MODULE: 18 - ReactJS for Full Stack

❖ **THEORY EXERCISE :-**

1. Introduction to React.js

Q-1] What is React.js? How is it different from other JavaScript frameworks and libraries?

- **React.js** is an open-source, component-based JavaScript library developed and maintained by Meta (Facebook) for building interactive and high-performance UIs, particularly for single-page applications (SPAs).

- React focuses **only on the View layer** of the MVC (Model–View–Controller) architecture. This makes React highly flexible, as it can be easily integrated with other libraries or frameworks for state management, routing, and backend communication.

- **Key Characteristics of React:**

  1. **Declarative UI** – Developers describe *what* the UI should look like for a given state, and React updates the DOM automatically.

  2. **Component-based architecture** – UI is broken into independent, reusable components.

  3. **Virtual DOM** – Improves performance by minimizing direct DOM manipulation.

  4. **Unidirectional data flow** – Data flows **from parent to child components**.

- **Differences from Other Frameworks and Libraries:-**

Aspect	**React Js**	**Angular Js**	**Vue Js**
Type	Library	Full Framework	Framework
Architecture	Component-based	MVC	MVVM
Data Binding	One-way	Two-way	Two-way
Learning Curve	Moderate	Steep	Easy
Performance	High (Virtual DOM)	Moderate	High

Q-2]: Explain the core principles of React such as the virtual DOM and component-based architecture.

- **Virtual DOM:**

The **Virtual DOM (VDOM)** is a **lightweight JavaScript object** that is a virtual representation of the real DOM. This drastically improves performance compared to direct DOM manipulation.

Working:

1. React creates a virtual DOM tree.
2. When state or props change, a new virtual DOM is created.
3. React compares the new and old virtual DOM using a process called **diffing.**
4. Only the changed parts are updated in the real DOM using reconciliation.

- **Component-Based Architecture:**

A **component** represents a **self-contained piece of UI** with:

- Its own logic
- Its own state
- Its own rendering

Benefits:

- Reusability
- Maintainability
- Separation of concerns
- Easy testing and debugging

Q-3]: What are the advantages of using React.js in web development?

The following are the advantages of using React js in web-development:-

- High performance due to Virtual DOM.
- Reusable UI components.
- Easy maintenance and scalability.
- Strong ecosystem and community.
- SEO friendliness with server-side rendering.
- Efficient updates with one-way data binding.
- Supports modern JavaScript (ES6+).

2. JSX -JavaScript XML

Q-4] What is JSX in React.js? Why is it used?

- **JSX (JavaScript XML)** is a **syntax extension of JavaScript** that allows developers to write **HTML-like code inside JavaScript**.
- JSX is not mandatory but highly recommended because it:
  - Improves readability
  - Combines logic and UI
  - Helps React create elements efficiently
- JSX is converted to JavaScript using **Babel**.

Q-5] How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?

- **Differences:**

JSX	**JavaScript**
HTML-like syntax	Only JS
Needs transpiling	Direct execution
More readable UI	Logic-focused

- You can write, JavaScript expressions inside JSX using {}.

Example:

<h2>{user.name.toUpperCase()}</h2>

Q-6] Importance of curly braces {} in JSX expressions

- Curly braces allows embedding **any valid JavaScript expression** like:-

Variables, Functions, Conditional expressions, Array mapping

- Without {}, JSX treats content as plain text.

3. Components (Functional & Class Components)

Q-7] What are components? Difference between functional and class components.

In React, components are the building blocks of a React application. They are reusable, self-contained pieces of code that return a React element (which describes how a section of the UI should appear). Components can accept input in the form of "props" (properties) and manage their own state.

❖ **Functional Components**:
- Defined as JavaScript functions that return JSX.
- Can use React Hooks (e.g., **useState**, **useEffect**) to manage state and lifecycle methods.
- Generally simpler and more straightforward to write.
- Encourage the use of hooks for managing state and lifecycle.

❖ **Class Components**:
- Defined using ES6 **class** syntax, extending **React.Component**.
- Must have a **render()** method that returns JSX.
- Have access to lifecycle methods such as **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount**.
- Typically more verbose than functional components.

Q-8| How do you pass data to a component using props?

➢ Props (properties) are **read-only inputs** passed from parent to child components.

Props enable:

- Data sharing
- Component customization
- Reusability

Q-9| Role of render() in class components

- Required method
- Returns JSX
- Called whenever state or props change
- Controls UI rendering

4. Props and State

Q-10] What are props? How are props different from state?

Props:

- Immutable
- Passed from parent to child
- Used for configuration

State:

- Mutable
- Managed internally
- Used for dynamic data

Q-11] Explain the concept of state in React

- State represents **data that can change over time** and affects rendering.
- React automatically re-renders the component when state changes.

Q-12] Why is this.setState() used in class components?

- State updates are asynchronous
- Batches multiple updates
- Ensures UI consistency

5. Handling Events in React

Q-13] How are events handled in React compared to vanilla JavaScript?

- React uses **Synthetic Events**, which:
  - Are wrapper around native events
  - Provide cross-browser compatibility
  - Improve performance

Q-14] Common event handlers in React.js

- onClick
- onChange
- onSubmit
- onMouseOver

Q-15] Why bind event handlers in class components?

Binding ensures:

- Correct this reference
- Access to state and props

6. Conditional Rendering

Q-16] What is conditional rendering?

- Conditional rendering means **displaying UI elements based on conditions** such as:
  - Authentication status
  - Data availability
  - User roles

Q-17] Conditional techniques in JSX

- if/else
- Ternary operator
- Logical &&

7. Lists and Keys

Q-18] Rendering lists and importance of keys

- Lists are rendered using .map() function.
- Keys are Uniquely identify elements, which Improve reconciliation and Prevents unnecessary re-rendering.

Q-19] What happens if keys are not unique?

- Incorrect DOM updates
- Performance degradation
- Warning messages
-

8. Forms in React

Q-20] Handling forms (Controlled Components)

- Controlled components in React are those where the form data is handled by the state of the component. In controlled components, the form elements such as **<input>**, **<textarea>**, and **<select>** derive their values from the component's state. This means that React takes control of the form inputs and allows you to manage the values through the component's state.

- **Key Features of Controlled Components:**

  1. **State Management**: The value of the input is stored in the component's local state. You can update this state through events.

2. **Event Handlers**: An **onChange** event handler is often used to update the state as the user types in the form element.
3. **Single Source of Truth**: The input values are derived from the React state, ensuring that the form fields reflect the current application state.

Q-21] Controlled vs Uncontrolled Components

1. **Controlled Components:-**
   - The component's state controls the input values.
   - The state is updated through event handlers such as **onChange**.
   - Provides a single source of truth, making it easier to control UI updates based on state.
   - Useful for validation, formatting, and enforcing rules on user input.
2. **Uncontrolled Components**:
   - The input's state is managed by the DOM rather than React.
   - You use a **ref** to access the input value after user interaction.
   - Doesn't require state updates on every keystroke, which can be beneficial for performance in certain scenarios.
   - More akin to traditional forms in HTML, where you do not rely on JavaScript for managing input values.

9. Lifecycle Methods

Q-22] Lifecycle methods and phases

1. Mounting
2. Updating
3. Unmounting

Q-23] Important lifecycle methods

- componentDidMount → API calls
- componentDidUpdate → respond to changes
- componentWillUnmount → cleanup

10. Hooks

Q-24] What are hooks? useState and useEffect

- Hooks are special functions in React that allow you to use state and lifecycle features in functional components without writing class components, introduced in React 16.8.
- Hooks allow:
  - State management
  - Side effects
  - Code reuse
- useState is used to **add and manage state in a functional component**.
  Syntax: const [state, setState] = useState(initialValue);
- useEffect is used to **handle side effects** in functional components.

Disadvantage include:

- API calls
- DOM updates
- Timers
- Subscriptions

Q-25] Problems hooks solves

1. Complex lifecycle methods
   - componentDidMount
   - componentDidUpdate
   - componentWillUnmount
2. Code duplication
   - Same logic repeated in multiple lifecycle methods
3. Difficult state sharing
   - Needed HOCs or render props
4. Confusing this keyword
   - Required binding methods
5. Large and complex components
6. Removes need for classes
7. Simplified logic sharing
8. Reduced boilerplate

Q-26] useReducer

Used for:

- Complex state logic
- Predictable state transitions

Q-27] & 28]: useCallback vs useMemo

❖ **useCallback:**

➢ useCallback is a hook that **memoizes a function** so that the **same function reference** is reused between renders unless dependencies change.

➢ Syntax: const memoizedFunction = useCallback(() => {
// logic
}, [dependencies]);

❖ **useMemo:**

➢ useMemo is a hook that **memoizes a computed value** so that **expensive calculations** are not recalculated on every render.

➢ Syntax: const memoizedValue = useMemo(() => {
return expensiveCalculation();
, [dependencies]);

Q-29] useRef

Used for:

- DOM access
- Persisting mutable values

11. Routing (React Router)

Q-30] What is React Router?

➢ React Router is a popular library for routing in React applications. It enables developers to build single-page applications (SPAs) with navigation capabilities, allowing users to move between different views or components without refreshing the entire page.

➢ React Router provides a declarative way to handle routing, making it easy to manage navigation and rendering of components based on the URL.

➢ **Key Features of React Router:**

1. **Declarative Routing**: You can define your application's routing configuration in a straightforward way using JSX syntax.
2. **Dynamic Routing**: React Router supports dynamic routing, allowing routes to be defined based on component state, user input, or other factors.

3. **Nested Routes**: It allows for defining routes that can have their own child routes, facilitating modular architecture and organization of components.
4. **Route Matching**: It provides various methods to match routes dynamically and conditionally render components based on the matched routes.
5. **Browser Compatibility**: React Router can work with various navigation strategies (HTML5 History API, hash-based routing, etc.), making it suitable for different types of applications.
6. **Links**: It provides a component that allows you to navigate between routes declaratively rather than using traditional anchor tags.
7. **Redirects**: The library supports redirection between routes, which is useful for scenarios like route authentication or fallback behavior.
8. **Route Parameters**: You can define parameters in the route paths that can be accessed in your components for dynamic data fetching or display.

Q-31] Components explanation

- Components in React Router:
  - **BrowserRouter**: This component uses the HTML5 history API to keep the UI in sync with the URL.
  - **Routes**: A container for defining multiple routes.
  - **Route**: Specifies a mapping from a URL path to a component.
  - **Link**: Renders a clickable link that navigates to a specified route.
  - **Redirect**: Used to redirect from one route to another.

12. JSON-Server & Firebase

Q-32] RESTful web services

- REST (Representational State Transfer) is an architectural style for building web services that allow communication between Client and Server using HTTP.

Key principles:

- Uses standard HTTP methods: **GET, POST, PUT, DELETE**
- Stateless (each request is independent)
- Uses URLs to access resources
- Data usually exchanged in **JSON** or **XML**

Example:-

- GET /users → Fetch users
- POST /users → Add new user
- PUT / users → Edit users
- DELETE / users → Remove users.

Q-33] JSON-Server

- JSON-Server is a **fake REST API tool** used mainly, for frontend development and testing.

Features:

- Uses a simple db.json file
- Supports CRUD operations
- No backend coding required

Q-34] Fetching data in React

- Fetching data in React is commonly done using the **fetch API ie: fetch()** or **Axios ie: axios()**, usually inside useEffect().

Q-35] Firebase features

- Authentication
- Realtime database
- Cloud functions
- Hosting

Q-36] Error and loading handling.

- Used to improve user experience during API calls.
- Common approaches includes : loading → shows spinner/message and error → displays error message.
- Essential for:
  - User experience
  - Stability
  - Debugging

13. Context API

Q-37] What is Context API?

- Context API is a **React feature** that Manages **global state** without prop drilling.
- **Used for:**
  - Authentication
  - Theme (dark/light)
  - Language settings

Q-38] createContext and useContext

- These are hooks used to implement Context API.
- Used to share data across components.
- **createContext**: Creates a context object
- **useContext**: Consumes the context data.

- **Example:-**
- const UserContext = createContext();
- const value = useContext(UserContext);

14. State Management (Redux & Recoil)

Q-39] Redux explanation

Redux provides:

- Predictable state
- Central store
- Debugging tools

Q-40] Recoil vs Redux

- Recoil:
  - Simpler
  - Atomic state
- Redux:
  - Scalable
  - Enterprise-level
