

FINAL PROJECT REPORT

SUBMITTED BY:

|  |  |
| --- | --- |
| **NAME** | **REGISTRATION NO** |
| HASNAIN WAZIR | 2024222 |
| ABDULLAH MUNIR | 2024042 |

SUBMITTED TO:

**SIR AHMED NAWAZ**

# Table of Content:

1. Introduction and Problem Statement………………………………….….3
2. System Design and Architecture...........................................3 - 4
3. Application of OOP Concepts in C++……………………………….........4
4. Implementation Details and Challenges………………………….…4 - 5
5. Screenshots or Outputs of the System……………………………….5 - 9
6. Conclusion and Future Improvements…………………………………....9

**Bank Management System in C++ Using OOPS**

# Introduction and Problem Statement:

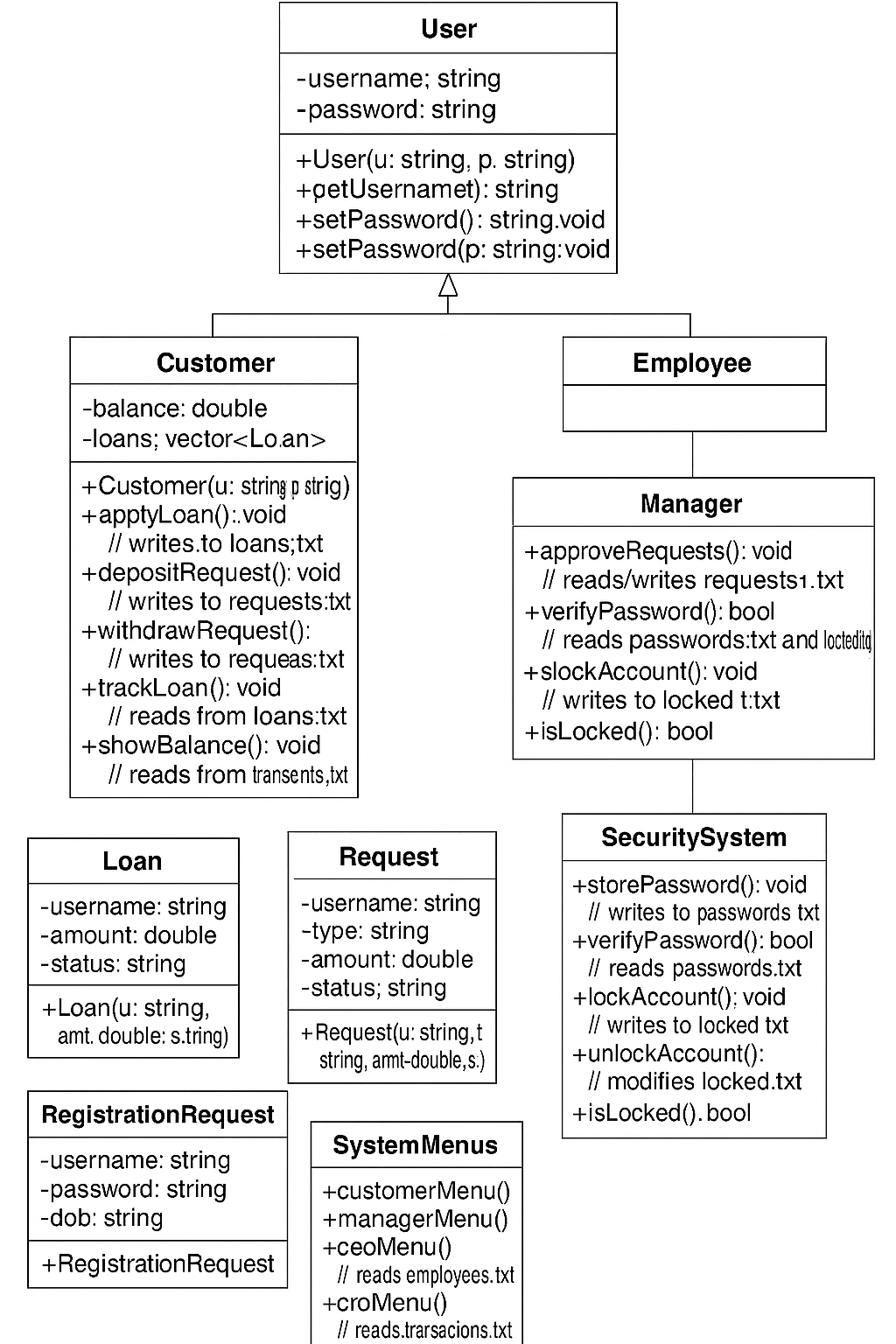
The Bank Management System (BMS) is a console-based software designed to streamline and manage banking operations using Object-Oriented Programming (OOP) in C++. The main aim of this project is to demonstrate how core banking functions like customer registration, account management, transaction processing, and employee role-based operations can be implemented efficiently using OOP principles. This system supports multiple roles like customers, managers, and higher-tier employees like CEO, CFO, CRO, and CISO, each with specific access and control features.

# System Design and Architecture:

The architecture of the BMS consists of multiple classes representing different roles and functionalities:

* User (base class)
* Customer, Employee (derived from User)
* Manager (derived from Employee)
* Separate functions for each employee role (CEO, CFO, CRO, CISO)
* File-based storage for persistence (e.g., users.txt, transactions.txt)

# UML Diagram:



# Application of OOP Concepts in C++:

The BMS project makes use of several key OOP concepts:

## Encapsulation:

Data and functions are encapsulated within classes like `Customer`, `Manager`, etc.

Inheritance:

Classes like `Customer` and `Employee` inherit from the base class `User`. And then again another class `Manager` is inherited from class `Employee`.

Polymorphism:

While not heavily used, role-based behavior switching simulates polymorphism.

Abstraction:

Role-based functions abstract away implementation details from the user interface.

# Implementation Details and Challenges:

The system was implemented in C++ using standard input/output streams and file handling for data persistence.

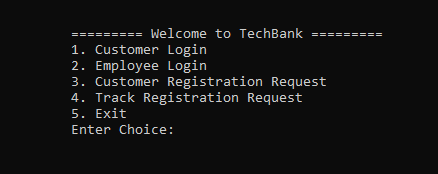
Key challenges included:

* Managing user sessions securely using simple file encryption.
* Ensuring correct role-based access to sensitive functions (e.g., loan approval, account lock).
* Tracking and managing registration requests and transaction records.
* Maintaining data integrity across multiple text files without using databases.

# Screenshots or Outputs of the System:

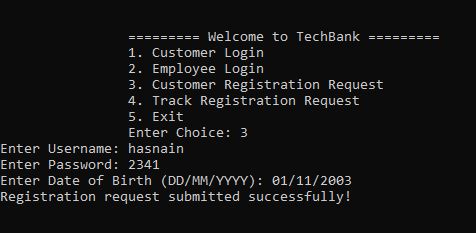
1. Picture One:

“It is Main Menu , which is displayed when we run the program.”

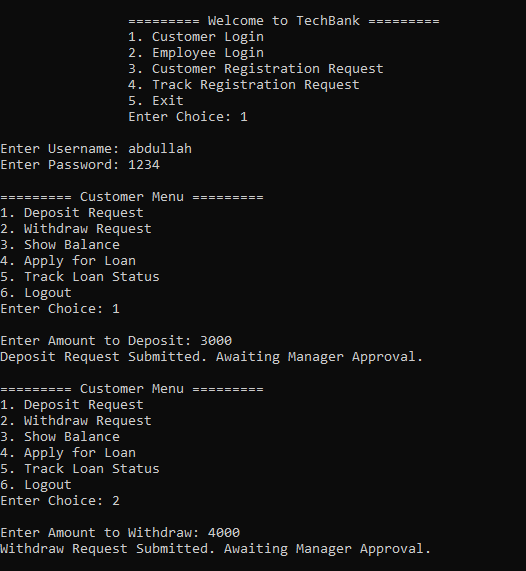


1. Picture Two:

“Here User is Requesting for opening a New Account”



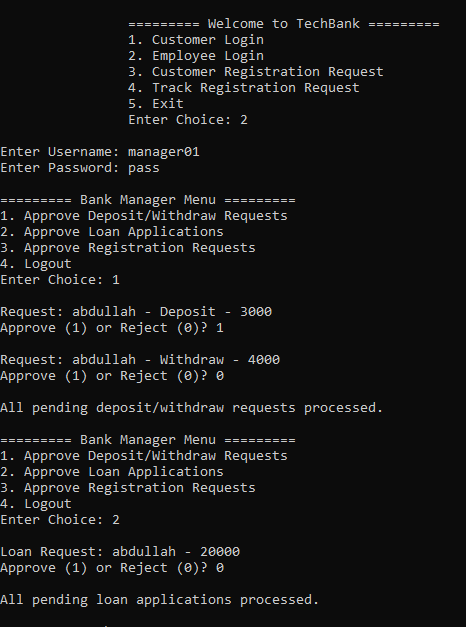
1. Picture Three:

“Here Customer logged in and then is requesting a transaction of two types, withdraw and deposit. While both are on hold until Manager approves/rejects the

1. Picture Four:

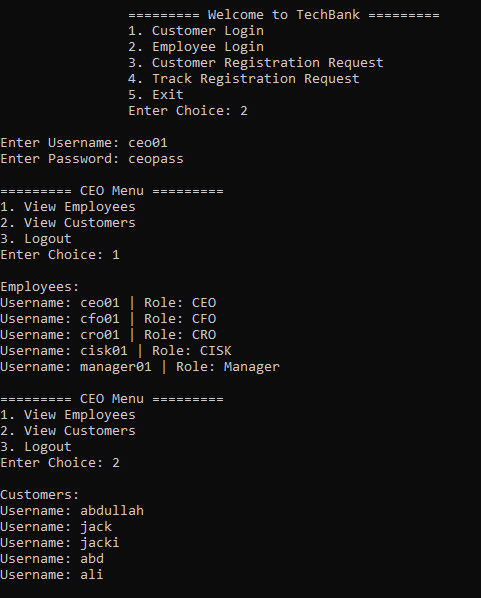
“Manager Menu , where he can accept/reject several things like:

* Withdraw/Deposit Requests
* Loan Requests
* New Account Registration Request”



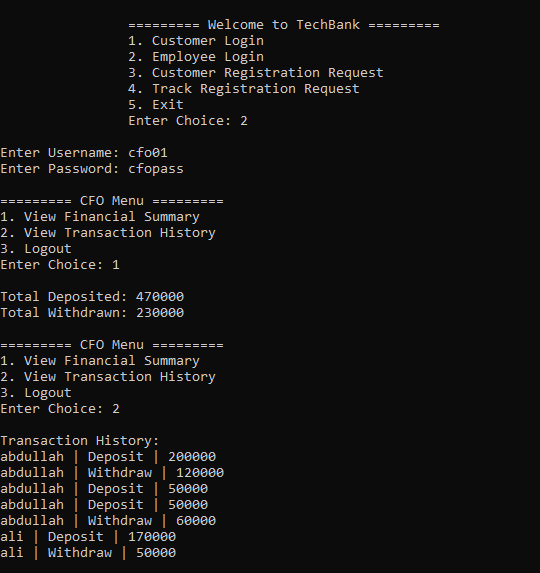
1. Picture Five:

“CEO Menu, where CEO can view all employees and customers list”

:

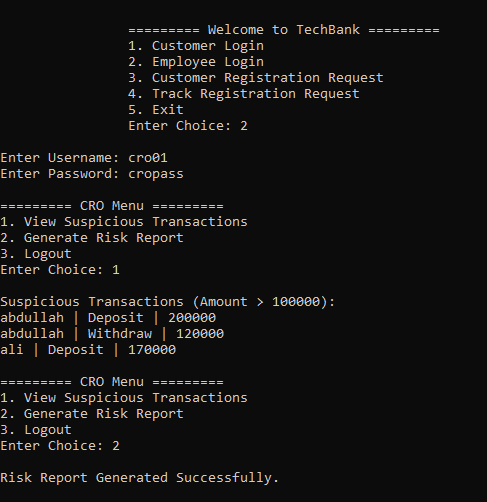
1. Picture Six:

“CFO Menu, where CFO can view Transaction History and Financial Summary”



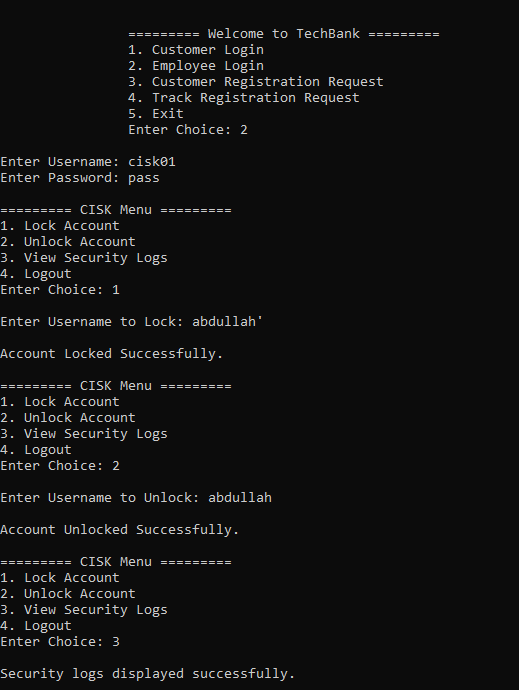
1. Picture Seven:

“CRO Menu, where CRO can view Suspicious Transactions and Risk Report”



1. Picture Eight:

“CISO Menu, where CISO can Lock and Unlock account sand can see Security Logs ”



1. Picture Nine:

“Here customer can check their Application’s Status”



# Conclusion and Future Improvements:

The Bank Management System successfully demonstrates the implementation of a functional banking backend system using OOP concepts. It effectively handles customer and employee workflows and ensures modular code structure.  
Future improvements may include:

* Migrating to a database system for better scalability and data integrity.
* Adding a GUI or web-based frontend.
* Implementing encryption using secure libraries (like hashing etc).
* Adding automated test cases and error logging.