Bank Application
Analysis and Design Document

Student: Dregan Anda-Denisa
Group: 30233

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

Using JAVA/C# API , 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.

The regular user can perform the following operations:

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

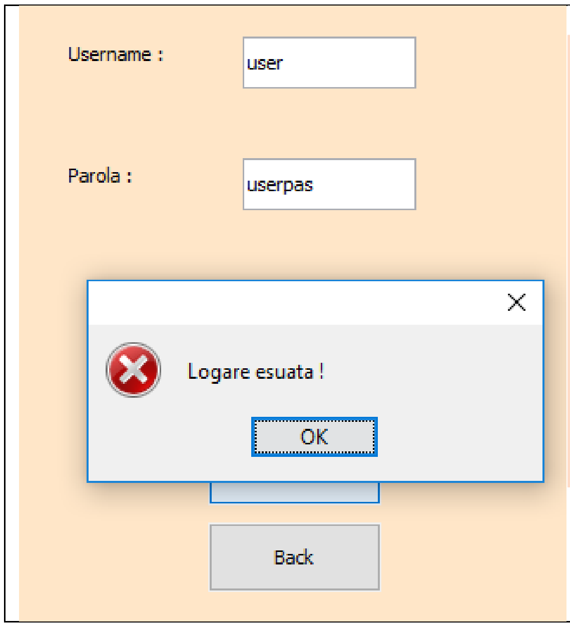
The administrator user can perform the following operations:

- CRUD on employees' information.
- Generate reports for a particular period containing the activities performed by an employee.

1.2 Functional Requirements

Firstly, the two users, user and administrator, must enter a valid name and password in order to make the operations allowed for each of them. If logging was not made, an error message appears and they can still try to log. Otherwise, there is a new window, where they can operate.

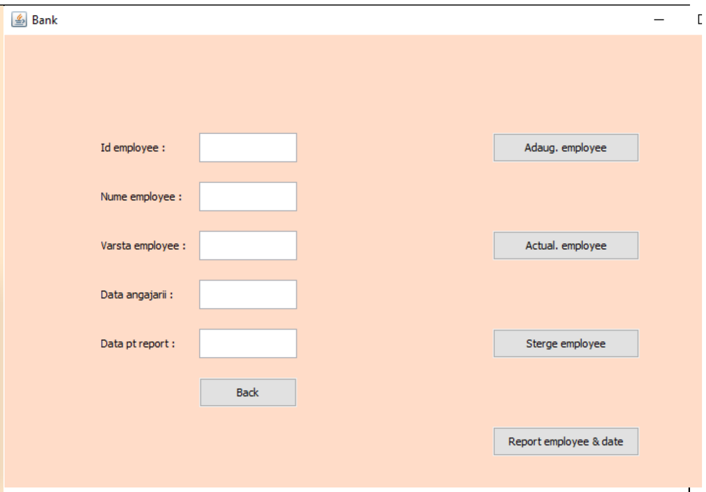
Below, you can see two cases: failed login admin, admin logged correctly.



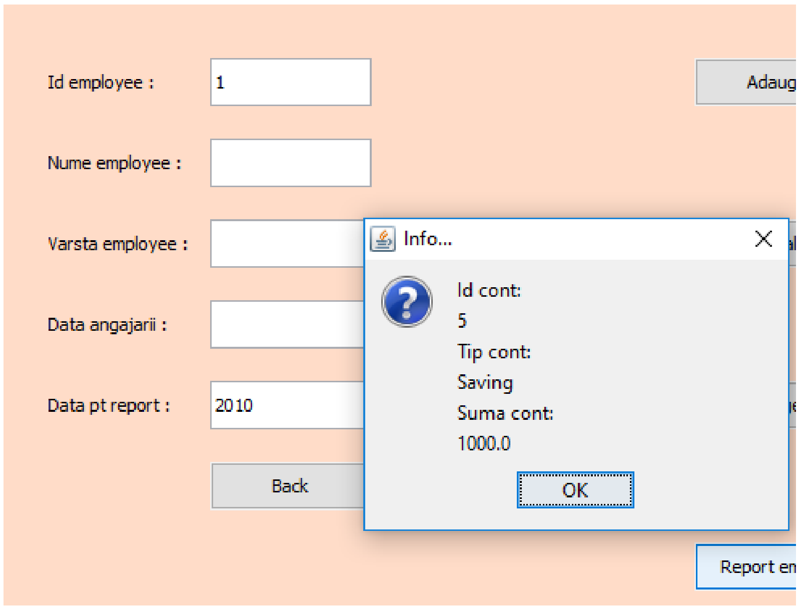
Username : user

Parola : userpas

Logare esuata !



For example, if I logged in as admin, I can execute CRUD operations for employees and generate a report of an employee in a given year, as in the picture.

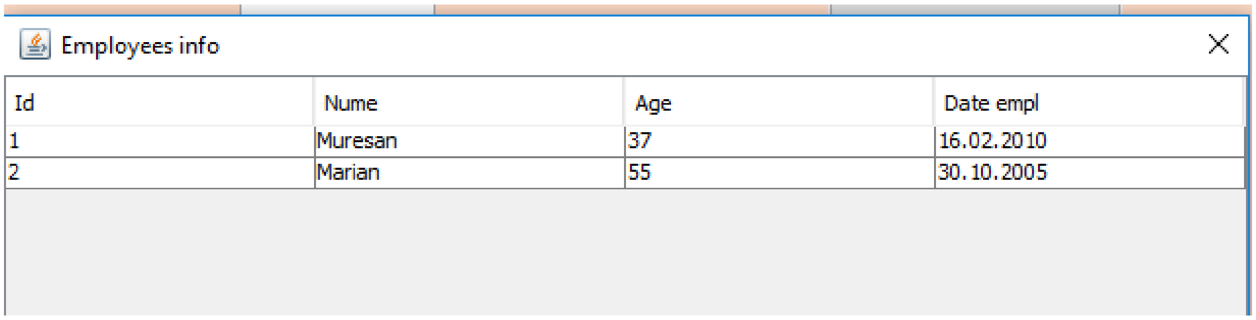


Id employee :	1
Nume employee :	
Varsta employee :	
Data angajarii :	
Data pt report :	2010

Id cont:
5
Tip cont:
Saving
Suma cont:
1000.0

When I am logged successfully as a user, I have several possibilities: the usual CRUD operations on accounts and customers, accompanied by information or alert messages. In addition, I can transfer a certain amount from one account to another and pay bills for a specified client.

Both users have access to the data tables with employees, and also accounts and customers (the last ones are for the regular user), just clicking the buttons "View employees ","View accounts" and "View clients". See the following example for employees' table:



Employees info

Id	Nume	Age	Date empl
1	Muresan	37	16.02.2010
2	Marian	55	30.10.2005

1.3 Non-functional Requirements

While the functional requierements describe what the system should do, the non-functional ones describe how the system works.

Among the known nonfunctional requierements, this project follows

*the minimum respons time : total amount of time it takes to respond to a request for service is small, meaning that if users make changes to the database tables available in the application interface will be updated in real time to verify the correctness of operations at the same time; it led to the next requirement :

*good processing speed;

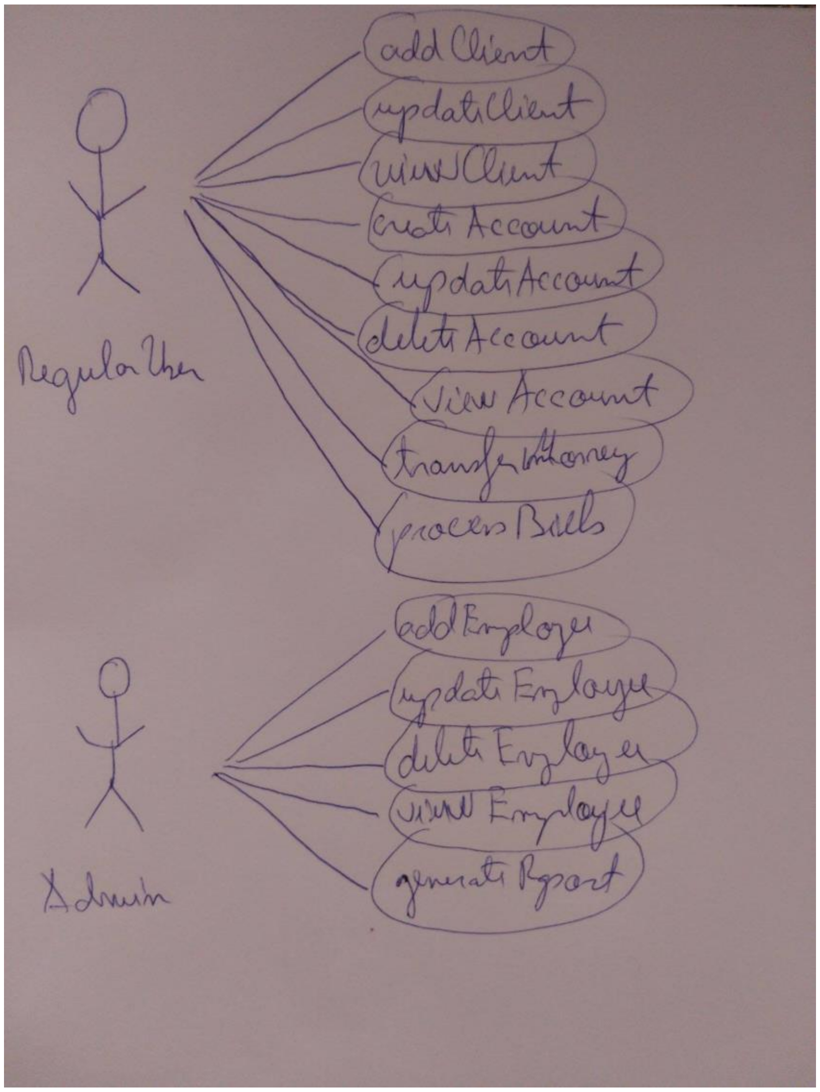
*scalability: this system is able to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth

*security : to avoid situations where some enivous users change data that should not change, everyone must log into their account

*usability : it is easy to operate with this system, because the interface is friendly and concise.

2. Use-Case Model

The use case diagrams are showed in the picture below:



Use case : Add new client and see the modification in the clients' table.
Level: subfunction
Primary actor : only regular user can do the CRUD operation on clients.
Main success scenario : the user log correctly into application, introduces all the infos required for adding a client : id, name, card number, personal number and the address. Then click on the button "Add client" and it is stored automatically in the database. By clicking on the button "View client", the user see the made modification into the table.
Extensions: a first failure in this use case could be user not knowing the username and the password for his account on the application.

3. System Architectural Design

3.1 Architectural Pattern Description

I chose a hybrid architectural pattern called Table Module for structuring the domain logic. The implementation has one java class per table on the database : Client, Employee, Account, Admin , User, and Operation which incapsulates the logic of the application. In this case, there is a single instance that handles the business logic for all rows in the database table.

The *Table Module* gives an explicit method-based interface that acts on the data resulting from SQL calls. Grouping the behavior with the table gives so many benefits of encapsulation in that the behavior is close to the data it will work on.

Using this architectural pattern means organizing the domain logic around tables rather than straight procedures provides more structure and makes it easier to find and remove duplication.

The difference between Table Module and Domain Model is represented by the fact that Domain Module would have one instance of client(for example) for each client stored on the database. We can say that this pattern, Table Module, is the "average" between Domain Model and Transition Script.

In the Layered pattern, this architectural pattern incorporates or form the Domain Logic stage,also called "business logic". There is all the work of the system and it involves methods that calculate based on inputs, validates the inputs and so on. So, to sum up, we can say that Domain Layer is the "real point of the system".

As a data source pure pattern, the application has Table Data Gateway and it fits perfectly with Table Module pattern and has one instance per table: ClientGateway,Account Gateway,EmployeeGateway, AdminGateway and UserGateway. In fact, Gateway separes the SQL acces from the classes from the domain logic and in this case, the access to the database is very easy to handle.

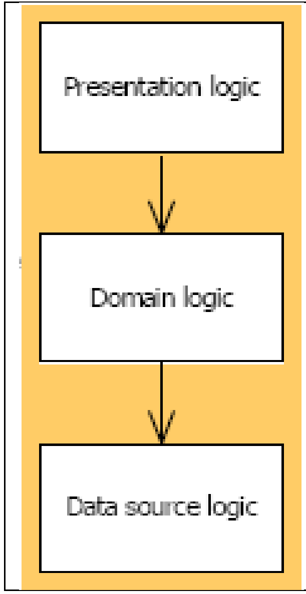
In the Layered pattern, this pattern fits into the Data Source Layer, that handle the communication with DB,transaction managers, messagins systems and so on.

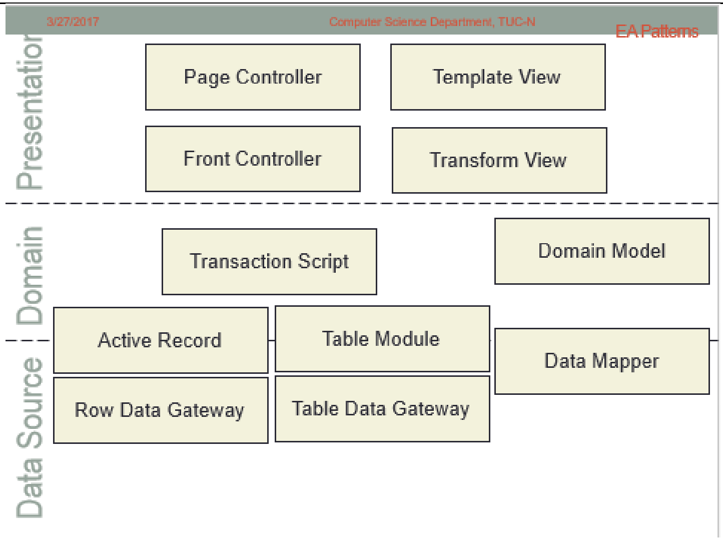
The third layer is the Presentation one which is responsible for handling all user interface and browser communication logic.

So all these stages in this architectural pattern are organized into horizontal layers, each performing a specific role within the application.

3.2 Diagrams

There are some schemas that show how the above architectural patterns are applied on this system :



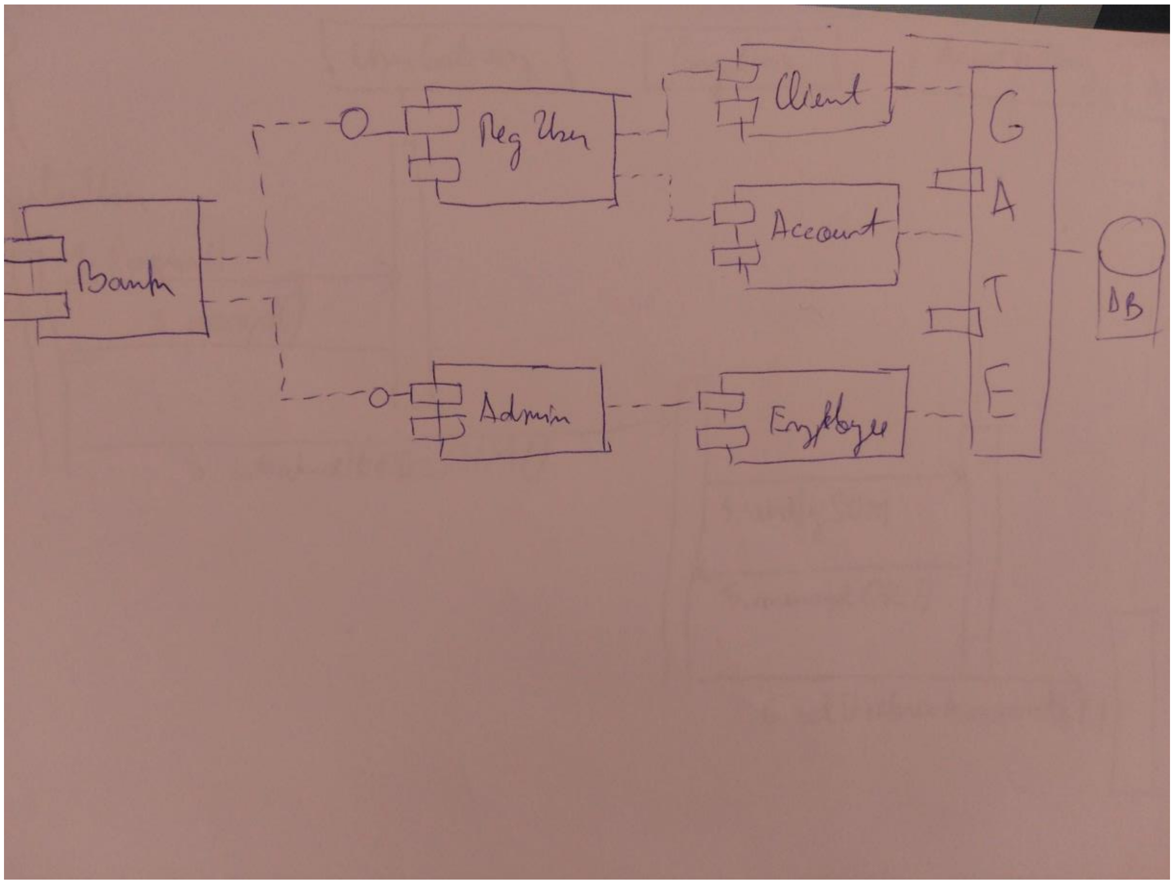


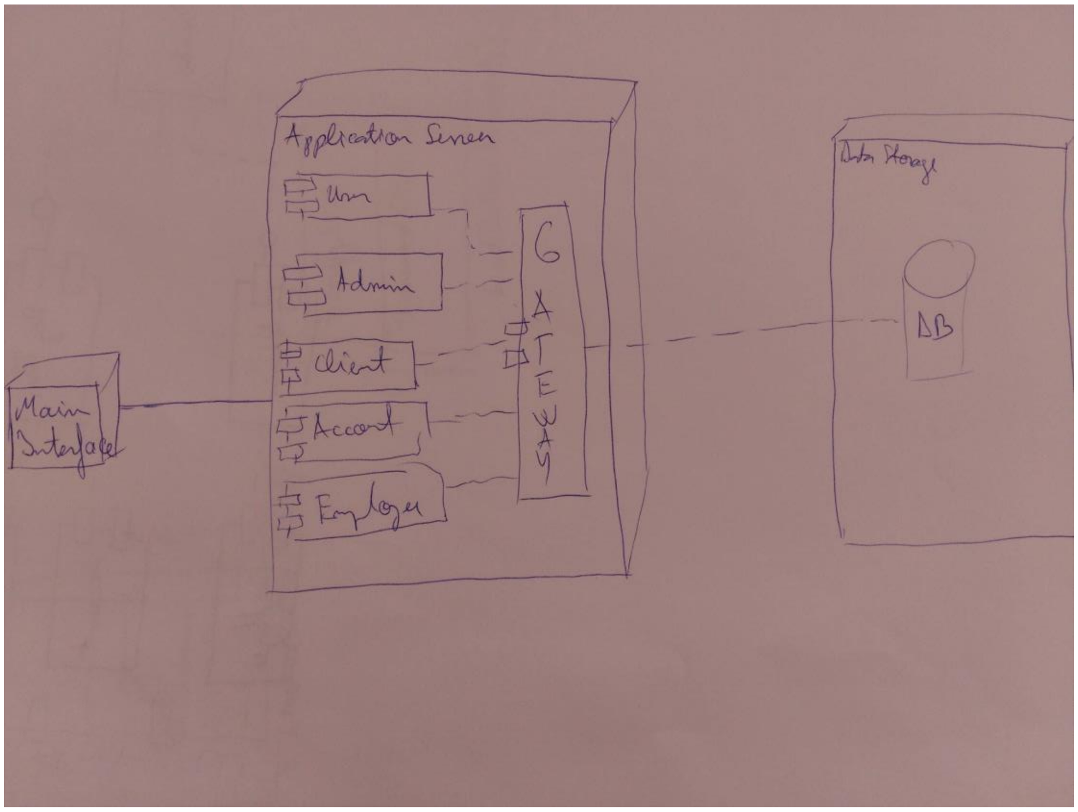
For the Presentation stage, the implementation contains GraphicControl class, which handle the interface, the buttons, labels, tables and all the stuff the user interact with.

For the next level, Domain, the Table Module hybrid pattern has, as I mentioned above, the classes: Client,Account , Operation, Employee,Admin and User.

The Data Source layer (and Table Data Gateway) has a Gateway class for each table in the database.

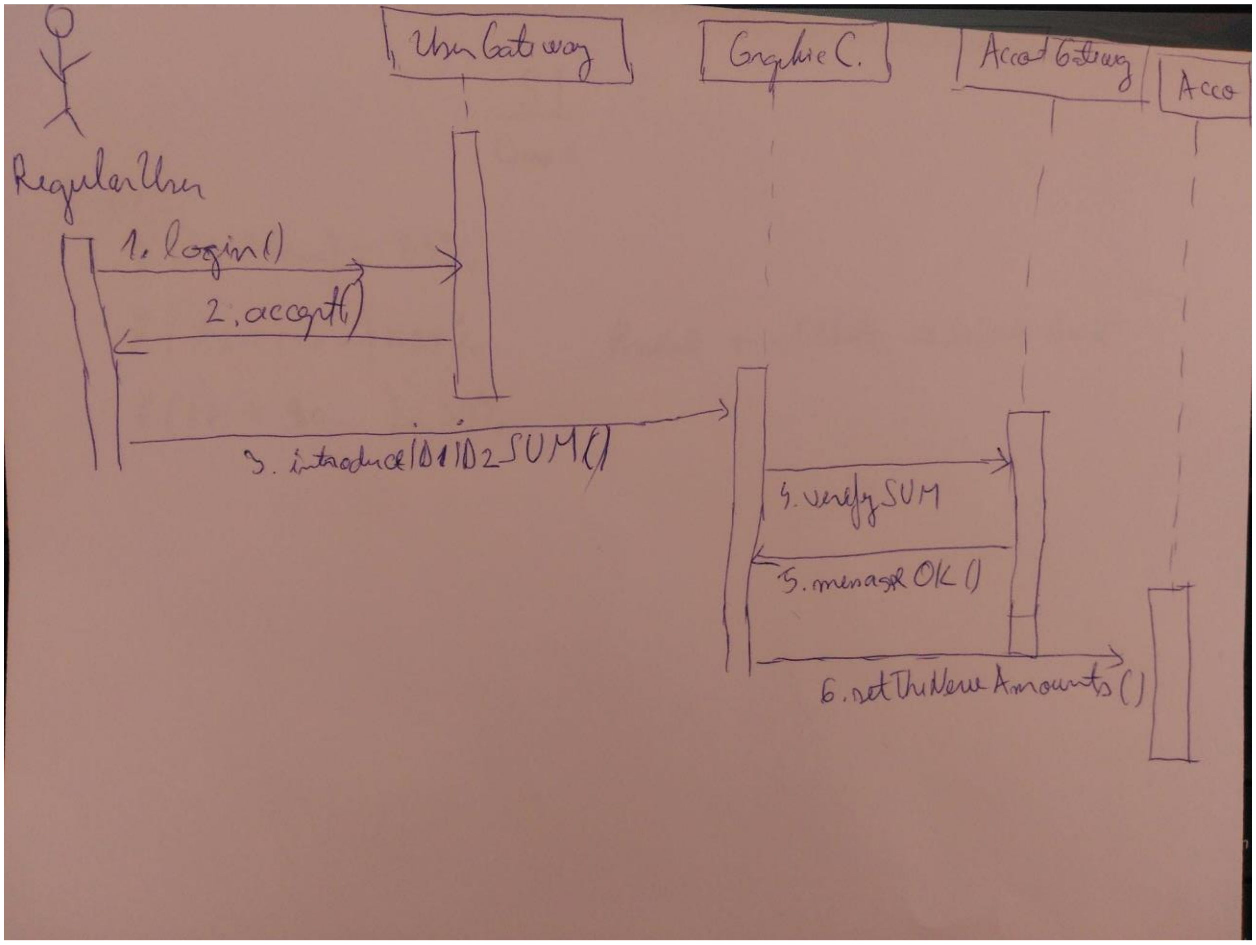
The pictures below show the component and the deployment diagrams for the application:


Reg User
Client
Account
Bank
Admin
Employee
GATE
DB


Application Server
User
Admin
Client
Account
Employee
GATEWAY
Data Storage
DB
Main Interface

4. UML Sequence Diagrams

The following sequence diagram represents the scenario of the transfer money from one account to another. The regular user successfully log into his application account, introduces the IDs, the sum and if it is enough money on the “fromAccount”, the operation is made : decreases this account’s amount of money and increases the amount on “toAccount”.



5. Class Design

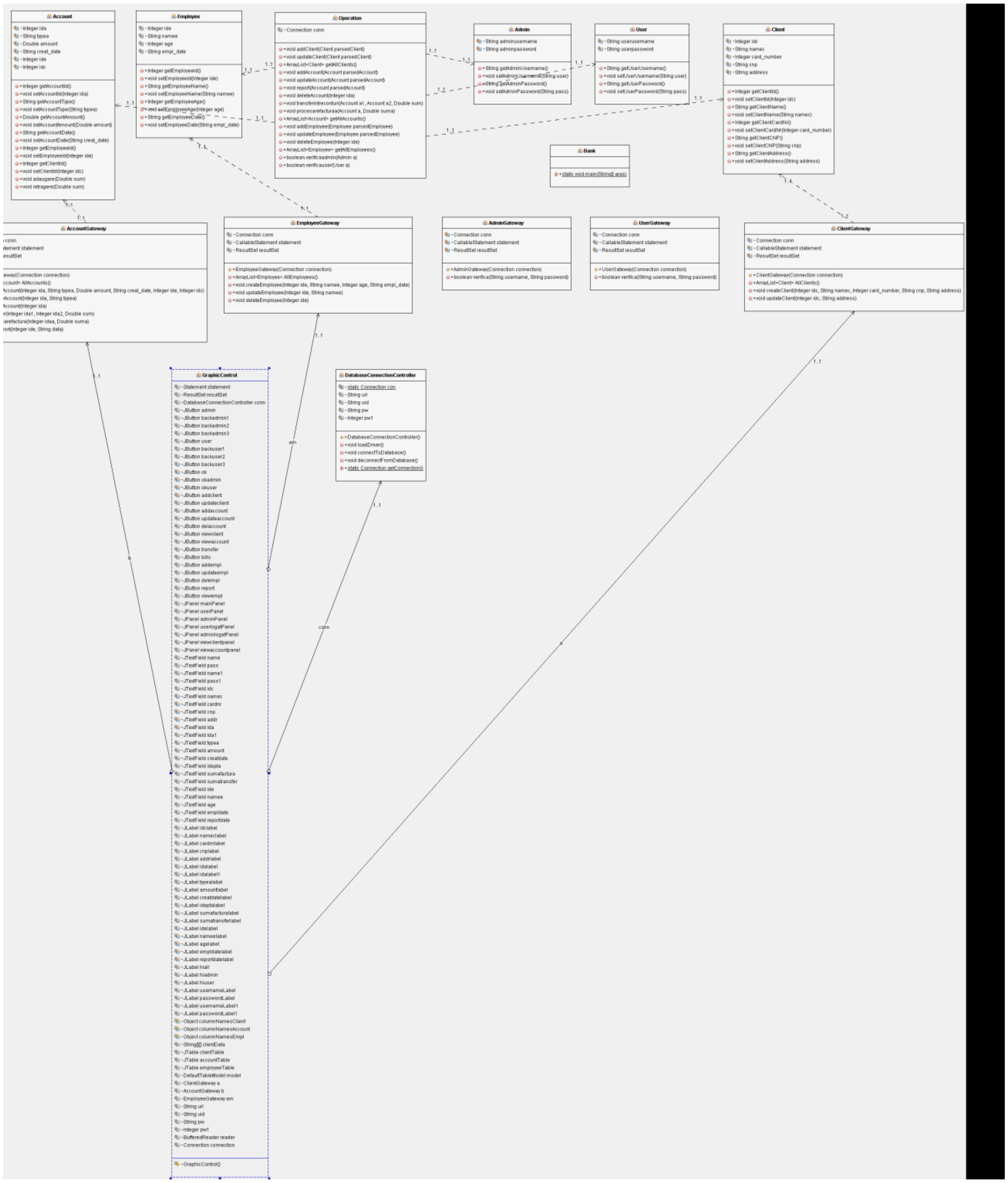
5.1 Design Patterns Description

As design patterns, I used in the implementation the pattern written above, where I made a parallel between Table Module design pattern, Table Data Gateway and the architectural one Layer pattern.

In addition, the MVC design pattern fits the realized application, because it is a close connection between the Layered Arch. Pattern and MVC one. The **Model** contains the core functionality and data. **Views** display information to the user. **Controllers** handle user input. Views and controllers together comprise the user interface.

5.2 UML Class Diagram

The following picture shows the class diagram of the application. It is easy to understand the concepts approached until now, because the diagram is structured as in the patterns' schema and are visible the layers and their incorporated classes.



6. Data Model

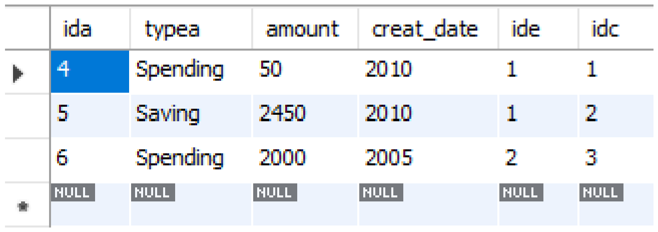
Data models define how data is connected to each other and how they are processed and stored inside the system. In other words, this concept define how a database is structured or modeled.

This system's implementation is based on Relational Model, which is the most popular data model in DBMS.

This data model is characterized by the following :

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain,

which can be validated in the picture below:



ida	typea	amount	creat_date	ide	idc
4	Spending	50	2010	1	1
5	Saving	2450	2010	1	2
6	Spending	2000	2005	2	3
NULL	NULL	NULL	NULL	NULL	NULL

7. System Testing

I have tested almost every operation / method that data is entered correctly. Here are some examples:

*I tested if the password and login name introduced by two users are valid.

*I have treated the cases where there is not enough money in the account to make the transfer or bill payment.

* I checked if there is activity in a period of time required by the administrator for a particular employee.

* I have treated the case where already exists in DB an id introduces by the user when he wants to add a new client , account or employee.

All these tests come along with error or alert messages for unfavorable cases and with information messages, plus perform the desired operation, in the favorable cases.

8. Bibliography

https://drive.google.com/drive/folders/0B-7Rn7Rec1VUaWlaVnNpUV9ROVk - Laboratory resources

http://users.utcluj.ro/~dinso/PS2017/Lectures/

https://martinfowler.com/eaaCatalog/tableModule.html