1. **ReactJS-HOL**

**What Is an SPA (Single-Page Application)?**

A Single-Page Application (SPA) is a web app that loads a single HTML page and dynamically updates content as the user interacts with the app. Instead of loading new pages from the server, an SPA rewrites parts of the current page, offering a smooth, app-like experience.    
**Benefits:**

* Fast user interactions after initial load.
* Seamless, uninterrupted navigation.
* Reduced server load—only relevant data is transferred.
* More fluid, desktop-like user experience.

**React: Definition and How It Works**

React is a popular, open-source JavaScript library for building user interfaces, especially for SPAs. It was developed by Facebook and is known for its component-based structure and use of the virtual DOM.    
**How React Works:**

* You divide your UI into independent, reusable pieces called “components.”
* When data changes, React uses the virtual DOM to update only the necessary parts of the page, leading to efficient and fast UI updates.

**Differences between SPA vs. MPA (Multi-Page Application)**

|  |  |  |
| --- | --- | --- |
| Aspect | SPA | MPA |
| Page Loads | Loads a single HTML file, updates content dynamically | Loads a new page (and HTML) for each action |
| Navigation | Seamless, fast, no full reloads | Each interaction reloads the whole page |
| Best for | Interactive, responsive apps | Large, content-heavy sites |
| Initial Load | Can be slower, but faster after | Faster for first page, but navigation is slower |

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

**Pros:**

* Fast, interactive user experience after first load.
* Bandwidth efficient—only data, not full pages, are reloaded.
* Easier to make dynamic, feature-rich apps.[3](https://www.netsolutions.com/insights/single-page-application/)

**Cons:**

* Initial load can be heavy (downloads most assets at once).
* SEO (search engine optimization) is more complex than MPA.
* If JavaScript is disabled, app may not work correctly.
* More potential for frontend security issues.

**About React**

React enables development of large, interactive web apps using simpler, reusable building blocks.

* Developed by Facebook in 2011, publicly released in 2013.
* Used for web (ReactJS) and mobile (React Native) development.
* Supported by a massive developer community and extensive ecosystem.

 Learn Once, Write Anywhere, Can be used for web, mobile, desktop (React Native)

**What Is the Virtual DOM?**

The virtual DOM is an in-memory, lightweight copy of the actual browser DOM.

* When a component in React changes, the framework updates the virtual DOM rather than updating the real DOM directly.
* React then compares the updated virtual DOM to the previous version (“diffing”) and updates only the necessary parts in the actual DOM.
* This process is efficient and minimizes slow, expensive DOM operations.

**Key Features of React**

* **Component-based architecture:** UI is built from modular, reusable “components.”
* **Virtual DOM:** Increases performance by minimizing direct DOM manipulations.
* **One-way data flow:** Predictable data movement from parent to child components.
* **JSX syntax:** JavaScript XML lets you write HTML-like code within JavaScript for describing UI.
* **Declarative UI:** Easily describe how the UI should look for any application state.

**Hands-On Lab: Setting Up React and Creating First App**

**Prerequisites**

* **Node.js** (includes NPM): Download and install from the [official site].
* **NPM**: Comes bundled with Node.js.
* **Visual Studio Code**: Popular, free code editor.

**Step-by-Step Instructions**

1. **Install Node.js and NPM**
   * Go to the official Node.js site and install the recommended version (which includes NPM).
2. **Create a New React App (“myfirstreact”)**
   * Open a command prompt or terminal.
   * Type and run:

npx create-react-app myfirstreact

1. **Navigate to Your Project**
   * Move into your new project folder:

cd myfirstreact

1. **Open in Visual Studio Code**
   * Run:

code .

(This opens the current folder in VS Code.)

1. **Edit the Main App Component**
   * Open the file: src/App.js.
   * Remove the current code and replace it with:

**function** App() {

**return** (

<div>

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

</div>

);

}

**export** **default** App;

1. **Run the Application**
   * Save all changes.
   * In the terminal, make sure you are in the project folder, then run:

npm start

* + This launches the React development server.

1. **View Your App**
   * Open a browser and visit http://localhost:3000.
   * You should see the heading:  
     “welcome to the first session of React”
2. **ReactJS-HOL**

**1. Explain React Components**

React components are the fundamental building blocks of a React application. A component is a reusable piece of code that determines how a part of your UI should appear and behave. Components can accept inputs (called props) and return JSX elements describing what should be rendered on the screen.

**2. Identify the Differences Between Components and JavaScript Functions**

|  |  |  |
| --- | --- | --- |
| Attribute | JavaScript Functions | React Components |
| Purpose | Perform logic, return values | Render UI, return JSX |
| Output | Numbers, strings, objects, etc. | React elements (JSX) |
| State/Lifecycle | No built-in state or lifecycle | Can have state, lifecycle methods, hooks |
| Reusability | Can be called anywhere | Can be composed into larger UIs |
| Side Effects | Manual (e.g., DOM manipulation) | Often handled via hooks or lifecycle methods |
| Example | function greet(name) { return "Hello " + name; } | function Greeting({name}) { return <h1>Hello {name}</h1>; } |

**3. Identify the Types of Components**

**React components come in two main types:**

* **Class Components**: Defined using ES6 classes, can use lifecycle methods and local state.
* **Function Components**: Defined as JavaScript functions, simpler syntax. With React Hooks, function components can now use state and side effects, making them the modern standard.

4. **Explain Class Component**

A class component is a JavaScript class that extends React.Component. It must include a render() method that returns JSX. Class components can have their own state and lifecycle methods.

**Example:**

import React, { Component } from 'react';

class Welcome extends Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

**5. Explain Function Component**

A function component is a JavaScript function that accepts props as an argument and returns JSX. With React Hooks, even function components can manage state and side effects.

**Example:**

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**6. Define Component Constructor**

The constructor is a special method in a class component that is called when an instance of the component is created. It is used to:

* Initialize internal state via this.state.
* Bind event handler methods (if needed).
* Always call super(props) before using this in the constructor.

**Example:**

constructor(props) {

super(props);

this.state = { count: 0 };

}

**7. Define render() Function**

The render() method is required in every React class component. It returns the (JSX) markup that should appear on the screen. The render() method is called automatically whenever the component’s state or props change.

**Example:**

render() {

return <div>Welcome to React</div>;

}

**Hands-On Lab**

**Prerequisites**

* Node.js and npm installed
* Visual Studio Code (or any code editor)

**Step-by-Step Instructions**

**1. Create a New React App**

Open your terminal and run:

npx create-react-app StudentApp

**2. Create a Components Folder**

Inside the src directory, create a new folder named Components.

3**. Create a Home Component**

Create src/Components/Home.js:

import React from 'react';

function Home() {

return (

<div>

<h2>Welcome to the Home page of Student Management Portal</h2>

</div>

);

}

export default Home;

**4. Create an About Component**

import React from 'react';

function About() {

return (

<div>

<h2>Welcome to the About page of the Student Management Portal</h2>

</div>

);

}

export default About;

**5. Create a Contact Component**

import React from 'react';

function Contact() {

return (

<div>

<h2>Welcome to the Contact page of the Student Management Portal</h2>

</div>

);

}

export default Contact;

**6. Edit App.js to Render All Components**

Edit src/App.js:

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;

**7. Run the Application**

Make sure your terminal is in the StudentApp directory, then run:

npm start

**8. View Your Portal**

Open your browser and go to http://localhost:3000.  
You can see three welcome messages, each from the Home, About, and Contact components, indicating your multi-component React app is working.

**3. ReactJS-HOL**

**What Are React Components?**

React components are the reusable building blocks of a React application. Each component controls a part of the app’s UI and logic, returning elements to display based on input data (called **props**) and internal state. Components enable modular, maintainable, and scalable code structures.

**Differences Between Components and JavaScript Functions**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **JavaScript Functions** | **React Components** |
| Purpose | Run logic, return data | Render UI, return JSX (UI structure) |
| Output | Primitive values, objects, etc. | React elements (JSX markup) |
| State/Lifecycle | No built-in state/lifecycle | Can have state, lifecycle methods/hooks |
| Reusability | Can be reused | Can be reused and composed into others |
| Example | function add(a, b) { return a + b; } | function Hello() { return <h1>Hello</h1>; } |

**Types of Components in React**

* **Function Components**: Defined with JavaScript functions. Can use React Hooks for state and side effects.
* **Class Components**: Defined with ES6 classes, can have local state and lifecycle methods.
* **Pure Components**: Subclass React.PureComponent, optimized for performance if output only depends on props and state.

**Class Component Explained**

A **Class Component** in React is defined using a class that extends React.Component. It can manage local state and use component lifecycle methods (like componentDidMount). Each class component **must** define a render() method that returns JSX.

**Example:**

import React, { Component } from 'react';

class Greeting extends Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

export default Greeting;

**Function Component Explained**

A **Function Component** is declared as a JavaScript function. It receives props as its argument and returns JSX describing the UI. Function components are simpler and, thanks to React Hooks, can use state and other features previously limited to class components.

**Example:**

function Greeting(props) {

return <h1>Hello, {props.name}</h1>;

}

export default Greeting;

**Component Constructor Defined**

The **constructor** is a special method in class components that runs when the component is created. It is used to initialize state and bind methods. It must call super(props) before using this in the component.

**Example:**

**constructor(props) {**

**super(props);**

**this.state = { message: "Welcome!" };**

**}**

**The render() Function Defined**

The **render()** function, required in every class component, returns JSX that tells React what to display in the UI. Whenever state or props change, React automatically calls render() to update the view.

**Example:**

render() {

return <div>{this.state.message}</div>;

}

**Hands-On Lab**

**Prerequisites**

* Node.js installed
* NPM installed
* Visual Studio Code

**Lab Tasks & Step-by-Step Solution**

Create a React App called scorecalculatorapp with a function component named CalculateScore that accepts:

* Name
* School
* Total
* Goal

1. Create the React App

Open VS Code terminal and run:

npx create-react-app scorecalculatorapp

2. Add a Components Folder

Inside src/:

* Create a folder: Components
* Inside it, create a file: CalculateScore.js

3. Code: CalculateScore.js

Paste the following code:

import React from 'react';

import '../Stylesheets/mystyle.css';

function CalculateScore(props) {

const average = props.total / props.goal;

return (

<div className="score-card">

<h2>Student Score Summary</h2>

<p><strong>Name:</strong> {props.name}</p>

<p><strong>School:</strong> {props.school}</p>

<p><strong>Total Marks:</strong> {props.total}</p>

<p><strong>Goal:</strong> {props.goal}</p>

<p><strong>Average Score:</strong> {average.toFixed(2)}</p>

</div>

);

}

export default CalculateScore;

4. Add Styles: mystyle.css

Inside src/, create a new folder: Stylesheets  
Add a file: mystyle.css with the following content:

css

.score-card {

border: 2px solid #4CAF50;

padding: 20px;

margin: 20px auto;

width: 400px;

background-color: #f9f9f9;

border-radius: 10px;

font-family: Arial, sans-serif;

}

.score-card h2 {

color: #4CAF50;

text-align: center;

}

5. Edit App.js

Replace the contents of App.js with:

jsx

import React from 'react';

import './App.css';

import CalculateScore from './Components/CalculateScore';

function App() {

return (

<div className="App">

<CalculateScore

name="Shashwat Dodamani"

school="Dayananda Sagar University"

total={450}

goal={5}

/>

</div>

);

}

export default App;

6. Run the React App

Open terminal and run:

cd scorecalculatorapp

npm start

7. Open in Browser

Navigate to:

arduino

http://localhost:3000

You will see a styled student score summary with calculated average score.

**4. ReactJS-HOL**

**Explain the Need and Benefits of Component Lifecycle**

**Need**

React components go through multiple stages: creation, updating, and removal. Lifecycle methods (also called lifecycle hooks) allow developers to **run specific code at each stage** of a component's life.

**Benefits**

* **Code organization** – Separate logic for mounting, updating, and unmounting
* **Controlled rendering** – Fetch data, apply side effects at the right time
* **Error handling** – Catch errors during rendering
* **Performance optimization** – Skip unnecessary rendering or clean up resources

**Identify Various Lifecycle Hook Methods**

**React class components have three main lifecycle phases:**

**A. Mounting *(When the component is added to the DOM)***

* constructor()
* static getDerivedStateFromProps()
* render()
* componentDidMount()

**B. Updating *(When props or state change)***

* static getDerivedStateFromProps()
* shouldComponentUpdate()
* render()
* getSnapshotBeforeUpdate()
* componentDidUpdate()

**C. Unmounting *(When the component is removed from the DOM)***

* componentWillUnmount()

**D. Error Handling**

* componentDidCatch(error, info)
* static getDerivedStateFromError(error)

**List the Sequence of Steps in Rendering a Component**

**A. Mounting Phase Sequence**

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

**B. Updating Phase Sequence**

(Triggered by setState() or new props)

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

**C. Unmounting Phase**

* componentWillUnmount()

**Hands-On Lab**

**Create a React app called blogapp and implement lifecycle hooks**:

* componentDidMount() to load posts from API
* componentDidCatch() to handle errors gracefully

**Prerequisites**

Ensure you have:

* Node.js and npm installed
* Visual Studio Code

**Step-by-Step Implementation**

**Create the React App**

npx create-react-app blogapp

cd blogapp

code .

**Create Post.js Component (Model)**

class Post {

constructor(userId, id, title, body) {

this.userId = userId;

this.id = id;

this.title = title;

this.body = body;

}

}

export default Post;

**Create Posts.js (Class Component)**

import React, { Component } from 'react';

class Posts extends Component {

constructor(props) {

super(props);

this.state = {

posts: [],

hasError: false

};

}

loadPosts = () => {

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

.then(response => response.json())

.then(data => {

this.setState({ posts: data });

})

.catch(error => {

console.error("Error fetching posts:", error);

this.setState({ hasError: true });

});

};

componentDidMount() {

this.loadPosts();

}

componentDidCatch(error, info) {

alert("Something went wrong: " + error);

this.setState({ hasError: true });

}

render() {

if (this.state.hasError) {

return <h2>Something went wrong while loading posts.</h2>;

}

return (

<div>

<h1>Blog Posts</h1>

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

<div key={post.id}>

<h3>{post.title}</h3>

<p>{post.body}</p>

<hr />

</div>

))}

</div>

);

}

}

export default Posts;

**Update App.js**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

</div>

);

}

export default App;

**Run the App**

npm start

Then open your browser and go to:

<http://localhost:3000>

1. **ReactJS-HOL**

Style the existing CohortDetails React component using:

* CSS Modules
* Inline styles based on the cohort status (e.g., "ongoing" or "completed")

**Step-by-Step Implementation**

**Setup**

* Unzip the provided React app
* Open a terminal and navigate into the app folder
* Run the following commands to install dependencies:

npm install

Open the app in VS Code

code .

**Create a CSS Module**

Inside the src/ folder (or components/ if used), create:

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;

}

**Edit the CohortDetails.js Component**

Import the CSS Module and apply conditional inline styles based on status:

import React from 'react';

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

function CohortDetails({ cohort }) {

const titleStyle = {

color: cohort.status === 'ongoing' ? 'green' : 'blue'

};

return (

<div className={styles.box}>

<h3 style={titleStyle}>{cohort.name}</h3>

<dl>

<dt>Status:</dt>

<dd>{cohort.status}</dd>

<dt>Start Date:</dt>

<dd>{cohort.startDate}</dd>

<dt>End Date:</dt>

<dd>{cohort.endDate}</dd>

</dl>

</div>

);

}

export default CohortDetails;

**Usage Example in App.js**

import React from 'react';

import CohortDetails from './CohortDetails';

function App() {

const cohorts = [

{ name: "React Bootcamp", status: "ongoing", startDate: "2025-06-01", endDate: "2025-08-01" },

{ name: "Node Mastery", status: "completed", startDate: "2025-01-01", endDate: "2025-03-01" }

];

return (

<div className="App">

{cohorts.map((c, idx) => (

<CohortDetails key={idx} cohort={c} />

))}

</div>

);

}

export default App;

**Final Output**

* Each cohort in a styled card with padding, border, and spacing
* Title (h3) is green for **ongoing** cohorts, blue otherwise
* Semantic tags (<dt>, <dd>) with dt bolded per CSS