TECHNICAL SPECIFICATION DOCUMENT

Assignment for MayBank

MARCH 16, 2025

FOR MAY BANK

matharaziz@gmail.com

Contents

1. Introduction

1.1 Purpose

This document aims to deliver a detailed technical specification for the Customer Management CRUD Application, which is built using Spring Boot and React.js. The application enables users to perform Create, Read, Update, and Delete operations on customer data, along with features to create customer accounts, deposit cash, and withdraw cash.

1.2 Scope

This document outlines the architecture, technologies, database schema and APIs implemented in the development of the application. It is intended for developers, testers, and anyone involved in maintaining or extending the system

1.3 Audience

This document is intended for:

- Development team members
- Technical support team
- Future developers maintaining the application

2. System Overview

2.1 Application Description

The Customer Management CRUD application is a web-based solution that enables administrators to manage customer records. The application is built using the Spring Boot framework for the backend, which provides RESTful APIs, and React.js for the frontend, handling the user interface components.

The following REST APIs have been implemented in Spring Boot:

1. **CreateCustomer**: Adds new customer data to the system.
2. **GetCustomers**: Retrieves a list of all customers.
3. **GetCustomerById**: Fetches a specific customer by the provided ID.
4. **UpdateCustomer**: Modifies customer information.
5. **DeleteCustomer**: Removes customer records from the database.
6. **CreateAccount**: Creates a new account for the customer.
7. **CloseAccount**: Closes a given account.
8. **DepositCash**: Deposits cash into an account.
9. **WithdrawCash**: Withdraws cash from an account.

3. Architecture Design

3.1 System Architecture

The system follows a client-server architecture consisting of the following components:

- **Frontend**: React.js application (user interface)
- **Backend**: Spring Boot application (RESTful API)
- **Database**: Relational database (e.g., MySQL, PostgreSQL), currently using an embedded H2 database for testing purposes.

The frontend (React.js) communicates with the backend (Spring Boot) through REST APIs to perform CRUD operations. The Spring Boot backend processes the business logic and interacts with the database to store and retrieve data.

3.2 Technology Stack

- **Frontend**:
  - React.js (JavaScript framework)
  - Bootstrap (for styling)
  - React Router (for navigation)
- **Backend**:
  - Spring Boot (Java framework)
  - Spring Data JPA (for database interactions)
  - Hibernate (ORM for managing relational data)
  - RESTful APIs (for communication with frontend)
- **Database**:
  - H2 (relational database)

4. Database Design

4.1 Database Schema

Customer Table:

Column Name	Data Type	Constraints
Id	BIGINT	AUTO_INCREMENT, PRIMARY KEY
Name	VARCHAR(100)	NOT NULL
Email	VARCHAR(100)	UNIQUE, NOT NULL
Phone	VARCHAR(15)	
date_of_birth	Date	

Account Table

Column Name	Data Type	Constraints
Id	BIGINT	AUTO_INCREMENT, PRIMARY KEY
account_number	VARCHAR(20)	NOT NULL, UNIQUE
balance	DECIMAL(15, 2)	DEFAULT 0.0
account_type	VARCHAR(20)	NOT NULL
status	VARCHAR(20)	DEFAULT 'Active'
customer_id	BIGINT	NOT NULL, FOREIGN KEY (customer_id) REFERENCES customer(id)

5. Backend Design

5.1 Spring Boot Overview

The backend is developed using **Spring Boot**, which simplifies the development of REST APIs. The Spring Boot application uses **Spring Data JPA** to communicate with the database and perform CRUD operations.

5.1.1 Controllers

- **CustomerController**: Handles the HTTP requests for CRUD operations.
  - **GET: /api/bank/status**: Test the server status
  - **POST: /api/bank/createCustomer**: Create a new customer in the system.
  - **GET: /api/bank/getCustomer/{id}**: Retrieve a specific customer by ID.
  - **GET: /api/bank/getCustomers**: Retrieve a list of all customers.
  - **PUT: /api/bank/updateCustomer/{id}**: Update the information of an existing customer by ID.
  - **DELETE: /api/bank/deleteCustomer/{id}**: Delete a customer by ID.
  - **POST: /api/bank/createAccount**: Create a new account for a customer.
  - **POST: /api/bank/depositCash**: Deposit cash into an account.
  - **POST: /api/bank/withdrawCash**: Withdraw cash from an account.
  - **POST: /api/bank/closeAccount**: Close an existing account.
  - **GET: /api/bank/getAccount/{accountNumber}**: Retrieve account details by account number.

These endpoints allow you to manage customers and accounts, including creating, updating, deleting, depositing, withdrawing, and closing accounts.

5.1.2 Services

- **CustomerService**: Contains business logic for managing customer data. This service interacts with the **CustomerRepository** to fetch and modify data in the database.
- **CustomerService**: Contains business logic for accounts data. This service interacts with the **AccountRepository** to fetch and modify data in the database.

5.1.3 Repositories

- **CustomerRepository**: A Spring Data JPA repository interface for interacting with the **Customer** table.
- **AccountRepository**: A Spring Data JPA repository interface for interacting with the **Account** table.

5.2 API Endpoints

HTTP Method	Endpoint	Description
GET	/api/bank/status	Check the server status either it is running or not
POST	/api/bank/createCustomer	Create a new customer in the system.
GET	/api/bank/getCustomer/{id}.	Retrieve a specific customer by ID
GET	/api/bank/getCustomers	Retrieve a list of all customers.
PUT	/api/bank/updateCustomer/{id}	Update the information of an existing customer by ID.
DELETE	/api/bank/deleteCustomer/{id}	Delete a customer by ID.
POST	/api/bank/createAccount	Create a new account for a customer.
POST	api/bank/depositCash	Deposit cash into an account.
POST	/api/bank/withdrawCash	Withdraw cash from an account.
POST	/api/bank/closeAccount	Close an existing account.
GET	/api/bank/getAccount/{accountNumber}	Retrieve account details by account number.

5.3 API Data Flow

- **Frontend (React.js)** sends HTTP requests (GET, POST, PUT, DELETE) to the **Backend (Spring Boot)** API.
- The backend processes the request, interacts with the **database**, and sends a response back to the frontend.

6. Frontend Design

6.1 React Components

- **Home**: Displays a list of customers in a table format with options to edit, delete, or create a new customer.
- **CreateCustomerForm**: A form to add new customers to the system.
- **EditCustomerForm**: A form to update an existing customer's details.

6.2 State Management

React uses the useState and useEffect hooks to manage the state and side effects in the application.

- **State Variables**: customers, loading, error, etc.
- **API Calls**: The application uses **fetch** to make HTTP requests to the backend.

6.3 Routing

React Router is used to handle navigation between different pages:

- **Home** page: Displays the list of customers.
- **CreateCustomerForm** page: For adding new customers.
- **EditCustomerForm** page: For editing existing customers.

7. Security

7.1 Authentication

To secure API endpoints, you can implement JWT (JSON Web Tokens) for authentication, ensuring that only users who are properly authenticated have access to the data.

7.2 Authorization

Additionally, we can introduce authorization to restrict specific actions, like creating or deleting customers, to only authorized users. However, this functionality is not currently implemented

8. Error Handling

- **Backend**: Handle API errors gracefully by returning appropriate HTTP status codes and error messages.
- **Frontend**: Display user-friendly error messages when API calls fail.

9. Testing

10.1 Unit Testing

- **Backend**: Use **JUnit** and **Mockito** for unit testing of service layer

10. Deployment

10.1 Backend Deployment

Here's a detailed guide on how to deploy a Spring Boot application on a machine:

1. Build the Spring Boot Application

Before deploying, you need to build the application using **Maven**

- Open the terminal or command prompt in the root directory (`/assignment`) of the Spring Boot project.

```
mvn clean install
```

This command will clean any previous build and install the required dependencies and packages. It will also create a `.jar` file in the `target` directory

2. Prepare the Server

Ensure that the server where you plan to deploy the application has Java installed.

- **Verify Java installation:**

```
java –version
```

If java is not installed, install it or Set JAVA_HOME variable. **JDK 17** is required for it

3. Run the Spring Boot Application on the Server

Once the .jar file is on the server, navigate to the directory where the file is located.

- **Run the application:**

```
java -jar assignment.maybank-0.0.1.jar
```

If Java is not installed, you can set the `JAVA_HOME` environment variable temporarily for the current Command Prompt session in Windows like this:

```
1. set JAVA_HOME= C:\path\to\java17
2. "%JAVA_HOME%\bin\java" -jar assignment.maybank-0.0.1.jar
```

4. Access the Application

Open a web browser and visit the application using the machine's IP address and the port on which it is running (default is 8089).

http://localhost:8089/api/bank/status

10.2 Front-End Deployment

Here is a detailed guide to deploying a React.js application to a web server or hosting service:

1. Install http-server globally

Using http-server is a simple and effective way to serve a static React.js application locally or on a remote server. It can be useful for testing the production build of your app without needing to configure complex web servers

Install the http-server package globally on your machine using the following command:

```
npm install -g http-server
```

2. Build the React Application

Before deploying, you need to create a production-ready build of your React application.

- Navigate to the root directory of your React project.
- Run the following commands to create an optimized build for production:

```
npm install

npm run build
```

This will generate a build folder containing the optimized and minified static files

After the build is complete, navigate to the build directory (inside your project folder), which contains the production-ready files.

```
cd build
```

3. Run Server

In the build folder, run the following command to start a simple HTTP server:

```
http-server -p 3000
```

This will start a server and the application will be available at the default address (http://localhost:3000).

4. Access the React App

Open a browser and go to http://localhost:3000 Your React application should now be live and accessible locally.

11. Conclusion

This document presents a comprehensive specification for the Customer Management CRUD Application, developed using Spring Boot and React.js. It covers the system architecture, database schema, API design, frontend components, deployment processes, and testing methodologies.