What is a Microservice?

The idea is to split your application into a set of smaller, interconnected services instead of building a single monolithic application. Each service handles a specific job, like handling user accounts or managing payments. Inside each service, there's a mini-world of its own, with its own set of rules (business logic) and tools (adapters). Some services talk to each other in different ways, like using REST or messaging. Others might even have their own website.

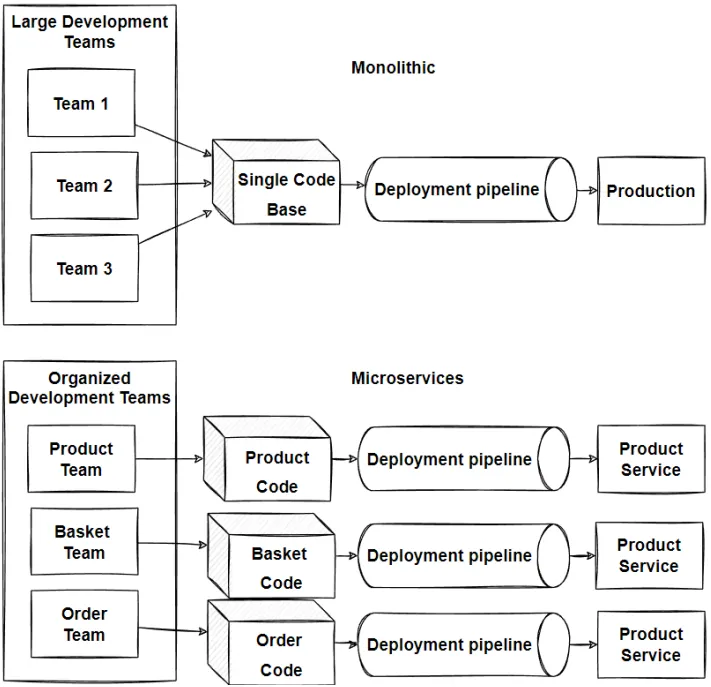
* **Simpler Development**: Microservices break down complex applications into smaller, easier-to-handle services. This makes development faster and maintenance easier.
* **Independent Teams**: Each service can be developed independently by a team focused on that specific task.
* **Flexibility in Technology**: Developers have the freedom to choose the best technologies for each service, without being tied to choices made at the project's start.
* **Continuous Deployment**: Microservices allow for independent deployment, enabling continuous deployment for complex applications.
* **Scalability**: Each service can be scaled independently, ensuring efficient resource usage.
* **Separation of Concerns**: With each task having its own project, the architecture stays organized and manageable.
* **Single Responsibility**: Every service has its own job, following the principle of single responsibility. This ensures focused and efficient development.

What is Monolith architecture?

In the past, we used to build large projects where everything was bundled together. Imagine building an entire application where all the code—APIs, user interface, database connections, authentication, even notification services— resides in one massive project with single code base.

* **Size and Complexity Limitation**: Monolithic applications become too large and complex to understand.
* **Slow Startup**: The application's size can slow down startup time.
* **Full Deployment Required**: Every update requires redeploying the entire application.
* **Limited Change Understanding**: It's hard to grasp the full impact of changes, leading to extensive manual testing.
* **Difficult Continuous Deployment**: Implementing continuous deployment is challenging.
* **Scaling Challenges**: Different modules may have conflicting resource needs, making scaling difficulty.
* **Reliability Concerns**: Bugs in any module can crash the whole application, affecting availability.
* **Adoption of New Technologies**: Making changes in frameworks or languages is expensive and time-consuming since it affects the entire application.

What is the difference between Monolith and Microservice?



What is Shimmer UI?

Shimmer UI is a technique that shows placeholder content while data is loading, reducing wait time and keeping users engaged. Instead of displaying a generic "loading" message, we'll integrate a **<shimmer/>**component within our app to provide visual feedback while data is loading. This concept is known as **‘conditional rendering’**.

What is CORS?

(Cross-Origin Resource Sharing) is a system, consisting of transmitting HTTP headers, that determines whether browsers block frontend JavaScript code from accessing responses for cross-origin requests.  
In simpler terms, CORS (Cross-Origin Resource Sharing) is a security feature implemented by browsers that restricts web pages from making requests to a different origin (domain) than the one from which it was served. Therefore, when trying to call Swiggy's API from localhost, the browser blocks the request due to CORS restrictions.