System Requirements Specification

Index

For

FORUM APP

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Version 1.0 1

Forum App 2

1 PROJECT ABSTRACT 3

2 CONSTRAINTS 4

Common Constraints 4

3 SYSTEM REQUIREMENTS 4

4 MICROSERVICES COMMUNICATION 5

5 REST ENDPOINTS 6

6 SEQUENCE TO EXECUTE 8

7 EXECUTION STEPS TO FOLLOW 9

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).

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

Parameter	-	
Return	List <UserDetailDTO>	

b. CommentController

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

c. PostController

URL Exposed		Purpose
1. /api/post/add/{userId}		Adds a new post for a specified user by user id
Http Method	POST **The post data to be created must be received in the controller using @RequestBody.**	
Path variable	String (userId)	
Return	PostDetailDto	
2. /api/post/get/{postId}		Retrieves a specific post by its ID
Http Method	GET	

Path variable	String (postId)	
Return	PostDetailDto	
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. 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