System Requirements Specification

Index

For

FORUM APP

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Forum App

System Requirements Specification

1 PROJECT ABSTRACT

The Forum Application is designed to create an interactive online platform where users can engage in discussions, share information, and connect with others through posts and comments. The application leverages microservices architecture built using Spring Boot, with dedicated microservices for handling users and posts. Each microservice operates independently with its own database and communicates seamlessly with others to provide a cohesive user experience.

Following is the requirement specifications:

	Forum App
Microservices	
1	User Micro-service
2	Post Micro-service
User Microservice	
1	Register User
2	Login into user
3	Check email is already in use
4	Get the user details by user id
5	Get all the users
Post Microservice	
Post	
1	Adds a new post for a specified user by user id
2	Get a specific post by post id
3	Get all posts associated with a specific user by user id
4	Add like to post by post id
Comment	
1	Add a comment to a specific post by user id
2	Add like to comment by comment id

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 EUREKA-NAMING-SERVER

This is a discovery server for all the registered microservices. Following implementations are expected to be done:

a. Configure the Eureka server to run on port: 8761.
b. Configure the Eureka server to deregister itself as Eureka client.
c. Add appropriate annotation to Enable this module to run as Eureka Server.

You can launch the admin panel of Eureka server in the browser preview option.

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. Following implementations are expected to be done:

a. Configure API Gateway to run on port: 6062.
b. Implement the routes and logging in this api-gateway.

3.3 USER-MICRO-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 and UserController.java classes. Following implementations are expected to be done:

a. Configure this service to run on port: 9091.

3.4 POST-MICRO-SERVICE

The post microservice is used to perform all the operations related to the post and comment. In this microservice, you have to write the logic for PostServiceImpl.java, CommentServiceImpl.java and PostController.java, CommentController.java classes. Following implementations are expected to be done:

a. Configure this service to run on port: 9092.
b. You are required to configure a feign proxy to fetch (Get User Details by User ID).

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 1 feign proxy to fetch (Get User Details by User ID) (Post-Micro-Service)

5 REST ENDPOINTS

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

a. USERCONTROLLER

URL Exposed		Purpose
1. /api/user/register		Create / Register a new user
Http Method	POST **The user data to be created must be received in the controller using @RequestBody.**	
Parameter 1	RegisterDto	
Return	UserDetailDTO	
2. /api/user/login		Authenticates a user by validating their login credentials
Http Method	POST **The user data to be created must be received in the controller using @RequestBody.**	
Parameter 1	LoginDto	
Return	UserDetailDTO	
3. /api/user/check-in-use/{emailId}		Checks if the email ID is already in use in the system or not
Http Method	GET	
Path Variable	String (emailId)	
Return	Boolean	
4. /api/user/get/{id}		Retrieves details of a user by their user ID
Http Method	GET	
Path Variable	Integer (id)	
Return	UserDetailDTO	
5. /api/user/all-users		Fetches all the registered users
Http Method	GET	
Parameter	-	
Return	List <UserDetailDTO>	

b. CommentController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1./api/comment/add/{postId}/{userId}
<table><tr><td>Http Method</td><td>POST

The comment data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Path Variable 1</td><td>postId</td></tr><tr><td>Path Variable 2</td><td>userId</td></tr><tr><td>Parameter 3</td><td>CommentDto</td></tr><tr><td>Return</td><td>CommentDetailDto</td></tr></table></td><td>Allows users to add a comment to a specific post</td></tr>
<tr><td>2./api/comment/like/{commentId}
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Path Variable</td><td>String (commentId)</td></tr><tr><td>Return</td><td>Integer</td></tr></table></td><td>Increments the like count for a specific comment and return the final count of likes</td></tr>
</table>

c. PostController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/post/add/{userId}
<table><tr><td>Http Method</td><td>POST

The post data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Path variable</td><td>String (userId)</td></tr><tr><td>Return</td><td>PostDetailDto</td></tr></table></td><td>Adds a new post for a specified user by user id</td></tr>
<tr><td>2. /api/post/get/{postId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Path variable</td><td>String (postId)</td></tr><tr><td>Return</td><td>PostDetailDto</td></tr></table></td><td>Retrieves a specific post by its ID</td></tr>
</table>

3. /api/post/get-all/{userId}		Retrieves all posts associated with a specific user by user id
Http Method	GET	
Path variable	String (userId)	
Return	PostDetailListDto	
4. /api/post/like/{postId}		Increments the like count for a specific post and return the final count of likes
Http Method	POST	
Path variable	String (postId)	
Return	Integer	

6 SEQUENCE TO EXECUTE

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

- eureka-naming-server
- api-gateway
- user-micro-service
- post-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. 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.
8. 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.
9. 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.
10. 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

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’

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

mvn test