**Exploring the world**

Q: What is Microservice?

Ans>

Microservice - also known as the microservice architecture - is an architectural and organizational approach to software development where software is composed of small independent services like database, server or a UI of the application, that communicate over well-defined APIs. These services are owned by small, self-contained teams. Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features. means we are dividing software into small, well-defined modules enables teams to use functions for multiple purposes.

**Benefits of Microservices:**

* Flexible Scaling
* Easy Deployment
* Technological Freedom
* Reusable Code
* Resilience

Q: What is Monolith architecture?

A: A Monolith architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications. A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming. means we are not dividing software into small, well-defined modules, we use every services like, database, server or a UI of the application, in one Application file.

Q: What is the difference between Monolith and Microservice?

A: With monolithic architectures, all processes are tightly coupled and run as a single service. This means that if one process of the application experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application’s features becomes more complex as the code base grows. This complexity limits experimentation and makes it difficult to implement new ideas. Monolithic architectures add risk for application availability because many dependent and tightly coupled processes increase the impact of a single process failure.

With a microservices architecture, an application is built as independent components that run each application process as a service. These services communicate via a well-defined interface using lightweight APIs. Services are built for business capabilities and each service performs a single function. Because they are independently run, each service can be updated, deployed, and scaled to meet demand for specific functions of an application.

Q: Why do we need a useEffect Hook?

A: useEffect Hook is javascript function provided by react. The useEffect Hook allows you to eliminate side effects in your components. Some examples of side effects are: fetching API data, directly updating the DOM, and setting up subscriptions or timers, etc can be lead to unwarranted side-effects. useEffect accepts two arguments, a callback function and a dependency array. The second argument is optional.

useEffect(() => {}, [])

The () => {} is callback function and [] is called a empty dependency array. If anything that we pass (suppose currentState) inside the [] it trigger the callback function and changes the state of the application.

useEffect(() => {

setCurrentState("true");

}, [currentState])

If we do not pass empty dependency array then the useEffect runs everytime when the UI is rendered.

useEffect(() => {})

**NOTE: There are 2 cases (Props change & State change), when component re-render**

Q: What is Optional Chaining?

A: Optional Chaining (?.) operator accesses an object's property or calls a function. If the object accessed or function called is undefined or null , it returns undefined instead of throwing an error. Optional Chaining (?.) is good way of accessing the object keys, it prevents the application from being crashed if the key that we are trying to access is not present. If the key is not present then instead of a throwing key error, it returns undefined.

Q: What is Shimmer UI?

A: A Shimmer UI resembles the page's actual UI, so users will understand how quickly the web or mobile app will load even before the content has shown up. It gives people an idea of what's about to come and what's happening (while UI currently loading) when a page full of content/data takes more than 3 - 5 seconds to load. Shimmer UI is a great way for loading the applications. Instead of showing a loading circle we can design a shimmer UI for our application that is good for user experience.

Q: What is the difference between JS expression and JS statement?

A: A JS expression returns a value that we use in the application. for example:

1 + 2 // expresses

"foo".toUpperCase() // expresses 'FOO'

console.log(2) // logs '2'

isTrue ? true : false // returns us a true or false value based on isTrue value

A JS statement, does not return a value. for example:

let x; // variable declaration

if () { } // if condition

If we want to use JS expression in JSX, we have to wrap in {/\* expression slot \*/} and if we want to use JS statement in JSX, we have to wrap in {(/\* statement slot \*/)};

Q: What is Conditional Rendering? explain with a code example.

A: Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them. for example:

// Using Ternary operator as a shorthand way or writing an if-else statement

{isLoggedIn ? (return <UserGreeting />) : (return <GuestGreeting />)};

// Using an if…else Statement

{

(if (isLoggedIn) {

return <UserGreeting />;

}else {

return <GuestGreeting />;

})

}

// Using Logical &&

{isLoggedIn && <button>Logout</button>}

Q: What is CORS?

A: Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS defines a way in which a browser and server can interact to determine whether it is safe to allow the cross-origin request.

Q: What is async and await?

A: **Async**: It simply allows us to write promises-based code as if it was synchronous and it checks that we are not breaking the execution thread. It operates asynchronously via the event loop. Async functions will always return a promise. It makes sure that a promise is returned and if it is not returned then JavaScript automatically wraps it in a promise which is resolved with its value.

**Await**: Await function is used to wait for the promise. It could be used within the async block only. It makes the code wait until the promise returns a result. It only makes the async block wait. for example:

// async function getRestaurant to fetch Swiggy API data

async function getRestaurants() {

const data = await fetch(

"Swiggy\_API\_URL"

);

const json = await data.json();

// we get the Swiggy API data in json format

console.log(json);

}

Q: What is the use of const json = await data.json(); in getRestaurants()?

A: The data object, returned by the await fetch(), is a generic placeholder for multiple data formats. so we can extract the JSON object from a fetch response by using await data.json(). data.json() is a method on the data object that lets you extract a JSON object from the data or response. The method returns a promise because we have used await keyword. so data.json() returns a promise resolved to a JSON object.