System Requirements Specification Index

For

Micro Pay Application

Version 1.0

TABLE OF CONTENTS

Micro Pay Application
System Requirements Specification

1 PROJECT ABSTRACT

The Micro Pay Application is an innovative financial transaction system built on a microservices architecture using Spring Boot 2.6.2 and Java 17. Comprising components like Order Service, Payment Service, and Eureka Server for service discovery, the application facilitates efficient order management, payment processing, and seamless communication between services. Leveraging Spring Cloud Open Feign for declarative REST client interactions and Hystrix for circuit breaking, it ensures scalability, fault tolerance, and modularity. The incorporation of Spring Data JPA enables smooth database integration. As a centralized service registry, Eureka Server promotes automatic service discovery. The application's modular design and utilization of advanced technologies exemplify a modern and robust solution for handling financial transactions in a scalable and maintainable manner.

Following is the requirement specifications:

	Micro Pay Application
Microservices	
1	Order-service
2	Payment-service
Order Microservice	
1	Add Order
2	Update an Order
3	Delete existing Order
4	Get the details of an Order
5	Get the details of all the Order
6	Save order to the database
7	Use Feign client to make a payment request
8	Update order status based on payment response
Payment Microservice	
1	Add Payment
2	Update Payment
3	Get Payment by Id
4	Fetch all Payment
5	Delete Payment
6	Use Feign client to update order status
7	Save payment to the database

2 CONSTRAINTS

Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in dto classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3 INTRODUCTION TO MICROSERVICES

3.1 EUREKA-NAMING-SERVER

This is a discovery server for all the registered microservices. This is fully implemented.

3.2 API-GATEWAY

This microservice is an api gateway to all the microservices. All the microservices can be accessed by using this common gateway. All the routes are already implemented in this api-gateway. All the rest endpoints can be accessed by this common gateway.

3.3 ORDER-SERVICE

The employee microservice is used to perform all the operations related to the order. In this microservice , you have to write the logic for OrderServiceImpl.java and OrderRestController.java classes. Rest all the classes and interfaces are fully implemented. All the dependencies are already configured.

3.4 PAYMENT-SERVICE

The skills microservice is used to perform all the operations related to payment. In this microservice, you have to write the logic for PaymentServiceImpl.java and PaymentRestController.java classes. Rest all the classes and interfaces are fully implemented. All the dependencies are already configured.

4 MICROSERVICES COMMUNICATION

The communication among the microservices is achieved by using the FeignClient. It is already configured in all the microservices. You can check in the proxy package of the microservice.

5 REST ENDPOINTS

Rest End-points to be exposed in the controller along with method details for the same to be created

5.1 ORDERRESTCONTROLLER(ORDER-SERVICE)

URL Exposed (Order Service)	Purpose
1. /order/ create	Add Order
2. /order /getAll	Fetches the list of all registered order
3. /order /get/{orderId}	Fetches the details of order by Id
4. /order /get/{orderId}	Update order
5. /order/update/{orderId}	Delete an order

5.2 PAYMENTRESTCONTROLLER (PAYMENT-SERVICE)

URL Exposed (Payment Service)	Purpose
1. /payment /makePayment	Add Payment
2. /payment /getAll	Fetches the list of all registered payment
3. /payment /get/{paymentId}	Fetches the details of payment by Id
4. /payment /update/{paymentId}	Update Payment
5. /payment /delete/{paymentId}	Delete an Payment

6 CONSIDERATIONS

A. You can perform the following 5 possible actions

Order Actions
Payment Actions
Use Feign client and Fallback
Discover all the services by Eureka
Use API Gateway to access all the microservices

7 SEQUENCE TO EXECUTE

The sequence has to be followed for step 8 for every microservice are given below:

- ? eureka-naming-server
- ? api-gateway
- ? order-service
- ? payment-service
- ? certificates-service

**Strictly follow the above sequence to follow step number 8.

8 EXECUTION STEPS TO FOLLOW

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
3. Kindly follow the sequence and run your commands through all the folders separately.
4. To build your project use command:
   mvn clean package -Dmaven.test.skip
5. To launch your application, move into the target folder (cd target). Run the following command to run the application:
   java -jar <jar-name>-0.0.1-SNAPSHOT.jar
6. This editor Auto Saves the code
7. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
8. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
9. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
10. To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.
11. Default credentials for MySQL:
    a. Username: root
    b. Password: pass@word1
11. To login to mysql instance: Open new terminal and use following command:
    a. sudo systemctl enable mysql
    b. sudo systemctl start mysql
    c. mysql -u root -p
       The last command will ask for password which is 'pass@word1'
12. Mandatory: Before final submission run the following command:
    mvn test
13. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.