Jon Guevara Banking System v0.50

Java NCIII 2025 Batch 3 Programming Course Completion Project

Prepared by:

Jonathan Guevara
Java NCIII 2025 Batch 3

Github:

`https://github.com/guevarajf/java_nciii_2025batch3.git`

Folder:

`Project/BankingSystem/`

TABLE OF CONTENTS:

1 Executive Summary

1.1 System Overview

The **Jon Guevara Banking System v0.50** is a desktop application developed as a project for the Java NCIII 2025 Batch 3 Programming Course completion.

The system demonstrates the basic Java programming competencies through implementation of a banking System using Object-Oriented Programming principles, Swing GUI, and relational database integration.

1.2 Primary Objectives

- Demonstrate mastery of Java Swing GUI development for course completion
- Implement role-based access control mechanisms as practical application
- Showcase database connectivity and transaction management skills
- Apply Model-View-Controller (MVC) architectural pattern in real-world context
- Exhibit professional software development practices acquired during the course

1.3 Target Audience

This technical documentation serves as a reference for:

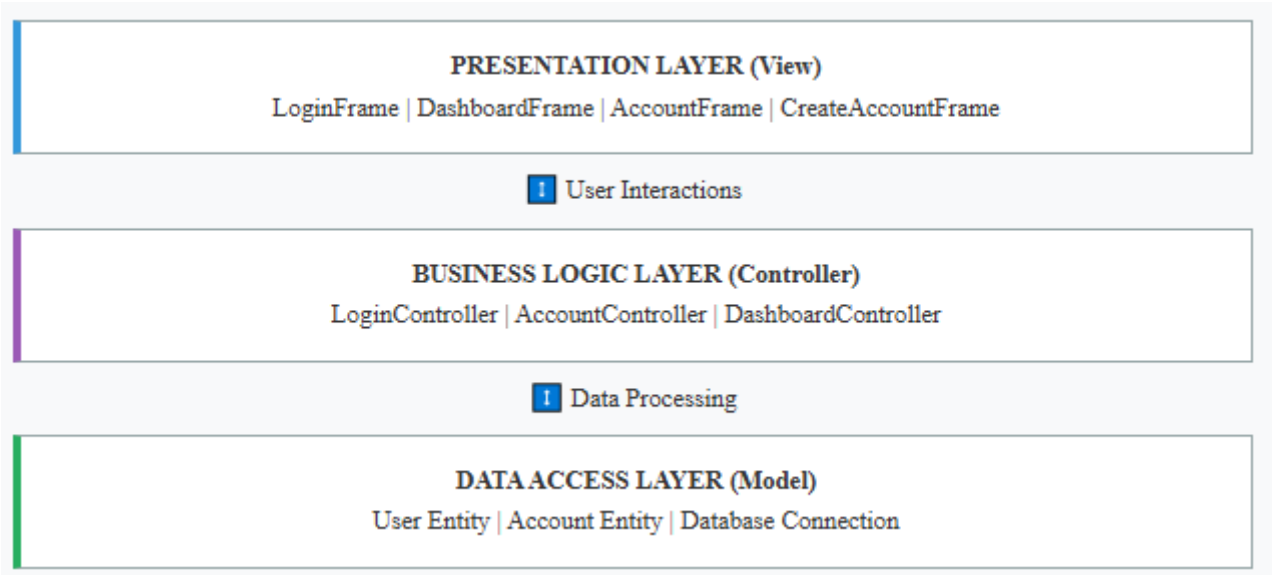
- Java NCIII course instructors evaluating course completion projects
- Academic evaluators assessing course completion requirements

2 System Architecture

2.1 Architectural Pattern

The application follows the Model-View-Controller (MVC) architectural pattern, ensuring clear separation of concerns and maintainable code structure.

MVC Architecture Diagram



2.2 Folder Structure

```
BankingSystem/
├── Main.java
├── README.md
├── UserRoles.xlsx
├── bank.db
├── controller
│   ├── AccountController.java
│   ├── DashboardController.java
│   └── LoginController.java
├── lib
│   ├── javafx.base.jar
│   ├── javafx.controls.jar
│   ├── javafx.fxml.jar
│   ├── javafx.graphics.jar
│   └── sqlite-jdbc-3.50.3.0.jar
├── model
│   ├── Account.java
│   ├── Database.java
│   └── User.java
└── view
    ├── AccountFrame.java
    ├── ChangePasswordFrame.java
    ├── CreateAccountFrame.java
    ├── DashboardFrame.java
    └── LoginFrame.java
```

2.3 Technology Stack

- Programming Language: Java
- GUI Framework: Java Swing
- Database System: SQLite
- Database Connectivity: JDBC (Java Database Connectivity)
- Architecture Pattern: Model-View-Controller (MVC)
- Build System: Manual compilation with classpath dependencies

3 Functional Specifications

3.1 User Role Management System

3.1.1 Role Hierarchy and Permissions

The system implements a four-tier role-based access control mechanism:

User Role	Create Account	Modify Password	Deposit Funds	Withdraw Funds	Balance Inquiry	Account Operations	System Logout
Customer	❌	❌	❌	❌	✅	❌	✅
Teller	✅	❌	✅	✅	✅	✅	✅
Manager	✅	✅	❌	❌	✅	✅	✅
Admin	✅	✅	✅	✅	✅	✅	✅

3.1.2 Test System Credentials

Login	Password	Role
admin	admin	Admin
qqqq	1111	Customer
admin1	Admin1@1	Admin
zzzz	Zzzz1111@	Manager
xxxx	Xxxx1111@	Teller

3.2 Core Banking Operations

3.2.1 Account Management Functions

- **Account Creation**: Automated account generation during first deposit transaction
- **Balance Inquiry**: Real-time account balance retrieval with currency formatting
- **Transaction Processing**: Deposit and withdrawal operations with validation

3.2.2 User Management Functions

- **User Authentication**: Credential verification against database records
- **User Registration**: New user account creation with role assignment
- **Session Management**: Secure session handling with user context preservation

4 Technical Implementation

4.1 Database Schema Design

The system utilizes a normalized relational database schema with two primary entities:

```
-- User Authentication Entity CREATE TABLE users ( id INTEGER PRIMARY KEY
AUTOINCREMENT, username TEXT UNIQUE NOT NULL, password TEXT NOT NULL, role
TEXT NOT NULL );

-- Banking Account Entity CREATE TABLE accounts ( id INTEGER PRIMARY KEY
AUTOINCREMENT, owner TEXT UNIQUE NOT NULL, balance REAL NOT NULL DEFAULT
0.0 );
```

4.2 Security Implementation

4.2.1 SQL Injection Prevention

All database interactions utilize parameterized queries through PreparedStatement objects:

```
PreparedStatement ps = connection.prepareStatement( "SELECT role FROM users
WHERE username=? AND password=?" ); ps.setString(1, sanitizedUsername);
ps.setString(2, sanitizedPassword);
```

4.2.2 Input Validation Framework

- Client-side Validation: Immediate user feedback for invalid inputs
- Server-side Validation: Business logic validation in controller layer
- Data Type Validation: Numeric format verification for monetary amounts
- Required Field Validation: Empty field detection and user notification

4.3 Exception Handling Strategy

4.3.1 Database Exception Management

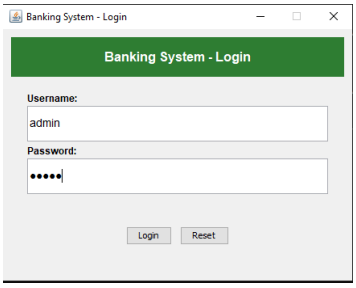
```
try (
    Connection connection = Database.connect();
    PreparedStatement statement = connection.prepareStatement(sql)
) {
    // Execute database operations here
} catch (SQLException exception) {
    exception.printStackTrace();
    return new TransactionResult(false, "Database error occurred", 0.0);
}
```

4.3.2 User Interface Exception Handling

- **Graceful Error Recovery**: System continues operation despite non-critical errors
- **User-Friendly Messages**: Technical exceptions translated to comprehensible messages
- **Resource Cleanup**: Automatic resource deallocation using try-with-resources pattern

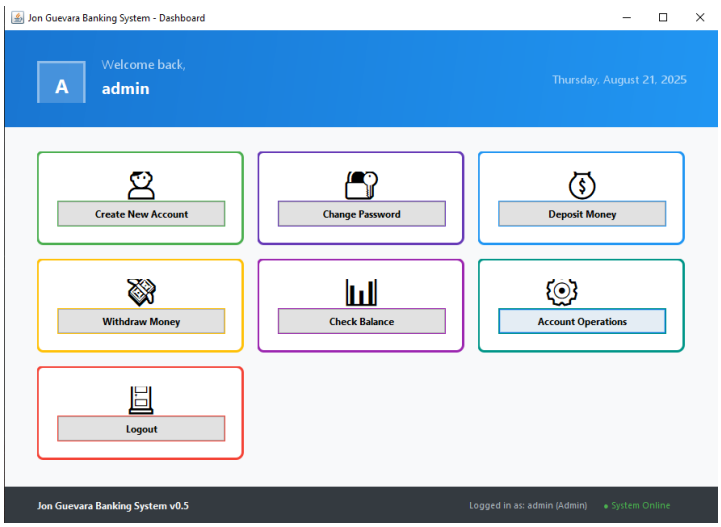
5 User Interface Design

5.1 Login Screen

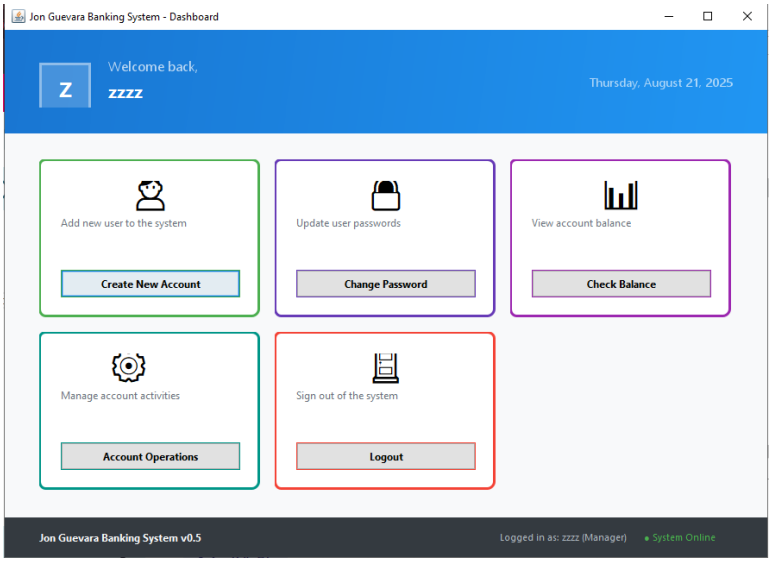


5.2 Dashboard

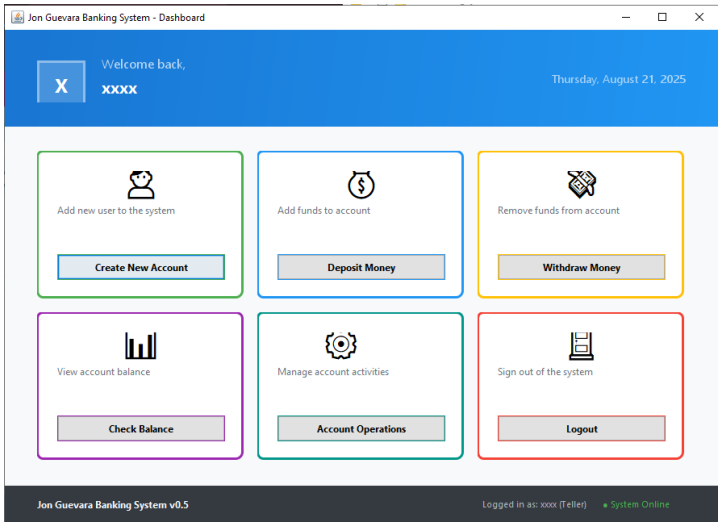
5.2.1 Dashboard - Admin



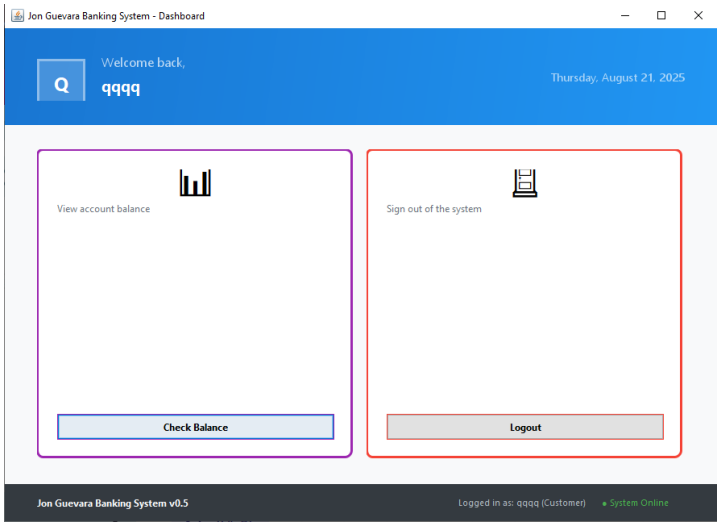
5.2.2 Dashboard – Manager



5.2.3 Dashboard – Teller

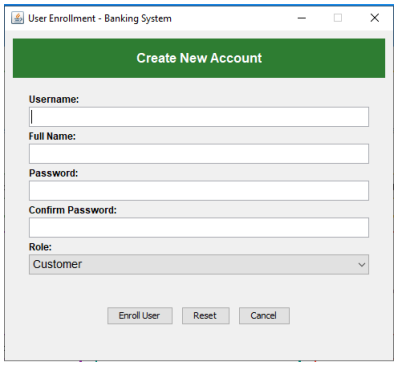


5.2.4 Dashboard – Customer

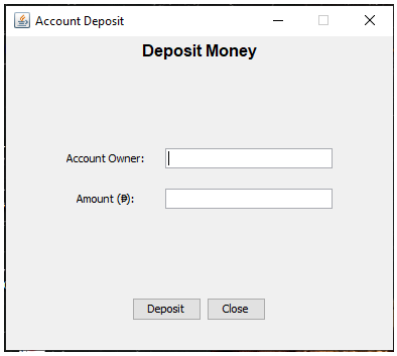


5.3 User Interface

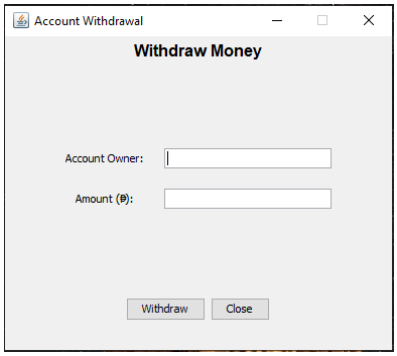
5.3.1 Create New Account



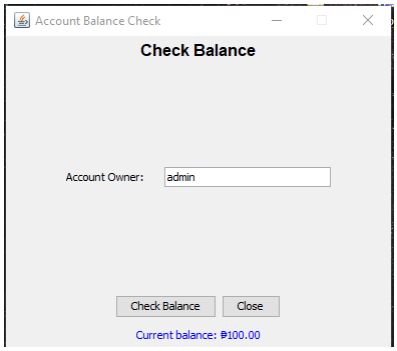
5.3.2 Deposit Money



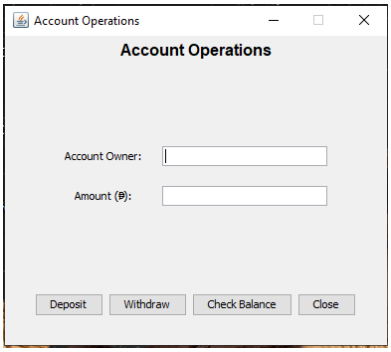
5.3.3 Withdraw Money



5.3.4 Check Balance

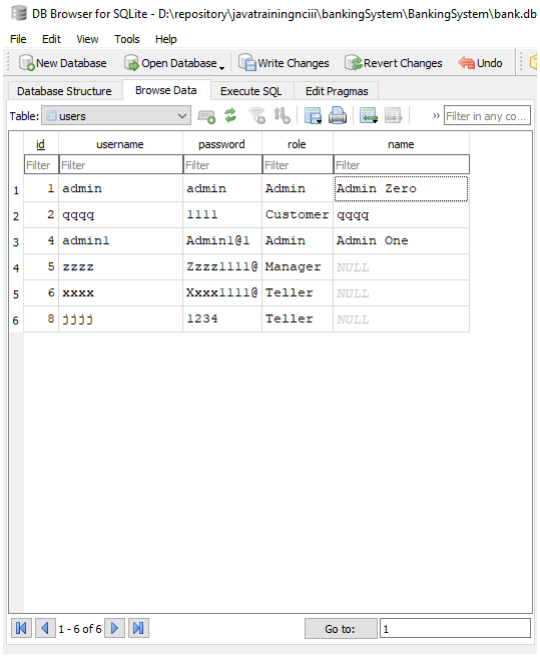


5.3.5 Account Operation



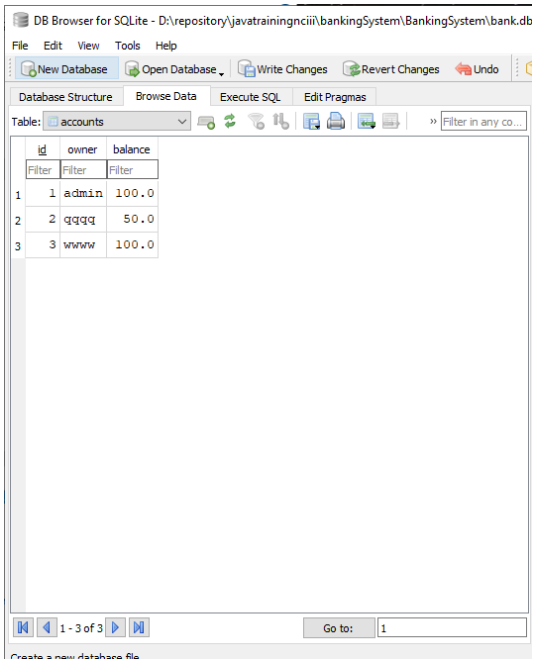
5.4 Database View

5.4.1 Table - Users

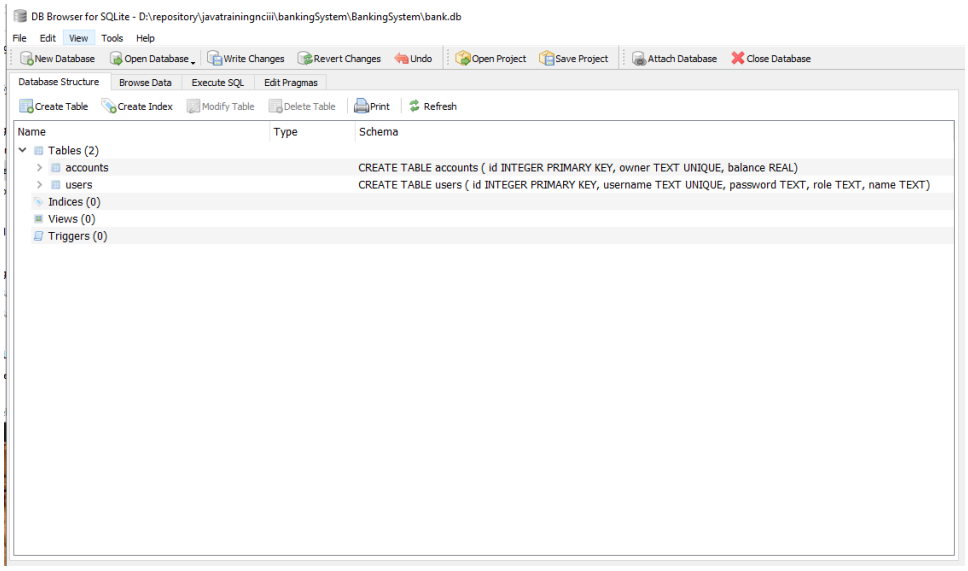


	id	username	password	role	name
1	1	admin	admin	Admin	Admin Zero
2	2	qqqq	1111	Customer	qqqq
3	4	admin1	Admin1@1	Admin	Admin One
4	5	zzzz	Zzzz1111@	Manager	NULL
5	6	xxxx	Xxxx1111@	Teller	NULL
6	8	jjjj	1234	Teller	NULL

5.4.1 Table - Account

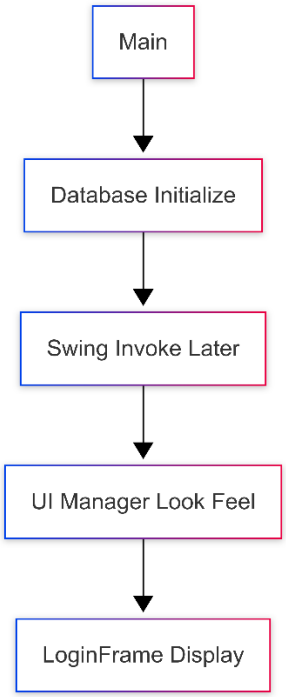


5.4.1 Schema

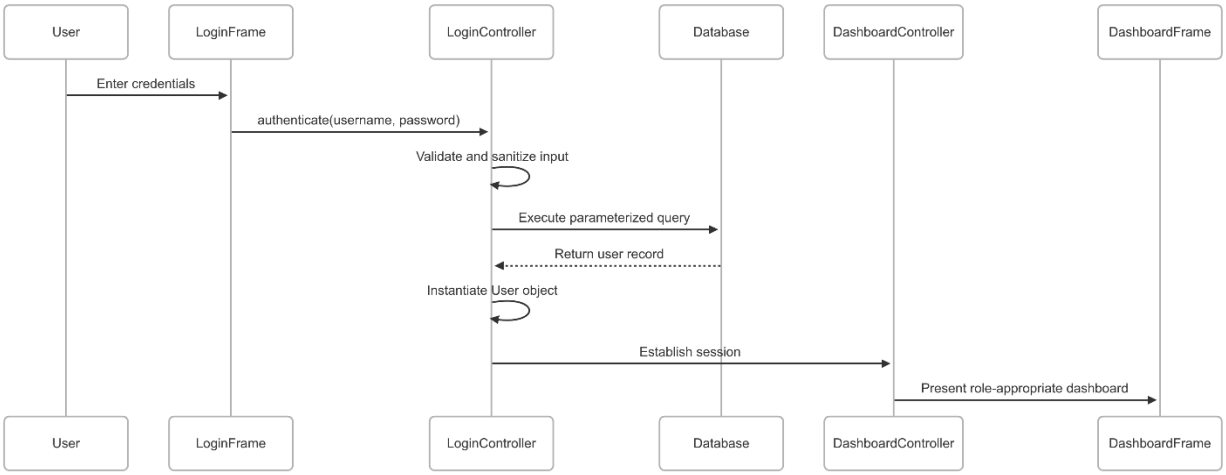


6 Data Flow Architecture

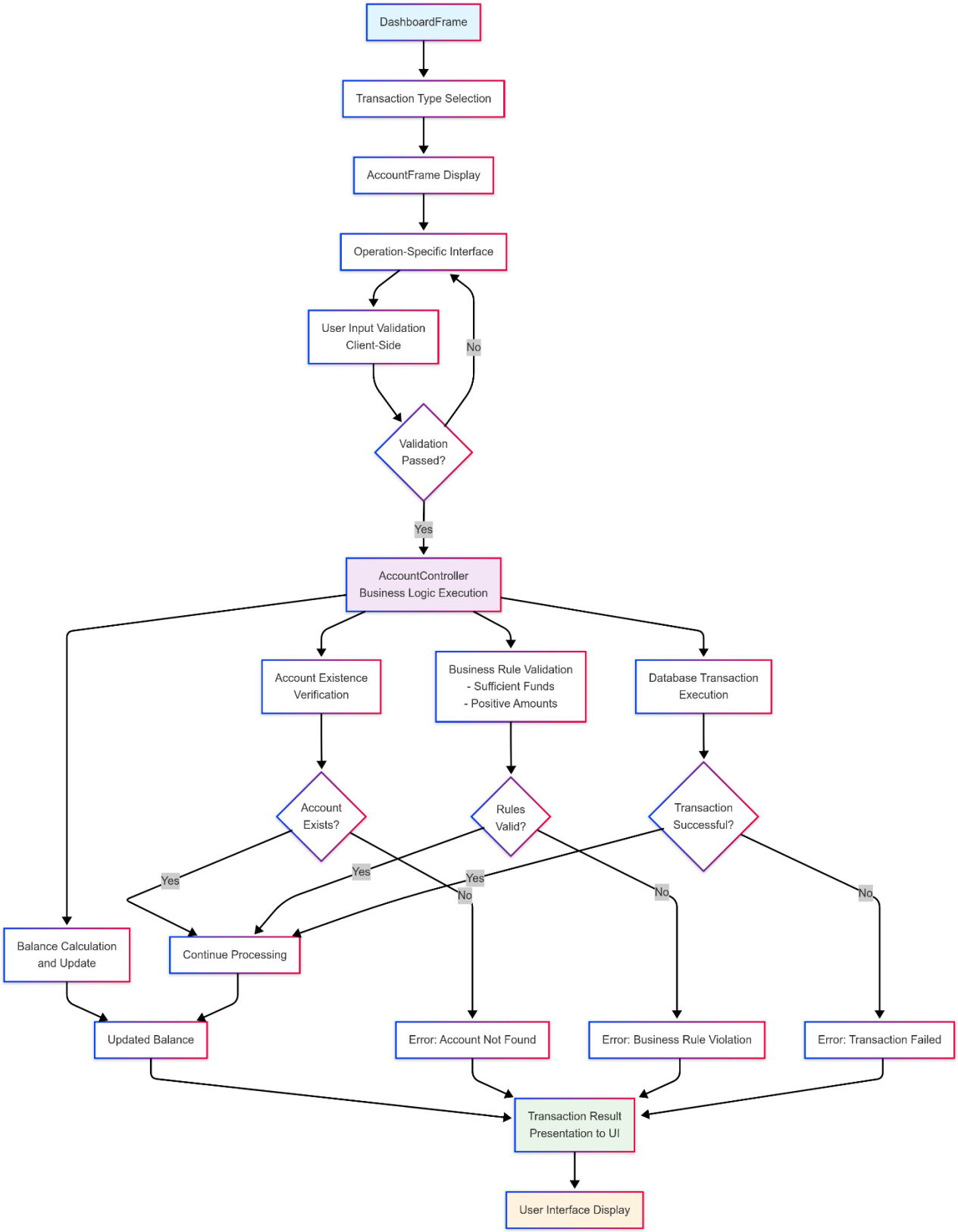
6.1 Application Initialization Sequence



6.2 6.2 Authentication Process Flow



6.3 6.3 Transaction Processing Workflow



7 Java NCIII Course Completion Competency

7.1 Object-Oriented Programming Mastery

7.1.1 Encapsulation Implementation

- **Private Field Access**: All entity fields declared with private visibility modifiers
- **Controlled Access Methods**: Public getter methods providing controlled field access
- **Data Validation**: Setter methods implementing business rule validation
- **Information Hiding**: Internal implementation details concealed from client code

7.1.2 Composition and Aggregation

- **Controller-Model Relationships**: Controllers utilize model classes for data operations
- **View-Controller Dependencies**: User interface components delegate to controller objects
- **Loose Coupling**: Interface-based communication between architectural layers

7.1.3 Static Method Utilization

- **Utility Methods**: Stateless operations implemented as static methods
- **Factory Patterns**: Database connection creation through static factory methods
- **Helper Functions**: Currency formatting and validation utilities

7.2 Database Integration

7.2.1 JDBC API Utilization

- **Connection Management**: Proper database connection lifecycle management
- **PreparedStatement Usage**: Parameterized query execution for security
- **ResultSet Processing**: Efficient data retrieval and processing
- **Transaction Management**: Atomic operation handling with rollback capabilities

7.2.2 SQL

- **Data Definition Language (DDL)**: Table creation with appropriate constraints
- **Data Manipulation Language (DML)**: INSERT, UPDATE, SELECT operation expertise
- **Query Optimization**: Efficient query structure for optimal performance
- **Data Integrity**: Constraint enforcement and referential integrity maintenance

8 System Requirements and Deployment

8.1 Technical Prerequisites

- Java Runtime Environment: JRE
- Operating System: Cross-platform compatibility (Windows, Linux, macOS)
- Memory Requirements: Minimum 512MB RAM, recommended 1GB RAM
- Storage Requirements: 50MB disk space for application and database files

8.2 Installation and Configuration

8.2.1 Compilation Process

a. Navigate to project root directory cd BankingSystem/

b. Compile all Java source files with SQLite dependency

```
javac -cp ".;lib\sqlite-jdbc-3.50.3.0.jar" model\*.java controller\*.java
view\*.java Main.java
```

c. Execute application with proper classpath configuration

```
java -cp ".;lib\sqlite-jdbc-3.50.3.0.jar" Main
```

8.2.2 Database Initialization

- **Automatic Schema Creation**: Database tables created automatically on first execution
- **Default Data Population**: Administrative user account created during initialization
- **File Location**: Database file (bank.db) created in application root directory
- **Backup Considerations**: Database file can be copied for backup purposes

**** END OF DOCUMENT ****