Q: What is a Microservice?
A: In Microservice architecture, we develop flexible, scalable and independently deployable software systems called services. Services are processes that communicate over network to achieve a common goal. Distributing responsibilities to these small services enhances cohesion and decreases coupling thus, making it easier to add a function to the system.

Q: What is Monolith Architecture?
A: In Monotlith Architecture, we develop a unified unit of software program. It has a single codebase which includes UI, APIs, backend, SMS sending system etc. So, when we make a small change then the whole codebase is deployed. This model is self-contained and independent from other applications.

Q: Difference between Monolith and Microservices?
A:

Monolith Architecture	**Microservce Architecture**
Develops a single project which includes UI and backend functionalities.	Develops various small projects for each service. One project for UI and a different for backend.
It deploys whole codebase even when there is a small change in one process. So, deployment is easy.	It deploys services independently, ensuring that only the service to which changes is made gets deployed again, making deployment relatively complex.
Cannot reuse modules of one system to another.	Microservices can be easily used in other software systems.
Constrained with the technologies already used in monolith.	Flexibility to use different tech stacks for different functions.

Q: Why do we need a useEffect hook?
A: useEffect hook lets us perform side-effects in a functional component . Data-fetching, manually updating DOM or setting up a subscription are examples of side-effects. useEffect performs the side-effect after the component has rendered. We can run useEffect based on dependencies that is when something has changed, after initial render and even after everytime render() is called. It also can include a cleanup function which is called right before component is unmounted.So, we can perform React lifecycle methods like componentDidMount(), componentDidUpdate() and componentWillUnmount() using useEffect hook.

Q: What is optional chaining?
A: It accesses an object's property or calls a function. If the object accessed or the function called returns null or undefined then it optional chaining returns undefined instead of throwing an error.
Ex: `let obj={
Name:"blair"
Cat: "Siamese"
}`
Accessing obj.Dog //throws error
But with optional chaining: obj?.Dog //returns undefined

Q: Shimmer UI? A: Basic skeleton of UI page which resembles actual UI page. It is used to improve the user experience by showing users Shimmer effect while the actual content is loading. It gives users idea

about what is expected to come on page and what is happening(page is loading). **This improves the user perceived performance.**

Q: What is difference between JS expression and JS statement?
A: A JS expression evaluates to a value, that is it returns a value but JS statements are lines of code that do not return a value, they are not evaluated.
Ex: JS Expression: a+b, func();
JS Statement: if(a)console.log(a)

Q: What is Conditional Rendering? Explain with code example.
A: Conditional rendering is rendering the elements onto UI based on if statement or conditional operators.
Ex: Showing a login page if user is not logged in else show home page.
`

```
function Body(){
Const [userid,setUserId] = useState(null);

if(userid) return <HomePage />
else return <LoginPage />
```

‘

Q: What is CORS?
A: CORS refers to cross-origin resource sharing. Whenever a user makes a request for a resource on a different origin then it must follow CORS policy. CORS checks whether it is safe to allow cross-origin resource sharing or not. If we try to make a request without enabling CORS then browser will block that request and show us CORS error.
From mdn: *Cross-Origin Resource Sharing 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 also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.*

Q: What is async and await?
A:
Async/Await lets us work with promises in a comfortable manner.
Async: Async keyword is used before a function and it indicates that a function always returns a promise and any non-promise value is automatically wrapped in a resolved promise.
Await: Await keyword is used before a promise and it make javascript wait until the promise gets settled and returns a value. The execution of the function is paused at the promise and resumes when the promise is either fulfilled or rejected. This does not cost CPU any resources because in the meantime javascript can execute other scripts, handle events etc.

Q: What is the use of `const json = await data.json();` in the getRestaurants()?
A: When we make a network request, a promise is returned to us then as soon as server responds with header the promise resolves with an object of the response class i.e, **data**. Now, this object(**data**) includes multiple promise based methods with which we can access the reponse body. data.json() is one such method which parses the data in json format. As soon as the promise data.json() is resolved we get our response body which we are storing in **const json.**