# Class 6: Assignment

▼ What is a Microservice?

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.

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

▼ What is Monolith architecture?

The **Monolithic** application describes a single-tiered **software** application in which different components are combined into a single program from a single platform.

Components can be:

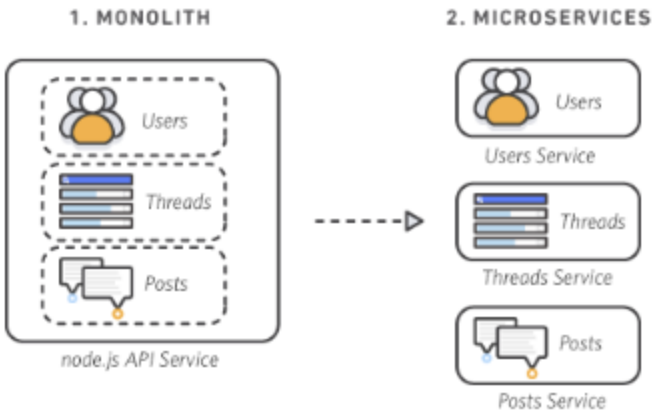
- Authorization — responsible for authorizing a user
- Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).
- Business logic — the application's business logic.
- Database layer — data access objects responsible for accessing the database.
- Application integration — integration with other services (e.g. via messaging or REST API). Or integration with any other Data sources.
- Notification module — responsible for sending email notifications whenever needed.

▼ What is the difference between Monolith and Microservice?

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.




▼ Why do we need a useEffect hook?

- useEffect is a React hook. We use useEffect to connect to an external system.
- Sometimes, your component might need to stay connected to the network, some browser API, or a third-party library, while it is displayed on the page. Such systems aren't controlled by React, so they are called *external.*
- To connect your component to some external system, call `useEffect` at the top level of your component.
- useEffect accepts `two arguments`, a `callback function` and a `dependency array`. The second argument is optional.

```
useEffect(() => {
    //API call here
    getRestaurants();
  }, []);
```

▼ What is Optional Chaining?

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` or `null`, the expression short circuits and evaluates to `undefined` instead of throwing an error.

▼ What is Shimmer UI?

A shimmer screen is a version of the UI that doesn't contain actual content. Instead, it mimics the page's layout by showing its elements in a shape similar to the actual content as it is loading and becoming available (i.e. when network latency allows).

A shimmer screen is essentially a wireframe of the page, with placeholder boxes for text and images.

## Usage

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 expression and JS statement?

Expression is a bit of JavaScript code that produces a value.

For example, these are all expressions:

- `1` → produces `1`
- `"hello"` → produces `"hello"`
- `5 * 10` → produces `50`

- `num > 100` → produces either `true` or `false`
- `isHappy ? "🙂" : "🙁"` → produces an emoji

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

Here are some examples of statements in JavaScript:

```
let hi = 5;
```

```
if (hi > 10) {
  // More statements here
}
```

statements are the rigid structure that holds our program together, while expressions fill in the details.

Statements often have "slots" for expressions. We can put any expression we like into those slots.

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 */)}`

▼ What is Conditional rendering? Explain with a code example

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.

```
{isLoggedIn
        ? <LogoutButton onClick={this.handleLogoutClick} />
        : <LoginButton onClick={this.handleLoginClick} />
      }
```

▼ What is CORS?

- **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 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.

▼ What is async and await?

- The `async function` declaration declares an async function where the `await` keyword is permitted within the function body.
- The `async` and `await` keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.
- The `await` operator is used to wait for a `Promise` and get its fulfillment value. It can only be used inside an async function or at the top level of a module.

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

- The `data` object, returned by the `await fetch()`, is a generic placeholder for multiple data formats.
- `data.json()` is a method of the Response object that allows a JSON object to be extracted from the response. The method returns a promise, so you have to wait for the JSON: `await data.json()`
- `data.json()` returns a promise resolved to a JSON object