**Banking System - OOP Project Report**

**Group:**

**Shivam 24K-0957**

**Jeevan 24K-0821**

**Yogita 24K-0929**

**1. Executive Summary**

**Overview:**

This project is a **Bank Management System** developed in C++ using Object-Oriented Programming (OOP) principles. It simulates real-world banking processes including money transfer, withdrawals, deposits, and account maintenance. Two user roles are supported: **Admin** and **Account Holder**, with various features such as encrypted passwords, file-based data storage, account age-based services, and a graphical user interface using **Raylib**.

**Key Findings:**

* Demonstrated core OOP concepts: encapsulation, abstraction, inheritance, and polymorphism
* Implemented secure, file-based record keeping and password encryption
* GUI makes banking operations user-friendly
* Integrated date/time logic to enable **account age–based services** (e.g., loan eligibility, bonus facilities)

**2. Introduction**

**Background:**

Banking systems are a classic example of OOP applicability due to their hierarchical structure and need for secure, modular design. This project uses C++ to simulate a functional bank management system, applying all OOP principles and enhancing user experience through GUI.

**Project Objectives:**

* Develop an OOP-based banking system with modular class structures
* Implement real-world features like account/card types, transactions, and admin control
* Secure data using file handling and encryption
* Use date/time logic for dynamic features
* Build an interactive GUI using Raylib

**3. Project Description**

**Scope:**

**Included Features:**

* User roles: Admin and Account Holder
* Account types: Saving and Current (via inheritance)
* Card types: Credit and Debit Cards (via inheritance)
* Transaction tracking
* Password encryption with a secret key
* GUI for login, transfers, etc.
* Account age calculation and conditional services

**Technical Overview:**

* **Language:** C++
* **IDE:** Visual Studio / Code::Blocks / VS Code
* **Libraries:** Raylib (GUI), chrono (date/time)
* **File Handling:** For user and transaction data
* **Security:** Password encryption using custom key logic

**4. Methodology**

**Approach:**

Agile-like approach:

* Start with CLI version for core logic
* Add GUI features after logic testing
* Weekly task planning and review
* Progressive feature testing and integration

**Roles and Responsibilities:**

**Shivam** - Classes and logic

**Yogita** - File Handling

**Jeevan** - GUI Handling

**5. Project Implementation**

**Design and Structure:**

* Classes:
  + Account → SavingAccount, CurrentAccount
  + Card → DebitCard, CreditCard
  + Transaction, Admin, AccountHolder, Bank
* GUI:
  + Login, deposit, withdraw, transfer, view transaction history
  + Admin dashboard for user control
* Data stored using file I/O
* Each account has a creation date; current date used to calculate age

**Functionalities Developed:**

* Account creation, login, and password encryption
* Deposit, withdraw, money transfer
* Transaction tracking through Transaction class
* Account age calculation using system date
* Age-based privileges: loan eligibility, bonus interest, etc.
* Full GUI integration using Raylib

**Challenges Faced:**

* Handling dynamic object storage with file I/O
* Encrypting/decrypting passwords with a fixed key
* GUI integration and event handling with Raylib
* Accurate date calculations for account age and benefits

**6. Results**

**Project Outcomes:**

* Fully working GUI-based banking system
* Accurate and secure file handling and encryption
* Modular and extensible design using OOP
* Account age-based service logic tested and verified

**Screenshots and Illustrations:**

* [Include screenshots of: Login page, Dashboard, Transaction history, Admin panel, etc.]

**Testing and Validation:**

* Unit testing for each class
* File testing for read/write accuracy
* GUI interaction testing for usability
* Manual testing of age-based conditions using different creation dates

**7. Conclusion**

**Summary of Findings:**

This project successfully simulates a robust banking system using OOP concepts, delivering a secure and intuitive experience. All key concepts such as inheritance, encapsulation, polymorphism, abstraction, and static variables were utilized effectively. The use of GUI and account age-based logic made the project stand out in terms of user engagement and realism.

**Final Remarks:**

This project not only reinforced core C++ and OOP skills, but also introduced real-world concepts like encryption, GUI programming, and user-oriented features. In the future, it can be expanded with mobile support or integration with online banking APIs.