System Requirements Specification Index

For

Online Banking System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Table of Contents

ONLINE BANKING APPLICATION

System Requirements Specification

1 Project Abstract

Online banking system is Spring boot application with MySQL, where it allows users to manage Users, let users create different types of Accounts and manages history of all types of Transactions. We can perform all basic CRUD operations along with the search functionality.

	Train Information Management System
Modules	
1	Account
2	Transaction
3	Users
Account Module Functionalities	
1	Create an Account
2	Update the existing Account details
3	Get the Account by Id
4	Get all Account
5	Delete an Account
6	Search the Account by username
Transaction Module Functionalities	
1	Create a Transaction
2	Update the existing Transaction details
3	Get the Transaction by Id
4	Get all Transaction
5	Delete a Transaction
6	Search the Transaction by type
Users Module Functionalities	
1	Create a User
2	Update the existing User details

3	Get the User by Id
4	Get all Users
5	Delete a User
6	Search the User by name

2 Assumptions, Dependencies, Risks / Constraints

2.1 ACCOUNT CONSTRAINTS

- When fetching an Account by ID, if the account ID does not exist, the operation should throw a custom exception.
- When updating an Account, if the account ID does not exist, the operation should throw a custom exception.
- When updating an Account, if request comes for updating account number then also it should throw a custom exception.
- When removing an Account, if the account ID does not exist, the operation should throw a custom exception.
- When creating an Account, if request comes with already created account number, then it should throw a custom exception.

2.2 TRANSACTION CONSTRAINTS

- When fetching a Transaction by ID, if the transaction ID does not exist, the operation should throw a custom exception.
- When updating a Transaction, if the transaction ID does not exist, the operation should throw a custom exception.
- When removing a Transaction, if the transaction ID does not exist, the operation should throw a custom exception.

2.3 USER CONSTRAINTS

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

- When creating a User, if request comes with already created username, then it should throw a custom exception.
- When updating a User, if request comes for updating username then also it should throw a custom exception.
- When removing a User, we should remove all details from Account and Transaction tables also.

COMMON CONSTRAINTS

- 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.
- Must not go and touch the test resources, as they will be used for Auto-Evaluation
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3 REST ENDPOINTS

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

3.1 ACCOUNTCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/accounts
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>List<Account></td></tr></table></td><td>Fetches all the accounts</td></tr>
<tr><td>/accounts
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Request body</td><td>Account</td></tr><tr><td>Return</td><td>Account</td></tr></table></td><td>Add a new account</td></tr>
<tr><td>/accounts/{id}
<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr></table></td><td>Delete account with given account id</td></tr>
</table>

Return	

/accounts/{id}	Fetches the account with the given id
Http Method	GET
Parameter 1	Long (id)
Return	Account

/accounts/{id}	Updates existing account by id
Http Method	PUT
Parameter 1	Long (id)
Body	Account
Return	Account

/accounts/search	Searches all accounts with given username
Http Method	GET
Request param	username
Return	List<Account>

1.2 TransactionController

URL Exposed	Purpose
/transactions	Fetches all the transactions
Http Method	GET
Parameter	-
Return	List<Transaction>

/transactions	Add a new transaction
Http Method	POST
Request body	Transaction
Return	Transaction

/transactions/{id}	Delete transactions with given transaction id
Http Method	DELETE
Parameter 1	Long (id)
Return	Transaction

/transactions/{id}	Fetches the transaction with the given id
Http Method	GET
Parameter 1	Long (id)
Return	Transaction

/transactions/{id}		Updates existing transaction by id
Http Method	PUT	
Parameter 1	Long (id)	
Request body	Transaction	
Return	Transaction	
/transactions/search		Searches all transaction on type basis
Http Method	GET	
Request param	type	
Return	Transaction	

1.3 UsersController

URL Exposed		Purpose
/users		Fetches all the users
Http Method	GET	
Parameter 1	-	
Return	List<User>	
/users		Add a new user
Http Method	POST	
Request body	User	
Return	User	
/users/{id}		Delete user with given user id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	User	
/users/{id}		Fetches the user with the given id
Http Method	GET	
Parameter 1	Long (id)	
Return	User	
/users/{id}		Updates existing user by id
Http Method	PUT	
Parameter 1	Long (id)	
Request body	User	
Return	User	
/users/search		Searches all users as per name basis
Http Method	PUT	
Request param	name	

Return	User	

1 Template Code Structure

1.1 Package: com.onlinebanking

Resources

OnlineBankingApplication (Class)	This is the SpringBoot starter class of the application.	Already Implemented

1.2 Package: com.onlinebanking.entity

Resources

Class/Interface	Description	Status
Account (class)	o Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. o Map this class with **account** table. o Generate the **Id** using the **IDENTITY** strategy.	Partially implemented.

Class/Interface	Description	Status
Transaction (class)	o Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. o Map this class with **transaction** table. o Generate the **Id** using the **IDENTITY** strategy.	Partially implemented.

Class/Interface	Description	Status
User (class)	o Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. o Map this class with **user** table. o Generate the **Id** using the **IDENTITY** strategy.	Partially implemented.

1.3 PACKAGE: COM.ONLINEBANKING.REPO

Resources

Class/Interface	Description	Status
AccountRepository (interface)	1. Repository interface exposing CRUD functionality for **Account** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented

Resources

Class/Interface	Description	Status
TransasctionRepository (interface)	1. Repository interface exposing CRUD functionality for **Transaction** Entity.	Partially implemented

	2. You can go ahead and add any custom methods as per requirements	

Resources

Class/Interface	Description	Status
UserRepository (interface)	1. Repository interface exposing CRUD functionality for **User** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented

1.4 PACKAGE: COM. ONLINEBANKING.SERVICE

Resources

Class/Interface	Description	Status
AccountService (interface)	Interface to expose method signatures for account related functionality. Do not modify, add or delete any method	Already implemented.
AccountServiceImpl (class)	• Implements **AccountService**. Contains template method implementation. • Need to provide implementation for account related functionalities	To be implemented.

	• Do not modify, add or delete any method signature	

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.
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.

Resources

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

1.5 PACKAGE: COM. ONLINEBANKING.CONTROLLER

Resources

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

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

Resources

Class/Interface	Description	Status

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

2 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 Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal**
3. **To build your project use command:**
   mvn clean package -Dmaven.test.skip
4. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**
   java -jar <application-name>-0.0.1-SNAPSHOT.jar
5. **This editor Auto Saves the code**
6. **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.**

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. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

   Note: The application will not run in the local browser
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**
    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.**