Namaste React Assignment6:

1. What is a microservice?

Ans: Microservice refers to a small, independent and modular component that performs a specific function in a larger application.

1. What is monolithic architecture?

Ans: In the monolithic architecture all the code will be present inside a single project. All the code means all the functionalities like UI, Database connectivity, Authentication etc all will be maintained inside a single project.

1. What is the difference between microservice and monolith?

Ans: In the monolithic architecture we will have all our code (functionalities) inside a single project and all the functionalities should use the same programming language or tech stack. Here maintenance is difficult because even for a single small change we need to build and deploy all our code and also making changes are difficult.

In the microservice architecture we will separate all our functionalities and make them communicate with each other using APIs when required. We can use different tech stack for each functionality (microservice). Here maintenance is easy because we can build and deploy only the necessary code (microservice) and also making changes are easy.

1. Why do we need a useEffect() hook?

Ans: useEffect() is used to perform the side effects in our component like making an API call. It takes two parameters first one is callback function and second one is dependency array. The callback function will be called immediately after our component is rendered.

1. What is optional chaining?

Ans: Optional chaining is a feature in JavaScript that allows you to access properties of an object or elements of an array without having to check whether the object or array is null or undefined at first. It is represented with **?. operator** and can be used to concisely access the deeply nested properties without having to write a long chain of conditional statements to check for null or undefined values.

Example1:

const user = {

name: “Kalyan”,

age: 24,

address:{

street: “Sirpur”,

city: “Nizamabad”,

state: “Telangana”,

zip: 503230

}

}

* Console.log(user.name) => Kalyan
* Console.log(user.address.street) => Sirpur
* Console.log(user.address.country) => undefined (with no error) -> undefined because country does not exist in address.

But we will get the error when we try to access property of an undefined

* Console.log(user.address.country.countryCode) => TypeError: Cannot read properties of undefined (reading 'countryCode')

And all the code after this line will not be executed, so in order to avoid this type of errors we can use the concept of Optional Chaining, with the help of Optional chaining we will not get any error we will just get undefined and rest of the code will be executed without getting blocked.

Optional chaining:

Console.log(user.address.country?.countryCode) => Undefined.

The above code means if country is exist (not null or not undefined) then only check countryCode else just return undefined without any error.

We are rewriting the above code as:

Console.log(User?.address?.country?.countryCode) => Undefined.

Example2:

Const Users = [

{name: “Kalyan”, age: 25},

{name: “Gupta”, age: 25},

{name: “Kalyan Gupta”, age; 25}

]

* Console.log(Users[1]) => {name: “Gupta”, age: 25}
* Console.log(Users[1].name) => Gupta
* Console.log(Users[3]) => Undefined
* Console.log(Users[3].name) => TypeError: Cannot read properties of undefined (Reading name) and rest of the code will not be executed, so inorder to avoid this type of situations we can use the concept of optional chaining.
* With optional chaining: console.log(Users[3]?.name) => undefined

The above means, only check for name when Users[3] exists (Users[3] is not null or not undefined) else return undefined.

Example 3:

const person = {

name: 'John',

address: {

city: 'New York',

zipcode: '10001',

coordinates: {

latitude: 40.7128,

longitude: -74.0060

}

}

};

// Without optional chaining

const latitude = person.address && person.address.coordinates ? person.address.coordinates.latitude : undefined;

// With optional chaining

const latitudeWithChaining = person.address?.coordinates?.latitude;

console.log(latitude); // Output: 40.7128

console.log(latitudeWithChaining); // Output: 40.7128

Example4:

let obj = {

person:{ name: “kalyan”},

func: () => {console.log(“Function called”)}

}

obj?.func?.()

Limitations of Optional Chaining:

* Optional chaining won’t work on non existing root object:

Example:

obj2?.func?.() => Uncaught ReferenceError: ‘obj2’ is not defined.

In such cases you have to do like this:

typeof obj2 !== ‘undefined’ && obj2?.func?.()

* We cannot modify the property that is not existing, it is not a limitation it is just a correct behaviour.

Obj?.person?.age = 25 => Uncaught syntax error: Invalid left-hand side in assignment

1. What is Shimmer UI?

Ans: The concept of Shimmer UI is used to improve the user experience, whenever the user is accessing a particular page instead of showing “loading or spinning bar” we can simply show him fake page with no data, the fake page structure is similar to the real page structure which the user is going to access. In the fake page we only show the shadow kind of structure without showing any proper data, for example when we open YouTube, YouTube directly will not show videos or loading bar in the UI, but it will show some dummy cards in gray colour just to improve the user experience and when the data is available to show to the user, it will quickly re-render the page with the real data.

1. What is the difference between the JS expression and JS statement?

Ans: JavaScript expression is a piece of code that produces a value. It can be a combination of variables, literals, operators and function calls that results into a single value.

//Arithmetic expression:

Let sum = 5+3

//Function call expression:

Let result = calculate(2,3);

//Conditional (ternary) expression:

Let max = (10>5) ? 10: 5;

JavaScript statement is a complete line of code that performs a specific action and they don’t necessarily produce a value.

//Variable declaration statement:

Let x=10;

//If statement:

If(x>5)

{

Console.log(“X is greater than 5”)

}

//For loop statement:

for(let i=0; i<5; i++)

{

Console.log(i);

}

//Function declaration statement:

Function greet(name)

{

Console.log(`Hello ${name} !`)

}

1. What is conditional rendering? Explain with an example?

Ans: Rendering the component based on the condition is called as conditional rendering.

Example:

Header.js:

Const Header = () => { return <h1>Swiggy Header</h1> }

Export default Header;

AppLayout.js:

Import Header from ‘Header.js’;

Const AppLayout = () =>{

Let data=10;

return(

<div>

{data>5? <Header></Header>: <h1>AppLayout without header</h1>}

</div>

)

}

1. What is CORS?

Ans: CORS stands for Cross Origin Resource Sharing. It is a security feature implemented by web browsers to control how web pages in one domain can request and interact with the resources hosted on the another domain.

The same-origin policy is a security measure that restricts web pages from making requests to a different domain than the one that served the original webpage.

However there are legitimate use cases where a webpage from one domain may need to make requests to another domain, such as when using third party APIs or fetching resources from a different server. CORS allow servers to specify which origins are permitted to access their resources, and browsers enforce these restrictions.

Here's a brief overview of how CORS works:

1. **Request with Origin Header:**
   * When a web page from one domain makes a cross-origin HTTP request (e.g., using JavaScript's XMLHttpRequest or the Fetch API), the browser includes an "Origin" header in the request, indicating the origin of the requesting page.
2. **Server-Side Handling:**
   * The server, in turn, needs to handle the CORS request. It examines the "Origin" header and determines whether to allow or deny the request. This decision is typically based on a server-side configuration.
3. **Response with CORS Headers:**
   * If the server allows the request, it responds with appropriate CORS headers, such as "Access-Control-Allow-Origin," indicating which origins are permitted. Other CORS headers may also be included, specifying allowed methods (e.g., GET, POST), headers, and more.
4. **Browser Enforcement:**
   * The browser checks the CORS headers in the server's response. If the server permits the request, the browser allows the web page to access the requested resource. If not, it blocks the request, and JavaScript on the page cannot access the response.

This mechanism helps prevent cross-origin attacks by giving servers control over which domains are allowed to access their resources. It is an important security feature in modern web development, particularly in the context of Single Page Applications (SPAs) and APIs.

1. What is async and await?

Ans: Async function is a function that returns a promise. The presence of the “Async” keyword before a function declaration indicates that the function will operate asynchronously allowing the use of await inside it.

The await keyword is used inside an async function to pause the execution of the function until the promise to which it is applied is settled (either resolved or rejected). It returns the resolved value of the promise.

Example:

async function fetchData() {

let response = await fetch('https://api.example.com/data');

let data = await response.json();

return data;

}

1. What is the use of “const json = await data.json();” in getRestaurants()

Ans: Both fetch() and .json() methods returns the promise.

Fetch Returns a Promise that resolves to a **Response** object representing the response to the request.

.json() Called on a **Response** object and it returns a Promise that resolves to the JavaScript representation of the JSON data in the response body.

So, when you use **await response.json()**, you're essentially waiting for the Promise returned by **.json()** to be resolved, and then you get the parsed JSON data that you can use in your JavaScript code.