<Assignment 1>

Analysis and Design Document

Student: Ștefănescu Marian

**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

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:

* 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.

# Non-functional Requirements

The application must comply in terms of:

* Scalability, in order to support a large number of clients and employees
* Reliability, as any errors will be handled locally and they should not affect the general performance at runtime.
* Data integrity, because the information stored is of high importance
* Security, because only the authorized users must be able to enter the application.

2. Use-Case Model

Use case: Accessing the members dashboard

Level: summary-level

Primary actor: Employee

Main success scenario: The employee accesses the application, logs in with correct credentials (otherwise it will be redirected back to the login screen) and reaches the dashboard where he can edit the information about the client or change his credentials.

Extensions: In the case his username and password correspond to admin credentials the user will be granted administrator rights. As administrator, he can see reports about the employees and change the information about them.

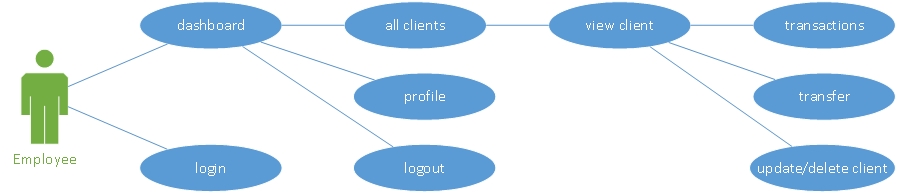


Figure Use case for the employee

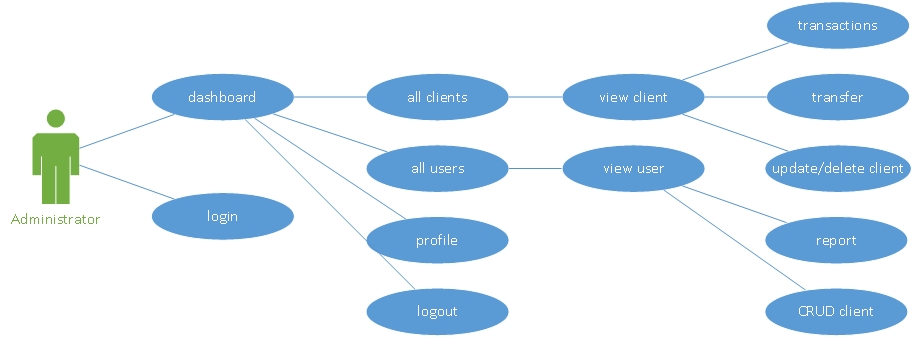


Figure Use case for the administrator

3. System Architectural Design

**3.1 Architectural Pattern Description**

I’ve used the Layered Architectural Style because my application can be split easily into three main components(Presentation, Business logic, Data source) thus forming a 3-layers architectural pattern.

The three layers are stacked vertically and follow closely the guidelines of a strict layered application. Thus, my layers aggregate incrementally, and the communication is established in a top-down manner with no direct communication between the top and bottom layer.

The biggest advantage of choosing this architecture is the separation of the functional layers, thus offering the possibility of working in collaboration with other developers without the fear that minor changes can ripple in classes that offer different functionalities.

**3.2 Diagrams**



Figure Diagrama de pachete

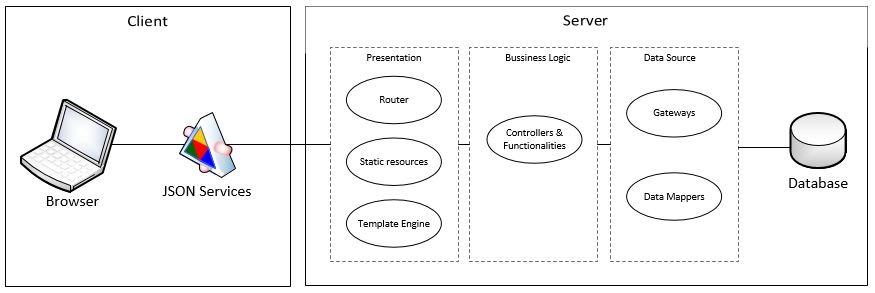


Figure Structura arhitecturala a sistemului

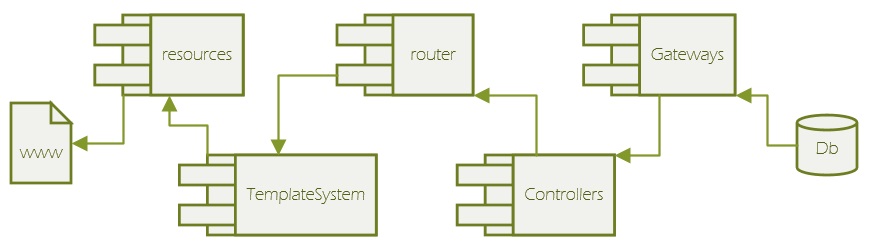
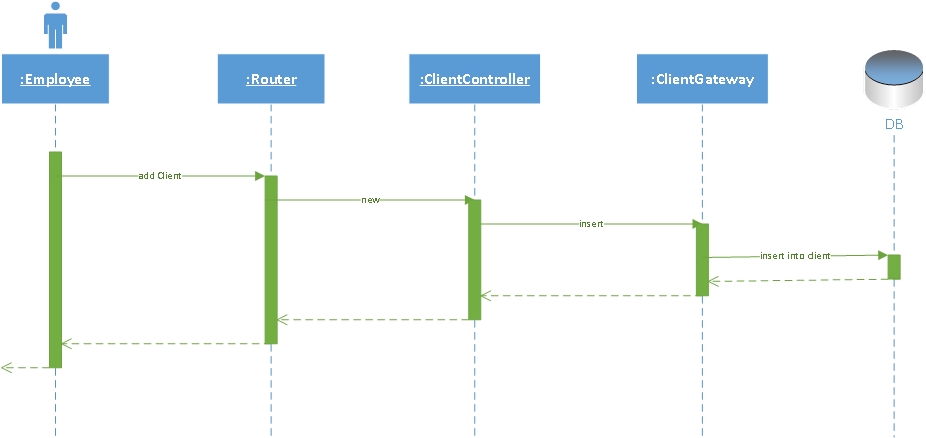


Figure Diagrama de componente

4. UML Sequence Diagrams

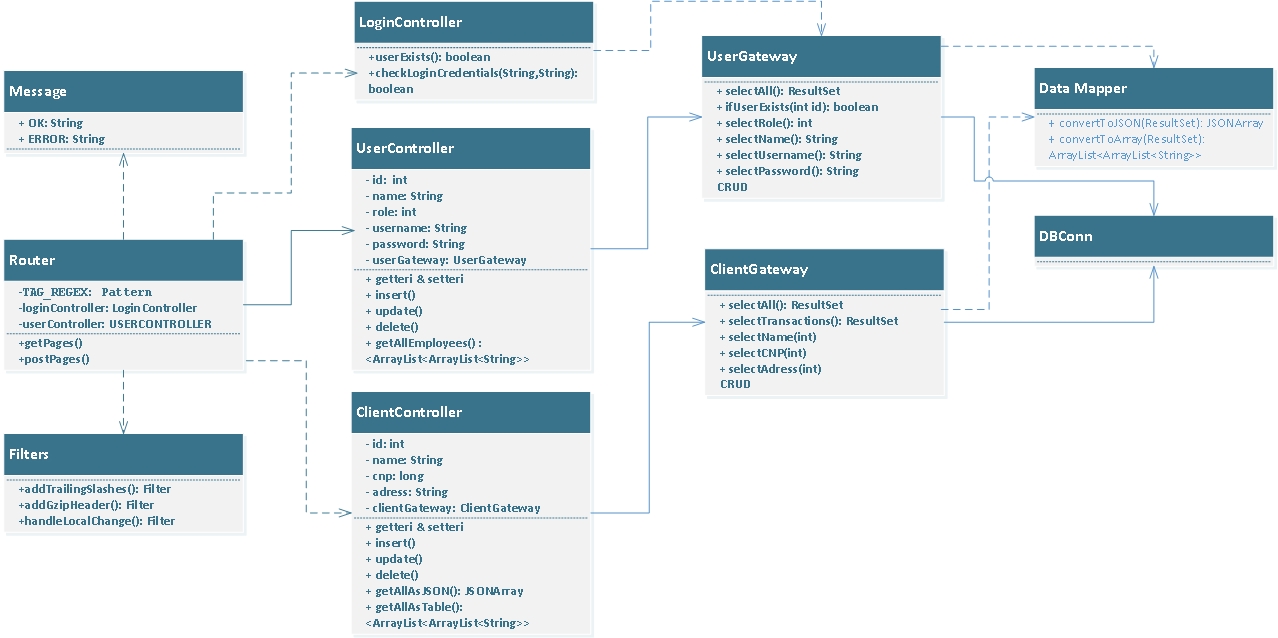


5. Class Design

**5.1 Design Patterns Description**

1. The system as a whole uses a 3-tier layered application which consist of:
   * Presentation layer
   * Business layer
   * Data layer
2. At the presentation level, I use a simple Template View pattern because I have a collection of static HTML pages and the dynamic content parts are inserted using a Template Framework called Velocity. This does a pre parse of the HTML code and searches for special template tags which are replaced with the content fetched from thru the business layer.

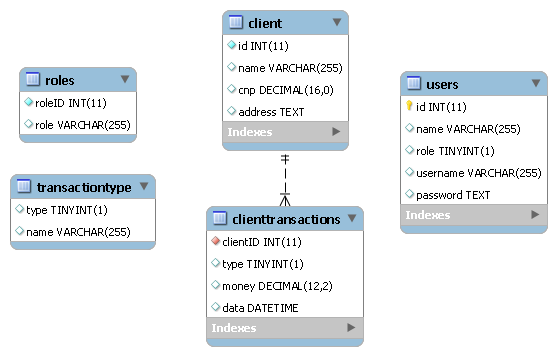
**5.2 UML Class Diagram**



I’ve chosen the structure shown above for this assignment as it follows the 3-tier layered design pattern. I’ve taken care into creating a strict top down structure which doesn’t have backward dependencies from the layers on the bottom toward the ones on the top.

Not shown in the diagram is the Template View pattern I used with the Velocity Framework, which is dependent only from the classes that are in the Presentation Layer. I find this important, as it reduces the interference between the java dynamic content and the static HTML to a minimum.

6. Data Model



For this assignment I’ve used a simple Database structure that uses two helper tables (roles and transactiontype). The two tables play the role of a LUT and they might have been implemented at the application level too. I’ve wanted to maintain though a clear separation between any type of data, that could in principle, be independent of the application used. In this way, I am sure certain traits will not be lost, if for example, the framework changes drastically.

At the application data layer, I’ve first used Active Record, but I’ve realized ( the hard way ) that certain classes started to lose their encapsulation, especially due to some static methods which intuitively would have seemed independent of any type of object. This problem forced me to redesign the application so I’ve chosen TableGateway, as it offers simple link with the database, maintaining in the same time a pure relationship with it.

I’ve created two TableGateways, for each of the major tables that can be find in the database too, clients and users. Both gateways implement finder methods along with the classical insert(), update(), delete().

I’ve found at the end useful to define a new class at the application layer, called DataMapper, which should in principle be on the long run, a purely static class because it’s purpose is solely to translate the data from one format to another.

7. System Testing

Because I don’t have a large scale system, I decided to skip unit testing, as I was pretty confident in the individual units of the application.

During the entire time of development, I’ve done integration testing in a top down manner, because it just so happened I had the high level modules developed first. This was an advantage in the sense I was more than confident in the routers and the methods that handled data transmission thru the services the presentation layer provided. The major downside however was that low level utilities were tested last and unfortunately, due to some code mistakes, I’ve found to be very hard and quite time consuming to find the sources of the data inconsistencies.

8. Bibliography

* Fowler, Martin (2002). Patterns of Enterprise Application Architecture. Addison-Wesley
* <http://thierryroussel.free.fr/java/books/martinfowler/www.martinfowler.com/isa/rowDataGateway.html>
* <http://thierryroussel.free.fr/java/books/martinfowler/www.martinfowler.com/isa/tableDataGateway.html>
* <http://richard.jp.leguen.ca/tutoring/soen343-f2010/tutorials/implementing-data-mapper/>
* <https://martinfowler.com/eaaCatalog/dataMapper.html>