**Case Study implementation**

Contents

[1. MicroServices Architecture: 2](#_Toc53161130)

[2. Feign Client: 3](#_Toc53161135)

[3. Microservices with corresponding DB’s: 3](#_Toc53161136)

[4. Repository Pattern: 3](#_Toc53161137)

[5. Ribbon Load Balancer: 4](#_Toc53161138)

[6. Eureka Naming Server: 4](#_Toc53161139)

[7. Zuul API Gateway Server: 4](#_Toc53161140)

[8. Spring Security 5](#_Toc53161141)

[9. Swagger Documentation 5](#_Toc53161142)

[10. Schedulers 5](#_Toc53161143)

[11. SMTP 5](#_Toc53161144)

[12. Exception Handling 5](#_Toc53161145)

[13. Circuit Breaker 5](#_Toc53161146)

1. MicroServices Architecture:

## Customer Service:

This is used to perform the CRUD operations on the customer’s info along with their policies and claims info. This microservice establishes a connection with DB for customer info and a feign client is used to talk with other microservices (Claim Service, Policy & Premium Service and Transaction Service).

|  |  |  |
| --- | --- | --- |
| ***Method*** | ***URI*** | ***Operation*** |
| GET | /customer/{id} | To retrieve the customer info |
| GET | /customer/{id}/claims | To retrieve the claim info for a customer |
| GET | /customer/{id}/claims/{claimId} | To retrieve a particular claim for customer |
| POST | /customer/{id}/fileClaim | To file a claim against a policy |
| POST | /customer/{id}/payPremium/{policyId} | To make a payment for a policy |
| GET | /customer/{id}/policy | To retrieve policy info for a customer |

## Claim Service:

This is used to perform the retrieve and create operations on the Claim DB for a particular customer. Whenever a claim is filed, a mail will be triggered to the customer with the claim info.

A scheduler runs at every 2 mins to process the pending claims.

|  |  |  |
| --- | --- | --- |
| ***Method*** | ***URI*** | ***Operation*** |
| GET | /{custId}/claims | To retrieve list of all claims for a customer |
| GET | /{custId}/claims/{id} | To retrieve a particular claim for customer |
| POST | /fileClaim | To file a claim against a policy |

## Policy & Premium Service:

This is used to fetch the Policy and Premium info along with the due date and due amount. This provides an option to make a payment by connecting to Policy and Premium tables in DB and establishing a link to Transaction Service using a Feign client to process a transaction.

|  |  |  |
| --- | --- | --- |
| ***Method*** | ***URI*** | ***Operation*** |
| GET | /{custId}/policy | To retrieve list of all policies for a customer along with transaction numbers |
| PUT | /policy/{id} | To update the policy info |
| GET | /{policyId}/premium | To retrieve the premium info for a policy |

## Transaction Service:

This is used to fetch and insert transactions done against a policy. A scheduler runs at every 2 mins to process the pending transactions.

|  |  |  |
| --- | --- | --- |
| ***Method*** | ***URI*** | ***Operation*** |
| GET | /{policyId}/transactions | To retrieve list of transactions for a policy |
| POST | /payment | To insert a transaction for customer policy |

# Feign Client:

Feign Client is enabled in the Customer Service by adding proxy classes instead of the traditional RestTemplate to provide the connection from the calling service to the child service.

# Microservices with corresponding DB’s:

MySQL database is used by the microservices to do the DB transactions.

Below are the corresponding tables to each service

|  |  |
| --- | --- |
| ***MicroService*** | ***Table Name*** |
| Customer Service | CUSTOMER |
| Claim Service | CLAIMS |
| Policy & Premium Service | POLICY, PREMIUM |
| Transaction Service | TRANSACTION |

# Repository Pattern:

All the microservices have their own POJO classes defined as entities to do the ORM and all the CRUD actions are performed using an interface extending the JPARepository.

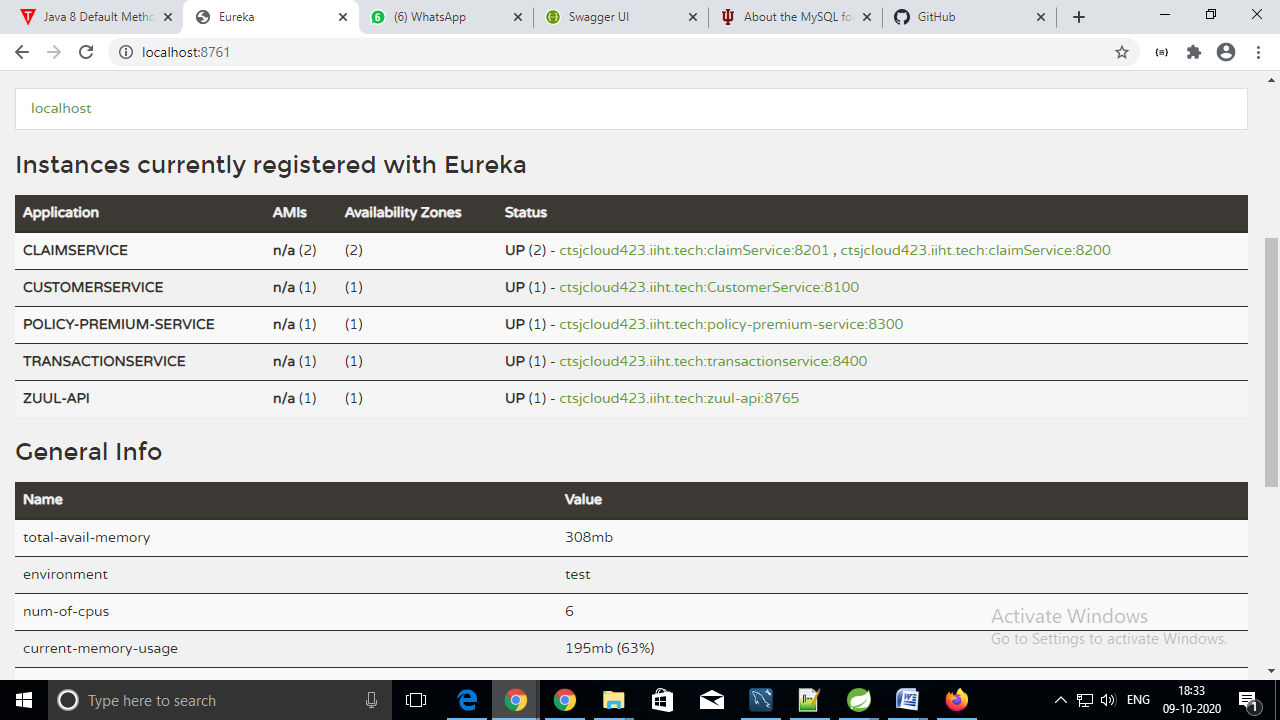
# Ribbon Load Balancer:

Ribbon LoadBalancer is used in the Customer services for distributing the request across the dependent microservices alongside the feign clients and instead of providing the list of servers for the Ribbon, the instances are managed by Eureka.

# Eureka Naming Server:

The Eureka server is provided to enable the microservices for registering themselves and discovering the other registered microservices. It also manages the instances for a microservice dynamically and provides them for Ribbon or Zuul to identify the port and call the service with their application name.

URL: <http://localhost:8761/>



# Zuul API Gateway Server:

The Zuul Server acts as a gateway for all the microservices. It provides the logging and security for application and routes the url for the corresponding microservice by discovering the application name from the Eureka Naming server.

Sample URL: <http://localhost:8765/customer-api/customer/101/claims>

# Spring Security

Spring security helps us in configuring the credentials for the USER and ADMIN.The Zuul API server implements security by using this configuration for mapping the URI’s to corresponding roles to restrict the USER’s from performing the ADMIN operations.

Sample URL: Actuator (Filters, Routes) <http://localhost:8765/actuator>

# Swagger Documentation

Swagger 2 is used to document the REST API provided by our microservices. The Swagger UI provides a beautiful and understandable interface for our API by exposing all the URI’s, Request methods, the attributes and its functionality along with an option to execute the service without going to the POSTMAN app.

Customer Service <http://localhost:8100/swagger-ui.html#/>

Claim Service <http://localhost:8200/swagger-ui.html#/>

Policy & Premium Service <http://localhost:8300/swagger-ui.html#/>

Transaction Service <http://localhost:8400/swagger-ui.html#/>

# Schedulers

Schedulers are enabled in the microservices Claim Service and Transaction Service to process the pending claims and transactions for every two minutes.

# SMTP

Java Mail Sender is implemented in the Customer Service to trigger the mails once the customer files a claim. Gmail SMTP is used to send the mail to the customer mail Id with the filed claim info.

# Exception Handling

Exceptions are handled and  shown to the end user(Customer) in an understandable way with the error reason and error code to check with the support team for resolving the issue.

# Circuit Breaker

Hystrix Circuit Breaker is used to take care of exceptions in case a dependent service fails or if it's taking too much time to return the response and a default response is given back to the user instead of the exception.