Banking Application: Detailed Project Report

# 1. Introduction

This report details the development of a simple banking application built using Spring Boot. The application provides essential banking functionalities such as creating accounts, depositing funds, withdrawing funds, transferring funds, and managing account status (locking/unlocking). Additionally, the application maintains a transaction history for all financial operations.

# 2. Business Specifications

## 2.1 Overview

The banking application is designed to simulate core banking functionalities. It is targeted at financial institutions needing to manage customer accounts and their associated transactions. The application ensures secure and accurate financial operations while providing transparency through transaction histories and account status management.

## 2.2 Key Business Requirements

- Account Management: Customers should be able to open new bank accounts.  
- Transactions:  
 - Deposits: Customers can deposit funds into their accounts.  
 - Withdrawals: Customers can withdraw funds from their accounts.  
 - Transfers: Customers can transfer funds between accounts.  
- Transaction History: Every transaction (deposit, withdrawal, transfer) must be recorded and retrievable.  
- Account Status Management: The application should allow accounts to be locked or unlocked based on certain conditions. Locked accounts should not be allowed to perform any transactions.

## 2.3 Use Cases

- Create Account: A customer or bank administrator can create a new account.  
- Deposit Funds: A customer deposits money into their account.  
- Withdraw Funds: A customer withdraws money from their account, provided they have sufficient funds.  
- Transfer Funds: A customer transfers money between two accounts.  
- View Transaction History: A customer or bank administrator can view the transaction history for any account.  
- Lock/Unlock Account: A bank administrator can lock or unlock a customer’s account to prevent or allow transactions.

# 3. Technical Specifications

## 3.1 System Architecture

The application is built on a standard Spring Boot architecture, leveraging Spring Data JPA for database interactions, and uses RESTful APIs for communication between the client and server.

## 3.2 Key Components

- Entities:  
 - Account: Represents a bank account with fields for ID, account holder name, balance, and account status.  
 - Transaction: Represents a financial transaction with fields for ID, account ID, transaction type (deposit, withdrawal, transfer), amount, timestamp, and description.  
- DTOs (Data Transfer Objects):  
 - AccountDto: Used to transfer account data between the client and server.  
 - TransactionDto: Used to transfer transaction data between the client and server.  
- Repositories:  
 - AccountRepository: Provides CRUD operations for the Account entity.  
 - TransactionRepository: Provides CRUD operations for the Transaction entity and custom query methods for retrieving transaction history by account ID.  
- Service Layer:  
 - AccountService: Defines the business logic for account creation, transactions (deposit, withdrawal, transfer), and account status management (lock/unlock).  
 - AccountServiceImpl: Implements the AccountService interface, handling all business operations and ensuring that account status checks are performed before any transaction.  
- Mappers:  
 - AccountMapper: Maps between Account and AccountDto.  
 - TransactionMapper: Maps between Transaction and TransactionDto.  
- Controller:  
 - AccountController: Provides REST endpoints for managing accounts and transactions, including creating accounts, performing deposits, withdrawals, and transfers, locking/unlocking accounts, and retrieving transaction histories.

## 3.3 Security Considerations

- Account Locking: Accounts can be locked to prevent unauthorized transactions. Locked accounts cannot perform any transactions until they are unlocked by an authorized user.  
- Transaction Validation: Transactions are validated for sufficient balance (in the case of withdrawals and transfers) and account status (must be active to perform any transaction).

## 3.4 Error Handling

The application provides comprehensive error handling to manage scenarios such as insufficient funds, account not found, and invalid transaction requests. Custom exceptions are thrown and handled gracefully to ensure the system remains robust and user-friendly.

## 3.5 Database Design

The application uses a relational database to store account and transaction data. Key tables include:  
- accounts:  
 - id: Primary key.  
 - account\_holder\_name: The name of the account holder.  
 - balance: The current balance of the account.  
 - status: The status of the account (ACTIVE or LOCKED).  
- transactions:  
 - id: Primary key.  
 - account\_id: Foreign key linking the transaction to an account.  
 - transaction\_type: Type of the transaction (DEPOSIT, WITHDRAWAL, TRANSFER\_IN, TRANSFER\_OUT).  
 - amount: The amount of the transaction.  
 - timestamp: The timestamp of the transaction.  
 - description: A brief description of the transaction.

## 3.6 API Endpoints

- Account Management:  
 - POST /api/accounts/addAccount: Create a new account.  
 - GET /api/accounts/getAccount/{id}: Retrieve account details by ID.  
- Transaction Operations:  
 - PUT /api/accounts/deposit: Deposit funds into an account.  
 - PUT /api/accounts/withdraw: Withdraw funds from an account.  
 - PUT /api/accounts/transfer: Transfer funds between accounts.  
 - GET /api/accounts/{accountId}/transactions: Retrieve transaction history for an account.  
- Account Status Management:  
 - PUT /api/accounts/lock/{accountId}: Lock an account.  
 - PUT /api/accounts/unlock/{accountId}: Unlock an account.

## 3.7 Enhancements and Future Work

- Overdraft Protection: Implementing overdraft protection features to allow or deny transactions based on configurable limits.  
- Multi-Currency Support: Enhancing the system to support accounts in multiple currencies.  
- User Authentication and Roles: Adding authentication and role-based access control to protect sensitive operations (like locking/unlocking accounts) and ensure that only authorized users can perform certain actions.  
- Reporting and Analytics: Implementing reporting features to provide insights into account activity, such as monthly statements and transaction summaries.

# 4. Conclusion

This banking application successfully meets the specified business requirements by providing essential banking functionalities such as account management, transaction processing, and account status control. The system's architecture is robust, with clear separation of concerns across its components, making it easy to extend and maintain. With potential enhancements like overdraft protection and multi-currency support, the application can be scaled to meet more complex banking needs.