**1. Define SPA and its benefits**

**SPA (Single Page Application)** is a web application that loads a single HTML page and dynamically updates content as the user interacts with the app, without reloading the entire page.

**Benefits:**

* Faster user experience
* Reduced server load
* Smooth navigation without page refresh
* Better performance after the initial load

**2. Define React and identify its working**

**React** is a JavaScript library developed by Facebook for building user interfaces, especially for SPAs.

**How it works:**

* Uses a **component-based** architecture
* Maintains a **virtual DOM** to efficiently update and render UI
* Responds to state or prop changes and re-renders components dynamically

**3. Identify the differences between SPA and MPA**

| **Feature** | **SPA (Single Page App)** | **MPA (Multi Page App)** |
| --- | --- | --- |
| Page Reloads | No full reloads | Full reload on each page |
| Speed | Fast after first load | Slower due to reloads |
| Navigation | Dynamic, handled by JS | Server-based, static links |
| Development | Complex with routing logic | Simpler structure |
| SEO | More complex | SEO-friendly |

**4. Explain Pros & Cons of Single-Page Application**

**Pros:**

* Seamless user experience
* Quick navigation
* Reduced server requests

**Cons:**

* Poor SEO by default
* Initial load time can be high
* Browser history and analytics tracking is harder

**5. Explain about React**

React is a **declarative**, **component-based** JavaScript library used for building interactive user interfaces. It encourages reusable components and is commonly used to build SPAs with dynamic data updates.

**6. Define virtual DOM**

**Virtual DOM** is a lightweight JavaScript object that is a copy of the real DOM. React uses it to track changes and update only the changed parts in the real DOM efficiently, improving performance.

**7. Explain Features of React**

* **JSX**: JavaScript + HTML syntax extension
* **Components**: Reusable, modular UI blocks
* **Virtual DOM**: Improves performance
* **Unidirectional Data Flow**: Predictable state management
* **Hooks**: Manage state and lifecycle in functional components
* **High Performance**: Due to virtual DOM diffing

**1. Explain React components**

**React components** are the building blocks of a React application. Each component is a self-contained piece of UI that can have its own structure, logic, and styling. Components can be reused and composed to build complex interfaces.

**2. Identify the differences between components and JavaScript functions**

| **Aspect** | **React Components** | **JavaScript Functions** |
| --- | --- | --- |
| Purpose | Build UI with state and props | Perform general tasks/calculations |
| Return value | Returns JSX (UI elements) | Returns data or performs actions |
| Lifecycle Methods | Can use lifecycle methods (class) | No lifecycle methods |
| React Integration | Designed for React rendering | Not specifically for UI |

**3. Identify the types of components**

There are two main types of React components:

* **Class Components**
* **Function Components**

**4. Explain class component**

A **class component** is a React component defined using ES6 classes. It includes a render() method and can manage its own state and lifecycle methods like componentDidMount, componentDidUpdate, etc.

class Welcome extends React.Component {

render() {

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

}

}

**5. Explain function component**

A **function component** is a simpler way to write components using JavaScript functions. It can accept props and, with React Hooks, can also manage state and side effects.

function Welcome(props) {

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

}

**6. Define component constructor**

In class components, the **constructor()** method is used to initialize state and bind methods. It’s called before the component is mounted.

constructor(props) {

super(props);

this.state = { count: 0 };

}

**7. Define render() function**

The **render()** function is a required method in class components. It returns the JSX that defines the component’s UI.

render() {

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

}

**1. Explain React components**

React components are **independent, reusable pieces of UI**. Each component defines how a part of the interface should look and behave. They can accept **props (inputs)** and manage **state (data)** to create dynamic UIs.

**2. Identify the differences between components and JavaScript functions**

| **Feature** | **React Component** | **JavaScript Function** |
| --- | --- | --- |
| Purpose | Builds UI using JSX | Performs logic or calculations |
| Return Type | Returns JSX (UI) | Returns values or performs tasks |
| React Integration | Used in React rendering | Not designed for UI |
| State & Lifecycle | Can use state and lifecycle methods/hooks | No concept of state or lifecycle |

**3. Identify the types of components**

React has **two main types of components**:

* **Class Components**: Use ES6 classes, can use lifecycle methods and state.
* **Function Components**: Use JavaScript functions, now support hooks for state and lifecycle.

**4. Explain class component**

A **class component** is a component defined using the class keyword. It has access to **state**, **props**, and **lifecycle methods**. It must include a render() method that returns JSX.

jsx

CopyEdit

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

render() {

return <h1>Count: {this.state.count}</h1>;

}

}

**5. Explain function component**

A **function component** is a simpler way to create components using a JavaScript function. It can use **React Hooks** (like useState, useEffect) to manage state and side effects.

function MyComponent(props) {

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

return <h1>Count: {count}</h1>;

}

**6. Define component constructor**

The **constructor** is a special method in class components used to:

* Initialize the component's state
* Bind methods to the component instance

It’s called once when the component is created.

constructor(props) {

super(props);

this.state = { name: "React" };

}

**7. Define render() function**

The **render()** function is a required method in class components. It returns the JSX that should be displayed on the screen.

render() {

return <div>Hello, React!</div>;

}

### **1. Explain the need and benefits of component lifecycle**

**Need:**  
React components go through different phases—**mounting, updating, and unmounting**. The lifecycle methods allow developers to run specific code during these phases (e.g., fetching data, cleaning up timers, logging, etc.).

**Benefits:**

* Efficient resource management (e.g., setting and clearing timers)
* Better control over rendering behavior
* Data fetching and DOM updates at the right time
* Cleanup actions before removing a component

**2. Identify various lifecycle hook methods**

In **Class Components**, lifecycle methods are grouped by phases:

* **Mounting (when component is added to the DOM):**
  + constructor()
  + static getDerivedStateFromProps()
  + render()
  + componentDidMount()
* **Updating (when props or state change):**
  + static getDerivedStateFromProps()
  + shouldComponentUpdate()
  + render()
  + getSnapshotBeforeUpdate()
  + componentDidUpdate()
* **Unmounting (when component is removed):**
  + componentWillUnmount()
* **Error Handling:**
  + componentDidCatch()
  + getDerivedStateFromError()

In **Function Components** (using Hooks):

* useEffect() (acts as componentDidMount, componentDidUpdate, componentWillUnmount)
* useLayoutEffect()
* useState(), useContext(), etc.

**3. List the sequence of steps in rendering a component**

**Mounting phase (initial render):**

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

**Updating phase (on prop/state change):**

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

**Unmounting phase:**

1. componentWillUnmount()

**1. Understanding the need for styling React components**

**Why styling is needed:**

* To make the UI visually appealing and user-friendly
* To reflect brand identity and improve usability
* To differentiate and organize components clearly

**In React:**  
Since React is component-based, styling needs to be **modular and scoped** to prevent conflicts and promote reusability.

**Benefits:**

* Enhances UI/UX
* Keeps styles scoped to specific components
* Allows dynamic styling using props or state

**2. Working with CSS Module and Inline Styles**

**CSS Modules:**

* A CSS file where classes and IDs are scoped **locally** to the component
* Prevents naming conflicts
* File extension: .module.css

**Example:**

/\* styles.module.css \*/

.heading {

color: blue;

}

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

function MyComponent() {

return <h1 className={styles.heading}>Hello!</h1>;

}

**Inline Styles:**

* Defined as a **JavaScript object** and applied using the style prop
* Useful for dynamic or quick styles

**Example:**

function MyComponent() {

const headingStyle = {

color: 'red',

fontSize: '24px'

};

return <h1 style={headingStyle}>Hello!</h1>;

}