**Circuit Breaker Pattern**

The **Circuit Breaker pattern** is essential in microservices architecture to prevent cascading failures. When a service fails repeatedly due to network issues, unavailability, or other reasons, the Circuit Breaker opens to prevent additional requests from hitting the failing service. This approach helps maintain the system's overall health and provides fallback mechanisms for graceful degradation.

When a failing service is hit in a Spring Boot application, the outcome depends on how the application is configured to handle such failures. Without specific error handling, the application will likely return an error response to the client. This response typically includes an HTTP status code indicating the type of error (e.g., 500 Internal Server Error) and potentially an error message or stack trace.

However, Spring Boot provides several mechanisms for more graceful error handling:

* **Global Exception Handling:**

Using @ControllerAdvice and @ExceptionHandler, you can define centralized exception handling logic that applies to all controllers. This allows you to catch specific exceptions thrown by the failing service and return custom error responses, log the error, or perform other actions.

* **Retry Mechanism:**

Spring Retry allows you to automatically retry failed requests to a service. This can be useful for handling transient errors, such as network glitches or temporary service outages.

* **Circuit Breaker:**

Libraries like Resilience4j implement the circuit breaker pattern, which prevents an application from repeatedly calling a failing service. When the service fails repeatedly, the circuit breaker "opens," blocking further requests until the service recovers. Fallback methods can be configured to provide a default response when the circuit is open.

If the failing service is an external dependency, it's crucial to implement proper error handling to prevent cascading failures and maintain the stability of your application.

**Circuit Breaker with Resilience4J**

The Circuit Breaker pattern is key to building resilient microservices. It acts as a fail-safe mechanism that prevents repeated failures in service calls by "breaking the circuit" and stopping the flow of requests when a service becomes unresponsive or fails consistently. This approach allows the system to fail fast and recover gracefully.

**How the Circuit Breaker Works**

* **Closed State:** The circuit is closed and allows requests to pass through. If failures occur and reach the threshold, the circuit opens.
* **Open State:** No requests can pass through. After a timeout, the circuit transitions to the half-open state.
* **Half-Open State:** A restricted number of requests are allowed to pass through. If they succeed, the circuit returns to the closed state; if they fail, it returns to the open state.

Resilience4J's Circuit Breaker monitors calls to the downstream service, tracks failure rates, and trips the circuit if the failure rate crosses the defined threshold.

**Why Use the Circuit Breaker Pattern?**

* **Prevents Cascading Failures:** Isolates failures to prevent a chain reaction that could bring down multiple services.
* **Graceful Degradation:** Provides fallback responses to ensure a better user experience when failures occur.
* **Improves System Stability:** Prevents continuous attempts to execute failing operations, thus improving overall stability.

**How Resilience4J Implements the Circuit Breaker**

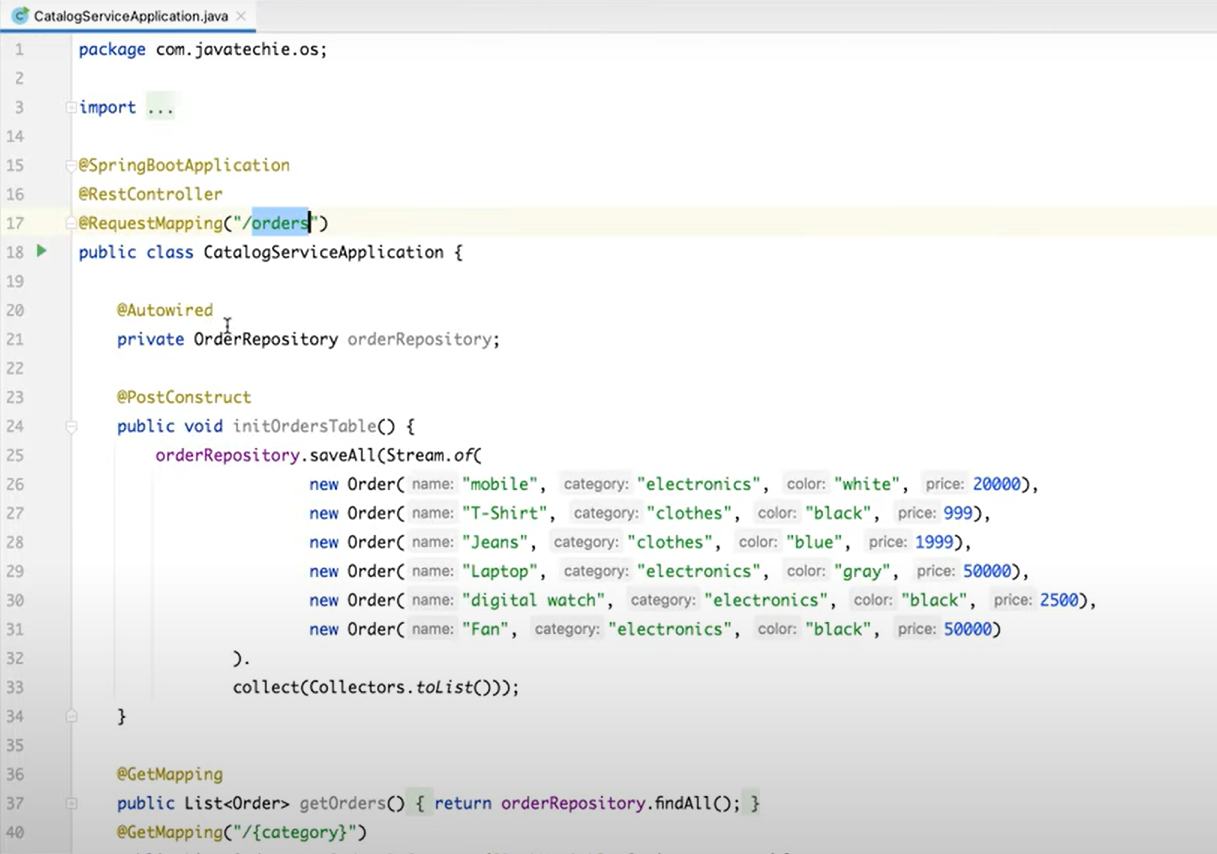
Resilience4J provides a comprehensive and flexible way to implement the Circuit Breaker pattern in Spring Boot applications. It monitors downstream service calls and automatically trips the circuit if the failure rate exceeds the defined threshold. By using annotations like @CircuitBreaker and defining properties in the application.properties or application.yml file, developers can control the behavior of the Circuit Breaker.

**Configuration Parameters in Resilience4J**

* **failureRateThreshold:** Sets the failure rate threshold percentage. When the failure rate meets or exceeds this percentage, the circuit transitions to the open state.
* **slidingWindowSize:** Defines the size of the sliding window used for calculating the failure rate. It can be count-based or time-based.
* **minimumNumberOfCalls:** Specifies the minimum number of calls needed before the Circuit Breaker begins calculating the failure rate. The Circuit Breaker will not open until this number is reached.
* **waitDurationInOpenState:** Duration the Circuit Breaker stays in the open state before transitioning to the half-open state.
* **permittedNumberOfCallsInHalfOpenState:** Number of test calls allowed in the half-open state to determine if the Circuit Breaker should close again.

Example:

CatalogService

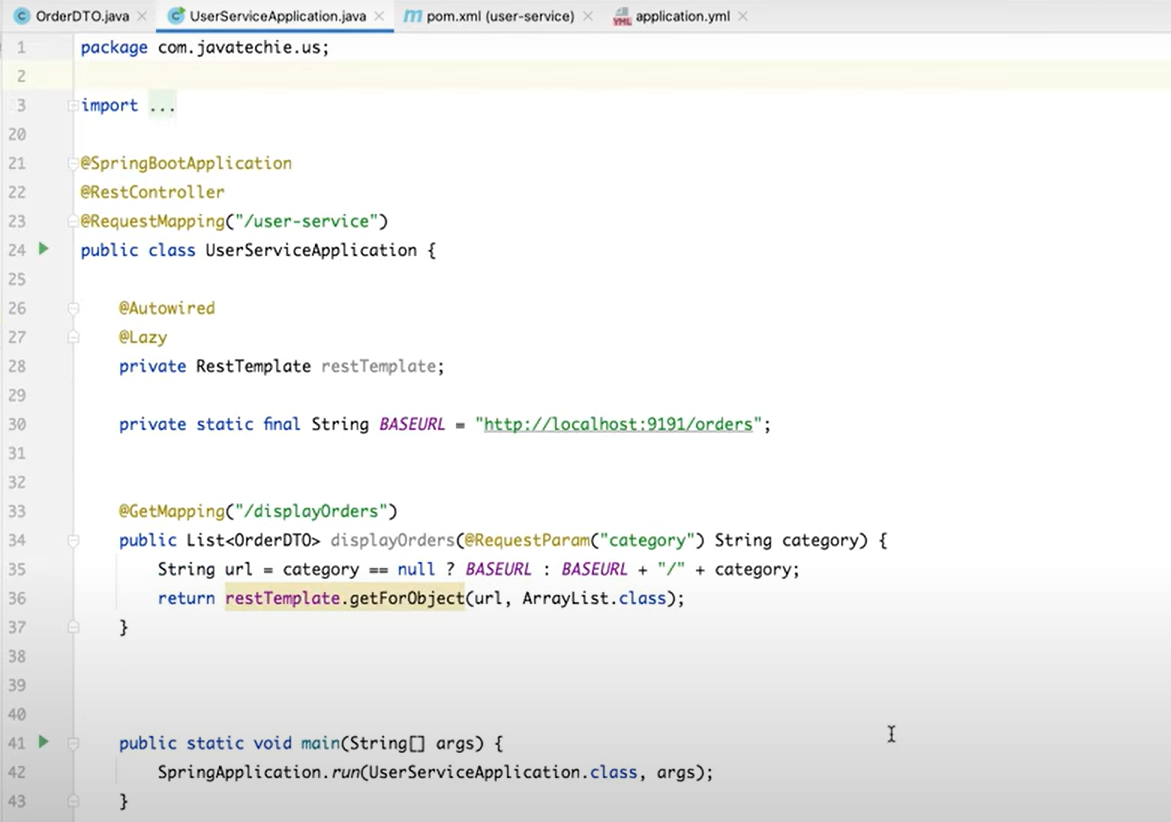


Repository



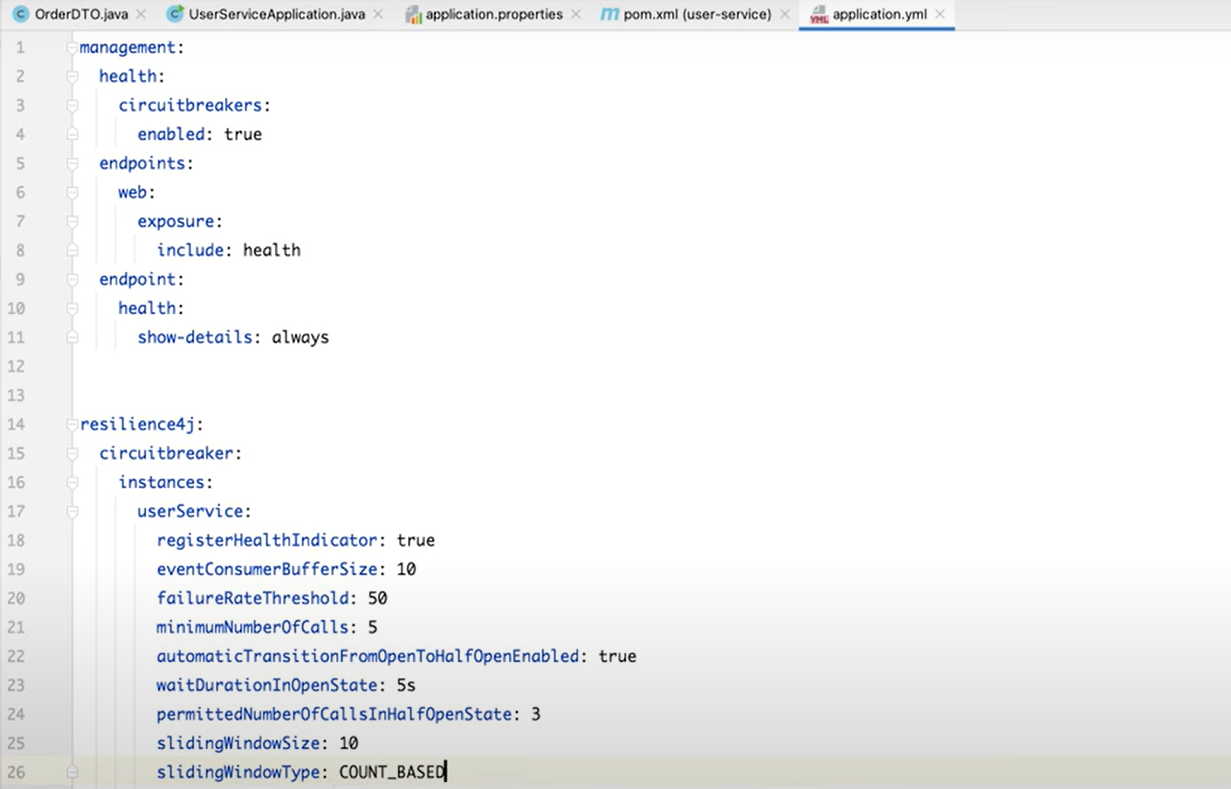


UserService

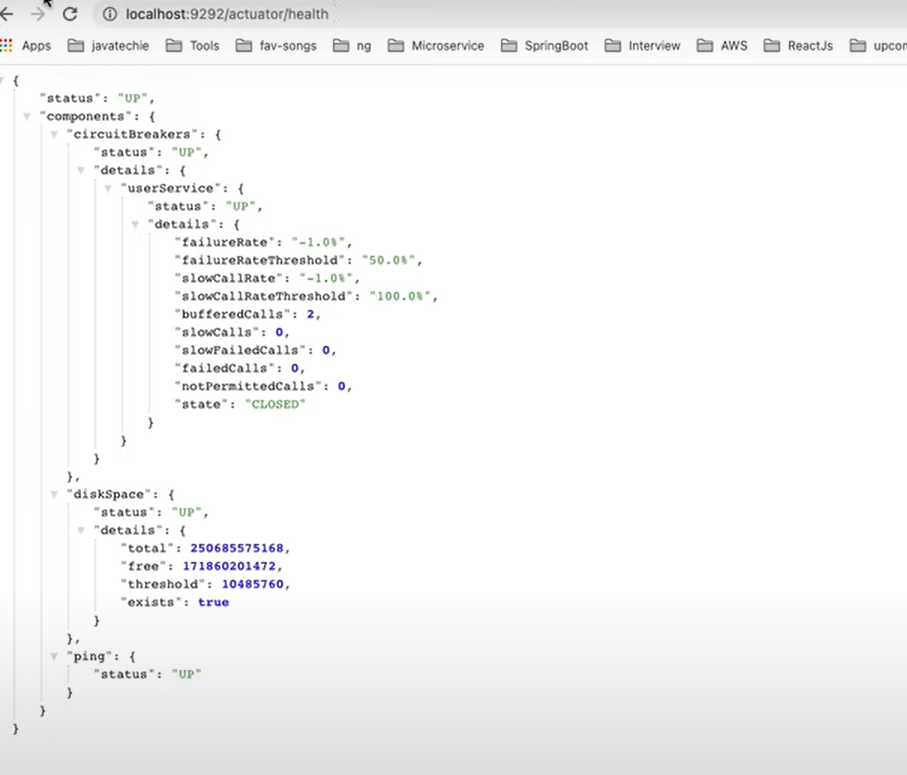


Pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.6.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.javatechie</groupId>  
 <artifactId>catalog-service</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>order-service</name>  
 <description>order microservice</description>  
 <properties>  
 <java.version>1.8</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <excludes>  
 <exclude>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </exclude>  
 </excludes>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>



When both the services are running



When CatalogService is closed



When the failedcalls exceed 50% of threshold then it will show open state:



