Contents

[1. CIRCUIT BREAKER Introduction: 1](#_Toc148441777)

[1.1 General 1](#_Toc148441778)

[2. Implementing Circuit Breaker Pattern : 1](#_Toc148441779)

[2.1 UI based Circuit Breakers: 2](#_Toc148441780)

[2.2 Backend based Circuit breakers: 2](#_Toc148441781)

[3. Rationale for choosing Resilience4J implementation: 2](#_Toc148441782)

[4. Resilience4J implementation: 2](#_Toc148441783)

[5. Actuator Outputs: 3](#_Toc148441784)

[6. Conclusion: 6](#_Toc148441785)

# CIRCUIT BREAKER Introduction:

# 1.1 General

* Circuit Breaker is a design pattern used in software engineering to handle failures in Microservices.
* It is used to detect and handle faults in communication between services, preventing them from cascading and causing further damage
* Circuit Breaker pattern has three states: Closed, Open, and Half-Open.
* **Closed**  
  In the closed state, the circuit breaker allows requests to flow through and execute the operation as normal.
* **Open**  
  In the open state, the circuit breaker returns a pre-configured fallback value instead of executing the operation.
* **Half-Open**  
  In the half-open state, the circuit breaker allows a limited number of requests to pass through to test if the operation is functioning correctly. If these requests succeed, the circuit breaker returns to the closed state. If they fail, the circuit breaker returns to the open state.

# Implementing Circuit Breaker Pattern :

* Circuit Breaker design pattern can be implemented both: UI or backend

# UI based Circuit Breakers:

* Following are the UI based implementations of Circuit breakers

1. opossum: <https://www.npmjs.com/package/opossum>
2. circuit-breaker-js : <https://www.npmjs.com/package/circuit-breaker-js>
3. @fastify/circuit-breaker: <https://www.npmjs.com/package/@fastify/circuit-breaker>

# Backend based Circuit breakers:

* netflix-hystrix
* resilience4j
* Sentinel

# Rationale for choosing Resilience4J implementation:

* Spring cloud Netflix-Hystrix is deprecated
* Resilience4J has more configuration options and offers granularity of control
* Resilience4J has ability to be integrated with other live monitoring systems
* Also, for time being, we are focusing on backend based Circuit Breaker implementation

# Resilience4J implementation:

* Add the following dependencies in booking-service pom.xml

A screenshot of a computer program

Description automatically generated

* Add the following configuration related to resilience4j: in booking-service application.properties, A screenshot of a computer program

  Description automatically generated
* In GlobalCompanyIdentiferSearchServiceHelper.java, add the @CircuitBreaker annotation and the relevant fallback method, which you want to execute if the call to the relevant service fails

A screenshot of a computer program

Description automatically generated

* In MultiSecurityConfig.java -> filterChain method, add the below line to allow the actuator url



# Actuator Outputs:

* Able to access actuator via : <http://localhost:8443/actuator>
* Actuator showing circuit breaker related links:

A screenshot of a computer

Description automatically generated

* To list the circuit breakers, we have added in our application: <http://localhost:8443/actuator/circuitbreakers>

A screenshot of a computer

Description automatically generated

* Also we can see the circuit breaker events: <http://localhost:8443/actuator/circuitbreakerevents>
* We can see the circuitbreakerevents are empty, before we launched the booking page:

A screenshot of a computer

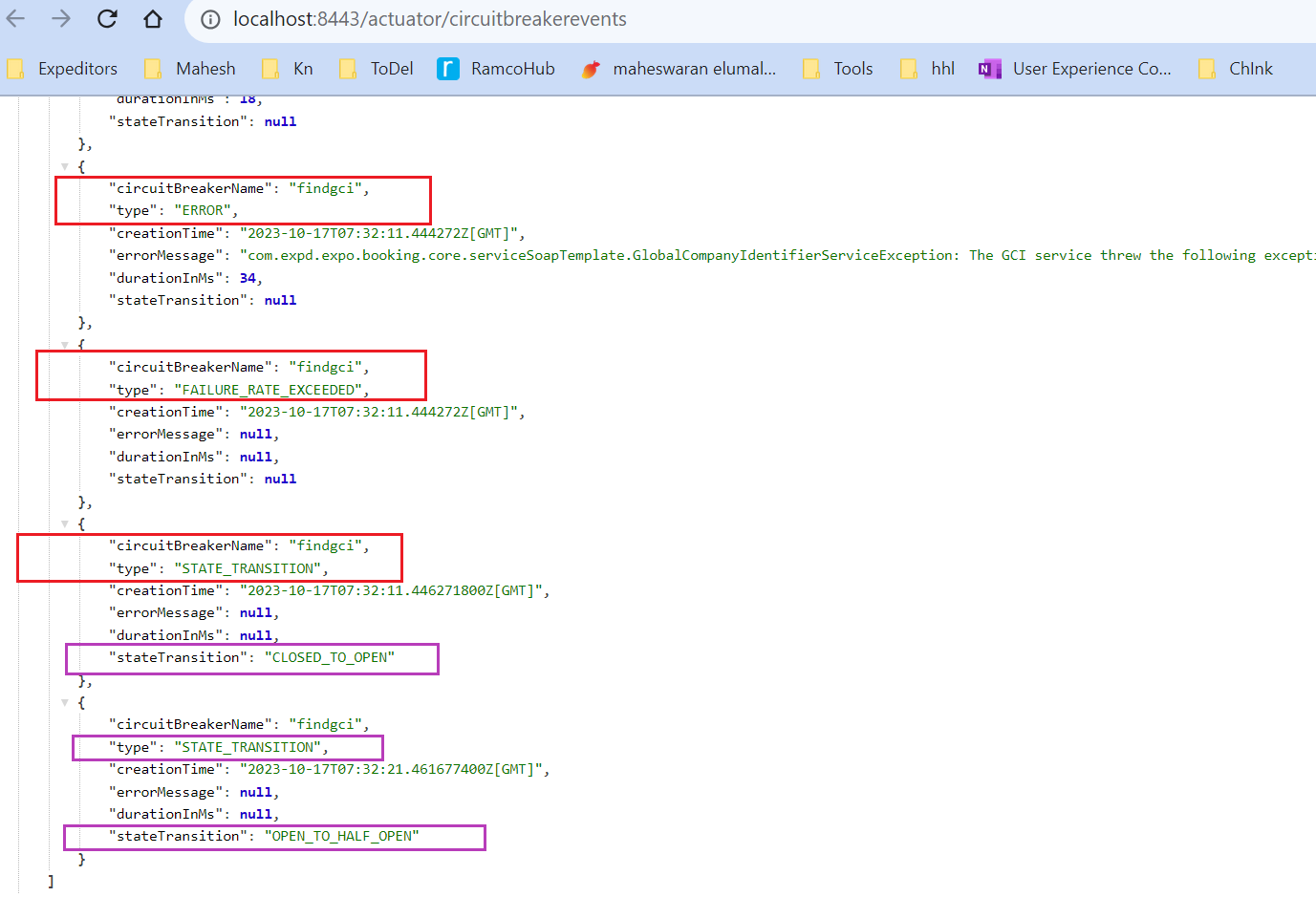
Description automatically generated

* **CLOSED STATE:** Initially we can see the circuitbreakers to be in closed state. Hence stateTransition is null and type is “SUCCESS”

A screenshot of a computer

Description automatically generated

* **CLOSED to OPEN and then from OPEN to HALF\_OPEN:** Now we bring the gci service down and see that findgci circuit breaker has changed to type: “Error” and we are able to see transition of states:



* Now we bring back gci service up and we can see that type has changed back to :”SUCCESS” and state transition is null, which means that gci circuitbreaker is back to CLOSED state now.

A screenshot of a computer

Description automatically generated

# Conclusion:

* As part of this exercise, We are able to successfully integrate resilience4j circuit breaker with booking application.
* Next step is to carefully identify relevant services/operations that need Circuit breaker implementations in the application and implement the same.
* *Also care should be taken to ensure that: we don’t implement Circuit breaker unnecessarily – since Circuit Breaker is implemented via AOP in underlying code and there are chances that it can affect the performance of the application.*