React-js

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

- React.js is a JavaScript library developed by Facebook for building user interfaces (UIs), especially for single-page applications where the UI changes dynamically.
- React focuses on the view layer of an application (the “V” in MVC), meaning it is mainly concerned with rendering the UI efficiently.

  Differences from other frameworks/libraries:
- **Library vs Framework**: React is a library, not a full framework like Angular. This means React is focused on UI, and you often need other libraries (like React Router or Redux) for full application functionality.
- **Declarative**: React allows developers to describe what the UI should look like for a given state, rather than how to update it step by step**.**
- **Component-based**: React uses reusable components to build UIs, unlike jQuery which manipulates the DOM directly.
- **Virtual DOM**: React uses a virtual DOM to optimize rendering and make UI updates faster.

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

1.Virtual DOM

- Instead of updating the real DOM directly (which is slow), React maintains a virtual representation of the DOM in memory.
- When the state changes, React diffs the virtual DOM with the previous version and only updates the parts of the real DOM that changed.
- Benefit: Fast, efficient updates.

2. **Component-based Architecture**:

- React apps are built using components, which are self-contained, reusable pieces of UI.
- Components can be functional (stateless or using hooks) or class-based (with state and lifecycle methods).
- **Benefit:** Easier code maintenance, reusability, and separation of concerns.

3.Unidirectional Data Flow:

- Data flows from parent to child components via props.
- **Benefit:** Predictable state management and easier debugging.

4.JSX (JavaScript XML):

- JSX allows you to write HTML-like code in JavaScript, making UI code readable and easier to write.

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

1. **High Performance:** Thanks to the virtual DOM, updates are fast and efficient.
2. **Reusable Components:** Components can be reused across the app, reducing code duplication.
3. **Easy to Learn:** React focuses only on the view layer, making it simpler than full frameworks.
4. **Strong Community Support**: React has a huge community, lots of libraries, and frequent updates.
5. **SEO Friendly:** React can be rendered on the server-side (with Next.js), improving SEO.
6. **Rich Ecosystem**: Integrates easily with tools for routing (React Router), state management (Redux, Recoil), and testing.
7. **Cross-platform Development:** With React Native, you can build mobile apps using React knowledge.

JSX (JavaScript XML)

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

- JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code inside JavaScript.
- React uses JSX to define the UI structure in a more readable and declarative way.
  Why it is used:

- Makes the code more readable and maintainable because HTML structure and JavaScript logic can coexist.
- Allows React to convert JSX into JavaScript function calls (React.createElement) that build the virtual DOM efficiently.
- Provides a clear visual representation of the UI in the code.

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

Differences from regular JavaScript:

- JSX looks like HTML but is not a string or HTML—it is compiled into JavaScript objects that React uses to create the virtual DOM.
- HTML attributes use camelCase in JSX (e.g., className instead of class, onClick instead of onclick).
- JSX requires closing tags even for self-closing elements (e.g., <img />, <input />).

  Using JavaScript inside JSX:

- Yes, you can write JavaScript expressions inside curly braces {} in JSX.

  Ex.

```
const name = "Sahil";

<h1>Hello,  {name}!</h1

<p>{2 + 3}</p>
```

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

- Curly braces allow you to embed JavaScript expressions inside JSX.
- Without {}, JSX treats content as plain text, not dynamic data.
- Curly braces make JSX dynamic and interactive, letting the UI respond to data changes.
  Ex.

```
const age = 20;

<p>My age is {age}</p

<p>Next year, I will be {age + 1}</p>
```

3.Components (fundamental & class Components)

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

Components in React:

- Components are reusable, self-contained pieces of UI in React.
- They allow you to split the UI into independent parts that can manage their own logic and rendering.
- Every React app is built by combining multiple components.

- **Types of Components:**

  Functional component:-

- A JavaScript function that returns JSX.
- Can use hooks (useState, useEffect) for state.
- Use hooks like useEffect.
- Simpler, easier to read.
- Render no required.

  Class Component:-

- A **JavaScript class** extending React.Component.
- Uses this.state to manage state.
- Uses built in lifecycle methods (componentDidmount , ComponentDidupdate) etc.
- More verbose , older style.
- Render required to return JSX.

Ex.

Functional Component:

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
 }
```

Class Component:

```
class Greeting extends React.Component {
  render() {
return <h1>Hello, {this.props.name}!</h1>;
  }

}
```

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

- Props (properties) are used to pass data from a parent component to a child component.
- Props are read-only inside the child component.

Ex.

```
Function child(props) {
   Return <p>welcome, {props.username} ! </p>;
}
```

```
Function parent() {
  Return <child username="sahil" />;
}
```

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

- In class components, render() is a mandatory method that returns the JSX to display in the UI.
- React calls render() whenever the state or props change, updating the virtual DOM.
- Without render(), the class component cannot display any content.

  Ex.

```
class Welcome extends React.Component {

  render() {

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

  }

}
```

4.props and state

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

props. :

- props are used to pass from a parent component to a child component.
- They are read only a child component cannot modify them.
- Props make component dynamic and reusable.

Difference between props and state:

Props:

- Data passed from parents to child.
- Mutability – read only
- Source – passed by parent
- Usage – used to make component resuable
- Ex. <child name = "xyz">

State:

- Data managed inside the component
- Mutability – mutable
- Source – defile inside the component
- Usage - used to manage dynamic behavio of a component
- Ex . this.state = {count : 0}

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

State:

- State is a JavaScript object that holds dynamic data for a component.
- When the state changes, React re-renders the component to reflect the new data.
- State allows components to remember information and react to user interactions.

Ex.

```
Import { usestate } from “react”;

 Function Counter() {

    Const [count , setcount ] = usestate(0);

 Retuen (

<div>

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

    <button onclick={0} => setcount{count + 1} > increment</button>

</div>

);

}
```

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

• In class components, you cannot modify state directly using this.state.count = 1.

• Instead, you use this.setState() to update state.

How it works:

- this.setState() merges the new state with the existing state.
- It triggers a re-render of the component, so the UI reflects the updated state.

Ex.

```
class Counter extends React.Component {
constructor() {
super();
this.state = { count: 0 };
}
increment = () => {
this.setState({ count: this.state.count + 1 });
}
render() {
return (
<div>
<p>Count: {this.state.count}</p>
<button onclick = {this.increment}>Increment</button>
</div>
);
}
}
```

5. Handling Events in React

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

Event handling in React:

- When you write an event handler in React (for example, onClick={handleClick}), React does not bind this directly to the DOM node in the same way as vanilla JavaScript.

- React attaches a single event listener at the root of the application.

- When an event occurs, React captures it, processes it, and then calls your component's event handler.

What are Synthetic Events?

1.Cross-browser consistency

- Synthetic events normalize browser differences, so properties like event.target, event.type, and event.preventDefault() behave the same in all browsers.

2. Performance optimization

- Because React uses event delegation and one listener at the root, it reduces the number of event listeners in the DOM and improves performance

3. Same interface as native events

- Synthetic events expose the same methods and properties as native DOM events (such as stopPropagation() and preventDefault()), so they feel familiar to JavaScript developers.

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

1. onClick – Triggered when an element is clicked.

```
 function Button() {

 const handleClick = () => alert("Button clicked!");

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

}
```

2. onChange – Triggered when the value of an input changes.

```
function InputField() {

 const handleChange = (event) => console.log(event.target.value);

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

}
```

3. onSubmit – Triggered when a form is submitted.

```
function Form() {

 const handleSubmit = (event) => { event.preventDefault();

 alert("Form submitted!");

 };

return (

</button>

 ); }
```

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

• In class components, this does not automatically refer to the component instance inside methods.

• Without binding, this in event handlers will be undefined or point to the wrong object.

• Binding ensures that this correctly refers to the component instance, so you can access this.state or this.props.

Ex

```
class Button extends React.Component {

 handleClick() {

 console.log(this.state);

}

 render() {

return

Click Me;

 }

 }
```

Alternative: Use arrow functions, which automatically bind this.

```
 handleClick = () => {
```

```
 console.log(this.state.count);

 }
```

6. Conditional Rendering

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

- Conditional Rendering means displaying different UI elements or components based on certain conditions (like state or props).

- In React, instead of manipulating the DOM manually, you use JavaScript logic inside JSX to decide what should be rendered.

Example:

```
 function Greeting({ isLoggedIn }) {

 if (isLoggedIn) {

return

<h1>Welcome Back!</h1>;

} else {

return

<h1>Please sign in.</h1>

; }

}
```

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

a) if-else statement

• Can be used outside JSX to decide what to render.

```
  function Greeting({ isLoggedIn }) {

  let message; if (isLoggedIn) {

  message = <h1>welcome Back;</h1>

  }
```

```
else

{

 message = <h1>Please sign in</h1>;

}

return

{message};

}
```

b) Ternary operator (condition ? true : false)

 • Useful inside JSX for inline conditional rendering

```
 function Greeting({ isLoggedIn }) {

return (

{isLoggedIn ?

<div>

<h1>Welcome Back!</h1>

</div>;

}
```

c) Logical AND (&&) operator

 • Renders an element only if the condition is true.

• If the condition is false, React renders nothing.

```
function Notification({ hasMessages }) {

 return (

<div>{hasMessages && <p>You have new messages!</p>

}

</div>

);

}
```