Bank Management System

Analysis and Design Document

Student: Onaciu Andreea

**Group: 30434**

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.

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.

# 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

• Secure access of confidential data.

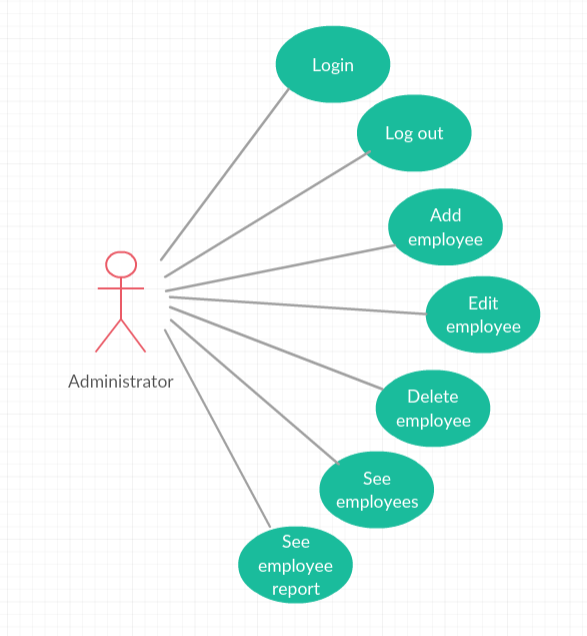
• 24X7 availability

• Better component design to get better performance at peak time

• Flexible service based architecture will be highly desirable for future extensions.

2. Use-Case Model

**2.1 Administrator use case diagram**



**2.2 Employee use case diagram**



Use case: Add client

Level: summary level;

Primary actor: employee;

Main success scenario: provide valid information for the new client;

Extensions: if invalid information is provided the client should not be saved.

Use case: View client

Level: summary level;

Primary actor: employee;

Main success scenario: provide a valid CNP for the client and then the information stored in the database about the client will be shown;

Extensions: if no user with that CNP is found in the database then no information is shown.

Use case: Update client

Level: summary level;

Primary actor: employee;

Main success scenario: search the client who’s information you want to edit and provide new provide valid information for the new client;

Extensions: if invalid information is provided the client should not be saved.

Use case: Add account

Level: summary level;

Primary actor: employee;

Main success scenario: search the client who wants to have a new account and provide valid information for the new account;

Extensions: if invalid information is provided the account should not be saved.

Use case: Delete account

Level: summary level;

Primary actor: employee;

Main success scenario: search the account you want to delete by introducing client and account information then select the option to delete it;

Extensions: if invalid information is provided for the client and account, a message should be shown;

Use case: View accounts

Level: summary level;

Primary actor: employee;

Main success scenario: select view account and by introducing the client information a table with all the accounts and data regarding the account should be provided;

Extensions: if invalid information is provided for the client an error message should be shown.

Use case: Update account

Level: summary level;

Primary actor: employee;

Main success scenario: search the account you want to update and provide valid information for the new account;

Extensions: if invalid information is provided the account should not be saved.

Use case: Transfer money between accounts

Level: summary level;

Primary actor: employee;

Main success scenario: enter the source and the destination account, enter also the amount of money to transfer, if all the information is valid and the source account has enough money in the account to transfer the wanted amount, the transfer is made, changing the status of the source account and of the destination account;

Extensions: if invalid information like invalid source account, invalid destination account, invalid sum of money (negative sum or greater than in the source account) the transfer shouldn’t be made and an error message indicating the problem should be shown.

Use case: Pay utility bills

Level: summary level;

Primary actor: employee;

Main success scenario: enter the account information and select the utility you want to pay and the sum of money you want to pay;

Extensions: if invalid information is provided the transfer should not be made;

Use case: Add employee

Level: summary level;

Primary actor: administrator;

Main success scenario: provide valid information for the new employee;

Extensions: if invalid information is provided the employee should not be saved.

Use case: Update employee

Level: summary level;

Primary actor: administrator;

Main success scenario: search the employee who’s information you want to edit and provide valid information for the new employee;

Extensions: if invalid information is provided the employee should not be saved.

Use case: Delete employee

Level: summary level;

Primary actor: administrator;

Main success scenario: search the employee you want to delete and press the delete button;

Extensions: if invalid information is provided no employee should be deleted.

Use case: See employees

Level: summary level;

Primary actor: administrator;

Main success scenario: a list with all the employees should be provided;

Extensions:-;

Use case: See user report

Level: summary level;

Primary actor: administrator;

Main success scenario: search the employee you want to see its report, a table with information about all his activity should be shown;

Extensions: if invalid information is provided no report should be shown and an error message should be shown;

3. System Architectural Design

**3.1 Architectural Pattern Description**

The project will be developed in a layered architecture pattern used in the majority of spring web applications. It respects the principles of Separation of concerns and The Keep It Simple Stupid (KISS) principle. The layers of the application will be:

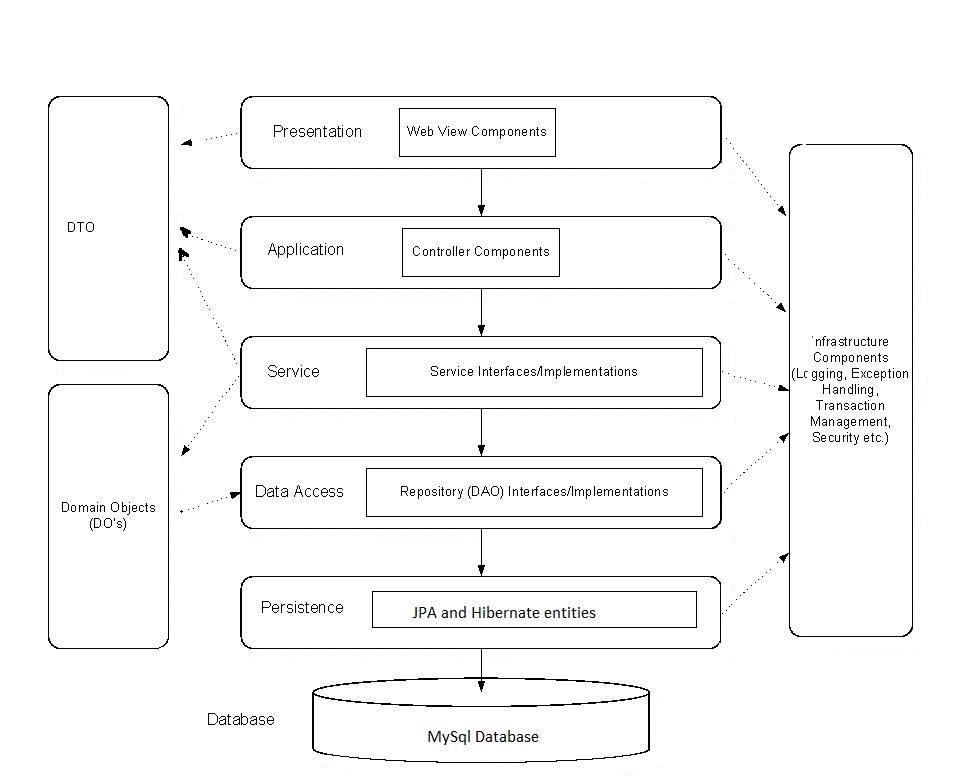
* The repository layer is the lowest layer of a web application. It is responsible of communicating with the used data storage.
* The service layer resides below the web layer. It acts as a transaction boundary and contains both application and infrastructure services. The application service provides the public API of the service layer. They also act as a transaction boundary and are responsible of authorization. The infrastructure services contain the “plumbing code” that communicates with external resources such as file systems, databases, or email servers. Often these methods are used by more than a one application service.
* The web layer is the uppermost layer of a web application. It is responsible of processing user’s input and returning the correct response back to the user. The web layer must also handle the exceptions thrown by the other layers. Because the web layer is the entry point of our application, it must take care of authentication and act as a first line of defense against unauthorized users.

The benefits of this kind of architecture is that it is a low coupling architecture, it offers the possibility of improving performance (sometimes).The maintenance of your application is easier because of the low coupling between layers. Adding more functionality to your application is made easier. Layers make your application more testable.

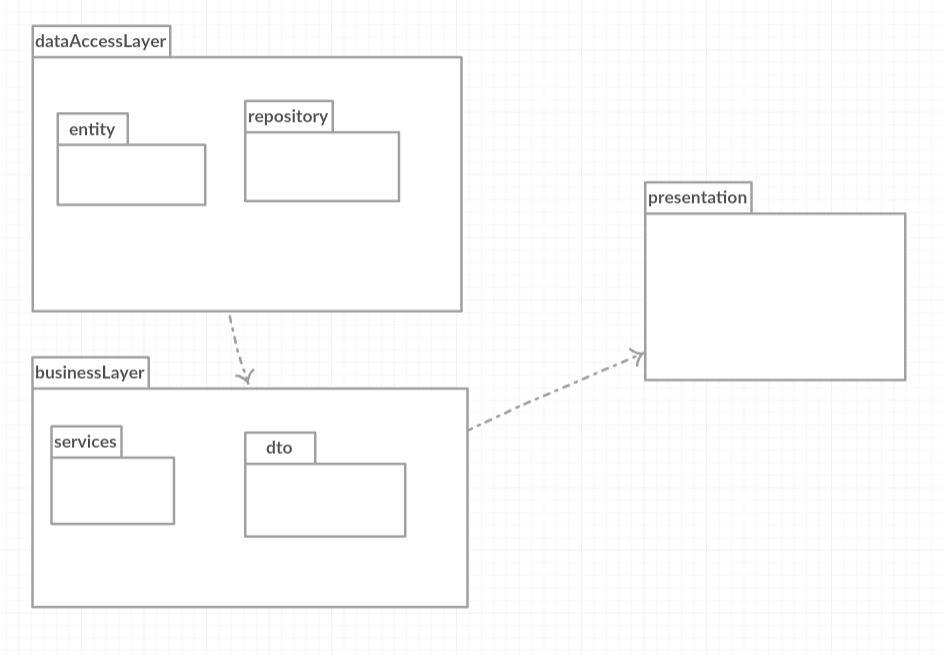
**3.2 Diagrams**

3.2.1 Conceptual architecture design

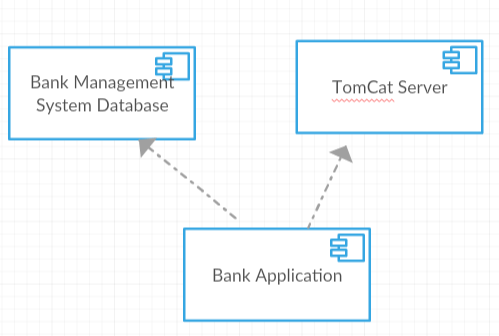
In the figure it can be seen how the architecture is structured on layers: presentation layer, controller layer, service layer, data access layer, database layer.



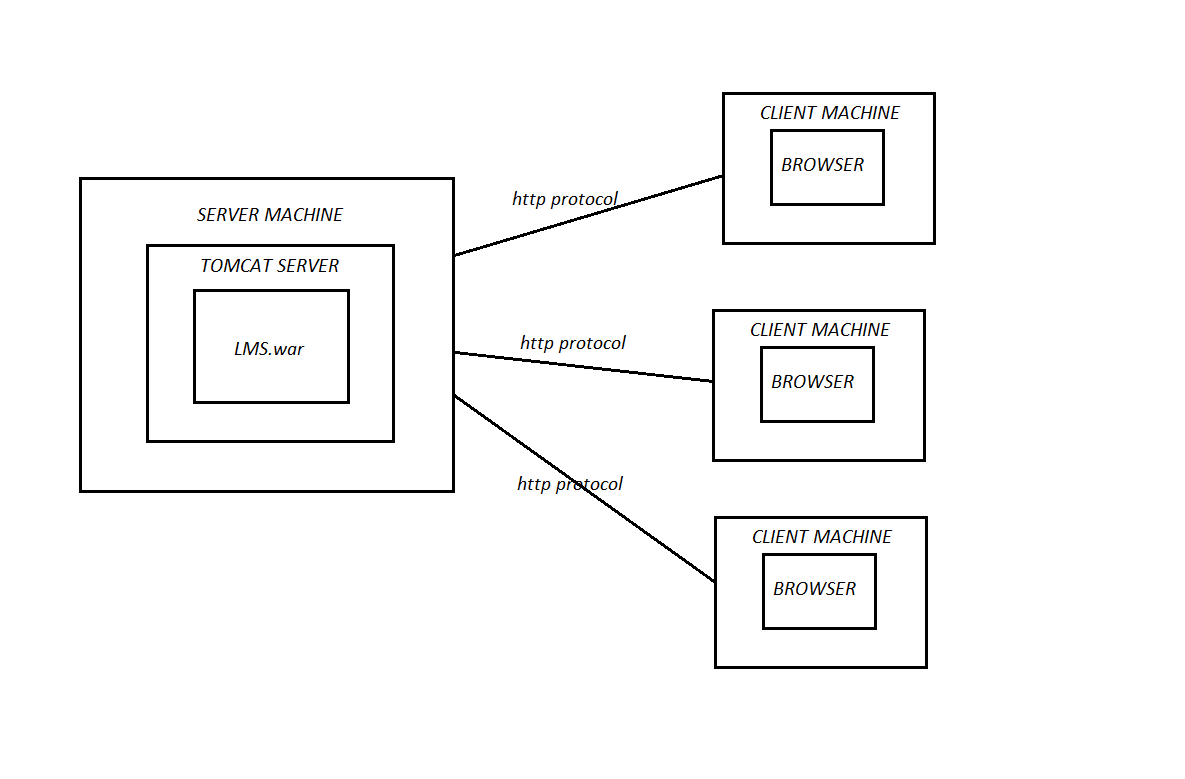
3.2.2 Package diagram

**

3.2.3 Component diagram

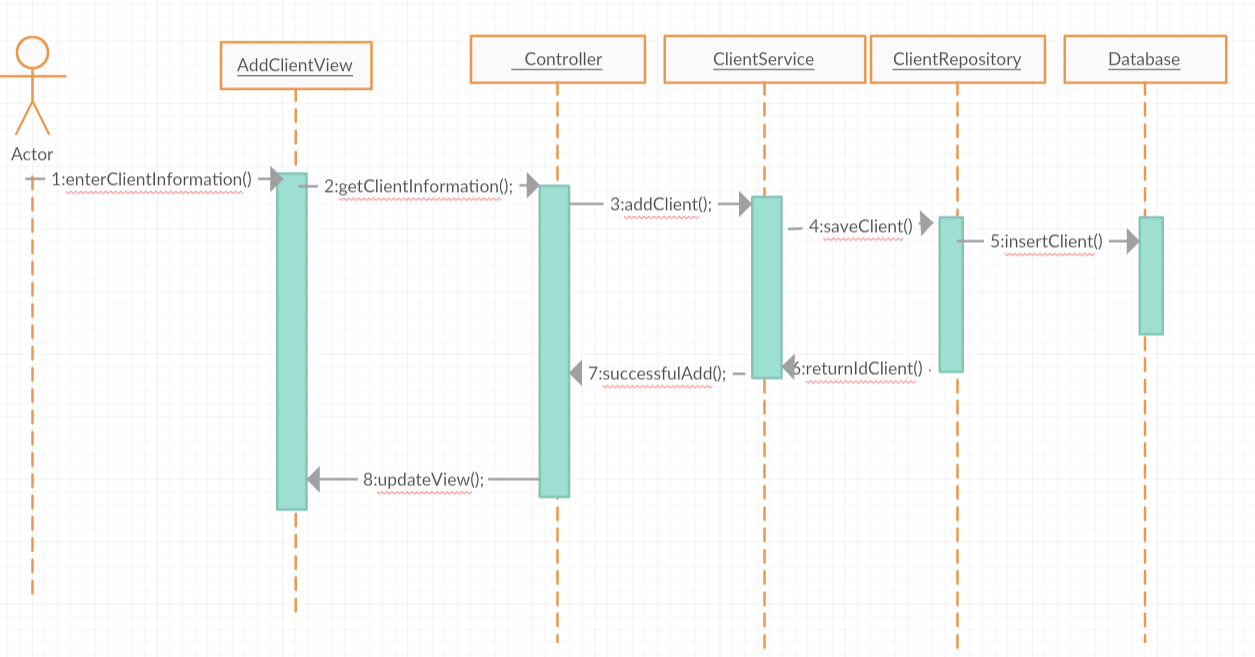


3.2.4 Deployment diagram

**

4. UML Sequence Diagrams

Add Client sequence diagram

**

5. Class Design

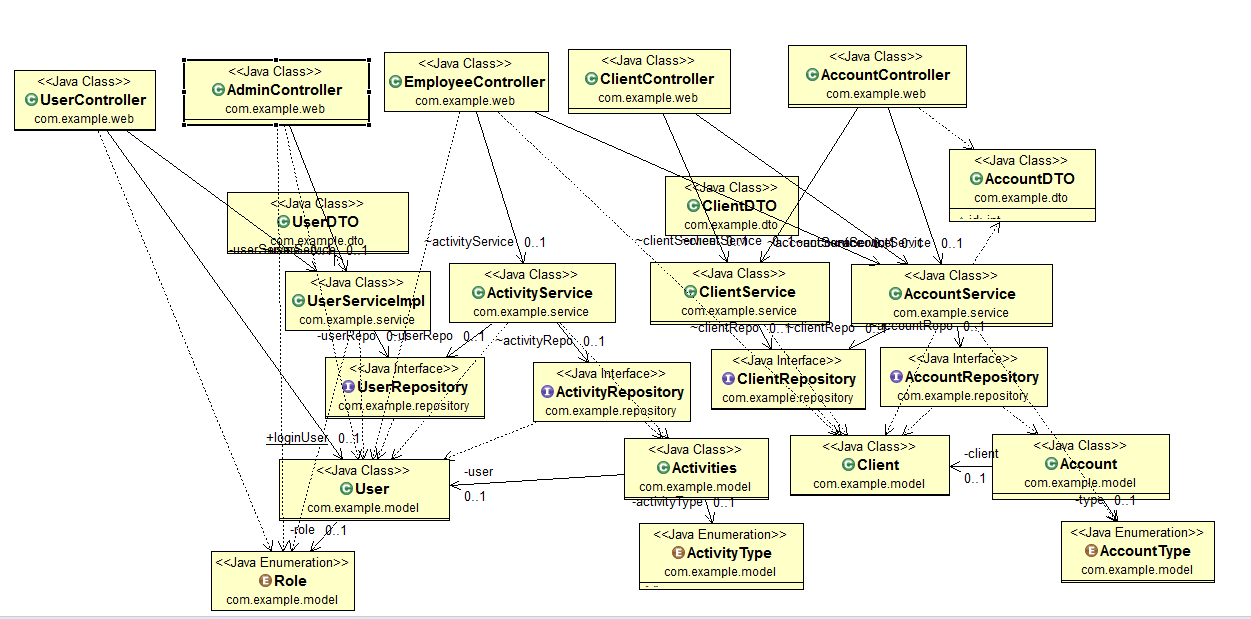
**5.1 Design Patterns Description**

For developing the application I used as some design patterns for each layer, some of them are used due to the frameworks in which the application is made. For the data access level I used the Spring Hibernate framework which uses the Data Model design pattern and Proxy Pattern. At runtime, Hibernate dynamically generates proxies from the entity code through bytecode generation. These proxies are aware of the ORM functionality and implement the object lifecycle, while pretending to belong the class that the programmer created.

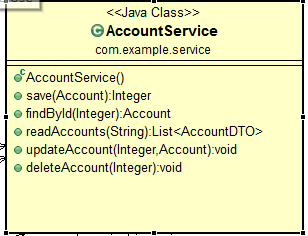
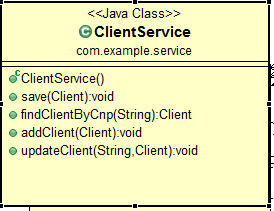
The business logic is created using the Services layer pattern, services for each domain model entity, they include all the needed logic to provide information for the user information and for saving into the database.

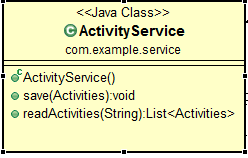
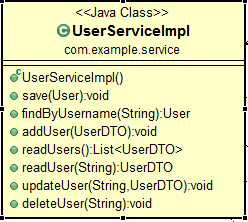
The presentation layer is created using MVC pattern. The Model handles the state of the application. The View is the representation of the user interface. User actions on the View are sanded to the Controller.

**5.2 UML Class Diagram**

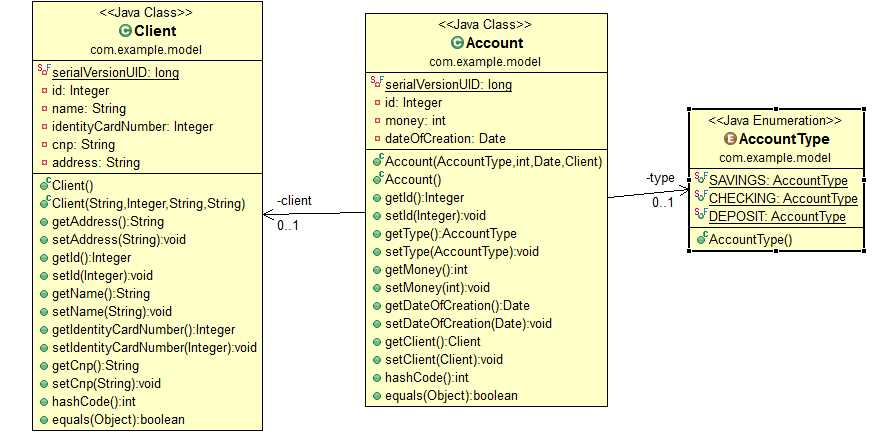


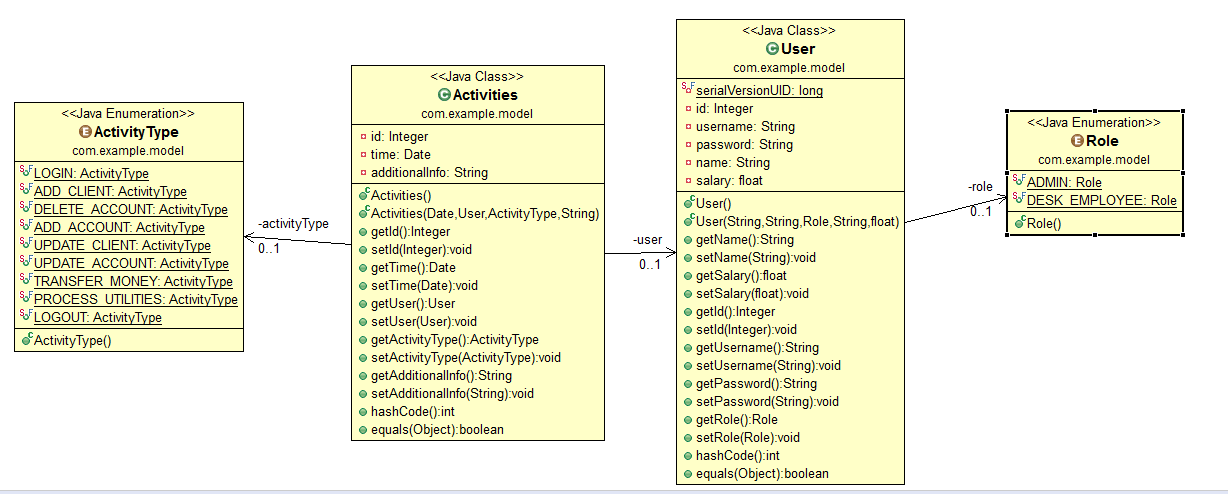
Services classes:

* *

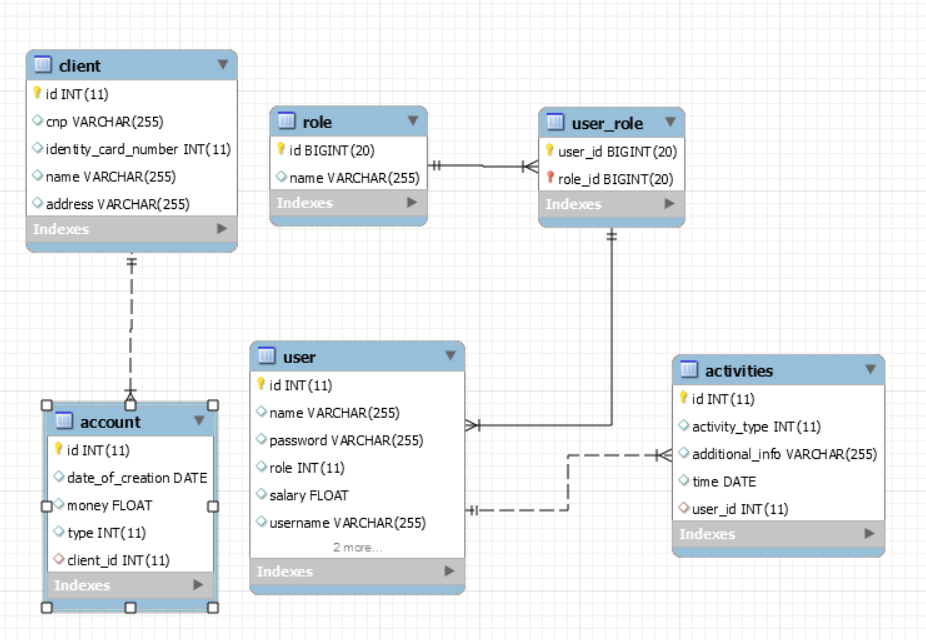
* *

Entities:





6. Data Model

**

7. System Testing

For testing the system the following strategies were used: unit testing, validation testing and testing methods like boundary analysis. The system contains unit tests for each service of the application.

Validation testing is made in the business logic of the application by testing each user input to have correct and valid values, if invalid values are placed then an error message which communicates the error is shown. The validation methods contain CNP validation, saving username and password validation, positive values for salaries and money transfer.

Boundary analysis was made for the transfer money use cases, to allow only valid amounts of money to be transferred from the source account, explicitly the sum of money transferred must be lower than the source account with maximum 1$, so as the source account to not remain empty after the transfer.

8. Bibliography

* <https://zeekat.nl/articles/mvc-for-the-web.html>
* <http://stackoverflow.com/questions/2637114/what-are-the-benefits-of-an-n-layered-architecture>