In a Microservice Architecture there is a complex call chain



What would happen if 1 of the service is down or very very slow ?

Lets say Microservice4 is down.

* There would be an impact on entire chain

In this case if 4 is down 3 , 2 will also get down .

Even if slow then there will be build call

So the requirement is can we return a fallback response if a service is down ?

Can we implement a circuit breaker pattern to reduce the load ?

Can I return a default response back without hitting the microservice ?

Can we retry a request in case of temporary failures?

Can we implement rate limiting? Only specific number of call to specific service for specific period of time

Solution :

Resilience4J

Resilience4j is a lightweight fault tolerance library designed for functional programming. Resilience4j provides higher-order functions (decorators) to enhance any functional interface, lambda expression or method reference with a Circuit Breaker, Rate Limiter, Retry or Bulkhead. You can stack more than one decorator on any functional interface, lambda expression or method reference. The advantage is that you have the choice to select the decorators you need and nothing else.

To implement in Currency Coversion Application follow below steps

Step1 : Go to currency Exchange service

Open POM file :

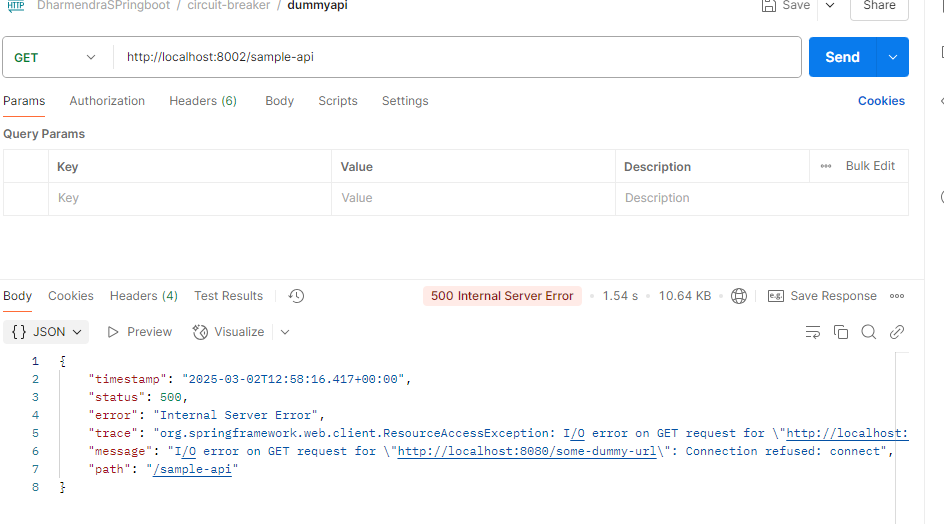
Add this

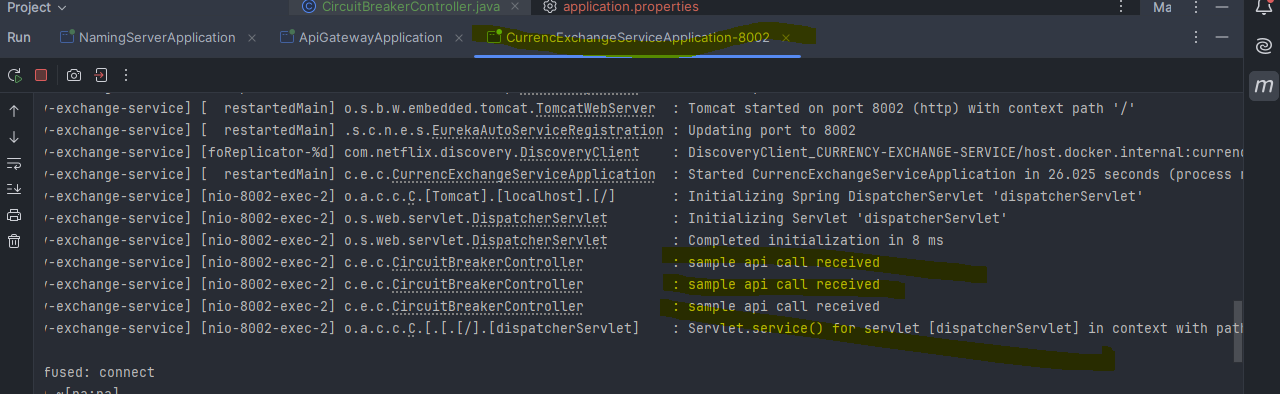
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-aop</artifactId>  
</dependency>

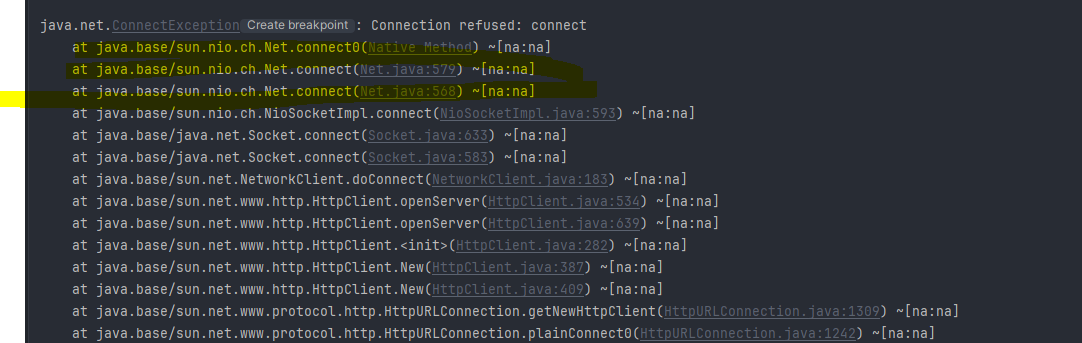
<dependency>  
 <groupId>io.github.resilience4j</groupId>  
 <artifactId>resilience4j-spring-boot2</artifactId>  
</dependency>

Ex:

*package* com.example.currenc\_exchange\_service;  
  
*import* io.github.resilience4j.retry.annotation.*Retry*;  
*import* org.slf4j.Logger;  
*import* org.slf4j.LoggerFactory;  
*import* org.springframework.http.ResponseEntity;  
*import* org.springframework.web.bind.annotation.*GetMapping*;  
*import* org.springframework.web.bind.annotation.*RestController*;  
*import* org.springframework.web.client.RestTemplate;  
  
  
*@RestController  
public class* CircuitBreakerController {  
 *private* Logger logger = LoggerFactory.getLogger(CircuitBreakerController.*class*);  
 *@GetMapping*("/sample-api")  
 *@Retry*(name = "sample-api")  
 *public* String sampleApi(){  
 logger.info("sample api call received");  
 ResponseEntity<String> forEntity = *new* RestTemplate()  
 .getForEntity("http://localhost:8080/some-dummy-url", String.*class*);  
 *return* forEntity.getBody();  
 }  
}





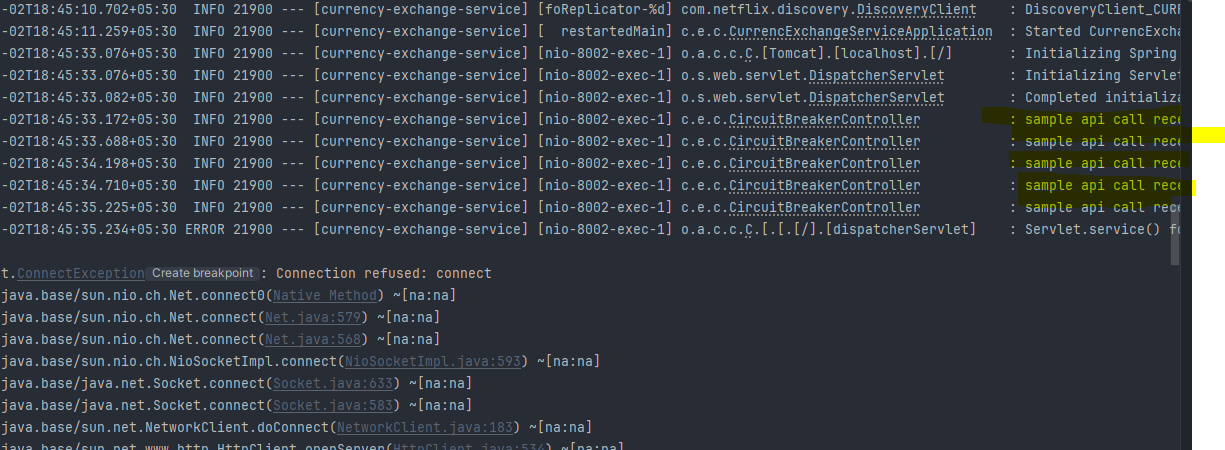


Step 2

default retry = 3.

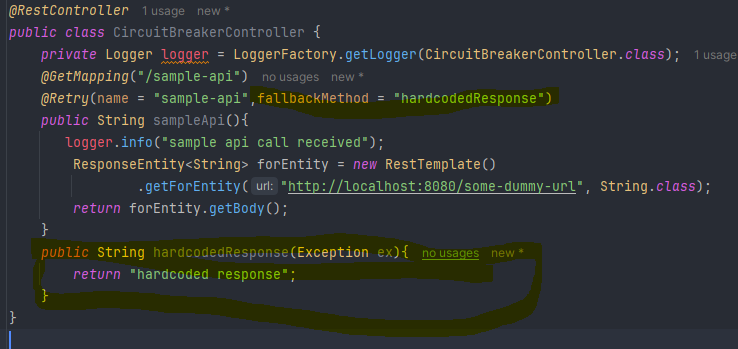
You can customize the retry in .property file as below :

spring.h2.console.enabled=true  
spring.jpa.show-sql=true  
spring.jpa.defer-datasource-initialization=true  
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka  
resilience4j.retry.instances.sample-api.maxRetryAttempts=5

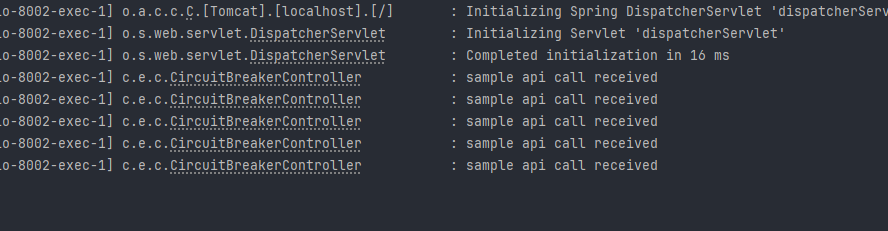


Step 3 :

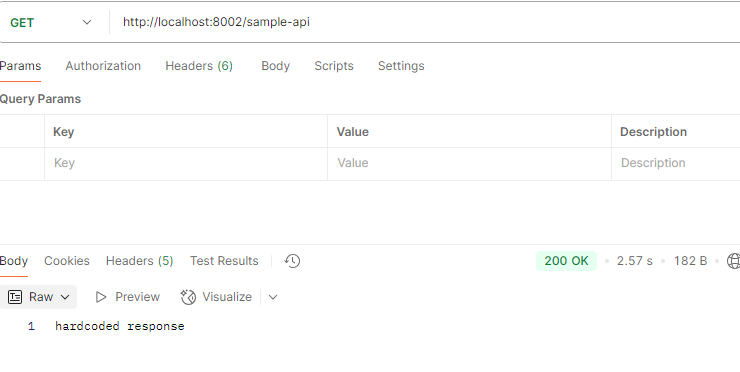
Setting up fallback method :



Console logs:



Api response

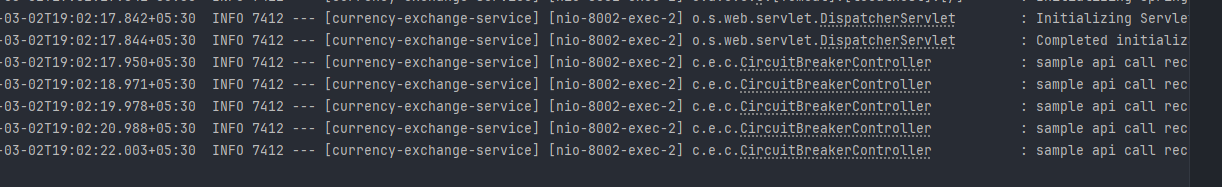


Step 4 :

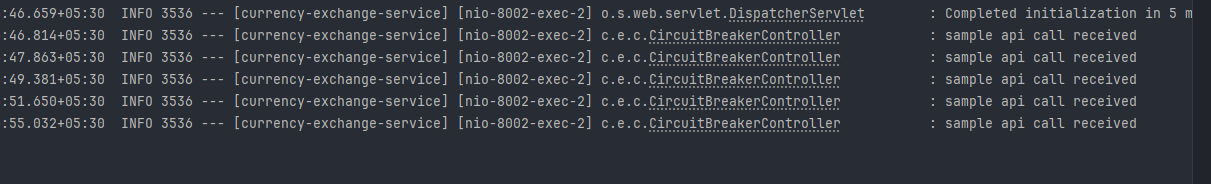
To configure timeout

What should be the interval between the retry api calls :

spring.jpa.defer-datasource-initialization=true  
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka  
resilience4j.retry.instances.sample-api.maxAttempts=5  
resilience4j.retry.instances.sample-api.waitDuration=1s



resilience4j.retry.instances.sample-api.enableExponentialBackoff=true



Conclusion:

