System Requirements Specification

Index

For

Banking Application

Version 1.0

TABLE OF CONTENTS

BANKING APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Banking Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a virtual bank and allow it to do all major operations.

Following is the requirement specifications:

	Banking Application
Modules	
1	User
2	Transaction
User Module Functionalities	
1	Get user by id
2	Create user
3	Update user by id
4	Delete user by id

Transaction Module Functionalities	
1	Add a transaction
2	Get all transaction for a user

2 Assumptions, Dependencies, Risks / Constraints

2.1 USER CONSTRAINTS

- When fetching a user by ID, if the user ID does not exist, the operation should throw a not found exception.
- When updating a user, if the user ID does not exist, the operation should throw a not found exception.
- When removing a user, if the user ID does not exist, the operation should throw a not found exception.

2.2 TRANSACTION CONSTRAINTS

- When fetching a transaction by ID for any user, if the user ID does not exist, the operation should throw a not found exception.

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 - User

- Name is not blank.
- AccountNumber is not blank.
- AccountType is not blank.

4 Business Validations - Transaction

- Amount is not blank.
- TransactionDate is not blank.
- User is not blank.

5 REST ENDPOINTS

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

5.1 USERCONTROLLER

URL Exposed		Purpose
1. /api/users/{id}		Get a user by id
Http Method	GET	
Parameter 1	Long (id)	
Return	User	
2. /api/users		Create a new user
Http Method	POST	
Parameter	-	
Return	User	
3. /api/users/{id}		Updates existing user by id
Http Method	PUT	
Parameter 1	Long (id)	
Return	User	
4. /api/users/{id}		Deletes a user by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

5.2 TRANSACTION CONTROLLER

URL Exposed		Purpose
1. /api/transactions		Creates a new transaction
Http Method	POST	
Parameter	-	
Return	Transaction	
2. /api/transactions/user/{userId}		Fetches a list of all transaction for any user
Http Method	GET	
Parameter 1	Long (userId)	
Return	List<Transaction>	

6 TEMPLATE CODE STRUCTURE

6.1 PACKAGE: COM.BANKINGAPPLICATION

Resources

BankingApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented

6.2 PACKAGE: COM.BANKINGAPPLICATION.REPOSITORY

Resources

Class/Interface	Description	Status
TransactionRepository (interface)	• Repository interface exposing CRUD functionality for transaction entity. • You can go ahead and add any custom methods as per requirements.	Partially implemented.
UserRepository (interface)	• Repository interface exposing CRUD functionality for user entity. • You can go ahead and add any custom methods as per requirements.	Partially implemented.

6.3 PACKAGE: COM.BANKINGAPPLICATION.SERVICE

Resources

Class/Interface	Description	Status
TransactionService (interface)	• Interface to expose method signatures for transaction related functionality. • Do not modify, add or delete any method.	Already implemented.
UserService (interface)	• Interface to expose method signatures for user related functionality. • Do not modify, add or delete any method.	Already implemented.

6.4 PACKAGE: COM.BANKINGAPPLICATION.SERVICE.IMPL

Class/Interface	Description	Status
TransactionServiceImpl (class)	• Implements TransactionService. • Contains template method implementation. • Need to provide implementation for transaction related functionalities. • Do not modify, add or delete any method signature	To be implemented.

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

6.5 PACKAGE: COM.BANKINGAPPLICATION.CONTROLLER

Resources

Class/Interface	Description	Status
TransactionController (Class)	• Controller class to expose all rest-endpoints for transaction related activities. • May also contain local exception handler methods	To be implemented
UserController (Class)	• Controller class to expose all rest-endpoints for user related activities. • May also contain local exception handler methods	To be implemented

6.6 PACKAGE: COM.BANKINGAPPLICATION.DTO

Resources

Class/Interface	Description	Status
TransactionDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.
UserDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.

6.7 PACKAGE: COM.BANKINGAPPLICATION.ENTITY

Resources

Class/Interface	Description	Status
Transaction (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 **transaction table**. • Generate the **id** using the IDENTITY strategy	Partially implemented.
User (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 **user table**. • Generate the **id** using the IDENTITY strategy	Partially implemented.

6.8 PACKAGE: COM.BANKINGAPPLICATION.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete the product/sell info which does not exist.	Already implemented.

	• Need to create Exception Handler for same wherever needed (local or global)	

1 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.**
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.