System Requirements Specification

Index

For

Payment Management

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Payment Management
System Requirements Specification

1 PROJECT ABSTRACT

The Payment Management System is a comprehensive platform designed to streamline the process of managing users, products, orders, and payments within an integrated system. Built using Spring Boot microservices architecture, this system divides functionalities across multiple microservices—User, Product, Order, and Payment. Each microservice operates independently with its own database, allowing for modular, scalable, and efficient operations. The microservices interact via RESTful APIs to provide seamless end-to-end payment management.

Following is the requirement specifications:

	Payment Management
Microservices	
1	User Micro-service
2	Product Micro-service
3	Order Micro-service
4	Payment Micro-service
User Microservice	
User	
1	Register an user
2	List all users
3	Update an user
4	Get user product information
Auth	
1	Generate token
2	Validate token
Product Microservice	
1	Add a new product
2	Get products by owner id
3	List all products

Order Microservice	
1	Creates a new order
2	Get orders by user id
3	List all products
Payment Microservice	
1	Creates a new payment
2	Get payments by user id
3	Get list of all payments

2 CONSTRAINTS

Common Constraints

- Do not change, add, remove any existing methods in the 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 System Requirements

3.1 PAYMENT-MANAGEMENT-GATEWAY

This microservice is a gateway to all the microservices. All the microservices can be accessed by using this common gateway. Following implementations are expected to be done:

a. Configure Payment Management Gateway to run on port: 8085.

b. Implement the routes and logging in this gateway.

3.2 USER-SERVICE

The user microservice is used to perform all the operations related to the user. In this microservice, you have to write the logic for UserServiceImpl.java, AuthServiceImpl.java and UserController.java, AuthController.java classes. Following implementations are expected to be done:

a. Configure this service to run on port: 8084.

b. You are required to configure a feign proxy to fetch (Get Product Info by Owner ID).

3.3 PRODUCT-SERVICE

The product microservice is used to perform all the operations related to the products. In this microservice, you have to write the logic for ProductServiceImpl.java and ProductController.java classes. Following implementations are expected to be done:

a. Configure this service to run on port: 8076.
b. You are required to configure a feign proxy to fetch (Get User Details by User ID).

3.4 ORDER-SERVICE

The order microservice is used to perform all the operations related to the orders. In this microservice, you have to write the logic for OrderServiceImpl.java and OrderController.java classes. Following implementations are expected to be done:

a. Configure this service to run on port: 8075.

3.5 PAYMENT-SERVICE

The payment microservice is used to perform all the operations related to the payments. In this microservice, you have to write the logic for PaymentServiceImpl.java and PaymentController.java classes. Following implementations are expected to be done:

a. Configure this service to run on port: 8077.
b. You are required to configure a feign proxy to fetch (Get User Details by User ID).

GIT based command (for reference):

git init: To initialize a git repository.

git add: To add/track changes done in repository.

git commit: To save and commit changes to repository

4 MICROSERVICES COMMUNICATION

Communication among the microservices needs to be achieved by using FeignClient. A Feign configuration class is created in the project, but you are required to implement the feign client method. You can check in the proxy package of the microservice.

- You are required to configure a feign proxy to fetch (Get User Details by User ID) (Product-Service) (Payment-Service).

- You are required to configure a feign proxy to fetch (Get Product Info by Owner ID) (User-Service).

5 REST ENDPOINTS

Rest Endpoints to be exposed in the controller along with method details for the same to be created

a. USERCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/users/register
<table><tr><td>Http Method</td><td>POST

The user data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter 1</td><td>UserDto</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Create / Register a new user in the system</td></tr>
<tr><td>2. /api/users/list
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>List<UserDto></td></tr></table></td><td>Retrieves a list of all registered users</td></tr>
<tr><td>3. /api/users/{userId}
<table><tr><td>Http Method</td><td>POST

The user data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Path Variable 1</td><td>Long (userId)</td></tr><tr><td>Parameter 2</td><td>UserDto</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Updates user information for a specific user identified by their user ID</td></tr>
<tr><td>4. /api/users/productInfo/{userId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Path Variable</td><td>Long (userId)</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Retrieves product-related information for a specific user, identified by their user ID.</td></tr>
</table>

b. AuthController

URL Exposed		Purpose
1./auth/token		Authenticates a user and generates an authentication token if credentials are valid.
Http Method	POST **The token data to be created must be received in the controller using @RequestBody.**	
Parameter 1	AuthRequest	
Return	String	
2./auth/validate		Validates a given JWT to ensure it's still valid and corresponds to an active session.
Http Method	Get	
Request param	String (token)	
Return	String	

c. ProductController

URL Exposed		Purpose
1. /api/products/create		Adds a new product to the system
Http Method	POST **The product data to be created must be received in the controller using @RequestBody.**	
Parameter 1	ProductDto	
Return	Response	
2. /api/products/owner/{ownerId}		Retrieves a list of products associated with a specific owner
Http Method	GET	
Path variable	Long (ownerId)	
Return	List<ProductDto>	
3. /api/products/all		Retrieves a paginated list of all products in the system
Http Method	GET	
Parameter	Pageable	
Return	Page<ProductDto>	

d. ORDERCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/orders/create
<table><tr><td>Http Method</td><td>POST

The order data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter 1</td><td>OrderDto</td></tr><tr><td>Return</td><td>Response</td></tr></table></td><td>Creates a new order in the system</td></tr>
<tr><td>2. /api/orders/{userId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Path variable</td><td>Long (userId)</td></tr><tr><td>Return</td><td>List<OrderDto></td></tr></table></td><td>Retrieves all orders associated with a specific user</td></tr>
<tr><td>3. /api/orders/all
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>Pageable</td></tr><tr><td>Return</td><td>Page<OrderDto></td></tr></table></td><td>Retrieves a paginated list of all orders in the system</td></tr>
</table>

e. PAYMENTCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/payments/create<table><tr><td>Http Method</td><td>POST

The payment data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter</td><td>PaymentDto</td></tr><tr><td>Return</td><td>Response</td></tr></table></td><td>Creates a new payment record in the system</td></tr>
<tr><td>2. /api/payments/{userId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Path variable</td><td>Long (userId)</td></tr><tr><td>Return</td><td>List<PaymentDto></td></tr></table></td><td>Retrieves all payment records associated with a specific user by user id</td></tr>
<tr><td>3. /api/payments/all<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>Pageable</td></tr><tr><td>Return</td><td>PaymentDto</td></tr></table></td><td>Retrieves a paginated list of all payments in the system.</td></tr>
</table>

6 SEQUENCE TO EXECUTE

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

- service-registry
- payment-management-gateway
- user-micro-service
- product-micro-service
- order-micro-service
- payment-micro-service

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

7 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 the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. cd into your backend project folder
4. To build your project use command:

   mvn clean package
5. To launch your application, move into the target folder (cd target). Run the following command to run the application:

   java -jar <your application jar file name>
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

12. To login to mysql instance: Open new terminal and use following command:

    a. sudo systemctl enable mysql

    b. sudo systemctl start mysql

    NOTE: After typing any of the above commands you might encounter any warnings.

    >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

    c. mysql -u root -p

    The last command will ask for password which is 'pass@word1'

13. Mandatory: Before final submission run the following command:

    mvn test

14. 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.

15. If the CTRL+Shift+B command is not working, you can manually push changes to Git using the following commands in your terminal:

    -> git status

    -> git add .

    -> git commit -m "Completed"

    -> git push