System Requirements Specification

Index

For

Banking Application

Version 1.0

TABLE OF CONTENTS

BANKING APPLICATION

System Requirements Specification

BACKEND-SPRING DATA RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Banking Application** is implemented using Spring Data 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 a user.
3	Update user by id.
4	Delete user by id.
5	Search users by name **(must use dynamic method).**

Transaction Module Functionalities	
1	Add a transaction **(must be transactional).**
2	Get all transactions for a user **(must return transactions by date in ascending order and that also in pages).**
3	Get user transactions with date range **(must use custom query).**
4	Get user transactions with amount range **(must use custom query).**

2 Assumptions, Dependencies, Risks / Constraints

2.1 USER CONSTRAINTS

- When fetching a user by ID, if the user ID does not exist, the service method should throw a NotFoundException with “User not found.” message.
- When updating a user, if the user ID does not exist, the service method should throw a NotFoundException with “User not found.” message.
- When removing a user, if the user ID does not exist, the service method should throw a NotFoundException with “User not found.” message.

2.2 TRANSACTION CONSTRAINTS

- When adding a transaction by user ID, if the user ID does not exist, the service method should throw a NotFoundException with “User not found.” message.

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 should not be blank.
- AccountNumber should not be blank.
- AccountType should not be blank.

Transaction

- Amount should not be null.
- TransactionDate should not be null.
- UserDTO 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 USERCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/users/{id}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Get a user by id</td></tr>
<tr><td>2. /api/users<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</td><td>-</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Create a new user</td></tr>
<tr><td>3. /api/users/{id}<table><tr><td>Http Method</td><td>PUT

The user data to be updated must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Updates existing user by id</td></tr>
<tr><td>4. /api/users/{id}<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Deletes a user by id</td></tr>
<tr><td>5. /api/users/search<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Request Parameter</td><td>name</td></tr><tr><td>Return</td><td>List<UserDTO></td></tr></table></td><td>Search users by name</td></tr>
</table>

4.2 TRANSACTION CONTROLLER

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

The transaction data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>TransactionDTO</td></tr></table></td><td>Creates a new transaction</td></tr>
<tr><td>2. /api/transactions/user/{userId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (userId)</td></tr><tr><td>Return</td><td>List<TransactionDTO></td></tr></table></td><td>Fetches a list of all transaction for any user</td></tr>
<tr><td>3. /api/transactions/amount-range
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Request Param1</td><td>minAmount</td></tr><tr><td>Request Param2</td><td>maxAmount</td></tr><tr><td>Return</td><td>List<TransactionDTO></td></tr></table></td><td>Fetches all transactions in amount range</td></tr>
<tr><td>4. /api/transactions/user/{userId}/date-range
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (userId)</td></tr><tr><td>Request Parameter 1</td><td>startDate</td></tr><tr><td>Request Parameter 2</td><td>endDate</td></tr><tr><td>Return</td><td>List<TransactionDTO></td></tr></table></td><td>Fetches all transactions in date range</td></tr>
</table>

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.BANKINGAPPLICATION

Resources

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

5.2 PACKAGE: COM.BANKINGAPPLICATION.REPOSITORY

Resources

Class/Interface	Description	Status

TransactionRepository (interface)	• Repository interface exposing CRUD functionality for transaction entity.
	• It must contain the methods for:
	  ○ finding all users order by transaction date ascendingly
	  ○ finding all transaction by amount in range
	  ○ finding all transaction by user id and transaction date range
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

UserRepository (interface)	• Repository interface exposing CRUD functionality for user entity. • It must contain the methods for: ○ finding all users by name • You can go ahead and add any custom methods as per requirements.	Partially implemented.

5.3 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.

5.4 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.

5.5 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

5.6 PACKAGE: COM.BANKINGAPPLICATION.DTO

Resources

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

5.7 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.

5.8 PACKAGE: COM.BANKINGAPPLICATION.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete the user/transaction info which does not exist. • Need to create Exception Handler for same wherever needed (local or global).	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**
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.**