**Karan Patel**

MERN Stack Assignment

| Module 7 – Mernstack – React JS |
| --- |

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

Answer:

| **Feature** | **React.js** | **Angular** | **Vue.js** |
| --- | --- | --- | --- |
| Type | Library | Framework | Framework |
| Developed by | Facebook | Google | Evan You (ex-Google) |
| Data Binding | One-way | Two-way | Two-way |
| DOM Handling | Virtual DOM | Real DOM with change detection | Virtual DOM |
| Learning Curve | Moderate | Steep | Easy to moderate |
| Flexibility | High (library + ecosystem) | Low (all-in-one solution) | Medium |
| Use of Templates | JSX (JavaScript + HTML) | HTML + Angular syntax | HTML + template syntax |

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

Answer:

**1. Virtual DOM**

* A **lightweight copy of the real DOM**.
* When state changes, React creates a new Virtual DOM and **compares it with the previous one** using a **diffing algorithm**.
* React updates only the parts of the actual DOM that changed — this improves **performance and speed**.

#### **2. Component-Based Architecture**

* UIs are built using **components** — small, reusable, and isolated pieces of code.
* Each component manages its own **state** and **props**.
* Types of components:  
  + **Functional components** (with hooks)
  + **Class components** (older approach)

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

Answer:

#### **🔹 1. Reusability with Components**

* Build once, use multiple times.

#### **🔹 2. Improved Performance**

* Thanks to the Virtual DOM.

#### **🔹 3. Unidirectional Data Flow**

* Easier to debug and manage data.

#### **🔹 4. Large Ecosystem & Community**

* Access to tools like Redux, React Router, Next.js.

#### **🔹 5. JSX Syntax**

* Combines JavaScript and HTML for more readable UI code.

#### **🔹 6. Support for Mobile (React Native)**

* Build native mobile apps using React knowledge.

#### **🔹 7. SEO Friendly**

* Can be optimized for server-side rendering (with tools like Next.js).

JavaScript XML

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

Answer:

**JSX (JavaScript XML)** is a **syntax extension** for JavaScript used in React. It looks like HTML and is used to describe the UI structure in a more readable way.

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

Answer:

| **JSX** | **JavaScript** |
| --- | --- |
| Looks like HTML | Uses plain JavaScript syntax |
| Used to define UI | Used for logic and functionality |
| Requires transpilation (via Babel) | Runs directly in browser |

#### **Can you write JavaScript inside JSX?**

**Yes** You can write JavaScript expressions **inside curly braces {}** in JSX.

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

Answer:

In JSX, **curly braces {}** are used to **embed JavaScript expressions** inside the JSX markup.

#### **Why are curly braces important?**

* Allow you to **insert dynamic content** (variables, functions, etc.).
* Help mix **logic and UI** together in a clean way.
* Enable **conditional rendering**, loops, etc.

Components (Functional & Class Components)

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

Answer:

In React, **components** are reusable building blocks used to build the UI. A React app is made up of multiple components, each responsible for rendering a part of the user interface.

#### **Functional Components**

* Written as JavaScript functions.
* Introduced with simpler syntax.
* Can use **React Hooks** (like useState, useEffect) for state and side effects.
* **Recommended** for modern React development.

#### **Class Components**

* Written using ES6 class syntax.
* Must extend React.Component.
* Use this.state for state and this.props for props.
* Use render() method to return JSX.

| **Feature** | **Functional Component** | **Class Component** |
| --- | --- | --- |
| Syntax | JavaScript Function | ES6 Class |
| State Management | Uses Hooks (useState) | Uses this.state |
| Lifecycle Methods | Hooks (useEffect) | Methods like componentDidMount() |
| this keyword | Not required | Required (e.g., this.state) |
| Simpler Syntax | ✅ | ❌ |
| Performance | Slightly better (lighter) | Heavier |

Question 2: How do you pass data to a component using props?

Answer:

**Props** (short for **properties**) are used to pass data from **parent** to **child** components.

// Parent Component

function App() {

return <Greeting name="John" />;

}

// Child Component

function Greeting(props) {

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

}

Question 3: What is the role of render() in class components?

Answer:

In **class components**, the render() method is **required**. It tells React **what to display** on the screen.

Props and State

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

Answer:

**Props (short for "properties")** are **read-only** inputs passed from a **parent component to a child component** in React.

They are used to **configure** or **customize** components and allow data to flow **one way** (from parent to child.

Difference Between Props and State:

| **Feature** | **Props** | **State** |
| --- | --- | --- |
| Mutability | Immutable (read-only) | Mutable (can be changed) |
| Usage | Passed from parent to child | Managed within the component |
| Purpose | To configure child components | To store and manage data |
| Ownership | External (parent controls it) | Internal (component controls) |

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

Answer:

State is an object in a React component that holds dynamic data.

It determines how a component behaves and renders.

When the state changes, the component re-renders automatically.

**import React, { useState } from 'react';**

**function Counter() {**

**const [count, setCount] = useState(0); // state variable**

**return (**

**<div>**

**<p>Count: {count}</p>**

**<button onClick={() => setCount(count + 1)}>Increment</button>**

**</div>**

**);**

**}**

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

Answer:

**this.setState()** is used in **class components** to **update the component’s state**.

* It **does not mutate** the state directly.
* It **schedules a re-render**, merging the new state with the current state.
* It ensures the component and UI stay in sync with the latest data.

**class Counter extends React.Component {**

**constructor() {**

**super();**

**this.state = { count: 0 };**

**}**

**increment = () => {**

**this.setState({ count: this.state.count + 1 }); // updating state**

**}**

**render() {**

**return (**

**<div>**

**<p>Count: {this.state.count}</p>**

**<button onClick={this.increment}>Increment</button>**

**</div>**

**);**

**}**

**}**

Handling Events in React

Question 1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.

Answer:

| **Feature** | **React** | **Vanilla JavaScript** |
| --- | --- | --- |
| Event System | Uses **Synthetic Events** | Uses **Native DOM Events** |
| Syntax | JSX (camelCase event names like onClick) | HTML (lowercase, like onclick) |
| Binding | Uses this.handleClick.bind(this) or arrow functions in class components | Direct or using .bind() |
| Event Delegation | Done internally for performance | Manually handled by developers |

**Synthetic Events in React:**

React wraps native browser events into SyntheticEvent objects to provide cross-browser compatibility.

This synthetic event system is consistent across different browsers and mimics native event behavior.

Question 2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.

Answer:

onClick – For button clicks

onChange – For form input updates

onSubmit – For form submissions

onClick

**function ClickExample() {**

**const handleClick = () => {**

**alert("Button clicked!");**

**};**

**return <button onClick={handleClick}>Click Me</button>;**

**}**

onChange

**function InputExample() {**

**const handleChange = (e) => {**

**console.log("Input value:", e.target.value);**

**};**

**return <input type="text" onChange={handleChange} />;**

**}**

onSubmit

**function FormExample() {**

**const handleSubmit = (e) => {**

**e.preventDefault(); // Prevent page reload**

**alert("Form submitted!");**

**};**

**return (**

**<form onSubmit={handleSubmit}>**

**<input type="text" />**

**<button type="submit">Submit</button>**

**</form>**

**);**

**}**

Question 3: Why do you need to bind event handlers in class components?

Answer:

In **class components**, this is not automatically bound to the component instance. When you pass a method (like this.handleClick) as a callback, this can become undefined unless bound.

**class MyComponent extends React.Component {**

**handleClick() {**

**console.log(this); // undefined**

**}**

**render() {**

**return <button onClick={this.handleClick}>Click</button>;**

**}**

**}**

**Example With Binding (Correct):**

**class MyComponent extends React.Component {**

**constructor() {**

**super();**

**this.handleClick = this.handleClick.bind(this);**

**}**

**handleClick() {**

**console.log(this); // Correct `this`**

**}**

**render() {**

**return <button onClick={this.handleClick}>Click</button>;**

**}**

**}**

**Conditional Rendering**

Question 1: What is Conditional Rendering in React? How can you conditionally render elements in a React component?

Answer:

**Conditional Rendering** in React means showing or hiding parts of the UI based on certain conditions (like if statements in JavaScript).

In React, you can use JavaScript logic (like if, ternary, &&, functions) **inside JSX** to decide which elements to render.

#### **🔸 Common Use-Cases:**

* Show a message if the user is logged in.
* Hide buttons when form is submitted.
* Display different UIs based on user roles (admin, guest, etc.)

Question 2: How if-else, ternary operators, and && (logical AND) are used in JSX for conditional rendering?

Answer:

Using if-else Statement (Outside JSX)

function Welcome(props) {

if (props.isLoggedIn) {

return <h1>Welcome back!</h1>;

} else {

return <h1>Please log in</h1>;

}

}

Using Ternary Operator (Inline in JSX)

function Greeting(props) {

return (

<div>

{props.isLoggedIn ? <p>Hello User</p> : <p>Hello Guest</p>}

</div>

);

}

Using && (Logical AND Operator)

function Message(props) {

return (

<div>

{props.show && <p>This is a secret message!</p>}

</div>

);

}

Lists and Keys

Question 1: How do you render a list of items in React? Why is it important to use keys when rendering lists?

Answer:

const fruits = ['Apple', 'Banana', 'Cherry']

function FruitList() {

return (

<ul>

{fruits.map((fruit, index) => (

<li key={index}>{fruit}</li>

))}

</ul>

);

}

### **Why is it important to use keys?**

* **Keys help React identify which items have changed, are added, or removed.**
* This allows **efficient updating** of the UI by avoiding unnecessary re-renders.
* Without keys, or with incorrect keys, React might **reorder or incorrectly reuse DOM elements**, leading to bugs or performance issues.

Question 2: What are keys in React, and what happens if you do not provide a unique key?

Answer:

**Keys** are **special string attributes** you need to include when creating lists of elements in React. Each key **must be unique** among siblings.

### What happens if you don’t use a unique key?

* React will **struggle to keep track** of which item corresponds to which DOM element.
* It may lead to:  
  + Incorrect item updates
  + Loss of component state (like input fields resetting)
  + Poor performance
  + React warnings in the console

Forms in React

Question 1: How do you handle forms in React? Explain the concept of controlled components.

Answer:

In React, **forms are handled using state** to keep track of input values. This approach is known as **controlled components**.

**import { useState } from 'react';**

**function MyForm() {**

**const [name, setName] = useState('');**

**const handleSubmit = (e) => {**

**e.preventDefault();**

**alert(`Submitted name: ${name}`);**

**};**

**return (**

**<form onSubmit={handleSubmit}>**

**<input**

**type="text"**

**value={name}**

**onChange={(e) => setName(e.target.value)}**

**/>**

**<button type="submit">Submit</button>**

**</form>**

**);**

**}**

**What is a Controlled Component?**

**A controlled component is an input form element (like <input>, <textarea>, etc.) whose value is controlled by React state. The input's value is set by state, and any changes are handled by an onChange event handler that updates the state.**

Question 2: What is the difference between controlled and uncontrolled components in React?

Answer:

| **Feature** | **Controlled Component** | **Uncontrolled Component** |
| --- | --- | --- |
| **Data source** | React state | DOM itself (via refs) |
| **Updates via** | onChange handler that sets state | Direct DOM access (using ref.current.value) |
| **Use case** | When you want full control over form behavior and validation | When minimal interaction or performance matters |
| **Example input binding** | value={stateValue} | No value, just a ref |
| **Form submission** | Handled through state | Access value via ref on submit |

Uncontrolled Component Example:

import { useRef } from 'react';

function MyForm() {

const inputRef = useRef();

const handleSubmit = (e) => {

e.preventDefault();

alert(`Submitted name: ${inputRef.current.value}`);

};

return (

<form onSubmit={handleSubmit}>

<input type="text" ref={inputRef} />

<button type="submit">Submit</button>

</form>

);

}

Lifecycle Methods (Class Components)

Question 1: What are lifecycle methods in React class components? Describe the phases of a component’s lifecycle.

Answer:

#### **What are Lifecycle Methods?**

Lifecycle methods are special methods in **React class components** that are automatically called at different stages of a component's life in the DOM. They allow developers to run custom code at specific points in the **component’s lifecycle**, such as when it is created, updated, or destroyed.

### **Phases of a Component’s Lifecycle**

React class component lifecycle has **three main phases**:

#### **1. Mounting (Component is being created and inserted into the DOM)**

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

#### **2. Updating (Component is re-rendered due to changes in state or props)**

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

#### **3. Unmounting (Component is removed from the DOM)**

* componentWillUnmount()

Question 2: Explain the purpose of componentDidMount(), componentDidUpdate(), and componentWillUnmount()

Answer:

#### **componentDidMount()**

* **When it runs:** After the component is mounted (added to the DOM).
* **Purpose:** Ideal for:

Fetching data from an API.  
Setting up subscriptions or timers.  
DOM manipulations.

#### **componentDidUpdate(prevProps, prevState)**

* **When it runs:** After the component is updated (re-rendered).
* **Purpose:** Used to:  
  + Respond to prop or state changes.
  + Perform side-effects based on changes.

**componentWillUnmount()**

**When it runs:** Just before the component is unmounted and destroyed.

**Purpose:** Used for:

* Cleaning up timers or subscriptions.
* Removing event listeners.