System Requirements Specification

Index

For

CrowdFunding Platform

Version 1.0

TABLE OF CONTENTS

CROWDFUNDING PLATFORM
System Requirements Specification

BACKEND-SPRING DATA RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **CrowdFunding Platform** is implemented using Spring Data with a MySQL database. In the dynamic world of crowdfunding, there's a growing need for modern platforms that connect project creators with potential backers. The CEO of a visionary startup, Mr. Patel, challenges a team of developers to create a Fullstack Crowdfunding Platform.

Your task is to develop a digital solution that empowers users to create and support / manage crowdfunding campaigns, facilitating the investments of innovative projects.

Following is the requirement specifications:

	Crowdfunding Platform
Modules	
1	Investment
2	Project
Investment Module Functionalities	
1	Can create/make investments **(must be transactional)**
2	Can update an investment **(must be transactional)**
3	Can delete an investment **(must be transactional)**
4	Get investment by investment id
5	Get investments by project id **(must use custom query)**
6	Get investments by investor name **(must use dynamic method)**
Project Module Functionalities	
1	Can create a project
2	Can update a project
3	Can delete a project
4	Get project by id
5	Get all projects **(must return all projects by name in ascending order and that also in pages)**

2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 INVESTMENT CONSTRAINTS

- When fetching an investment by id, if the investment ID does not exist, the service method should throw a “Investment not found” message in the ResourceNotFoundException class.
- When updating an investment , if the investment ID does not exist, the service method should throw a “Investment not found” message in the ResourceNotFoundException class.
- When removing an investment , if the investment ID does not exist, the service method should throw a “Investment not found” message in the ResourceNotFoundException class.

2.2 PROJECT CONSTRAINTS

- When fetching a project by id, if the project ID does not exist, the service method should throw a “Project not found” message in the ResourceNotFoundException class.
- When updating a project , if the project ID does not exist, the service method should throw a “Project not found” message in the ResourceNotFoundException class.
- When removing a project , if the projectID does not exist, the service method should throw a “Project not found” message in the ResourceNotFoundException class.

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 Business Validations

Investment:

- Id must be of type id.
- Investment amount should not be null and must be at least 1.
- Investor name should not be null and max 255 characters.

Project:

- Id must be of type id.
- Project name should not be blank and max 255 characters.
- Project description should not be blank and max 2000 characters.
- Goal amount should not be null and must be at least 1.
- Amount raised should not be null.

4 Rest Endpoints

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

4.1 InvestmentController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/investments/{investmentId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>Long (investmentId)</td></tr><tr><td>Return</td><td>InvestmentDTO</td></tr></table></td><td>Fetches the investment by investmentId</td></tr>
<tr><td>2. /api/investments/project/{projectId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (projectId)</td></tr><tr><td>Return</td><td>List<InvestmentDTO></td></tr></table></td><td>Fetches the investments by projectId</td></tr>
<tr><td>3. /api/investments/
<table><tr><td>Http Method</td><td>POST

The investment data to be created should be received in @RequestBody</td></tr><tr><td>Parameter</td><td>InvestmentDTO</td></tr><tr><td>Return</td><td>InvestmentDTO</td></tr></table></td><td>Creates a new investment</td></tr>
<tr><td>4. /api/investments/{investmentId}</td><td></td></tr>
</table>

Http Method	PUT **The investment data to be updated should be received in @RequestBody**	Updates an investment by id
Parameter 1	Long (investmentId)	
Parameter 2	InvestmentDTO	
Return	InvestmentDTO	
5. /api/investments/{investmentId}		Deletes an investment by id
Http Method	DELETE	
Parameter 1	Long (investmentId)	
Return	-	
6. /api/investments/investor/{investorName}		Fetches the investments by investor name
Http Method	GET	
Parameter	String (investorName)	
Return	List<InvestmentDTO>	

4.2 ProjectController

URL Exposed		Purpose
1. /api/projects/{projectId}		Fetches the project by project id
Http Method	GET	
Parameter	Long (id)	
Return	ProjectDTO	
1. /api/projects/		Fetches all the projects
Http Method	GET	
Parameter	-	
Return	Page<ProjectDTO>	
3. /api/projects/		Creates a new project
Http Method	POST **The project data to be created should be received in @RequestBody**	
Parameter	ProjectDTO	
Return	ProjectDTO	

4. /api/projects/{projectId}		Updates a project by id
Http Method	PUT **The project data to be updated should be received in @RequestBody**	
Parameter 1	Long (projectId)	
Parameter 2	ProjectDTO	
Return	ProjectDTO	
5. /api/projects/{projectId}		Deletes a project by id
Http Method	DELETE	
Parameter 1	Long (projectId)	
Return	-	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.CROWDFUNDING

Resources

CrowdFundingPlatformApplication (Class)	This is the Spring Boot starter class of the application.	Already implemented.

5.2 PACKAGE: COM.CROWDFUNDING.REPOSITORY

Resources

Class/Interface	Description	Status
InvestmentRepository (interface)	• Repository interface exposing CRUD functionality for investment Entity. • You can go ahead and add any custom methods as per requirements. • It must contain the methods for: ○ Finding all investments associated with a specific	Partially implemented.

	project ID.. ○ Finding all investments made by an investor with the specified name.	
ProjectRepository (interface)	• Repository interface exposing CRUD functionality for project Entity. • You can go ahead and add any custom methods as per requirements.	Already implemented.

5.3 PACKAGE: COM.CROWDFUNDING.SERVICE

Resources

Class/Interface	Description	Status
InvestmentService (interface)	• Interface to expose method signatures for investment related functionality. • Do not modify, add or delete any method.	Already implemented.
ProjectService (interface)	• Interface to expose method signatures for project related functionality. • Do not modify, add or delete any method.	Already implemented.

5.4 PACKAGE: COM.CROWDFUNDING.SERVICE.IMPL

Class/Interface	Description	Status
InvestmentServiceImpl (class)	• Implements InvestmentService. • Contains template method implementation. • Need to provide implementation for investment related functionalities. • Do not modify, add or delete	To be implemented.

	any method signature	
ProjectServiceImpl (class)	• Implements ProjectService. • Contains template method implementation. • Need to provide implementation for project related functionalities. • Do not modify, add or delete any method signature	To be implemented.

5.5 PACKAGE: COM.CROWDFUNDING.CONTROLLER

Resources

Class/Interface	Description	Status
InvestmentController (Class)	• Controller class to expose all rest-endpoints for investment related activities. • Should also contain local exception handler methods	To be implemented
ProjectController (Class)	• Controller class to expose all rest-endpoints for project related activities. • Should also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.CROWDFUNDING.DTO

Resources

Class/Interface	Description	Status
InvestmentDTO (Class)	• Use appropriate annotations for validating attributes of this class.	Partially implemented.

ProjectDTO (Class)	• Use appropriate annotations for validating attributes of this class.	Partially implemented.

5.7 PACKAGE: COM.CROWDFUNDING.ENTITY

Resources

Class/Interface	Description	Status
Investment (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. • Map this class with an **investment** table. • Generate the **id** using the IDENTITY strategy	Partially implemented.
Project (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. • Map this class with a **project** table. • Generate the **id** using the IDENTITY strategy	Partially implemented.

5.8 PACKAGE: COM.COM.CROWDFUNDING.EXCEPTION

Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the investment, project info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already implemented.
ErrorResponse (Class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **InvalidDataException** class. • Use this as a reference for creating exception handler for other custom exception classes	Already implemented.
RestExceptionHandler (Class)	• RestControllerAdvice Class for defining rest exception handlers. • Contains Exception Handler for **ResourceNotFoundException** class. • Use this as a reference for creating exception handler for other custom exception classes	Already implemented.

6 EXECUTION STEPS TO FOLLOW FOR BACKEND

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 -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 <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.Please use 127.0.0.1 instead of localhost to test rest endpoints.**
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.