Bank

Analysis and Design Document

Student: Pop Laura-Maria

**Group: 30233**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 4

3. System Architectural Design 6

4. UML Sequence Diagrams 7

5. Class Design 7

6. Data Model 10

7. System Testing 11

8. Bibliography 11

1. Requirements Analysis

# Assignment Specification

Use JAVA/C# API to design and implement an application for the front desk employees of a bank. The application should have two types of users (a regular user represented by the front desk employee and an administrator user) which have to provide a username and a password in order to use the application.

# Functional Requirements

The regular user can perform the following operations:

1. Add/update/view client information (name, identity card number, personal numerical code, address, etc.).
2. Create/update/delete/view client account (account information: identification number, type, amount of money, date of creation).
3. Transfer money between accounts.
4. Process utilities bills.

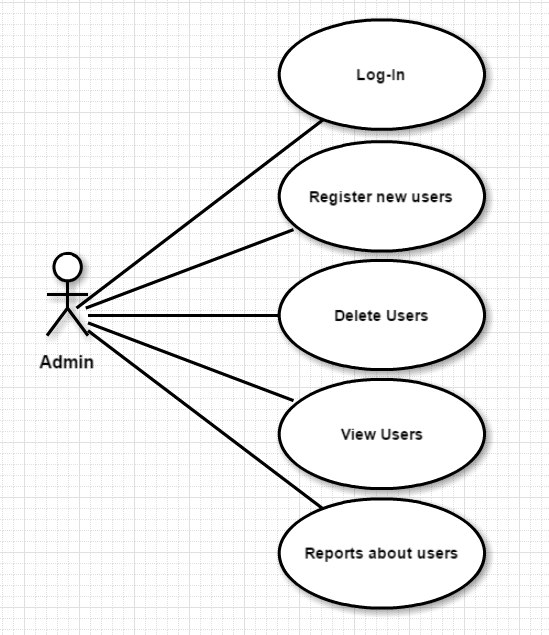
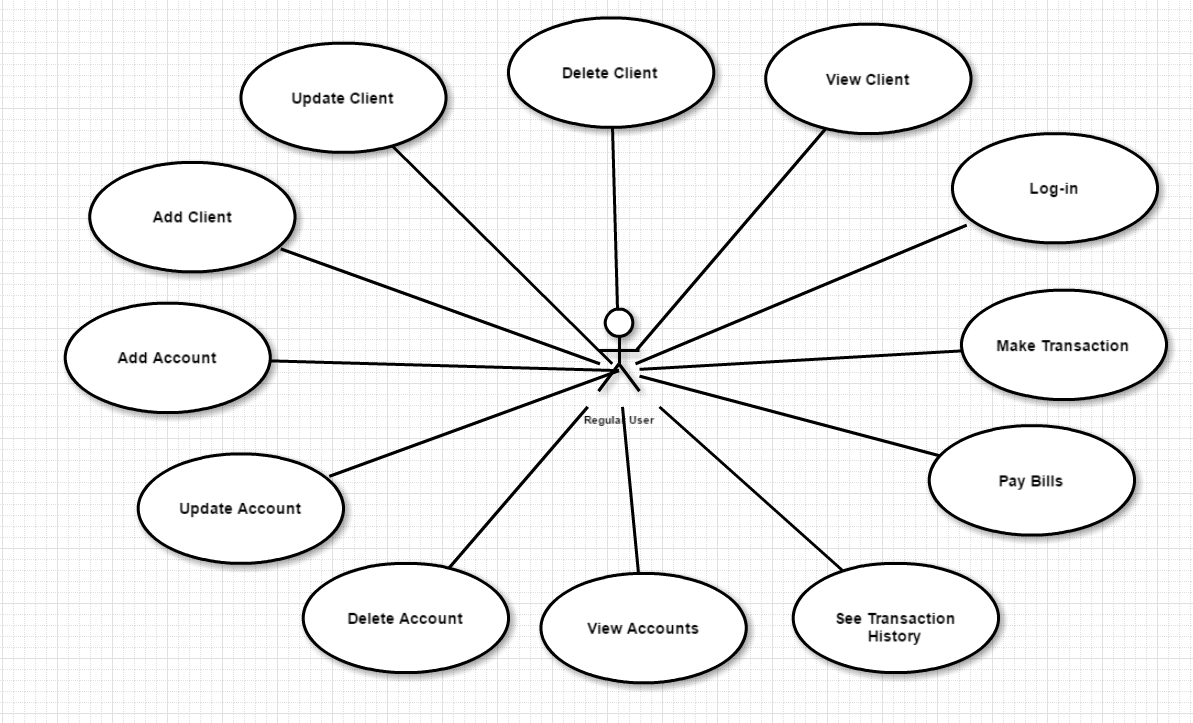
The administrator user can perform the following operations:

1. CRUD on employees’ information.
2. Generate reports for a particular period containing the activities performed by an employee.

# Non-functional Requirements

The application has to be accessible and secure because of the clients information. It also should be extensible (like to add new features and upgrades) and easy to use.

2. Use-Case Model



***Use case: LOG-IN***

***Level: user-goal***

***Primary actor: both Regular User and Admin***

***Main success scenario: the actor fills in the email and the password fields and if these are correct then the actor will see a menu of actions he/she can perform***

***Extensions: if the actor does not have an account, he will receive one from the Admin (only for Regular User)***

3. System Architectural Design

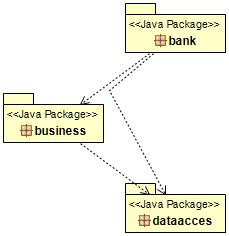
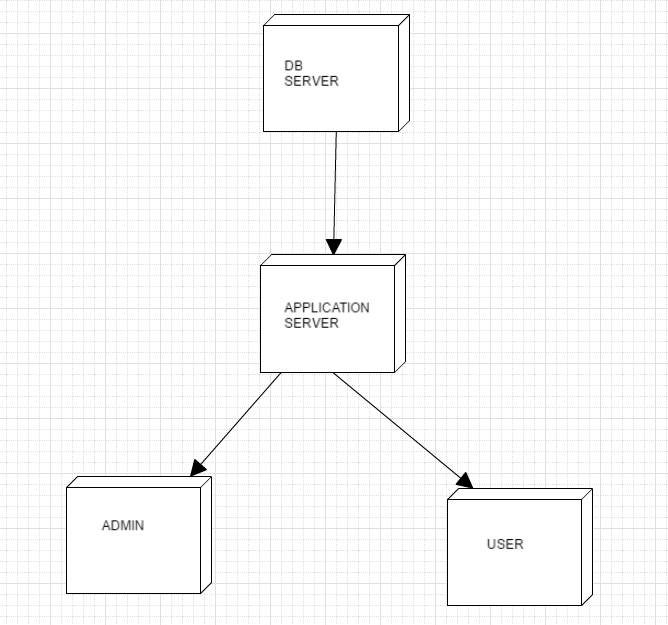
**3.1 Architectural Pattern Description**

The **Layers** architectural pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction. [1]

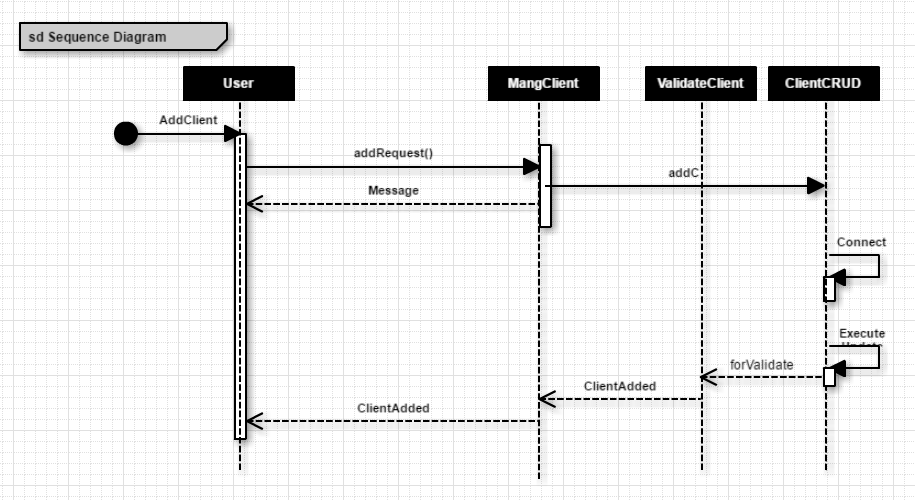
Layers in this project:

* Presentation (which represents GUI->responsible for the delivery and formatting information)
* Business logic (or domain logic->is the part where the program determinates how data can be created and stored)
* Data Access (CRUD and DB operation)

**3.2 Diagrams**



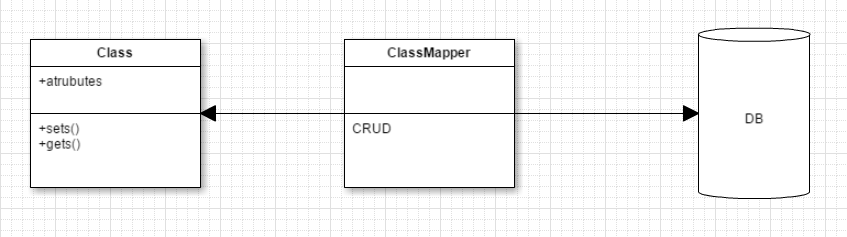
4. UML Sequence Diagrams



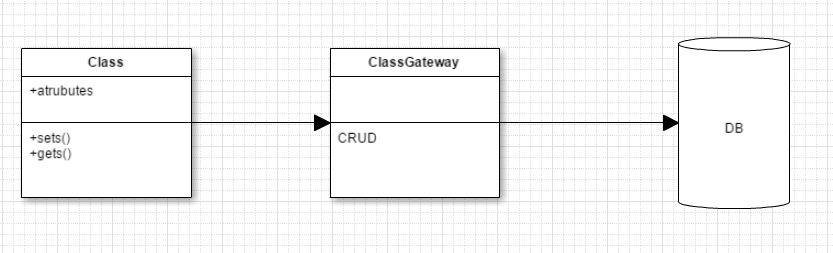
5. Class Design

**5.1 Design Patterns Description**

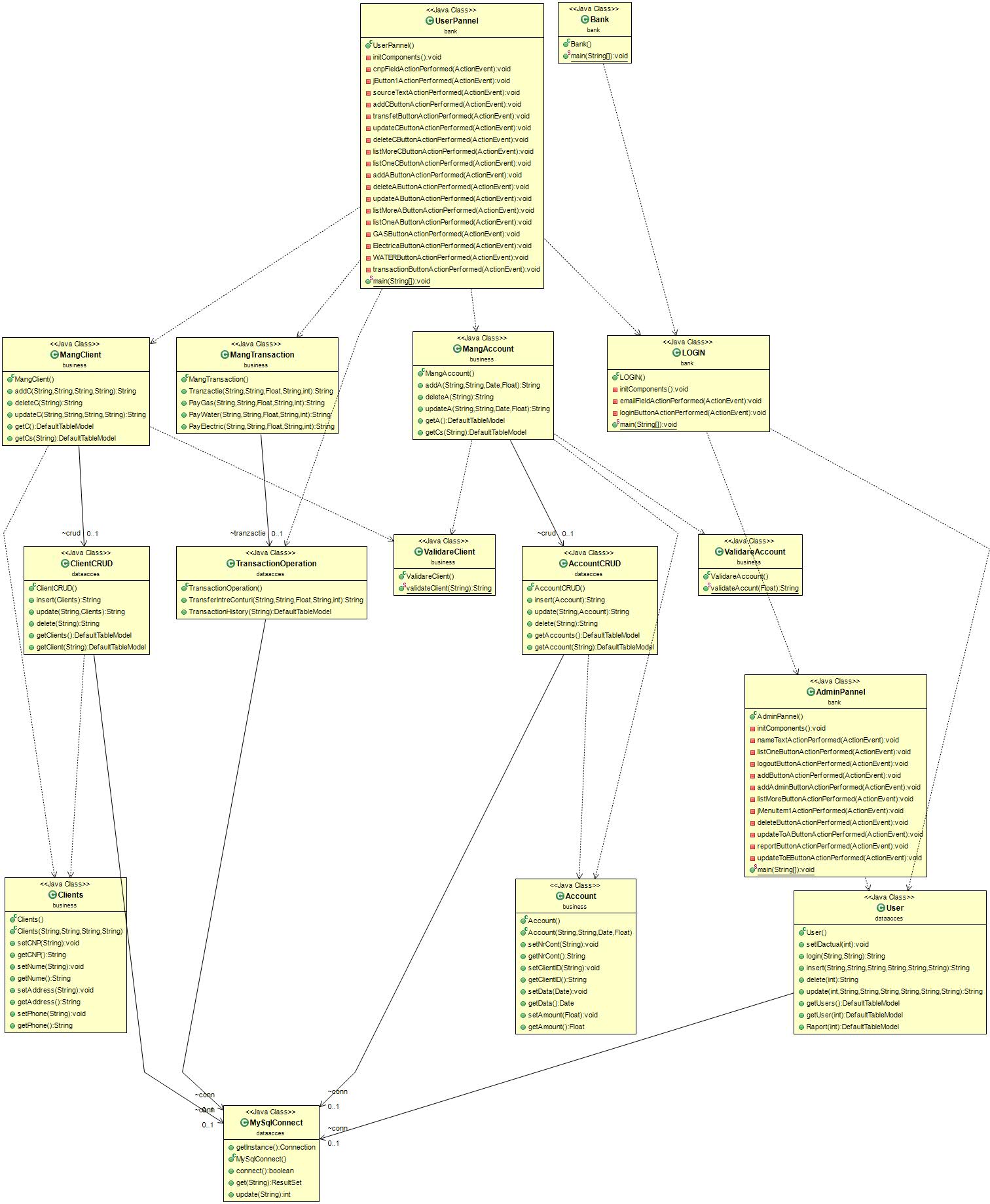
A Data Mapper is a Data Access Layer that acts as an intermediary between Domain Models and DB. It allows Domain Models and Data Sources classes to be independent of each other. It can either access the DB or make use of Table Data Gateway. It does not contain Domain Logic.



A Table Data Gateway hides all the SQL queries for accessing a single DB table or view (selects, updates, deletes). One instance handles all the rows in the table.

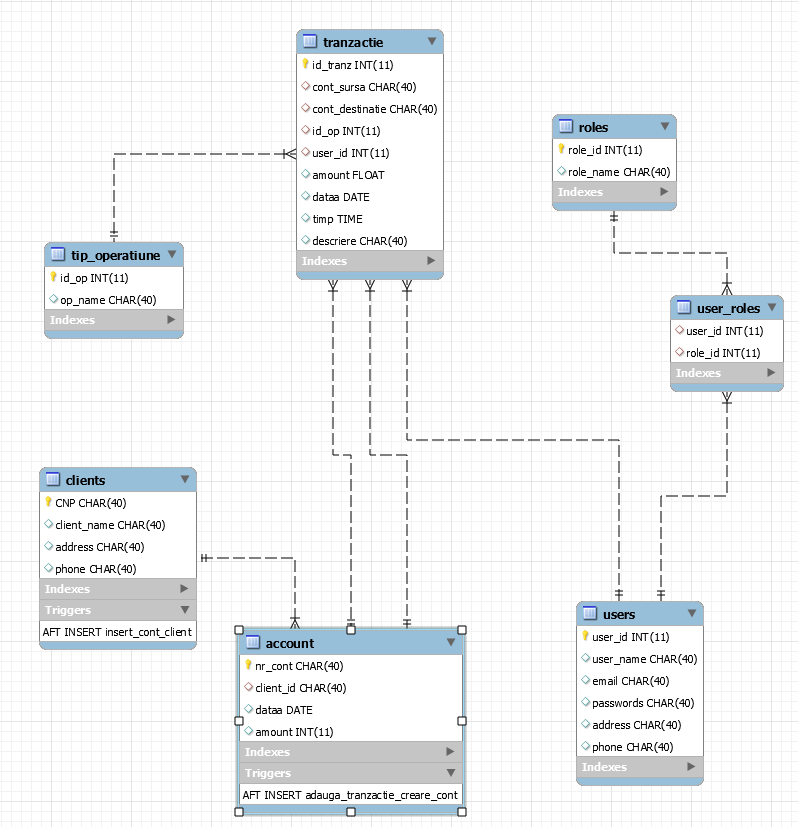


**5.2 UML Class Diagram**



For every (relevant) table in the data base there is a class for CRUD and a mapper class which is made for the data extract from the DB. After that, the data can be processed and used for resolving the request.

6. Data Model



Here, is important to mention two things:

1. Roles table: Every user has a role. Right now there are only two kinds of roles: regular user and admin, but in the future the table can be upgraded (extensible)
2. The triggers. There is a trigger that automatically inserts an account when a new client is added in the DB and a trigger that automatically inserts a transaction in the transaction table when a new account is added with the “New Account” description (from type operation table).

7. System Testing

The application was tested manually for all operation.

8. Bibliography

[1] Software Design Course, Prof. Dr. Eng. Mihaela Dinsoreanu

[2] Martin Fowler et. al, Patterns of Enterprise Application Architecture

[3] Data Bases Course, Senior lecturer Eng. Cosmina Ivan

[4] <http://www.mkyong.com/java/java-how-to-get-current-date-time-date-and-calender/>

[5] <https://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html>

[6] <http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html>

[7] <https://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html>

[8] <https://www.tutorialspoint.com/plsql/plsql_triggers.htm>

[9] <https://www.tutorialspoint.com/plsql/plsql_procedures.htm>