Bank Client – Server System
FINAL PROJECT
Documentation

PRESENTED BY :

Youssef Mohamed Hamdy

INTRODUCTION

1. Purpose

The purpose of this program to facilitate the handling of Bank Database and handling it as an admin user or to manipulate the account as a standard bank user.

2. Scope

This project supports the following operations:

- Get Account Number.
- View Account Balance.
- View transaction history.
- Make Transaction (standard user).
- Transfer Balance from standard user (logged in) account to another stander user account.
- View Bank whole database (Admin User).
- Create New User (Admin User).
- Delete User (Admin User).
- Update User (Admin User).
- All data Fetched and manipulated is set to JSON file.
- Server can handle multiple clients, but one request at a time.

Architecture

1. System Architecture

This project is based on Server Client Model, Where the client act as a terminal for the user and take inputs from the user and handles it and send it in a specific manner to the server. And the server handles all requests and manipulates all the client requests and send response back to the server.

2. Communication protocol

In this project communication between the server and the client is based on TCP/IP Socket. Using a specified IP Address and a specific PORT.

SECURITY

1. Authentication

In this Bank system User (Standard/Admin) must login before Opening the menu for them. To login user much choose either if they are a standard user or admin user. Then the credentials would be checked by the server if correct user will be allowed to use the application.

2. Authorization

Based on the User mode choice (Standard/User), A different menu will appear giving him the authority to do some operations.

SERVER

1.Server Setup

To make sure server is running smoothly, please provide **JSON File Path & Name found in JsonHandler.h** file. Run application from Qt6.6 preferably.

2.Database Integration

Json Database is manipulated through a utility class "JsonHandler" which provides general functions which manipulates the database based on certain arguments given to the functions. Each function is locked by a mutex to ensure thread safety which unlocks automatically when the mutex is out of scope.

- static bool CheckUserFound(const QString &fileName, const QString &UserName);

Description: This function Check if a username is found in a .json file.

Arguments: json file to search in, Username to search by it.

Return: bool either true (found) or false (not found).

- bool CheckLogin(const QString &fileName, const QString &UserName, const QString &UsrPassword)

Description: This function Check for login credentials of a specific user where it matches the username & password given to the existing data found in the database.

Arguments: json file to look in, username used in logging in, password used in logging in.

Return: bool either true (correct credentials) or false (incorrect credentials).

- bool GetField(const QString &fileName, const QString &UserName, const QString &Field, QString &FieldValue)

Description: This is a general function which gets the value a specific json field of a specifc json object.

Arguments: json file, username to fetch data of, required field to fetch, Variable to assign the field's value to.

Return: true if operation done successfully or false if not.

- bool SetField(const QString &fileName, const QString &UserName, const QString &Field, const QString &NewValue)

Description: This is a general function which Sets the value of a specific field of a specific json object.

Arguments: json file, username to set field's new value, field to set, new field value.

Return: bool true if done successfully or false if not.

- bool PrintDataBase(const QString &fileName, QString &JsonDataBase)

Description: This is a general function which Gets all data found in a json file.

Arguments: json file to get its data, Variable to assign the data of the json file in it.

Return: bool true if done successfully or false if not.

- bool DeleteUser(const QString &fileName, const QString &UserName)

Description: This is a general function which delete an entire json object from json file.

Arguments: filename to delete object in it, username to delete.

Return: bool true if done successfully or false if not.

- bool SetTransactionHistory(const QString &filename, const QString &UserName, const QString &Transaction)

Description: This function is used to only set “Transaction History” field specifically.

Arguments: filename of the desired json file, Username to set his transaction history, Transaction amount string with the sign.

Return: true if set successfully or false if not.

- bool GetTransactionHistory(const QString &filename, const QString &UserName, const QString Count, QString &HistoryData)

Description: This function is used to only Get “Transaction History” field specifically with certain count.

Arguments: filename of the desired json file, Username to get his transaction history, transaction count to show, variable to set the transaction data in it.

Return: true if done successfully or false if not.

- bool CreateNewUser(const QString &fileName, const QString &NewUserName, const QVariantMap &UserData)

Description: This function is used to create a new user and set it in the json file.

Arguments: json file to create new user in, Username which must be unique, a map variable which contains all new user data.

Return: true if created successfully or false if not.

- QString AccountNumberGenerator(const QString& UserName)

Description: This function is used to generate a unique account number based on a unique string.

Arguments: Username to create his account number.

Return: account number as a string.

3.Core Banking functions

- Void Admin::CreateNewUser(void)

<u>Description</u>: This function is used to create a new user based, Admin can choose the type of the new user (Standard/Admin), Username which must be unique so a request is sent to the server to check if user name is available or not, if available other parameters such as full name, password ... are given by the user.

Then data is sent to the server which generates a unique account number then insert the new data to the specified file. Then it shows for the admin user if created successfully of not.

- Void Client::GetBalance(void)

<u>Description</u>: This function is used to get the balance of a specific field, but first it's checked if username is found or not (in Admin Mode), send request for the server then receives the Balance, then it's shown to the user.

- Void User::MakeTransaction(void)

<u>Description</u>: This function is used to let the standard user choose either to deposit or withdrawal, if withdrawal is chosen user must have sufficient amount to cover the operation. Then Request is sent to the server & server save it in the transaction history. Then it's shown to the user the status of the operation.

- Void Client:: ViewTransactionHistory(void)

<u>Description</u>: This function takes count to show the most recent transaction based on the count taken, if the count takes more than the transaction all transaction are printed, if admin user is logged in a valid user must be enter to show the transaction history.

Client

1.Client setup

Make sure port is set to 1234, and IP Address is set to local host. Run application from Qt6.6 preferably.

2. User Interface

Review these videos for user interface

For Standard user interface: https://youtu.be/OTOqlHKx8uM

For Admin user interface: https://youtu.be/yott6ZSR4OI

3. Client – Side Validation

The client makes sure that user choose a valid option for the request to send, and the inputs are as they should be entered.

Maintenance

These are some points that will be implemented in upcoming versions:

- User must press enter twice to login, bug with QTextStream to be solved.
- Server can handle multiple clients at the same time, but one request at a time. Threads to be implemented to solve this.
- Implement clear screen for more human interface.
- Implement GUI for the Client interaction.

Conclusion

In conclusion, the development of the bank server – client program represents a stride towards providing efficient banking services. The thorough documentation outlined above serves as a comprehensive guide for understanding the architecture, security measures and functionalities of the system. By implementing authentication and authorization mechanisms, we prioritize the confidentiality and integrity of sensitive financial data.

The server's integration with the database ensures reliable storage and retrieval of banking information, while client's intuitive user interface empowers users to perform essential transaction with ease. With testing server and client functions, we ensure that system is reliable.