React JS

Q1. Framework vs Library.

Both libraries and frameworks are reusable code written by someone else.

When you use a library, you are in charge of the flow of the application. You are choosing when and where to call the library. When you use a framework, the framework is in charge of the flow. It provides some places for you to plug in your code, but it calls the code you plugged in as needed. This behaviour is called inversion of control.

Q2. Is React JS a library or framework?

ReactJS is a JavaScript library. Both React and ReactDOM are available over CDN or we can use Create React App tool which sets up the development environment.

The terms opinionated or unopinionated attempt to define the level of freedom a developer has when structuring their code. Example, AngularJS framework is highly opinionated as compared to Vue.js framework.

React JS is unopinionated.

Q3. Why should we use ReactJS. Explain its features.

* Library based approach provides flexibility to developers.
* Reusable components: It splits the user interface into independent, reusable parts that can be processed separately.
* ReactJS comes with two important features: JSX and Virtual DOM
* **High performance:** React updates only those components that have changed, rather than updating all the components at once.
* **Unidirectional data flow:** React follows a unidirectional data flow. This means that when designing a React app, we often nest child components within parent components. And since the data flows in a single direction, it becomes easier to debug errors and know where the problem occurs in an application at the moment.

Q4. Explain JSX syntax of React JS and its features.

JSX stands for JavaScript Extension, it is provided by React. JSX is a combination of HTML and JavaScript. We can embed JavaScript code inside HTML elements.

Example: <h1>Welcome to {name}</h1>

Web browsers cannot read JSX directly. Browsers can only read regular JS objects and JSX is not a regular JavaScript object. For a web browser to read a JSX file, the file needs to be transformed into a regular JavaScript object. Babel compiler transcompiles JSX to JS code.

Our browser cannot understand JS syntax above ES5, it can only render Vanilla JS. Compilers like Babble will convert all the modern JS syntax like JSX into browser understandable syntax.

Q5. Explain Virtual DOM used by ReactJS.

DOM stands for Document Object Model. DOM represents an HTML document with a logical tree structure.

Usually, JavaScript Frameworks updates the whole DOM at once, which makes the web application slow. But react uses Virtual DOM which is a lightweight representation of actual DOM.

React uses two virtual DOMs to render the user interface. One of them is used to store the current state of the objects and the other to store the previous state of the objects. Whenever the virtual DOM gets updated, react compares the two virtual DOMs and gets to know about which virtual DOM objects were updated. After knowing which objects were updated, react renders only those objects inside the real DOM.

Q6. How to create a React app?

First install NodeJS because we need npm, that contains many JavaScript libraries including React and ReactDOM.

Then use the Create-React-App tool: npx create-react-app my-app

Q7. Explain Rendering in React.

There is one simple rule: when you call setState in a component, that component and all of its children are marked as dirty. That means render() method will be called for each of these dirty components. While doing this, React does not even check whether props/state is actually changed, it just marks them as dirty blindly.

React only cares about state not props while deciding on whether to render a component, this is because the state variables are eventually passed as props into the sub-components.

This behaviour of auto re-rendering a child component can be finetuned using the lifecycle method: shouldComponentUpdate(nextProps, nextState).

Q8. What are components in ReactJS?

A component can be understood as one fraction of UI. Using components, we can split the UI into independent, reusable parts that can be processed separately.

We have two types of components in React: Functional and Class component.

Q9. Class component vs Functional component. When do we prefer using class components in React JS over functional components?

In the case of class components, when the state of the component is changed then the render method is called. Whereas, function components render the interface whenever the props are changed.

Although we should prefer using function components most times because React now allows using state with function components since React version 16.8 and later by using the useState hook.

But for the older react versions, we can use class components where we have a dependency on the state of the components.

We can do everything with a functional component that can be done using a class component with one exception.

Error Boundaries are a special type of component made for catching errors in your React app. Instead of the app crashing for the user, you can catch the error and display an error screen for example. And Error Boundaries can only be class components.

Q10. Compare lifecycle for a class and a functional component.

In a class component you access its lifecycle by implementing methods like componentDidMount or componentWillUpdate. But with functional components, all of its lifecycle lives in one single hook, useEffect.

The useEffect hook will first run at the beginning of the life of the component, then will run again depending on the dependency array you provided, and then will run its return function at the end of the lifecycle of the component.

useEffect(() => {

// this will run once at the beginning and then

// every time `someVariable` updates

console.log(‘I’m running!’)

// this will run at the end of the life of the component

return () => console.log(‘end of lifecycle’)

}, [someVariable]);

Q11. Explain lifecycle methods of React JS.

The lifecycle of a react component is divided into 3 phases: Mounting phase, Updating phase and Unmounting phase.

Q5. Explain session management in React JS.

Q6. Explain state management in React JS.

Q7. Explain rendering in React JS.

What are keys in react?

Controlled vs Un-Controlled components.

What is State in a React component?

How does the react rendering work?

React Concepts

Setting up a React Project:

* Create new react project: 'npx create-react-app myapp'
* Adding a dependency to our project: 'npm install react-router-dom'. This command installs the dependency and adds it to package.json file automatically.
* Installing all required dependencies to an existing react project as specified in package.json: 'npm install react-scripts --save'
* Inside the project directory: 'npm start'

React JSX

JavaScript XML is a template used by react, that allows you to write HTML elements within JavaScript and place them in the DOM without any createElement()  and/or appendChild() methods.

const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const myElement = <h1>I Love JSX!</h1>;

With JSX you can write expressions inside curly braces { }. The expression can be a React variable, or property, or any other valid JavaScript expression.

const myElement = <h1>React is {5 + 5} times better with JSX</h1>;

To write HTML on multiple lines, put the HTML inside parentheses. The HTML code must be wrapped in ONE top level element. JSX follows XML rules, and therefore HTML elements must be properly closed.

const myElement = (

<>

<p>I am a paragraph.</p>

<p>I am a paragraph too.</p>

</>

);

The class attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX. Use attribute className instead.

React Hooks

1. useState

2. useEffect: The Effect Hook lets you perform side effects in function components. It accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

By using this Hook, you tell React that your component needs to do something after render. React will remember the function you passed (we’ll refer to it as our “effect”), and call it later after performing the DOM updates.

There are two common kinds of side effects in React components: those that don’t require cleanup, and those that do.

Example 1: Effect without cleanup.

function Example() {

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

useEffect(() => {

document.title = `You clicked ${count} times`;

});

}

Example 2: Effect that requires cleanup.

useEffect(() => {

// this will run once at the beginning and then every time state of component updates.

console.log(‘I’m running!’)

// this will run at the end of the life of the component

return () => console.log(‘end of lifecycle’)

});

This is the optional cleanup mechanism for effects. Every effect may return a function that cleans up after it. This lets us keep the logic for adding and removing subscriptions close to each other.

React performs the cleanup when the component unmounts. However, as we learned earlier, effects run for every render and not just once. This is why React also cleans up effects from the previous render before running the effects next time.

In some cases, cleaning up or applying the effect after every render might create a performance problem.

In class components, we can solve this by writing an extra comparison with prevProps or prevState inside componentDidUpdate.

componentDidUpdate(prevProps, prevState) {

if (prevState.count !== this.state.count) {

document.title = `You clicked ${this.state.count} times`;

}

}

You can tell React to skip applying an effect if certain values haven’t changed between re-renders.

useEffect(() => {

document.title = `You clicked ${count} times`;

}, [count]); // Only re-run the effect if count changes

Note: We can use multiple Effects to separate concerns. This is not something that can be achieved using the lifecycle methods in a class component.

React will apply every effect used by the component, in the order they were specified.