returns JSX .
   * Can initialize internal state within a constructor(props)—you must call super(props) and then define this.state = { ... }
   * Support lifecycle methods (e.g. componentDidMount) for organizing setup, updates, cleanup.
2. **Function Components**
   * Defined as plain JavaScript functions that return JSX.
   * Cannot use lifecycle methods or internal state by default—but with **Hooks** like useState and useEffect, they can manage state and side effects.
   * Modern React prefers function components with Hooks for most use cases.

**🔍 Class vs Function Components**

| **Feature** | **Class Component** | **Function Component** |
| --- | --- | --- |
| Syntax | class MyComp extends React.Component { ... } | function MyComp(props) { return <JSX /> } |
| State initialization | In constructor(props), initialize this.state = {...} | Use const [state, setState] = useState(...) (Hook) |
| JSX Return | Must use render() method | Return JSX directly |
| Lifecycle methods | Available (componentDidMount, etc.) | Use Hooks (useEffect) |
| Complexity & Boilerplate | More verbose, uses this, requires binding methods | Simpler, more concise, no this confusion |

Class components are still supported but generally reserved for specific cases (e.g., **error boundaries**). Function components with Hooks are lighter, easier, and preferred for most modern development.

**🔧 Constructor and Render Method (Class Components)**

* **Constructor (constructor(props))**
  + Called when an instance of the class component is created.
  + Use super(props) to ensure the base class initializes properly.
  + Initialize component state here via this.state = {...}.
  + Avoid side effects here, use lifecycle methods instead
* **render() Method**
  + Mandatory method in class components; returns JSX describing what should be shown.
  + React calls render() after any state or prop updates.

**🛠 Hands-On Lab: Creating Components**

**Project Setup**

1. **Initialize the React app**

npx create-react-app StudentApp

cd StudentApp

code .

1. **Directory Structure & Component Files**  
   Create src/Components/ and inside it:
   * Home.js
   * About.js
   * Contact.js

**Example for a Class Component: Home.js**

import React from 'react';

class Home extends React.Component {

constructor(props) {

super(props);

// (optional) initialize state if needed

}

render() {

return (

<div>

<h1>Welcome to the Home page of Student Management Portal</h1>

</div>

);

}

}

export default Home;

Similarly, About.js and Contact.js should each export a class component rendering their respective messages.

**Modify App.js to Import/Render All Components**

import React from 'react';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

return (

<div>

<Home />

<About />

<Contact />

</div>

);

}

export default App;

**Run & Test**

npm start

Navigate to http://localhost:3000 to see all three messages displayed.

**✅ Summary of Objectives**

* **Explain React components** as reusable UI entities accepting props and rendering JSX.
* **Components vs JS functions:** Both process inputs and output UI, but components have structural conventions in React.
* **Component types:** Class (older with state/lifecycle), Function (modern, simpler, Hooks-enabled).
* **Class component:** Includes constructor (for state), and render() to output UI.
* **Function component:** Just a function returning JSX; state via Hooks only.

This setup should take ~30 minutes once prerequisites are installed. Let me know if you'd like to add routing or interactivity next!

1. **ReactJS-HOL**

**🧠 React Component Concepts**

**What Is a React Component?**

A **component** in React is an independent, reusable building block—similar to a JavaScript function—that encapsulates logic and UI structure. It accepts inputs (called **props**) and returns JSX to render UI elements.

**Differences Between React Components and JavaScript Functions**

* Conceptually, components act like JavaScript functions: they take inputs (props) and return output (JSX or React elements)
* However, **class components** are ES6 classes extending React.Component, offering features like internal state and lifecycle methods. **Function components** are plain JavaScript functions without internal state unless you use Hooks.

**⚙️ Types of React Components**

1. **Class Components**
   * Implemented as ES6 classes inheriting from React.Component.
   * Must define a render() method that returns JSX .
   * Can initialize internal state within a constructor(props)—you must call super(props) and then define this.state = { ... }
   * Support lifecycle methods (e.g. componentDidMount) for organizing setup, updates, cleanup.
2. **Function Components**
   * Defined as plain JavaScript functions that return JSX.
   * Cannot use lifecycle methods or internal state by default—but with **Hooks** like useState and useEffect, they can manage state and side effects.
   * Modern React prefers function components with Hooks for most use cases.

**🔍 Class vs Function Components**

| **Feature** | **Class Component** | **Function Component** |
| --- | --- | --- |
| Syntax | class MyComp extends React.Component { ... } | function MyComp(props) { return <JSX /> } |
| State initialization | In constructor(props), initialize this.state = {...} | Use const [state, setState] = useState(...) (Hook) |
| JSX Return | Must use render() method | Return JSX directly |
| Lifecycle methods | Available (componentDidMount, etc.) | Use Hooks (useEffect) |
| Complexity & Boilerplate | More verbose, uses this, requires binding methods | Simpler, more concise, no this confusion |

Class components are still supported but generally reserved for specific cases (e.g., **error boundaries**). Function components with Hooks are lighter, easier, and preferred for most modern development.

**🔧 Constructor and Render Method (Class Components)**

* **Constructor (constructor(props))**
  + Called when an instance of the class component is created.
  + Use super(props) to ensure the base class initializes properly.
  + Initialize component state here via this.state = {...}.
  + Avoid side effects here, use lifecycle methods instead
* **render() Method**
  + Mandatory method in class components; returns JSX describing what should be shown.
  + React calls render() after any state or prop updates.

**🛠 Hands-On Lab: Creating Components**

**Project Setup**

1. **Initialize the React app**

npx create-react-app StudentApp

cd StudentApp

code .

1. **Directory Structure & Component Files**  
   Create src/Components/ and inside it:
   * Home.js
   * About.js
   * Contact.js

**Example for a Class Component: Home.js**

import React from 'react';

class Home extends React.Component {

constructor(props) {

super(props);

// (optional) initialize state if needed

}

render() {

return (

<div>

<h1>Welcome to the Home page of Student Management Portal</h1>

</div>

);

}

}

export default Home;

Similarly, About.js and Contact.js should each export a class component rendering their respective messages.

**Modify App.js to Import/Render All Components**

import React from 'react';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

function App() {

return (

<div>

<Home />

<About />

<Contact />

</div>

);

}

export default App;

**Run & Test**

npm start

Navigate to http://localhost:3000 to see all three messages displayed.

**✅ Summary of Objectives**

* **Explain React components** as reusable UI entities accepting props and rendering JSX.
* **Components vs JS functions:** Both process inputs and output UI, but components have structural conventions in React.
* **Component types:** Class (older with state/lifecycle), Function (modern, simpler, Hooks-enabled).
* **Class component:** Includes constructor (for state), and render() to output UI.
* **Function component:** Just a function returning JSX; state via Hooks only.

This setup should take ~30 minutes once prerequisites are installed. Let me know if you'd like to add routing or interactivity next!

**4.React Js- HOL**

**🧠 Conceptual Objectives**

**✅ Why Component Lifecycle Matters (Need & Benefits)**

The lifecycle methods provide a structured way to run code at key stages of a component’s existence—before render, after mounting, on updates, and before removal. This allows you to initialize state, fetch data, manage subscriptions, optimize rendering, and perform cleanup, leading to better organization and performance in your app.

**📝 Common Lifecycle Methods & Their Sequence**

**Lifecycle Phases & Methods (Class Components):**

**Mounting Phase (initial render sequence):**

1. **constructor(props)** – initialize state and bind methods
2. static getDerivedStateFromProps(props, state) – optionally update state based on props before render
3. render() – mandatory method generating the JSX/UI
4. componentDidMount() – called once after component is inserted into the DOM; ideal for fetching data or side effects

**Updating Phase (on props/state changes):**

* static getDerivedStateFromProps()
* shouldComponentUpdate(nextProps, nextState)
* render()
* getSnapshotBeforeUpdate(prevProps, prevState)
* componentDidUpdate(prevProps, prevState, snapshot)

Unmounting Phase:

* componentWillUnmount() – cleanup before removal, e.g., clear timers, cancel network requests

Error Handling Phase:

* static getDerivedStateFromError(error)
* componentDidCatch(error, info) – execute custom logic when an error is thrown in child components, e.g., show fallback UI or log errors

**🔧 Hands‑On Lab: blogapp**

**Goal**

Create a class component named Posts that fetches posts from https://jsonplaceholder.typicode.com/posts using Fetch API in componentDidMount(), and handles errors with componentDidCatch().

**Step-by-Step Instructions (Estimated ~60 minutes)**

**1. Create React App**

npx create-react-app blogapp

cd blogapp

code .

**2. Create Post.js**

Define a simple component structure for a single post (e.g., render title and body—this is up to your implementation style).

**3. Create Class Component Posts.js**

import React from 'react';

import Post from './Post';

class Posts extends React.Component {

constructor(props) {

super(props);

this.state = {

posts: [],

error: null,

};

}

loadPosts() {

fetch('https://jsonplaceholder.typicode.com/posts')

.then(res => {

if (!res.ok) throw new Error('Network error');

return res.json();

})

.then(data => this.setState({ posts: data }))

.catch(error => { throw error; });

}

componentDidMount() {

this.loadPosts();

}

componentDidCatch(error, info) {

this.setState({ error });

alert(`Error loading posts: ${error.message}`);

}

render() {

if (this.state.error) {

return <div>Error occurred.</div>;

}

return (

<div>

{this.state.posts.map(p => (

<Post key={p.id} title={p.title} body={p.body} />

))}

</div>

);

}

}

export default Posts;

* constructor: initializes state object containing posts and error.
* loadPosts(): uses Fetch API to retrieve posts and update state.
* componentDidMount(): triggers loadPosts() after mounting.
* componentDidCatch(): handles rendering errors in children, displays an alert, sets error state. ([GeeksforGeeks](https://www.geeksforgeeks.org/reactjs/reactjs-componentdidmount-method/?utm_source=chatgpt.com), [BairesDev](https://www.bairesdev.com/blog/react-lifecycle-methods-hooks/?utm_source=chatgpt.com))

**4. Update App.js to Render Posts**

import React from 'react';

import Posts from './Posts';

function App() {

return (

<div>

<h1>Blog Posts</h1>

<Posts />

</div>

);

}

export default App;

5. Run and Test

npm start

Open http://localhost:3000. You should see fetched posts displayed (title and body). If an error occurs during fetch or rendering, an alert will appear and fallback UI is shown.

**🧾 Summary**

| **Objective** | **How Covered** |
| --- | --- |
| Explain need & benefits of lifecycle | Enables controlled operations at mount, update, unmount phases for setup, cleanup, and data flow |
| Identify lifecycle hook methods | Discussed constructor, getDerivedStateFromProps/Error, render, componentDidMount, componentDidCatch, etc. |
| Sequence of steps in rendering | constructor → getDerivedStateFromProps → render → componentDidMount |
| Implement componentDidMount() | Used to load posts after mounting |
| Implement componentDidCatch() | Handles child render errors by showing alert and fallback UI |

**🧠 Why This Matters**

* **componentDidMount()** ensures data is fetched or setup performed *after* DOM insertion—central for async operations and side effects
* **componentDidCatch()** helps isolate UI crashes to specific parts, preventing entire app failure and allowing graceful error handling .

1. **React JS- HOLY**

**🎯 Objectives & How They’re Met**

**• Why Style React Components?**

Styling enhances user experience, readability, and helps visually distinguish component states (e.g. “ongoing” vs “completed”). With React, scoped or inline styles let you modularize and conditionally render styles effectively.

**• CSS Modules vs. Inline Styles**

* **CSS Modules**: Scoped .module.css files enable local class names and prevent style collisions. Perfect for component-specific styles.
* **Inline styles**: Defined as JavaScript objects and passed via the style prop. Great for dynamic, conditional styling.

**🛠 Hands-On Lab: Styling a Cohort Dashboard**

**Scenario Overview**

You have a React app that renders cohort details using components. Your task: style these components using **CSS Modules** and **inline styling**.

**Step 1: Project Setup & Imports**

* Unzip and cd into the React app folder.
* Run npm install to restore packages.
* Open in VS Code.

**Step 2: Create CSS Module**

Make a file: CohortDetails.module.css

/\* CohortDetails.module.css \*/

.box {

width: 300px;

display: inline-block;

margin: 10px;

padding: 10px 20px;

border: 1px solid black;

border-radius: 10px;

}

dt {

font-weight: 500;

}

* .box class styles the container.
* Tag selector dt { font-weight: 500; } targets <dt> elements.([create-react-app.dev](https://create-react-app.dev/docs/adding-a-css-modules-stylesheet/?utm_source=chatgpt.com))

**Step 3: Import & Apply CSS Module in Boundary Component**

// CohortDetails.js (or similar component)

import React from 'react';

import styles from './CohortDetails.module.css';

const CohortDetails = ({ name, status, ...details }) => {

const titleStyle = { color: status === 'ongoing' ? 'green' : 'blue' };

return (

<div className={styles.box}>

<h3 style={titleStyle}>{name} ({status})</h3>

<dl>

{Object.entries(details).map(([label, value]) => (

<React.Fragment key={label}>

<dt>{label}</dt>

<dd>{value}</dd>

</React.Fragment>

))}

</dl>

</div>

);

};

export default CohortDetails;

* className={styles.box} applies the .box class from CSS Module.
* <h3> uses inline style via style={titleStyle}, coloring based on status.

**🔁 Combining Both Approaches**

* **CSS Modules** handle structural and consistent styles like layout, borders, margins.
* **Inline styles** handle dynamic styling (like color changes based on status) directly in JSX.

**🧠 Why This Matters**

| **Concept** | **Benefit** |
| --- | --- |
| Scoped CSS with Modules | Prevents style name collisions across components |
| Inline Styles for Dynamic UI | Enables easy conditional styling without extra CSS classes |
| Maintainable & Readable Code | Separates static styles and conditional logic cleanly |

* CSS Modules keep modules self-contained and reduce global CSS pollution.
* Inline styles simplify conditional styles (ternary operators) based on props/state.

**✅ Summary**

* You used **CSS Modules** to define styles scoped to the CohortDetails component.
* Applied .box styling for layout and used a CSS tag selector for <dt>.
* Used **inline styling** to dynamically set the <h3> color based on cohort status.
* This demonstrates both static and dynamic styling methods in React side by side.

If you'd like, we can extend this lab to include:

* Conditional class names using libraries like classnames (popular in React projects)
* Styling with CSS-in-JS (e.g. styled-components), or modular theme support.
* Responsiveness, hover effects, or animations within components.

Let me know your next direction!