Bank - Assignment 1

Student: Topan Bianca-Corina

**Group: 32711**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 4

1.3 Non-functional Requirements 4

2. Use-Case Model 4

3. System Architectural Design

3.1 Architectural Patterns Description 6

3.2 Diagrams 7

4. UML Sequence Diagrams 7

5. Class Design

5.1 Design Patterns Description 8

5.2 Uml Class Diagram 9

6. Data Model 10

7. System Testing 10

8. Bibliography 10

1. Requirements Analysis

# Assignment Specification

1. **Objective**

The objective of this assignment is to allow students to become familiar with architectural patterns.

1. **Application Description**

Use Swing/C# API to design and implement an application for the clients and employees of a bank. The application should have two types of users: a regular user which would be a client of the bank and an administrator user represented by an employee of the bank. Both users have to provide a username and a password in order to use the application.

The regular user can perform the following operations:

- View all accounts (account information: identification number, type, amount of money, date of creation).

- Transfer money between accounts.

The administrator user can perform the following operations:

- CRUD on clients information (name, identity card number, personal numerical code, address, etc.).

- Create/update/delete/view clients' accounts.

1. **Application Constraints**

The data will be stored in a database. Use the Layers architectural pattern to organize your application. Use a domain logic pattern (transaction script or domain model) / a data source hybrid pattern (table module, active record) and a data source pure pattern (table data gateway, row data gateway, data mapper) most suitable for the application.

1. **Requirements**

- Create the analysis and design document (see the template).

- Implement and test the application.

# Functional Requirement

The bank application have two types of users: **client** and **administrator**. Both users can login at the application and perform the following operations:

* **client**: view all accounts, transfer money between accounts;
* **administrator**: CRUD on clients information, Create/update/delete/view clients' accounts.

# Non-functional Requirements

Non-functional requirements are often called "[quality attributes](https://en.wikipedia.org/wiki/List_of_system_quality_attributes)" and can be divided into two categories:

* Execution qualities, such as safety, security and usability, which are observable during operation (at run time).
* Evolution qualities, such as [testability](https://en.wikipedia.org/wiki/Software_testability), extensibility, which are embodied in the static structure of the system. [1]

The application :

* must be easy to use,
* have a good response time,
* must provide security.

2. Use-Case Model

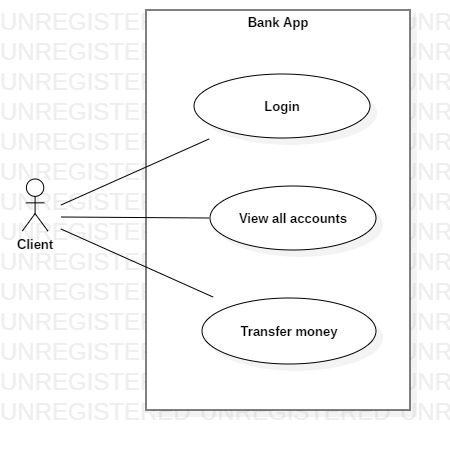
*Use-Case description format:*

*Use case: Client*

*Level: user-goal level*

*Primary actor: Client*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*

*Use-Case description format:*

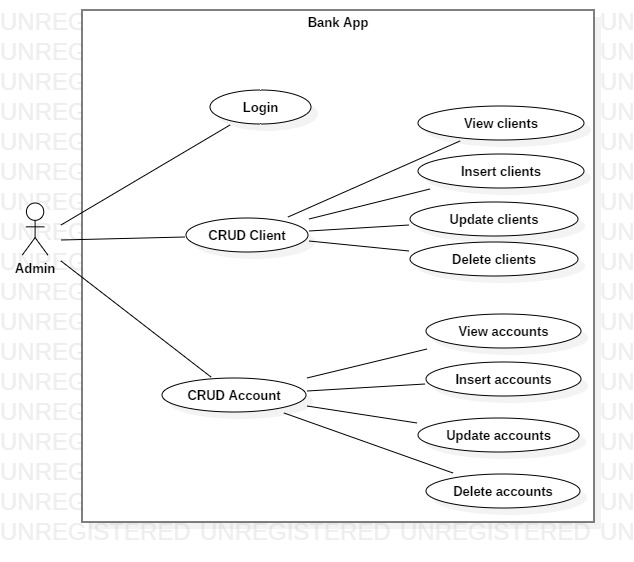
*Use case: Admin*

*Level: user-goal level*

*Primary actor: Admin*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*



3. System Architectural Design

**3.1 Architectural Pattern Description**

The basic idea of a **layered architecture** is to divide the design into small pieces. Each **layer** adds to the services provided by the lower **layers** in such a manner that the highest **layer** is provided a full set of services to manage communications and run the applications. [2]

The architectural models used:

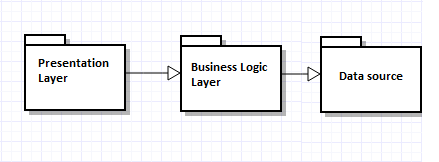
* Layers architectural pattern:

- Data Source layer

- Business Logic layer

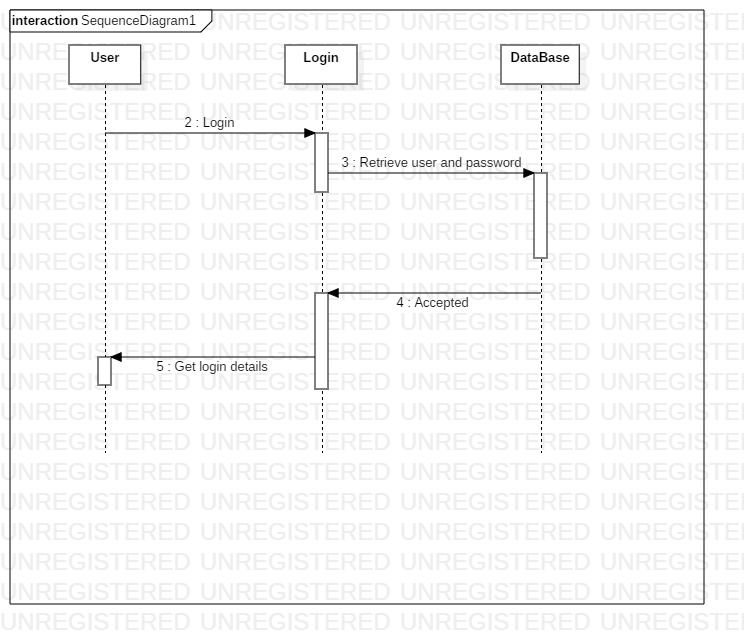
- Presentation layer

**3.2 Diagrams**

**

4. UML Sequence Diagrams

Login scenario.



5. Class Design

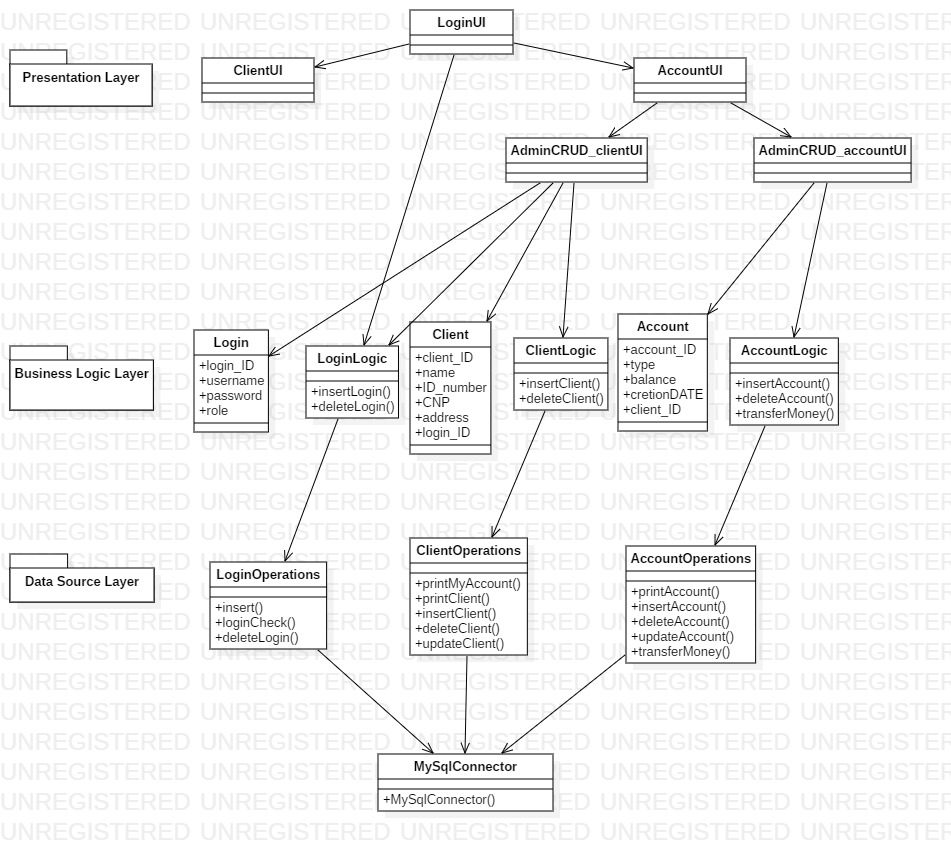
**5.1 Design Patterns Description**

The basic idea of a **layered architecture** is to divide the design into small pieces. Each **layer** adds to the services provided by the lower **layers** in such a manner that the highest **layer** is provided a full set of services to manage communications and run the applications.

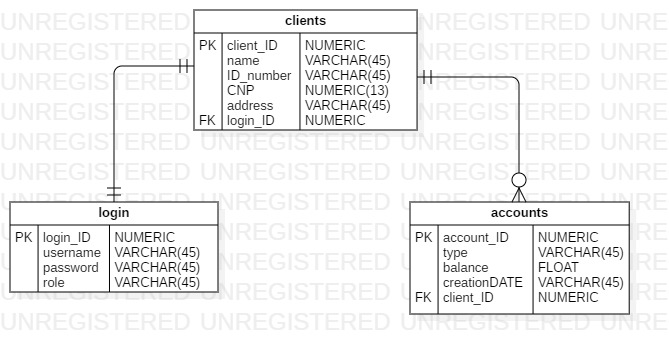
**Data Mapper** pattern is used for connection between objects and database still keeping them independent. Such objects will not even know that there is a database contain no SQL code, all of which are achieved through the Mapper.[3]

The **singleton pattern** is a [software design pattern](https://en.wikipedia.org/wiki/Software_design_pattern) that restricts the [instantiation](https://en.wikipedia.org/wiki/Instantiation_(computer_science)) of a [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system. [4]

* 1. **UML Class Diagram**



6.Data Model

**

**7. System Testing**

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

8. Bibliography

[1] <https://en.wikipedia.org/wiki/Non-functional_requirement>

[2] <https://www.slideshare.net/hemangkothari/lecture-1-introduction-to-computer-network-2013>

### [3] <http://www.informit.com/articles/article.aspx?p=1398618&seqNum=4>

[4] <https://en.wikipedia.org/wiki/Singleton_pattern>

<https://www.w3schools.com/java/java_interface.asp>

[https://stackoverflow.com](https://stackoverflow.com/)

<https://www.youtube.com/watch?v=-hpX9oEvoXc>