Bank Application

Analysis and Design Document

Student: Abrudan Darius

**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 3

3. System Architectural Design 4

4. UML Sequence Diagrams 7

5. Class Design 7

6. Data Model 7

7. System Testing 8

8. Bibliography 8

1. Requirements Analysis

# Assignment Specification

Design and implement an application for the front desk employees of a bank using JAVA/C# API. 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:

* Add/update/view client information (name, identity card number, personal numerical code, address).
* 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.

# Non-functional Requirements

Some of the non-functional requirements are privacy, security while using a log in and a log out mode of using the application, so that not everybody could use this application, just the employees and the administrator who have control on employees.

Maintainability and extensibility can be also seen by having the possibility of adding new clients with their associated accounts, or deleting some clients together with their accounts in case they decide to close the account and move to another bank. In time, some employees can resigned and others people can take their places, functionality maintained by the administrator.

Efficiency and usability are another non-functional requirement that are present by the functionality of getting a report for a specific employee with all the activities performed through a period of time, transfer money between account and process bills.

2. Use-Case Model

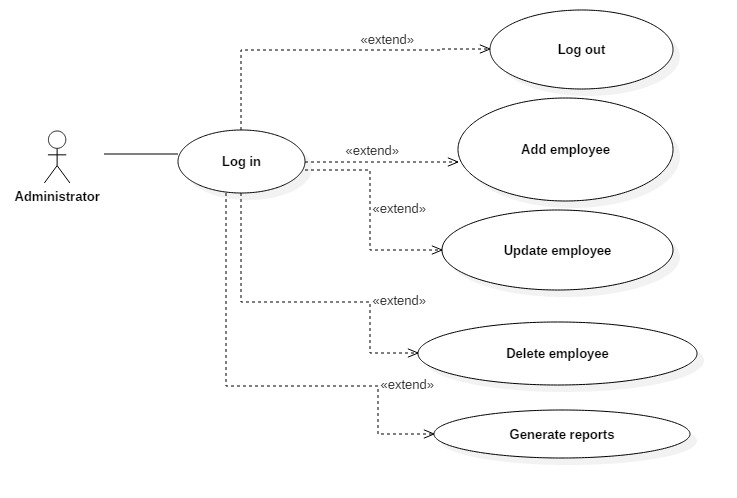
Use case: transfer money between accounts

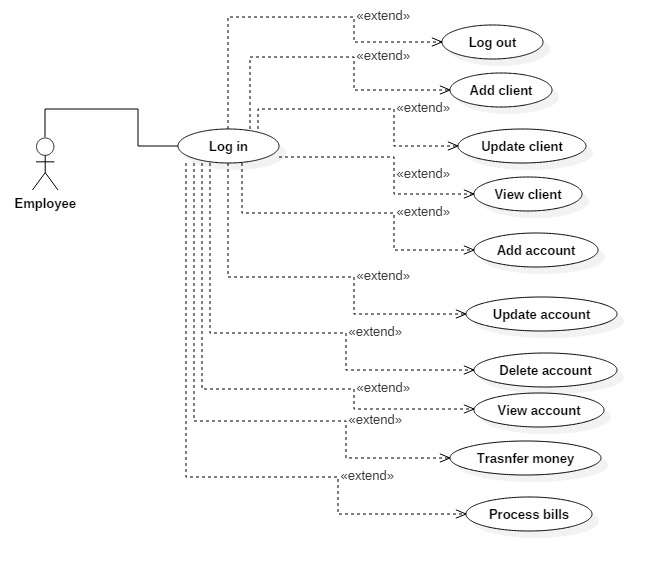
Level: user-goal level

Primary actor: employee

Main success scenario: there are enough money to transfer from source account to destination account

Extensions: not enough money in the source account requested to transfer





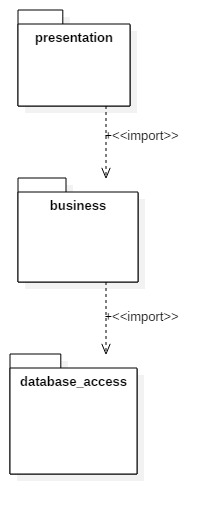
3. System Architectural Design

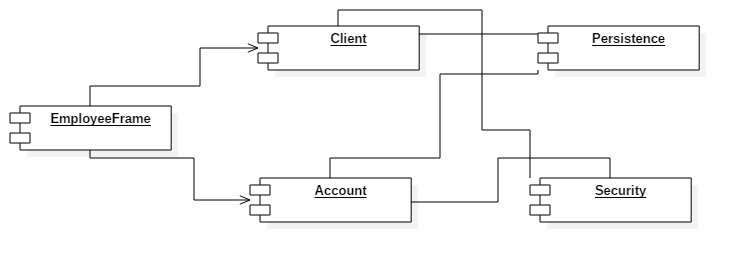
**3.1 Architectural Pattern Description**

I used the Layered Architectural Pattern or so called n-tier. I have three different layers, called presentation, business and database access. The first one is responsible for the user interface (UI) classes. The second one deals with business logic of the application and the third one is responsible for the connection and access to the database.

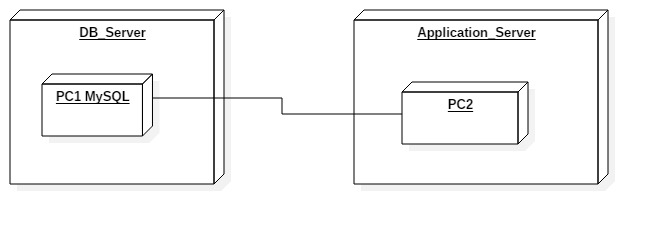
**3.2 Diagrams**

In the diagrams below, it is shown how the layered architectural pattern is applied to the bank application. Each layer is represented by a package with the same name.

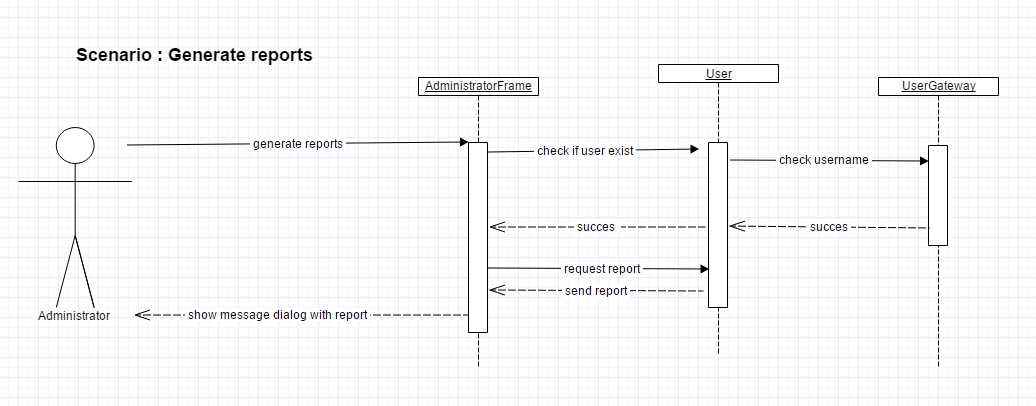


Component diagram:

Deployment diagram:



4. UML Sequence Diagrams

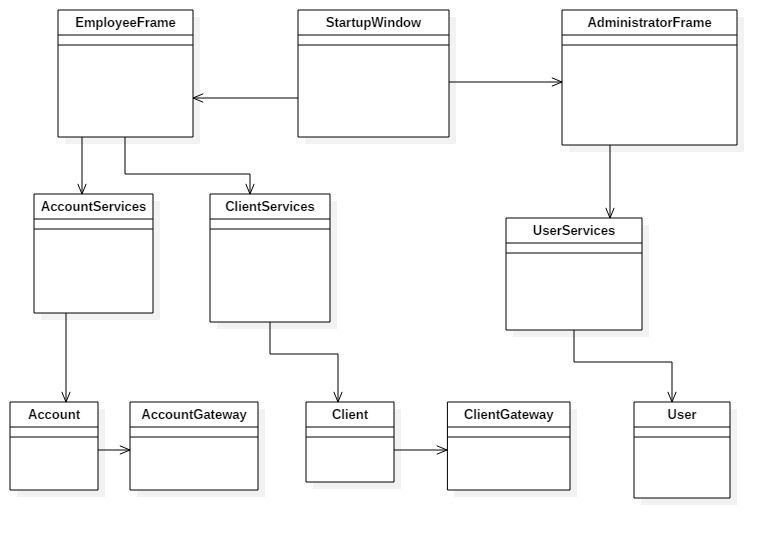


5. Class Design

**5.1 Design Patterns Description**

In the application created, I used the hybrid Data Source Pattern Table Module for accessing every table from the database. I created a correspondent class for each table form DB. I also used the pure Data Source Pattern Table Data Gateway. According to this pattern, I created a Gateway class in the database access layer corresponding to the class from business layer. The Gateway class is the only one who access the database and make queries such as select, insert, update or delete. The class from the business layer does not access the DB, it just call the methods responsible for CRUD from the correspondent Gateway class.

**5.2 UML Class Diagram**



6. Data Model

In the database, I created 4 tables: one for clients, one for accounts, one for users and one named user role. The client table contains id, name, identity card number, CNP and address. The account table contains id, identification number, type, amount of money, date of creation. A client can have multiple accounts, so I put a foreign key in the accounts table, such that for an account to know the client who own it. The user table has an id, username and a password. Here are stored all employees and the administrator. The table user role is responsible for storing the role of each user, if it is an employee or an admin.

7. System Testing

I designed a couple of tests such that all the inputs of the application (inputs introduced in J Text Fields) will be validated against invalid data before submitting the data and saving it in the database. If a Text Field is not completed, or if there is required a number and the user will introduce a string, it will output a J Option Pane with an appropriate message so that the user could know exactly which field was not completed correct. I also verified for the log in Startup Window such that the username exist and the password for a specific username is correct. In case of such an inconsistence, a message will appear.

8. Bibliography

1.Martin Fowler et. al, Patterns of Enterprise Application Architecture, Addison Wesley, 2003

2.<https://stackoverflow.com/>

3. <https://docs.oracle.com/javase/tutorial/>

4. <https://www.tutorialspoint.com/>