System Requirements Specification Index

For

Mobile Banking Application

Version 1.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

Table of Contents

Mobile Banking Application

System Requirements Specification

PROJECT ABSTRACT

In the fast-paced world of banking, there's a growing demand for modern mobile banking solutions. The CEO of a leading financial institution, Ms. Johnson, challenges a team of developers to create a Fullstack Mobile Banking Management Application.

Your task is to develop a digital solution that empowers users with seamless mobile banking capabilities.

BACKEND-JAVA

1. Problem Statement

The **Mobile Banking Application** is a Java-based RESTful Web API utilizing Spring Boot, with MySQL as the database. The application aims to provide a comprehensive platform for managing and organizing new features for mobile banking.

To build a robust backend system that powers the Mobile Banking Management Application. Here's what the developers need to accomplish:

Following is the requirement specification:

	Mobile Banking Application
1	Mobile Banking
Transaction Module Functionalities	
1	Get all mobile banking transactions
2	Get mobile banking transactions by id
3	Create a new mobile banking transaction
4	Update a mobile banking transaction by id
5	Delete a mobile banking transaction by id

2. Assumptions, Dependencies, Risks / Constraints

2.1 Mobile Constraints

- When fetching mobile transaction by id, if transaction ID does not exist, the service method should throw a “Mobile Banking not found” message in the ResourceNotFoundException class.
- When updating a transaction, if the transaction ID does not exist, the service method should throw a “Mobile Banking not found” message in the ResourceNotFoundException class.

- When removing a transaction , if the transaction ID does not exist, the service method should throw a “Mobile Banking not found” message in the ResourceNotFoundException class.

2.2 Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exceptions 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

- Amount should not be null and must be positive.
- Description should not be blank.
- Username should not be blank.
- Fullname should not be blank.

4. REST ENDPOINTS

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

4.1 Transaction Controller

URL Exposed		Purpose
1. /api/transactions		Fetches all the mobile transactions
Http Method	GET	
Parameter	-	
Return	List<TransactionDTO>	
2. /api/transactions/{id}		Fetches a mobile banking by id
Http Method	GET	
Parameter 1	Long (id)	
Return	TransactionDTO	
3. /api/transactions		Creates a new transaction
Http Method	POST **The transaction data to be created should be received in @RequestBody**	
Parameter	-	
Return	TransactionDTO	
4. /api/transactions/{id}		Updates a transaction by id
Http Method	PUT **The transaction data to be updated should be received in @RequestBody**	
Parameter 1	Long (id)	
Return	TransactionDTO	
5. /api/transactions/{id}		Deletes a transaction by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

5. Template Code Structure

5.1 Package: com.mobilebanking

Resources

Class/Interface	Description	Status
MobileBankingApplication (Class)	This is the Spring Boot starter class of the application.	Already implemented.

5.2 Package: com.mobilebanking.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.	Already implemented.

5.3 Package: com.mobilebanking.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.

5.4 Package: com.mobilebanking.service.impl

Resources

Class/Interface	Description	Status

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

5.5 PACKAGE: COM.MOBILEBANKING.CONTROLLER

Resources

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

5.6 PACKAGE: COM.MOBILEBANKING.DTO

Resources

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

5.7 PACKAGE: COM.MOBILEBANKING.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 **featureId** as primary key. ● Map this class with a **transactions** table.	Partially implemented.

5.8 PACKAGE: COM.MOBILEBANKING.EXCEPTION

Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	● Custom Exception to be thrown when trying to fetch, update or delete the mobile transaction info which does not exist. ● Need to create Exception Handler for same wherever needed (local or global)	Already implemented.

FRONTEND-REACT SPA

1 PROBLEM STATEMENT

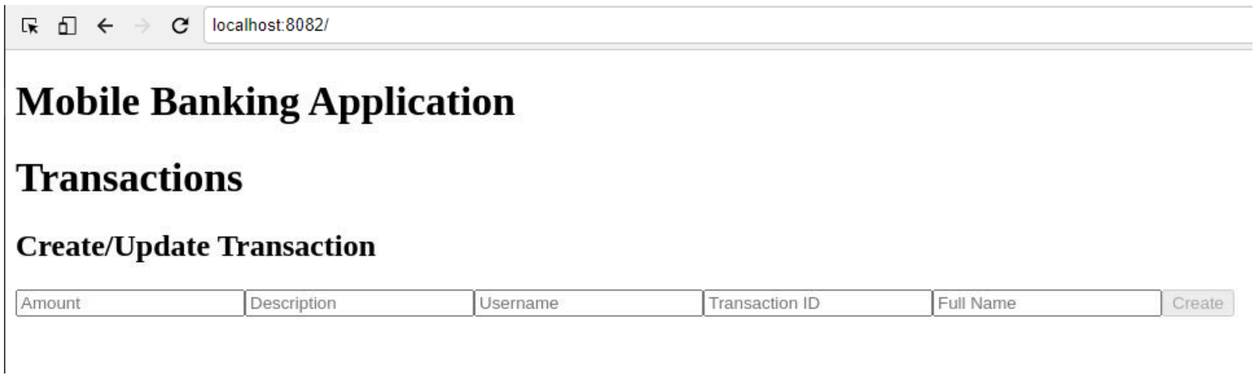
The **Mobile Banking Application** frontend is a Single Page Application (SPA) built using React. Here's what the frontend developers need to achieve:

The frontend should provide an intuitive user interface for easy navigation and efficient mobile banking operations.

2 PROPOSED MOBILE BANKING WIREFRAME

UI needs improvisation and modification as per given use case and to make test cases passed.

2.1 HOME PAGE



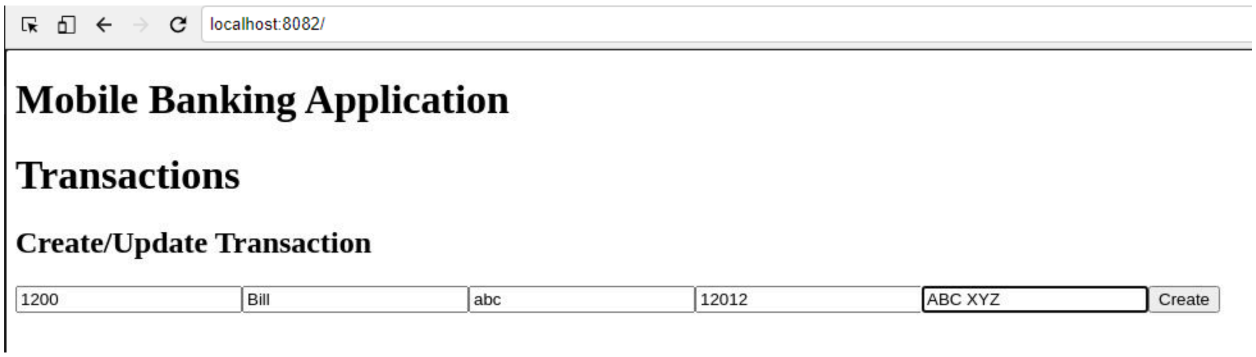
localhost:8082/

Mobile Banking Application

Transactions

Create/Update Transaction

**** write any id as string in "Transaction ID" field like below 12012**



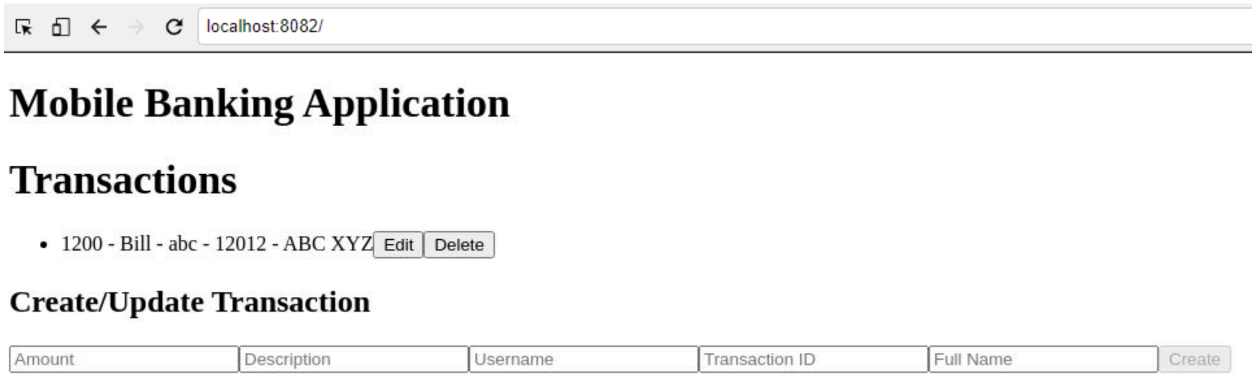
localhost:8082/

Mobile Banking Application

Transactions

Create/Update Transaction

1200	Bill	abc	12012	ABC XYZ	Create



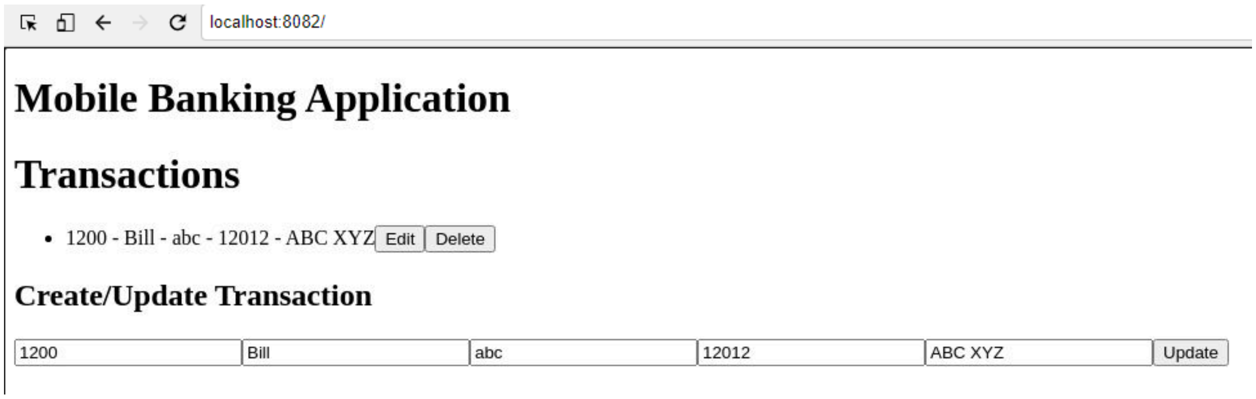
localhost:8082/

Mobile Banking Application

Transactions

- 1200 - Bill - abc - 12012 - ABC XYZ Edit Delete

Create/Update Transaction



localhost:8082/

Mobile Banking Application

Transactions

- 1200 - Bill - abc - 12012 - ABC XYZ Edit Delete

Create/Update Transaction

1200	Bill	abc	12012	ABC XYZ	Update

3 BUSINESS-REQUIREMENT:

As an application developer, develop the Mobile Banking Application (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Home Page	As a user I should be able to visit the Home page as the default page.
US_01	Home Page	As a user I should be able to see the homepage and perform all operations: Acceptance criteria: 1. Should have "Mobile Banking Application" as heading in h2. 2. Should have a "Transactions" as heading in h2. 3. Should show a list of all transactions with "Edit" & "Delete" button in each of the transactions.. 4. As a user I should be able to furnish the following details at the time of creating/updating a transaction. 1.1 Amount 1.2 Description 1.3 Username 1.4 Transaction ID 1.5 Full Name 5. All fields should be required fields to add a transaction 6. Until all fields are filled, create button should be disabled.

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.

Execution Steps to Follow For Frontend

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. This is a web-based application, to run the application on a browser, use the internal browser in the environment.
4. You can follow series of command to setup React environment once you are in your project-name folder:
   a. npm install -> Will install all dependencies -> takes 10 to 15 min
   b. npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:8082 to open project in browser -> takes 2 to 3 min
   c. npm run jest -> to run all test cases and see the summary
   d. npm run test -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min
5. 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.