**React JS - HOL**

**Hands on 1.**

**A)SPA (Single-Page Application) and Its Benefits**

Definition:

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 full page reloads.

Benefits:

Fast user experience: Only necessary content is updated, leading to smoother and faster interactions.

Reduced server load: Most logic runs in the browser; only data is fetched from the server.

Seamless navigation: No full page refreshes, making transitions between pages feel more like a native app.

Easier caching: Resources such as HTML, CSS, JS are loaded once and reused.

**B)React and Its Working**

Definition:

React is a popular open-source JavaScript library for building user interfaces, especially suited for SPAs. It was developed by Facebook.

How React Works:

React uses a component-based architecture. Each part of the UI is a component (function or class).

It uses a virtual DOM to efficiently render and update only the parts of the interface that change, instead of reloading the whole page.

**C)Differences Between SPA and MPA**

|  |  |  |
| --- | --- | --- |
| **Feature** | **SPA** | **MPA (Multi-Page Application)** |
| Page Loading | Loads a single HTML page; updates content dynamically | Loads a new HTML page from the server each time |
| Navigation | Seamless, without reloads | Full page reloads on navigation |
| Performance | Fast after initial load; less server requests | Slower experience due to frequent reloads |
| SEO | More challenging (requires extra setup) | Easier, as each page can be indexed separately |
| Development Approach | Frontend-heavy (more JS) | Balanced between frontend and backend |

**D)Pros & Cons of Single-Page Application**

Pros:

Fast and responsive user interface.

Feels like a native app.

Easy to turn into a mobile app (via frameworks like React Native).

Cons:

Initial loading can be slower (large JS bundle).

SEO (Search Engine Optimization) is harder compared to MPAs.

Browser back/forward navigation can be tricky (requires handling).

Potentially more complex state management.

**E)About React**

React is used for building interactive and reusable UI components.

It is declarative (you describe what UI should look like, not how to do it).

React handles updates and rendering efficiently using its virtual DOM.

It supports features like hooks, context, and efficient state management.

**F)Virtual DOM**

The virtual DOM is a lightweight, in-memory representation of the real DOM elements.

When state changes in React, a new virtual DOM tree is created and compared (using a "diffing" algorithm) to the previous tree.

Only the differences are updated in the actual DOM, which makes updates very fast and efficient.

G)Features of React

Declarative syntax: Describe UI as components.

Component-based architecture: Build complex UIs from small, reusable pieces.

Virtual DOM: Efficient rendering and updates.

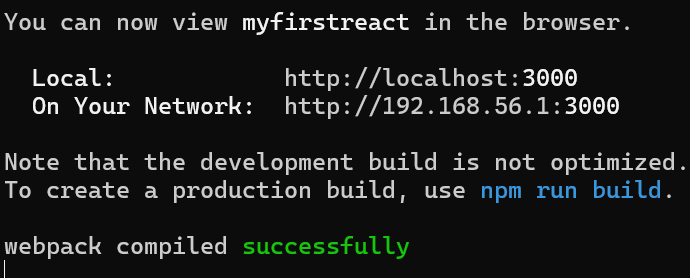
Unidirectional data flow: Data flows down from parent to child components.

JSX: Write UI code using a syntax that mixes JavaScript and HTML.

Ecosystem: Large base of tools, libraries, and community support.

Hooks: Modern feature for handling state and lifecycle in function components.

Cross-platform: Can be used for web, mobile (React Native), and desktop (React Native for Windows/macOS).





**Hands-on 2.**

## React Components Explained

**React components** are the core building blocks of any React application. They are independent, reusable pieces of UI that control their own structure and behavior. Think of them as custom HTML elements that you create and use to compose your interface. Components accept inputs called **props** and return React elements (usually written in JSX) that describe what appears on the screen.

## Differences: React Components vs. JavaScript Functions

|  |  |  |
| --- | --- | --- |
|  | **JavaScript Function** | **React Component** |
| Purpose | Executes logic, calculations, manipulates data | Defines part of the UI and manages rendering |
| Returns | Any JavaScript value | JSX (React elements, which get displayed in UI) |
| Usage | Called anywhere in JS code | Used like custom HTML tags (<MyComponent />) |
| Lifecycle | No lifecycle (runs once per call) | Has lifecycle phases (especially in class components) |
| React Features | Cannot directly use React features (like state or hooks) | Can use state, props, hooks, lifecycle methods |

React components look like functions, but they work in the “React ecosystem”—they control how things appear on screen and interact with user actions, state, and props.

## Types of React Components

React has **two main types** of components, each with its own syntax and recommended use cases:

1. **Function Components**

Declared using a JavaScript function.

Return JSX directly.

Can use React Hooks to manage state and lifecycle.

Preferred for simplicity and modern React development.

## 2. ****Class Components****

Declared using ES6 class syntax (“extends React.Component”).

Must define a render() method that returns JSX.

Able to manage state and have access to lifecycle methods.

Mostly used in older codebases before Hooks were introduced.

## Class Component: Key Features

A **class component**:

Is defined as a JavaScript class that extends React.Component.

Can maintain its own mutable state using this.state.

Can use lifecycle methods—special methods that run at specific points in a component’s “life” (for example, componentDidMount, componentDidUpdate).

Must have a render() method, which returns the JSX to display.

## Function Component: Key Features

A **function component**:

Is a JavaScript function that returns JSX.

Can manage state and side effects using **React Hooks** (e.g., useState, useEffect), introduced in React 16.8.

Has a more concise and readable syntax, making it the default choice for most components today.

## The Component Constructor

The **constructor** is a special method used only in class components.

It is called before the component mounts (is rendered to the DOM).

Most often used to:

Initialize the component’s state (this.state = { ... })

Bind event handlers (so this refers to the right context)

If you use a constructor, you **must** call super(props) as the first statement to give your component access to properties from React.Component.

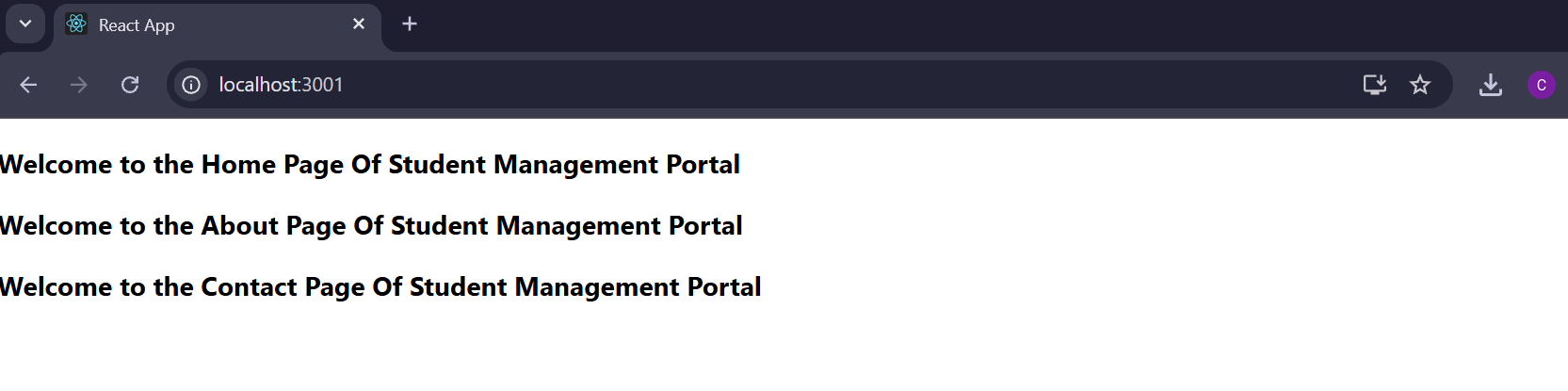
## The render() Function

The render() function is **required** in every class component.

Its purpose is to return the JSX that represents the component’s UI.

It is called automatically whenever the component needs to update (on state or prop changes).

You cannot cause side effects (like HTTP requests) or modify state directly inside render(); only calculate and return what should be displayed.



**Hands-on 3.**



**Hands-on 4.**

## Need and Benefits of Component Lifecycle in React

**Component lifecycle** refers to the series of methods that get called in different stages of a component's existence—right from its creation to its removal from the DOM. React provides this lifecycle to give developers precise control over what happens at each phase and to manage side effects, state, and resource cleanup efficiently.

## Why is the Component Lifecycle Needed?

**Resource Management:** Allows initialization (e.g., network requests, event listeners) when the component appears, and cleanup (like removing listeners) when it disappears.

**DOM Interactions:** Ensures code runs after DOM elements are created, for tasks like measuring or modifying the interface.

**Performance Optimization:** Updates or renders only when needed, avoiding unnecessary re-renders.

**Side Effects Control:** Ideal places to run code that interacts with APIs, timers, or browser storage.

**Consistency:** Maintains the app's expected behavior during updates, mounts, and unmounts.

**Benefits:**

Predictable updates and control over component behavior.

Cleaner code with separation of initialization, updates, and cleanup logic.

Easier debugging of rendering issues and performance bottlenecks.

## Various Lifecycle Hook Methods

In React, lifecycle methods (for class components) or hooks (for function components) correspond to phases:

## For ****Class Components****

**Mounting:** When a component is being inserted into the DOM

constructor()

componentDidMount()

**Updating:** When props or state change

shouldComponentUpdate()

componentDidUpdate(prevProps, prevState)

**Unmounting:** When a component is removed from the DOM

componentWillUnmount()

**Error Handling:**

componentDidCatch(error, info)

static getDerivedStateFromError(error)

For **Function Components** (with Hooks):

useEffect(() => { ... }, []) - Run after mounting (like componentDidMount)

useEffect(() => { ... }) - Run after every render/update (like componentDidUpdate)

Cleanup function inside useEffect simulates componentWillUnmount

useLayoutEffect, useMemo, useCallback for optimized behaviors

## Sequence of Steps in Rendering a Component

Below is the typical **sequence of lifecycle steps** for rendering a class component in React:

## Mounting (Initial Render)

## ****constructor()**** — Setup initial state, bindings.

**static getDerivedStateFromProps()** (rarely used) — Sync state to props (optional).

**render()** — Returns JSX describing the UI.

**componentDidMount()** — Run after first DOM insertion (ideal for side effects, API calls).

## Updating (on props/state change)

**static getDerivedStateFromProps()** — Optional state sync.

**shouldComponentUpdate()** — Determines if re-render is necessary.

**render()** — Updates the JSX.

**getSnapshotBeforeUpdate()** — Capture info (like scroll position) before changes (optional).

**componentDidUpdate()** — Runs after DOM and updates (side effects, fetching new data).

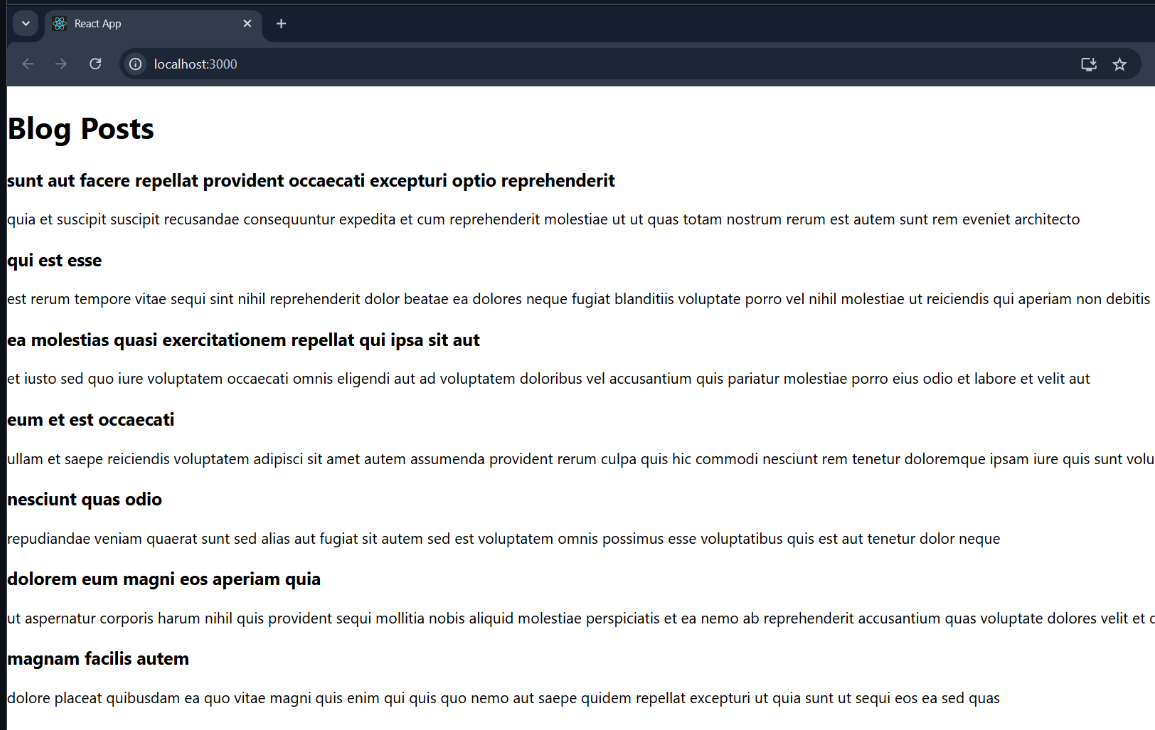
## Unmounting (Component Removal)

**componentWillUnmount()** — Cleanup resources (events, timers, etc.)

<div> <b>Note:</b> With function components and <code>useEffect</code>, you handle mounting, updating, and unmounting within the effect and cleanup function. </div>

## Quick Reference Table

|  |  |  |
| --- | --- | --- |
| **Phase** | **Class Method** | **Function Hook Equivalent** |
| Mount | constructor, componentDidMount | useEffect (with empty deps) |
| Update | shouldComponentUpdate, componentDidUpdate | useEffect (depends on state/props) |
| Unmount | componentWillUnmount | Cleanup function in useEffect |
| Error | componentDidCatch | (useErrorBoundary libraries) |



**Hands-on 5.**

