Student:Stefan Olimpiu

**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 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The application is designed for the front-desk empolyees of a bank. In this way, they will manage the clients and their accounts. Also there will be an administrator which will manage the employees.

# Functional Requirements

The regular user cand do the following: CRUD on client info, CRUD on client account, transfer money between accounts, process utilities bills.

The admin user can do the following: CRUD on employee’s info, generate reports for a particular employee

# Non-functional Requirements

Security is an important requirement as the application is for a bank. The passwords should require a high complexity and should not be highlighted when introduced. Also the application is accessible through its interface and extensible as new features cand be easily added because of the using of an architectural pattern. Readability of the source code is high enough also because of the structured way of writting it. Testing is supported both in manual and automated ways.

2. Use-Case Model

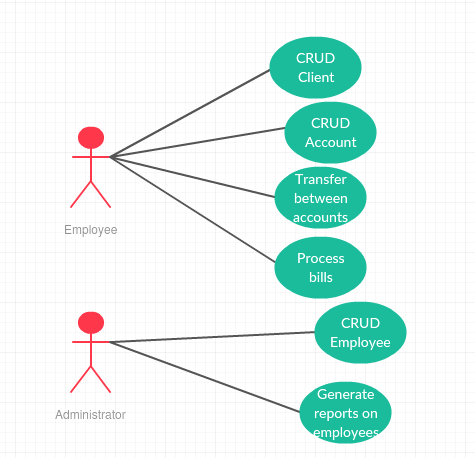
Use case: <transfer money between accounts>

Level: <user-goal level>

Primary actor: <regular user (employee)>

Main success scenario: <the user successfully logins, and then transfers the money between accounts>

Extensions: <the user wrongly introduce his credentials, after that he finally login, but the amount he wants to transfer is smaller the the amount of the account. An error message is displayed>



3. System Architectural Design

**3.1 Architectural Pattern Description**

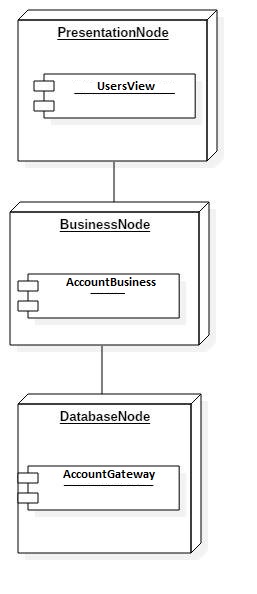
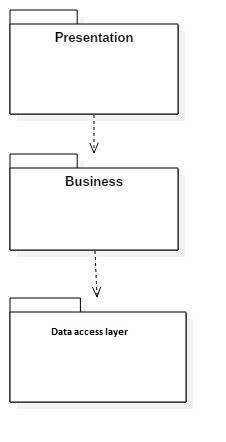
Three-tier architecture is a client–server software architectural pattern in which the [user interface](https://en.wikipedia.org/wiki/User_interface) (presentation), [functional process logic](https://en.wikipedia.org/wiki/Business_logic_layer) ("business rules"), [computer data storage](https://en.wikipedia.org/wiki/Computer_data_storage) and [data access](https://en.wikipedia.org/wiki/Data_access) are developed and maintained as independent [modules](https://en.wikipedia.org/wiki/Modular_programming)

Presentation logic is about how to handle the interaction between the user and the software. This can be as simple as a command-line or text-based menu

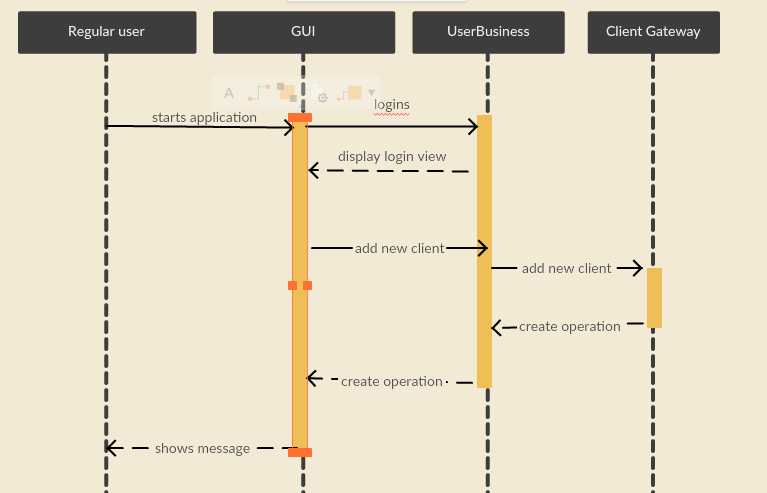
Data source logic is about communicating with other systems that carry out tasks on behalf of the application. These can be transaction monitors, other applications, messaging systems, and so forth. For most enterprise applications the biggest piece of data source logic is a database that is primarily responsible for storing persistent data.

The remaining piece is the domain logic, also referred to as business logic. This is the work that this application needs to do for the domain you’re working with. It involves calculations based on inputs and stored data, validation of any data that comes in from the presentation, and figuring out exactly what data source logic to dispatch, depending on commands received from the presentation.

**3.2 Diagrams**



4. UML Sequence Diagrams



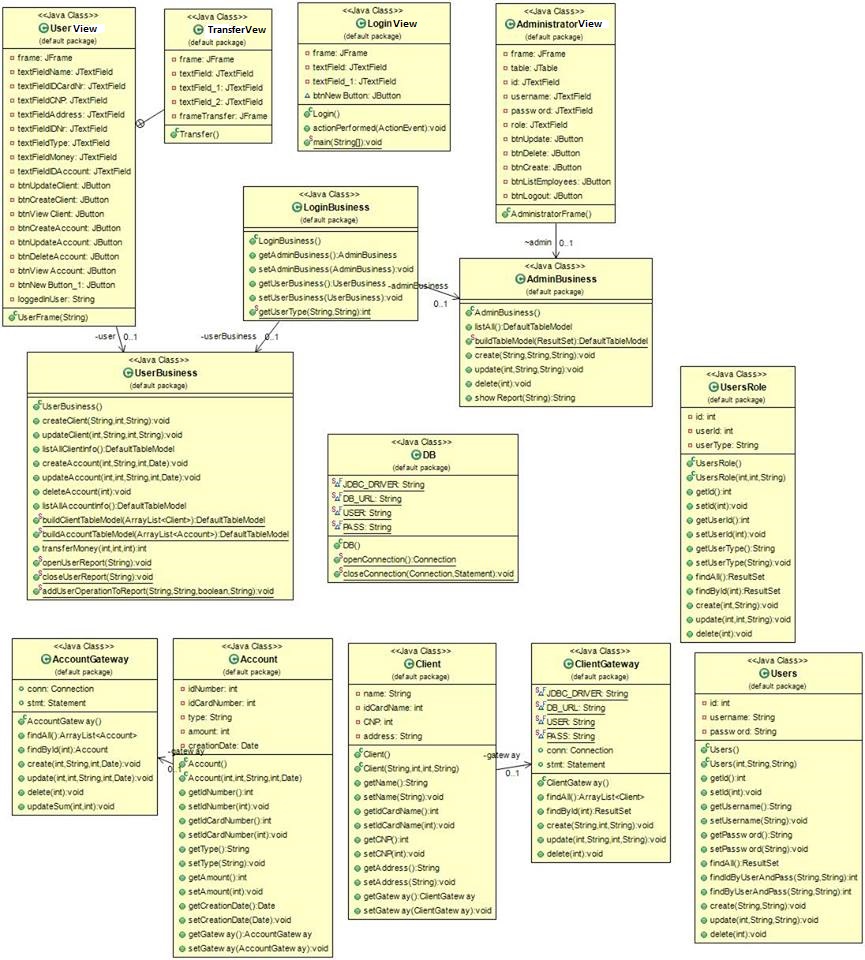
5. Class Design

**5.1 Design Patterns Description**

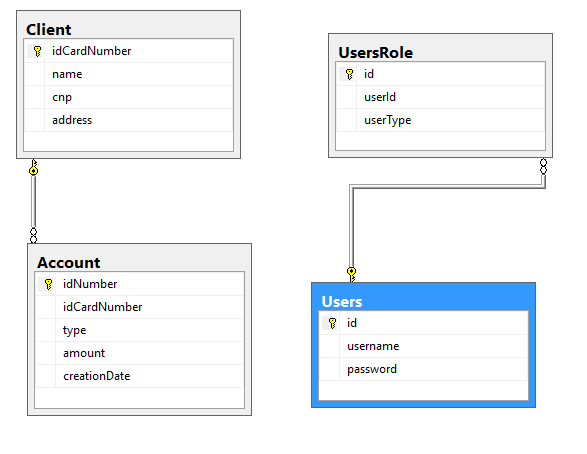
In “Active data record”, an object carries both data and behavior. Much of this data is persistent and needs to be stored in a database. Active Record uses the most obvious approach, putting data access logic in the domain object. This way all people know how to read and write their data to and from the database.

Table Data Gateway holds all the SQL for accessing a single table or view: selects, inserts, updates, and deletes. Other code calls its methods for all interaction with the database.

**5.2 UML Class Diagram**



6. Data Model



7. System Testing

For the main operations the system supports tests: delete, update, create, retrieve etc. If something is going wrong the application send an error message to inform the user. In particular, I tested the outcome of the following operations:

1. on login button click, from login view – this takes the user to admin view or user view depending on the introduced credentials
2. on create, list, update, logout, delete, report buttons click, from admin view
3. on create, update, view, logout, delete, transfer buttons click, from user view – the transfer button click opens another frame where user can give the accounts ids and click transfer. After that, if the list button is clicked, the user can see that the accounts’ amounts were updated.

8. Bibliography

* https://www.martinfowler.com/eaaCatalog/activeRecord.html
* https://www.martinfowler.com/eaaCatalog/tableDataGateway.html
* http://tutorialspoint.com/
* https://www.w3schools.com/