# **COMS-280 Final Project**

## **Architecture and Design Document**

### Introduction

The purpose of this document is to outline the overall architecture of the Banking System. This system will support essential banking functionalities such as transaction history, account management, and user authentication. The architecture will be designed for scalability, security, and maintainability while leveraging object-oriented principles, including operator overloading, to enhance code usability and readability.

### System Architect

The banking system has a multi-tiered architecture consisting of the following layers:

* Presentation Layer which provides the user interface (UI) for customers and administrators. It includes web or desktop interfaces that interact with the business logic layer.
* Business Logic Layer is the core component handles account operations, transaction processing, and validation.
* Data Access Layer manages interactions with the database, ensuring data integrity and efficient querying.
* Database Layer which stores the system’s data in a relational database. ACID (atomicity, consistency, isolation, durability) properties to ensure reliability.

### Functional Components

**User Authentication**

* Secure login with hashed passwords using encryption techniques.
* Multi-factor authentication for enhances security
* Role-based access control to restrict functionalities base on user type (customer, administrator).

**Account Management**

* User can create, update, and delete bank accounts.
* Support for different account type (savings, checking).
* Balance inquires and account details retrieval

**Transaction History**

* Each transaction (deposit, withdrawal, transfer) is recorded with timestamps.
* Users can view and filter transactions by date, type, or amount.
* The system ensures transaction integrity and prevents duplicate processing.

### Use of Operator Overloading

Operator overloading will be used to improve the usability and readability of the code. Some key implementations include:

* Overloading the + operator can be used to facilitate the addition of funds to an account, makin the operation more intuitive.

Account operator+(double amount)

{

this->balance += amount;

return \*this;

}

* Overloading the – operator can help simplify withdrawal operations.

Account operator-(double amount)

{

If (this->balance >= amount)

{

this->balance -= amount;

}

Else

{

Throw std::runtime\_error(“Insufficient funds”);

}

return \*this;

}

* Overloading the << operator to allow easy printing of account details.

Ostream& operator<<(ostream &out, const Account &acc)

{

Out << “Account ID: “ << acc.accountID << “\nBalance: “

Return out;

}

### Conclusion

This Banking System is structured for security, scalability, and efficiency. By implementing advanced banking functions and leveraging operator overloading, the system ensures a seamless user experience with improved code maintainability.