Module 18 – Reactjs for Full Stack

1) Introduction to React.js

Question 1: What is React.js? How is it different from other JavaScript frameworks and libraries?

React.js is a popular **JavaScript library** developed by **Facebook** for building **user interfaces**, especially **single-page applications** (SPAs). It allows developers to create **reusable UI components** and manage dynamic views efficiently.

Key Features of React:

- **Component-Based Architecture**: UI is broken into small, reusable pieces.
- **Virtual DOM**: Improves performance by minimizing real DOM updates.
- **Declarative Syntax**: You describe *what* the UI should look like, and React takes care of *how* to render it.
- **Unidirectional Data Flow**: Data flows from parent to child components via props.
- **React Hooks**: Enable state and side-effects in functional components.
- **JSX Syntax**: Allows HTML-like code within JavaScript for clearer UI structure.

How React is Different from Other JS Frameworks/Libraries

Feature	**React.js**	**Angular**	**Vue.js**
Type	**Library**	Framework	Framework
Developer	Facebook	Google	Evan You (ex-Google)
Architecture	Component-based	Component + MVC pattern	Component-based
DOM	**Virtual DOM**	Real DOM with change detection	Virtual DOM
Data Binding	**One-way (unidirectional)**	Two-way	Two-way (with option for one-way)
Learning Curve	Moderate	Steep	Easy to Moderate
Size & Flexibility	Small & Flexible	Large & Opinionated	Lightweight & Flexible
Use Case	UI layer of web apps	Full app with routing, services	UI layer or full apps

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

React is built on a few powerful and efficient principles that make it highly scalable and fast for building modern web applications. The **two most important concepts** are:

1. Component-Based Architecture

What It Means:

React breaks the UI into **independent, reusable components—**each responsible for rendering a small, specific part of the user interface.

Example:

You might have components like:

- <Header />
- <Navbar />
- <ProductCard />
- <Footer />

Each component can have its own:

- **HTML structure (via JSX)**
- **Styling**
- **Logic (state, events)**

2. Virtual DOM (Document Object Model)

What It Is:

The **Virtual DOM** is a **lightweight copy** of the real DOM kept in memory. React uses it to **efficiently update the UI**.

Working :

1. When state or props change, React **creates a new virtual DOM**.
2. It **compares** the new virtual DOM to the previous one (**diffing**).
3. React then **calculates the minimal changes** needed.
4. It **updates only the changed parts** in the real DOM (not the entire page).

3. Declarative UI

- You describe **what** the UI should look like based on current data, not how to change it.

jsx

return <h1>{isLoggedIn ? 'Welcome!' : 'Please log in'}</h1>;

4. Unidirectional Data Flow

- Data flows **from parent to child** via **props**.
- Makes the app predictable and easier to debug.

Summary

Principle	**Description**
Component Architecture	UI is divided into reusable, isolated components

Principle	Description
Virtual DOM	Efficient UI updates by comparing a virtual copy to the real DOM
Declarative UI	Focus on **what** the UI should show, not **how** to update it
Unidirectional Data Flow	Predictable state and props flow from top to bottom

Question 3: What are the advantages of using React.js in web development?

React.js offers a powerful, efficient, and developer-friendly approach to building modern web applications. Here are the main benefits:

1. Fast Rendering with Virtual DOM

- React uses a Virtual DOM to update only the parts of the page that change.
- This makes UI updates fast and efficient, even in large, dynamic applications.

2. Component-Based Architecture

- Build UIs using reusable, self-contained components.
- Makes code more modular, easier to maintain, and scalable for large projects.

3. Declarative Syntax

- You define what the UI should look like, and React handles the rendering.
- Leads to clearer, more readable code and fewer bugs.

4. Unidirectional Data Flow

- Data flows in one direction (parent → child), making the app logic predictable and easier to debug.

5. State Management with Hooks

- With React Hooks (useState, useEffect, etc.), managing component state is simpler and cleaner.
- Hooks allow you to write logic without converting to class components.

6. Large Ecosystem & Community

- Rich ecosystem of tools like:
    - React Router for routing
    - Redux or Context API for state management

    - Next.js for server-side rendering
- Huge community support and regular updates.

7. SEO-Friendly (with tools like Next.js)

- React by itself is a client-side library, but when paired with Next.js, it supports server-side rendering, which improves SEO and initial load time.

8. Cross-Platform Development

- With React Native, you can build mobile apps using the same React principles and components.

9. Developer Tools

- React Developer Tools (browser extension) helps with:
    - Inspecting components
    - Monitoring props and state
    - Debugging efficiently

Summary Table

Advantage	Benefit
Virtual DOM	Fast UI updates
Component-Based Structure	Reusability and maintainability
Declarative UI	Simpler code and fewer bugs
Unidirectional Data Flow	Predictable and easier to debug
React Hooks	Manage state in functional components
Large Ecosystem	Tools and libraries for every need
SEO Support (with Next.js)	Better search engine visibility
React Native	Web + mobile development with the same logic
Dev Tools	Efficient debugging and inspection

2) 2. JSX (JavaScript XML)

Question 1: What is JSX in React.js? Why is it used?

JSX stands for JavaScript XML. It is a syntax extension for JavaScript that allows you to write HTML-like code inside JavaScript, which React then transforms into React elements.

Why Use JSX?

JSX makes it easier to:

- Visualize the UI structure
- Write cleaner, more readable code
- Combine markup and logic in the same place

Example:

Without JSX (using React.createElement):

jsx

CopyEdit

```
React.createElement('h1', null, 'Hello, world!');
```

With JSX (simpler and clearer):

jsx

CopyEdit

```
<h1>Hello, world!</h1>
```

This gets compiled into React.createElement() behind the scenes.

Benefits of JSX:

Feature	Why It's Useful
HTML-like Syntax	Makes UI easier to understand
Tighter Integration	Logic + layout live together in one place
Compile-Time Error Checking	JSX helps catch syntax issues early
React Element Creation	Transforms into React.createElement() for performance

Question 2: How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?

JSX and **regular JavaScript** are closely related, but there are a few key differences between them.

1. Syntax:

- **JSX** is an **HTML-like syntax** used within JavaScript code.
- **Regular JavaScript** is more straightforward and does not allow HTML-like tags directly inside it.

Example:

Jsx:

```
const element = <h1>Hello, World!</h1>;
```

javascript:

```
const element = document.createElement('h1');
element.textContent = 'Hello, World!';
```

2. Embedding Expressions in JSX:

In JSX, you can **embed JavaScript expressions** inside {}.

3. Attributes:

- In JSX, **HTML attributes** are written in **camelCase**, while in regular JavaScript they use the **hyphenated style**.

Yes, you can write JavaScript inside JSX using curly braces {}. This allows you to embed expressions, including variables, functions, and even conditional logic, directly within the JSX.

Examples:

1. Variables inside JSX:
2. JavaScript Functions inside JSX:
3. Conditional Rendering inside JSX:
4. Loops inside JSX (using map):

Feature	**JSX**	**Regular JavaScript**
Syntax	HTML-like syntax within JavaScript	Standard JavaScript syntax
Expressions	Can embed JavaScript expressions with {}	JavaScript expressions outside HTML
Attributes	Uses **camelCase** (e.g., className)	Uses **kebab-case** (e.g., class)
Event Handlers	Written like HTML attributes (e.g., onClick={handleClick})	JavaScript functions assigned to event listeners (addEventListener)

Question 3: Discuss the importance of using curly braces {} in JSX expressions.

In JSX, curly braces {} are used to embed JavaScript expressions inside the JSX markup. This is a key feature of JSX that allows you to integrate dynamic content and logic into your UI, making it interactive and flexible.

Why Use Curly Braces {}?

1. Embed JavaScript Expressions:
   Curly braces allow you to insert JavaScript expressions directly within JSX. This is crucial because JSX itself is an HTML-like syntax, but React needs a way to mix dynamic data with the static structure of the UI.
   - JavaScript Expressions include variables, functions, arithmetic operations, ternary operators, and more.

2. Dynamic Content:
   Curly braces allow you to render dynamic content based on the component's state or props.

3. Conditions and Logic in JSX:
   You can use conditional rendering and expressions inside curly braces to control what content to display based on certain conditions.

4. Loops and Mapping:
   Curly braces also allow you to loop through data (using functions like map()) and render a list of elements dynamically.

5. Embedding Function Calls:
   You can also call functions inside the curly braces to return values that will be rendered dynamically.

3) Components (Functional & Class Components)

Que 1. What are components in React? Explain the difference between functional components and class components.

Ans: Components are the building blocks of a React application. They let you split the UI into independent, reusable pieces that can be managed separately. Each component defines a part of the UI and can maintain its own state, accept inputs (called props), and render UI elements.

Types of Components in React

React has two main types of components:

1. **Functional Components**
2. **Class Components**

---

1. Functional Components

- These are **JavaScript functions** that return JSX (a syntax extension for rendering UI).
- **Stateless** in older React, but with **React Hooks** (like useState, useEffect), they can now have state and side effects.
- Preferred for most modern development due to simplicity and better performance.

2. Class Components

- Use **ES6 classes** and extend from React.Component.
- Must use a **constructor** for setting state and this keyword for referencing class members.
- Used more before React Hooks were introduced.
- **Key Differences**

Feature	Functional Components	Class Components
Syntax	JavaScript function	ES6 class
State Management	`useState` (Hooks)	`this.state` and `setState`
Lifecycle Methods	`useEffect` and other hooks	`componentDidMount`, etc.
`this` keyword	Not required	Required
Code Length	Shorter and cleaner	More verbose
Performance	Slightly better	Slightly heavier

Feature	Functional Components	Class Components
Modern Usage	Highly recommended (Hooks)	Older approach

Que 2. How do you pass data to a component using props?

In React, **props** (short for "properties") are used to pass data from a **parent component** to a **child component**. Props are **read-only** and cannot be modified by the child.

Working

1. **Pass props in the parent component**
2. **Access props in the child component using parameters (props or destructuring)**

Example

1. Parent Component (App.js)

```
import React from 'react';

import Greeting from './Greeting';

function App() {

  return (

    <div>

      <Greeting name="Alice" />

      <Greeting name="Bob" />

    </div>

  );

}

export default App;
```

2. Child Component (Greeting.js)

```
import React from 'react';

// Using destructuring

function Greeting({ name }) {

  return <h2>Hello, {name}!</h2>;

}
```

```
export default Greeting;
```

Que 3. What is the role of render() in class components?

In React class components, the render() method is required and plays a central role. It defines what the UI should look like for that component.

Key Points:

- The render() method returns JSX (or null) that tells React what to display on the screen.
- It is called automatically whenever the component's state or props change.
- It should be a pure function—meaning it should not modify the component's state or interact with external systems (like APIs).

4) Props and State

Question 1: What are props in React.js? How are props different from state?

Props (short for "properties") in React are a way to pass data from a parent component to a child component. They are like function arguments in JavaScript and are used to customize or configure components.

Key Features of Props:

- Read-only (immutable inside the child)
- Passed from parent to child
- Can be of any data type (string, number, object, array, function, etc.)
- Accessed in components using props or destructuring

Props vs State in React

Feature	Props	State
Definition	Data passed from parent to child	Internal data managed by the component
Mutability	**Immutable** (read-only)	**Mutable** (can be updated)
Source	Provided by parent component	Defined and updated within the component
Usage	Customize child components	Manage component behavior/data over time
Component Type	Used in **both** functional and class components	Mainly used in components that require interactivity
Example	`<Greeting name="John" />`	`this.setState({ count: 1 })` or `useState(1)`

Question 2: Explain the concept of state in React and how it is used to manage component data.

State in React is a built-in object used to **store dynamic data** within a component. Unlike props, which are passed **from parent to child**, state is **managed within the component itself** and can **change over time**, causing the component to **re-render** and update the UI.

State is used to:

- Track user input

- Handle UI updates (e.g., show/hide elements)
- Store data that changes (like counters, form inputs, API responses)

State in Functional Components (with Hooks)

React uses the useState() hook to manage state in functional components.

State in Class Components

In class components, state is defined as an object and updated with this.setState().

Props vs State (Quick Recap)

Feature	**Props**	**State**
Source	Passed from parent	Defined inside the component
Mutability	Immutable	Mutable (can be updated)
Usage	Pass static or dynamic data	Store and manage dynamic data

Question 3: Why is this.setState() used in class components, and how does it work?

In React class components, this.setState() is used to update the component's state. It's the only correct way to change state and trigger a re-render of the component with the new data.

How It Works

- State should never be modified directly (e.g., this.state.count = 1).
- this.setState() schedules an update to the component's state and tells React to re-render the component.
- React merges the new state with the existing state object