##### Handle scenario when downstream application is down very frequently

Handling scenarios where a **downstream application** is frequently unavailable is crucial to maintain the resilience and reliability of your system. Here are some strategies you can adopt, categorized into **short-term** and **long-term solutions**, along with **technical design patterns** to make your system more fault-tolerant.

**1. Short-Term Solutions (Immediate Handling)**

These approaches ensure that your application can respond gracefully to intermittent failures:

**1.1. Retry Mechanism**

* Implement a retry policy with **exponential backoff** to avoid hammering the downstream service when it’s down.
* Example in Java using Spring:

java

Copy code

@Retryable(

value = { HttpServerErrorException.class },

maxAttempts = 5,

backoff = @Backoff(delay = 1000, multiplier = 2))

public String callService() {

// Logic to call the downstream service

}

**Tip:** Use a **circuit breaker** with retries to avoid excessive load on the downstream service.

**1.2. Circuit Breaker Pattern**

* The **circuit breaker** acts like a switch. If the downstream service keeps failing, the circuit remains open to stop further calls temporarily.
* Example: Use **Resilience4j** with Spring Boot:

java

Copy code

@CircuitBreaker(name = "downstreamService", fallbackMethod = "fallbackResponse")

public String callDownstreamService() {

// Call downstream service

}

public String fallbackResponse(Throwable t) {

return "Fallback response as the service is down!";

}

**When to use:** If failures happen frequently, the circuit breaker reduces load on the failing service by preventing redundant requests.

**1.3. Fallback Mechanism**

* Provide an **alternative response** when the downstream service is down (e.g., cached data or mock data).
* Example:
  + If you are fetching user details from a service, fall back to a **local cache** of recently fetched data.

**1.4. Asynchronous Calls with Queuing**

* Use **message queues** (like Kafka or RabbitMQ) to **decouple** your application from the downstream service.
* If the downstream is unavailable, store the requests in the queue and process them later when the service is back up.