Customer Account Management System

Technical Specification Documentation

Naga Vidyadhar

bandinaga.bnv@gmail.com

Table of Contents

1. Introduction

The Customer Account Management System is a Spring Boot application designed to manage banking operations for customers and their accounts. The system enables customer registration, account creation, and financial transactions such as deposits and withdrawals, with comprehensive audit logging.

Key Business Functions:

- Customer management (registration, updates, retrieval)
- Account management (creation, closure)
- Financial transactions (deposits, withdrawals)
- Balance inquiries
- Audit trail for all account operations

2. System Architecture

2.1 Technology Stack

- Framework: Spring Boot
- Database: H2 in-memory database
- ORM: Hibernate with JPA
- API Documentation: OpenAPI/Swagger
- Build Tool: Maven (implied)
- Testing: JUnit 5 with Mockito
- AOP: AspectJ for cross-cutting concerns

2.2 Architecture Pattern

- The system implements a standard N-tier architecture with clear separation of concerns:
- Presentation Layer: REST Controllers
- Business Logic Layer: Service implementations
- Data Access Layer: JPA Repositories
- Domain Model: Entity classes

3. Module Breakdown

3.1 Core Application

- CustomeraccountmanageApplication: Main Spring Boot application entry point

3.2 Configuration Modules

- AopConfig: Enables AspectJ proxy-based AOP
- DatabaseConfig: Configures JPA repositories and entity scanning
- OpenAPIConfig: Sets up Swagger documentation

3.3 Controller Layer

- CustomerController: REST endpoints for customer operations
- AccountController: REST endpoints for account operations

3.4 Service Layer

- CustomerService: Business logic for customer operations
- AccountService: Business logic for account operations and transactions

3.5 Repository Layer

- CustomerRepository: Data access for customer entities
- AccountRepository: Data access for account entities
- AuditLogRepository: Data access for transaction logs

3.6 Utility Modules

- GlobalExceptionHandler: Centralized exception handling
- ResourceNotFoundException: Custom exception for resource not found scenarios
- ErrorResponse: Standardized error response structure

3.7 Aspect Modules

- MethodExecutionTimeAspect: Performance monitoring aspect

4 API Reference

4.1 Customer API Endpoints

Endpoint	**Method**	**Description**	**Success Response**	**Error Response**
/customer/createCustomer	POST	Create new customer	201 (Created)	400 (Bad Request)
/customer/getCustomer/{id}	GET	Retrieve customer by ID	200 (OK)	404 (Not Found)
/customer/getCustomers	GET	Retrieve all customers	200 (OK)	404 (Not Found)
/customer/updateCustomer/{id}	PUT	Update customer by ID	200 (OK)	404 (Not Found)
/customer/deleteCustomer/{id}	DELETE	Delete customer by ID	204 (No Content)	404 (Not Found)

4.2 Account API Endpoints

Endpoint	**Method**	**Description**	**Success Response**	**Error Response**
/account/createAccount	POST	Create new account	201 (Created)	400 (Bad Request)

Endpoint	Method	Description	Success Response	Error Response
/account/depositCash	POST	Deposit cash	200 (OK)	404 (Not Found)
/account/withdrawCash	POST	Withdraw cash	200 (OK)	404 / 400
/account/closeAccount	POST	Close an account	200 (OK)	404 (Not Found)
/account/getAccount/{accountNumber}	GET	Retrieve account details	200 (OK)	404 (Not Found)

5 Transaction Flows

5.1 Customer Registration Flow

1. Client submits POST request to /customer/createCustomer with customer details
2. System validates input data
3. CustomerService creates a new Customer entity
4. CustomerRepository persists the entity
5. System returns customer data with generated ID

5.2 Account Creation Flow

1. Client submits POST request to /account/createAccount with customer ID and account type
2. System validates input data
3. AccountService verifies customer existence
4. System generates unique account number using UUID
5. AccountRepository persists the new account with zero balance
6. AuditLogRepository creates an audit entry for the account creation
7. System returns the created account details

5.3 Cash Deposit Flow

1. Client submits POST request to /account/depositCash with account number and amount
2. AccountService validates account existence and active status
3. System updates account balance by adding deposit amount
4. Changes are persisted in a transaction
5. AuditLogRepository creates an audit entry for the deposit operation
6. System returns updated account details

5.4 Cash Withdrawal Flow

1. Client submits POST request to /account/withdrawCash with account number and amount
2. AccountService validates account existence and active status
3. System verifies sufficient balance
4. If valid, updates account balance by subtracting withdrawal amount
5. Changes are persisted in a transaction
6. AuditLogRepository creates an audit entry for the withdrawal operation
7. System returns updated account details

6 Cross-Cutting Concerns

6.1 Exception Handling

The application implements a global exception handling strategy using Spring's @ControllerAdvice

6.2 Method Execution Time Logging

The system uses AOP to monitor and log execution time for all methods.

6.3 Transaction Management

Service methods are annotated with @Transactional to ensure data consistency.

6.4 Input Validation

The system uses Jakarta Bean Validation annotations on entities and DTOs.

7 Datebase Configuration

7.1 H2 Database

The application uses H2 in-memory database for development and testing.

- spring.datasource.url=jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
- spring.datasource.driverClassName=org.h2.Driver
- spring.datasource.username=sa
- spring.datasource.password=

7.2 Connection Pooling

HikariCP is configured for optimal database connection management.

- spring.datasource.hikari.maximum-pool-size=5
- spring.datasource.hikari.minimum-idle=2
- spring.datasource.hikari.connection-timeout=20000

7.3 JPA Configuration

JPA/Hibernate setting for optimal ORM operation.

- spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
- spring.jpa.hibernate.ddl-auto=create-drop
- spring.jpa.show-sql=true

8 Testing Strategy

8.1 Unit Tests

The application includes comprehensive unit tests using JUnit 5 and Mockito.

8.1.1 Service Layer Tests

- Tests for AccountService methods
- Tests for CustomerService methods
- Mocked repository layer

8.1.2 AOP Tests

- Testing for the aspect functionality

8.2 Integration Tests

The application includes a test configuration for Spring Boot integration tests.

9 Performance Considerations

9.1 Database Optimization

- Optimized HikariCP connection pool settings

- Batch size configuration for Hibernate
- Fetch size optimization for queries

9.2 Server Configuration

Tomcat server is configured for optimal performance.

server:

- port: 8090
- shutdown: graceful

tomcat:

- connection-timeout: 30s
- max-connections: 200
- accept-count: 100

threads:

- max: 200
- min-spare: 10

9.3 Method Execution Monitoring

AOP-based monitoring to identify performance bottlenecks.

10 Appendix API Documentation

The system includes Swagger UI for interactive API documentation.

Access points:

API-docs.path=/api-docs

Swagger-ui.path=/swagger-ui.html

11 Deployment

11.1Build the Spring Boot Application

- Before deploying, need to build the application using Maven.
- Open the terminal or command prompt in the root directory 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

11.2 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 or Set JAVA_HOME variable.
- JDK 17 is required for it.

11.3 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 customeraccountmanage-0.0.1-SNAPSHOT.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:
- set JAVA_HOME= C:\path\to\java17
- "%JAVA_HOME%\bin\java" -jar customeraccountmanage-0.0.1-SNAPSHOT.jar

11.4 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).