**What is a Microservice? What is Monolith architecture? What is the difference between Monolith and Microservice?**

Imagine a large, delicious cake. Now, slice that cake into smaller, independent pieces, each representing a specific flavour or function. These are microservices!

* Definition: An architectural style where a single application is broken down into a collection of small, independent services. Each service owns its own logic, database, and technology stack. They communicate with each other through APIs.
* Benefits:
  + Scalability: Individual services can be scaled independently based on demand.
  + Agility: Updates and deployments are faster and easier for individual services.
  + Resilience: Failure in one service doesn't bring down the entire application.
  + Flexibility: Different teams can work on different services using different technologies.
* Drawbacks:
  + Complexity: More moving parts mean increased complexity in development and maintenance.
  + Increased operational overhead: Monitoring and managing multiple services requires more effort.
  + Distributed data management: Data consistency across services can be challenging.

Monolith Architecture:

Back to the cake. Now, imagine the cake as a single, solid piece. That's a monolith architecture!

* Definition: A traditional approach where the entire application is built as a single, tightly coupled unit with all functionalities packed into one codebase.
* Benefits:
  + Simplicity: Easier to develop and understand due to its centralized structure.
  + Faster initial development: Getting a basic version up and running is quicker.
  + Lower operational overhead: Fewer moving parts make monitoring and management easier.
* Drawbacks:
  + Scaling challenges: Difficult to scale individual components as the entire application needs to be scaled together.
  + Tight coupling: Changes in one part can impact the entire application, slowing down development.
  + Less resilient: Issues in one module can bring down the entire application.
  + Limited technology choices: The entire application is constrained by the chosen technology stack.

Choosing between Microservices and Monolith:

The best choice depends on your specific needs and project requirements.

* Microservices are preferable for:
  + Large and complex applications.
  + Applications with high user traffic and demand.
  + Need for fast updates and deployments.
  + Teams with diverse technology expertise.
* Monolith is better suited for:
  + Smaller, simpler applications.
  + Applications with lower traffic and complexity.
  + Faster initial development and deployment time.
  + Need for a more centralized and cohesive structure.

**Why do we need a useEffect Hook?**

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 led 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 is passed in the dependency array so upon change in its value it trigger the callback function and useEffect() is called. If we remove the empty dependency array every time the state changes of the component useEffect() will be called. If we want to avoid this, we have to give it an empty dependency array.

**What is** **Optional Chaining?**

Optional Chaining allows to safely access properties of objects or call functions that may potentially be null or undefined. It returns undefined instead of throwing an error. Gracefully handles data that might not be available immediately, such as data fetched from APIs.

**What is Shimmer UI?**

 A Shimmer UI resembles the page's actual UI, so users will understand how the web or mobile app will look even before the actual content has been loaded. It gives people an idea of what's about to come and what's happening (while UI currently loading), 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.

**What is the difference between JS expression and JS statement ?**

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 \*/)};

**#NOTE :-**

*In JSX, we said that we can write JS inside JSX by wrapping it inside “ {} “. However, the actual truth is :- we can only write JS expressions (those piece of JS code that produces a value -> if you type that piece of code in browser console and hit enter, if it produces a value, then it’s an expression, else it’s a statement) and not JS statements.*

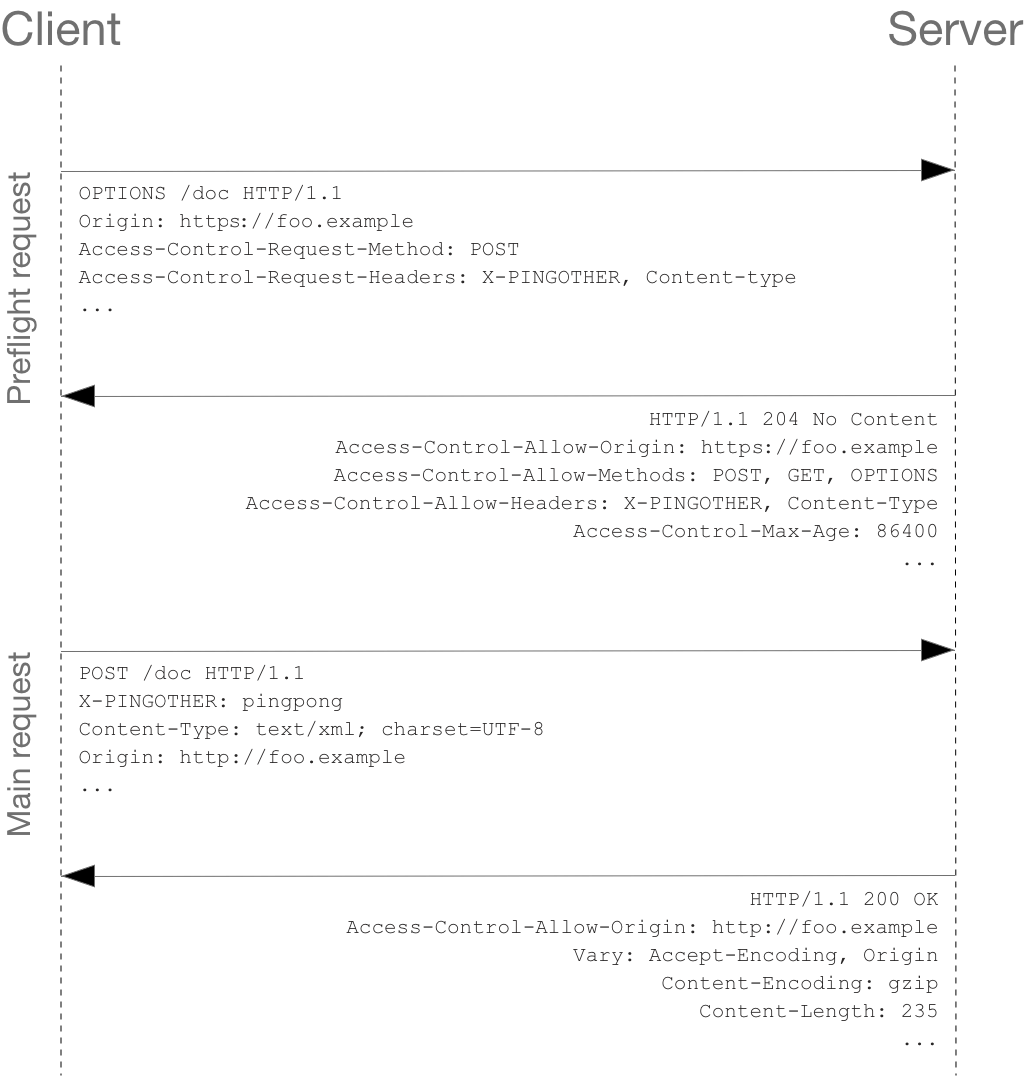
**What is Conditional Rendering, explain with a code example?**

Conditional rendering in React works the same way conditions (if..else or conditional operator -> ? :) work in JavaScript. We often need our components to display different things depending on different conditions.

1. if (data\_not\_fetched\_yet) => load Shimmer UI
2. if (data\_fetched) => load actual UI

**What is CORS?**

Cross origin resource sharing (CORS) is a HTTP header-based mechanism that comes into the play when we want to access the resources from another domain into one domain. Before hitting the actual request, the browser sends a *“Pre-flight”* request to the server in order to check that the server will permit the actual request. In the preflight request the browser sends the header details in it many attributes are there but 2 most important one are 1) *Access-control allow Origin* (i.e. contains domain name) A) 2) *Access-control request method* (i.e. contain GET, POST, UPDATE, DELETE). Now at the server side if the Access-control allow origin attribute contains either the domain name or \* in it and the requested Method is present in the *Access-control allow Method* it sends a +ve confirmation to the browser. After this the browser sends an actual request. In React we can configure package.json to proxy the API requests.



**What is async and await?**

* *Async functions always return a promise*: This means they handle asynchronous operations (like fetching data) without blocking the main thread, ensuring a smooth user experience.
* *Async and await work together for promise handling*: async marks a function as asynchronous, allowing the use of await within it. await pauses execution until a promise resolves, making code more readable and sequential.
* *Await is restricted to async functions and promises*: It can only be used within async functions and must be applied to a promise.
* *Await pauses async functions, managing call stack*: When await encounters a promise, the async function is temporarily suspended and removed from the call stack. It's re-added when the promise resolves, maintaining code execution order.
* *Fetch returns a promise with a response object*: The fetch API returns a promise that resolves with a response object, containing details like status code and response body (often a readable string).

Additional Insights:

* *Error handling*: Consider using try...catch blocks within async functions to gracefully handle potential errors during promise resolution.
* *Chaining promises*: Employ .then() and .catch() methods for chaining multiple promises, creating a sequence of asynchronous actions.
* *Common use cases*: Async/await is frequently used for:
  + Fetching data from APIs
  + Reading/writing files
  + Handling user interactions (e.g., waiting for clicks)
  + Running time-consuming tasks without freezing the UI

**What is the use of `const json = await data.json();` in getRestaurants()**

* const json =: Declares a constant variable named json to store the parsed JSON data.
* await: Pauses the function's execution until the promise returned by data.json() resolves.
* data.json(): A method called on a Response object (usually obtained from a Fetch API request). It:
  + Parses the response body as JSON.
  + Returns a promise that resolves with the parsed JSON data if successful.
  + Rejects with an error if parsing fails.