# Banking System Project Documentation

## Table of Contents



## 1. Introduction

The Banking System is a client-server application developed using C++ and the Qt framework. It provides a secure platform for users to perform various banking operations and for administrators to manage user accounts.

### 1.1 Project Objectives

- Develop a robust client-server banking system
- Implement secure user authentication
- Provide basic banking operations for users
- Offer administrative capabilities for account management
- Ensure data integrity and system reliability

### 1.2 Key Features

- User and Admin login
- View account balance
- View transaction history
- Make transactions
- Transfer money between accounts
- Administrative functions (view database, create/delete/update users)

## 2. System Architecture

The Banking System follows a client-server architecture:

### 2.1 Server-side Components

- TCP Server: Handles client connections and requests

- Database Manager: Manages user and admin databases
- Request Processor: Processes client requests and generates responses
- Logger: Records system activities and transactions

### 2.2 Client-side Components

- GUI: User interface for login and banking operations
- Network Client: Manages communication with the server
- Request Manager: Formats and sends requests to the server
- Response Handler: Processes and displays server responses

## 3. Class Diagrams

[Insert the class diagram generated from the PlantUML code provided earlier]

### 3.1 Server-side Classes

- Server: Main server class handling connections and requests
- DatabaseManager: Manages database operations
- RequestProcessor: Processes various types of client requests
- Logger: Handles logging of system activities

### 3.2 Client-side Classes

- Client: Manages network communication with the server
- LoginWindow: Handles user authentication
- MainWindow: Main interface for user and admin operations

## 4. State Diagrams

[Insert the state diagram generated from the PlantUML code provided earlier]

### 4.1 Client Application States

- Not Connected
- Connecting
- Connected
- Logged Out
- Logging In
- Logged In (with sub-states: Idle, Processing Request)

## 5. Sequence Diagrams

### 5.1 Login Process

[Insert a sequence diagram for the login process]

### 5.2 Transaction Process

[Insert a sequence diagram for a typical transaction]

# 6. Database Structure

## 6.1 User Database (user_database.json)

```
{
    "transactions": [
        {
            "account_number": "123456",
            "amount": 100,
            "date": "2024-07-17"
        },
        {
            "account_number": "123456",
            "amount": -50,
            "date": "2024-07-16"
        }
    ],
    "users": [
        {
            "account_number": "123456",
            "age": 25,
            "balance": 1000,
            "fullname": "User One",
            "password": "pass1",
            "username": "user1"
        },
        {
            "account_number": "234567",
            "age": 30,
            "balance": 2000,
            "fullname": "User Two",
            "password": "pass2",
            "username": "user2"
        },
        {
            "account_number": "10201",
            "balance": 621525,
            "password": "2010",
            "username": "ali"
        }
    ]
}
{
    "admins": [
        {
            "username": "admin",
            "password": "adminpass",
            "account_number": "admin001",
            "fullname": "Admin User",
            "age": 35
        }
    ]
}
```


7. API Documentation
   7.1 Client-Server Communication Protocol

All requests and responses are formatted as strings with colon-separated values.

7.2 Request Types

LOGIN: username:password
GET_ACCOUNT_NUMBER: username
VIEW_BALANCE: username
VIEW_TRANSACTION_HISTORY: username:count
MAKE_TRANSACTION: account_number:amount
TRANSFER_AMOUNT: from_account:to_account:amount
VIEW_DATABASE: user_type
CREATE_USER: user_type:username:password:account_number:initial_balance
DELETE_USER: user_type:account_number
UPDATE_USER: user_type:username:new_password:new_balance

7.3 Response Format

Responses are prefixed with the request type or status, followed by a colon and the response data.

8. User Guide

8.1 Logging In

Launch the client application
Enter your username and password
Select "User" as the user type
Click "Login"

8.2 Viewing Account Balance

Select "View Balance" from the dropdown menu
Click "Send Request"

8.3 Making a Transaction

Select "Make Transaction" from the dropdown menu
Enter the transaction amount
Click "Send Request"

8.4 Transferring Money

Select "Transfer Amount" from the dropdown menu
Enter the recipient's account number and the amount
Click "Send Request"

8.5 Viewing Transaction History

Select "View Transaction History" from the dropdown menu
Enter the number of transactions to view
Click "Send Request"

9. Admin Guide

9.1 Logging In as Admin

Launch the client application
Enter your admin username and password
Select "Admin" as the user type
Click "Login"

9.2 Viewing User Database

Select "View Database" from the dropdown menu
Click "Send Admin Request"

9.3 Creating a New User

Select "Create User" from the dropdown menu
Fill in the user details (username, password, account number, initial balance)
Click "Send Admin Request"

9.4 Deleting a User

Select "Delete User" from the dropdown menu
Enter the username to delete
Click "Send Admin Request"

9.5 Updating a User

Select "Update User" from the dropdown menu
Enter the username, new password, and new balance
Click "Send Admin Request"

10. Security Considerations
    10.1 Password Handling

Passwords are currently stored in plain text. In a production environment, they should be hashed and salted.

10.2 Data Transmission

All data is currently transmitted in plain text. Implement SSL/TLS for secure communication in a production environment.

10.3 Input Validation

Implement thorough input validation on both client and server sides to prevent injection attacks and ensure data integrity.

11. Testing Strategy
    11.1 Unit Testing

Implement unit tests for individual classes and methods using Qt Test framework.

11.2 Integration Testing

Test the interaction between client and server components.

11.3 System Testing

Perform end-to-end testing of all banking operations.

11.4 Security Testing

Conduct penetration testing to identify potential vulnerabilities.

12. Deployment Instructions
    12.1 Server Deployment

Compile the server application using Qt Creator or qmake.
Ensure the database files (user_database.json and admin_database.json) are in the same directory as the server executable.
Run the server executable.

12.2 Client Deployment

Compile the client application using Qt Creator or qmake.
Distribute the client executable along with necessary Qt libraries.
Ensure users have the correct server IP address and port configured.

13. Future Enhancements

Implement multi-factor authentication
Add support for different types of accounts (e.g., savings, checking)
Implement a more sophisticated transaction history system
Develop a mobile application for the banking system
Implement real-time notifications for account activities

14. Conclusion
    This Banking System project demonstrates a basic implementation of a client-server banking application using C++ and Qt. While it provides core functionality, there are many areas for improvement and expansion to make it production-ready.```