**Technical Specification Document**

**1. Introduction**

**This document outlines the design, development, and implementation of a banking application featuring a Spring Boot backend and a React JS frontend. It encompasses design decisions, data models, entity relationships, error handling strategies, logging practices, security measures, and API documentation.**

**2. Design Decisions**

**2.1 Architecture The application employs a multi-tier architecture:**

**Backend: Spring Boot MVC with RESTful API services.**

**Frontend: React JS for a responsive and interactive user interface.**

**Database: H2 in-memory database for development and testing efficiency.**

**2.2 Technology Stack**

**Backend:**

**Spring Boot: A comprehensive framework for building Java-based applications.**

**Spring Data JPA: Facilitates data access through JPA-based repositories.**

**H2 Database: A lightweight, in-memory database for rapid development and testing.**

**SLF4J: Provides logging capabilities.**

**Frontend:**

**React JS: A JavaScript library for constructing dynamic user interfaces.**

**Axios: Manages HTTP requests to interface with the backend services.**

**2.3 Key Design Considerations**

**Scalability: The design accommodates future enhancements, including additional services or complex banking transactions.**

**Maintainability: Modular code with clear separation of concerns to simplify maintenance and updates.**

**Security: Fundamental security measures are integrated to safeguard data and control access.**

**3. Data Models**

**3.1 Customer Entity**

* **Attributes:**
* id: auto generated(Primary Key)
* name: String

**3.2 Account Entity**

* **Attributes:**
* accountNumber: auto generated(Primary Key)
* customerId: (Foreign Key to Customer)
* accountType: String
* balance: Decimal
* status: String ("Active", "Closed")

**4.2.1 Create Customer**

* **Endpoint:** POST /api/customers
* **Request Body:**

json

Copy code

{

"name": "Azhad"

}

* **Response:**

json

Copy code

{

"id": "1",

"name": "Azhad"

}

**4.2.2 Inquire Customer**

* **Endpoint:** POST /api/customers/get

{

"id": "1"

}

* **Response:**

json

Copy code

{

"id": "1",

"name": Azhad"

}

**4.2.3 Create Account**

* **Endpoint:** POST /api/accounts
* **Request Body:**

json

Copy code

{

"customerId": "1",

"accountType": "Current"

}

* **Response:**

json

Copy code

{

"accountNumber": "1",

"customerId": "1",

"accountType": "Current",

"balance": 0.00,

"status": "Active"

}

**4.2.4 Deposit Cash**

* **Endpoint:** POST /api/accounts/deposit
* **Request Body:**

json

Copy code

{

"accountNumber": "any valid id",

"amount": 100.00

}

* **Response:**

json

Copy code

{

"accountNumber": "any valid id",

"balance": 100.00

}

**4.2.5 Withdraw Cash**

* **Endpoint:** POST /api/accounts/withdraw
* **Request Body:**

{

"accountNumber": "any valid id",

}

json

Copy code

{

"accountNumber": "any valid id",

"amount": 50.00

}

* **Response:**

json

Copy code

{

"accountNumber": "any valid id",

"balance": 50.00

}

**4.2.6 Close Account**

* **Endpoint:** POST /api/accounts/close
* **Request Body:**

{

"accountNumber": "any valid id",

}

json

Copy code

{

"accountNumber": "any valid id"

}

* **Response:**

json

Copy code

{

"accountNumber": "any valid id",

"status": "Closed"

}

**4.2.7 Inquire Account**

* **Endpoint:** POST /api/accounts/get
* request

{

"accountNumber": "any valid id",

}

* **Response:**

json

Copy code

{

"accountNumber": "any valid id",

"customerId": "uuid",

"accountType": "Checking",

"balance": 50.00,

"status": "Active"

}

**5.2 Response Codes**

* **200 OK:** Success responses
* **201 Created:** Resource created
* **400 Bad Request:** Invalid input or parameters
* **404 Not Found:** Resource not found
* **500 Internal Server Error:** General server error

**6. Logging Strategies**

**6.1 Logging Framework**

* **SLF4J with Logback**: For logging across the application.
* **Log Levels:** DEBUG, INFO, WARN, ERROR

**6.2 Logging Practices**

* **Log Key Operations:** Record important actions, such as customer creation or account transactions.
* **Error Logging:** Log exceptions with stack traces to diagnose issues.
* **Sensitive Data:** Avoid logging sensitive information such as customer details or transaction amounts.

java

Copy code

private static final Logger logger = LoggerFactory.getLogger(AccountService.class);

public Account createAccount(Long customerId, String accountType) {

logger.info("Creating account for customer ID: {} with account type: {}", customerId, accountType);

// Implementation

}

public Account deposit(Long accountNumber, double amount) {

logger.info("Depositing amount: {} to account number: {}", amount, accountNumber);

// Implementation

}

java

Copy code

import javax.validation.constraints.NotEmpty;

import javax.validation.constraints.Positive;

public class DepositRequest {

@NotEmpty

private Long accountNumber;

@Positive

private double amount;

// Getters and setters

}