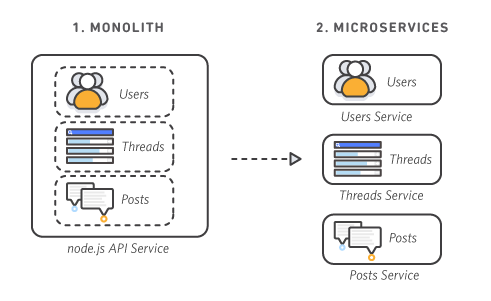
What are MicroServices?

* Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.
* It can be developed, tested, and deployed individually. This also enables development teams to work on different microservices in parallel, leading to faster time-to-market for new features and improvements.
* Each microservice can be implemented using a different technology stack, so the choice of technology can be optimized for the specific requirements of each microservice.
  + Loosely coupled
  + Organized around business capabilities
  + Owned by a small team
  + Highly maintainable and testable
  + Independently
  + Deployable

What is Monolith Architecture?



* A monolithic application is built as a single unified unit while a microservices architecture is a collection of smaller, independently deployable services. Which one is right for you? It depends on a number of factors.
* 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.

What is the difference between Micro Services and Monolith architecture?

| **Sr. No.** | **Key** | **Monolithic architecture** | **Microservices architecture** |
| --- | --- | --- | --- |
| 1 | Basic | Monolithic architecture is built as one large system and is usually one code-base | Microservices architecture is built as small independent module based on business functionality |
| 2 | Scale | It is not easy to scale based on demand | It is easy to scale based on demand. |
| 3 | Database | It has a shared database | Each project and module have its own database |
| 4 | Deployment | Large code base makes IDE slow and build time gets increase. | Each project is independent and small in size. So overall build and development time gets decrease. |
| 5 | Tightly Coupled and Loosely coupled | It extremely difficult to change technology or language or framework because everything is tightly coupled and depend on each other | Easy to change technology or framework because every module and project is independent |

Why do we need useEffect?

* *Hooks* are a new addition in React 16.8.
* Its a function, we call this function by passing another function
* The another function is a call back function
* useEffect(()=>{})
* After every render it will cal the function that we pass inside it.
* useEffect(()=>{

console.log(“render”);

},[]); ->. Called only once after initial render

* 2 params - call back function and dependency array
* Need to import useEffect using named import from react library

Import {useEffect} from “react”;

* Const [searchText,setSearchText]=useState(“”);

useEffect(()=>{

console.log(“render”);

},[searchText]);

Whenever the state searchText changed then call back function is called

-> when you want to do anything on change of the state variable we use effect

-> when you want to execute function only once during initial render, can use useEffect

What is Optional chaining ?

const adventurer = {

name: 'Alice',

cat: {

name: 'Dinah'

}

};

const dogName = adventurer.dog?.name;

console.log(dogName);

// Expected output: undefined

console.log(adventurer.someNonExistentMethod?.());

// Expected output: undefined

The optional chaining (?.) operator accesses an object's property or calls a function. If the object accessed or function called using this operator is [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) or [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null), the expression short circuits and evaluates to [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) instead of throwing an error.

The ?. operator is like the . chaining operator, except that instead of causing an error if a reference is [nullish](https://developer.mozilla.org/en-US/docs/Glossary/Nullish) ([null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null) or [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined)), the expression short-circuits with a return value of undefined. When used with function calls, it returns undefined if the given function does not exist.

What is shimmer UI?

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 (it's currently loading) when a page full of content/data takes more than 3 - 5 seconds to load.

What is the difference between js expressions and js statement ?

Expressions can be assigned or used as operands, while statements can only be declared.

At its core, an expression is a bit of JavaScript code that produces a value.

Eg: var sum = 2+3;

Statements create side effects to be useful, while expressions are values or execute to values

A JavaScript program is a sequence of statements. Each statement is an instruction for the computer to do something.

Eg: let hi = 5;

if (sum > 5) {

console.log("The sum is greater than 5.");

} else {

console.log("The sum is less than or equal to 5.");

}

What is conditional rendering? Explain with code ?

* Its a technique to conditionally render content on a web page based on certain condition.
* In react, you can conditionally render jsx using js syntax like if statement, &&, ? :(ternary operator)

import React from 'react';

function App() {

const isLoggedIn = true;

return (

<div>

{isLoggedIn && <h1>Welcome back!</h1>}

{!isLoggedIn && <h1>Please log in.</h1>}

</div>

);

}

import React from 'react';

function App() {

const isMorning = true;

return (

<div>

{isMorning ? <h1>Good morning!</h1> : <h1>Good afternoon!</h1>}

</div>

);

}

What is CORS?

* CORS defines a way in which a browser and server can interact to determine whether it is safe to allow the cross-origin request. It allows for more freedom and functionality than purely same-origin requests, but is more secure than simply allowing all cross-origin requests.
* CORS stands for Cross-Origin Resource Sharing. It is a security feature implemented by web browsers that restrict web pages or web applications from making requests to a different domain or server than the one from which the original resource was served.
* CORS is a mechanism that allows web pages to make requests to a different domain or server while maintaining the security of the user's data. It allows developers to create more interactive and dynamic web pages by making it possible to share data and resources between different domains.
* CORS can be implemented on the server side by setting appropriate Access-Control headers, and on the client side by using the appropriate XMLHttpRequest or Fetch API methods.

What is async and await ?

* Async/await is a feature in JavaScript that allows you to work with asynchronous code in a synchronous style. This makes it easier to read and write asynchronous code, and makes it less prone to errors that can occur with traditional callback-based asynchronous code.
* The async keyword is used to declare an asynchronous function, which always returns a promise. The await keyword is used to pause the execution of an async function until a promise is resolved or rejected.

async function getData() {

try {

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

const data = await response.json();

console.log(data);

} catch (error) {

console.error(error);

}

}

* In this example, the getData function is declared as an asynchronous function using the async keyword. The function makes a request to an API using the fetch method, which returns a promise. The await keyword is used to pause the function execution until the promise is resolved, and the response data is then parsed using the json method.
* If an error occurs, it is caught in the catch block and logged to the console.
* Async/await makes it easier to work with asynchronous code because it allows you to write code that looks and behaves like synchronous code. Instead of dealing with callbacks or promises, you can simply use the await keyword to pause the function execution until a promise is resolved or rejected. This makes it easier to reason about the code and reduces the potential for errors.

What is the use of ‘const json = await data.json();’ in getRestaurents ?

declared as an asynchronous function using the async keyword. The function makes a request to an API using the fetch method, which returns a promise. The await keyword is used to pause the function execution until the promise is resolved, and the response data is then parsed using the json method.