Bank Application for Employees

Analysis and Design Document

Student:Roman Bogdan-Ovidiu

**Group:30431**

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:

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

The administrator user can perform the following operations:

1. CRUD on employees’ information.
2. Generate reports for a particular period containing the activities performed by an employee.

# Non-functional Requirements

1. Performance – 1 sec response time for every operation from database like add remove view list . Aprox 13 seconds to run the application on server.
2. Availability- 24 h/day , 353 days/year, 98.9%
3. Reliability-99.9% correct given outputs from the system.
4. Security- Spring security with log in and admin user
5. Manageability- depends from case to case, depends on how the team works and how big the bug is.

2. Use-Case Model

*Use case: <Delete client>*

*Level: <one of: summary level, user-goal level, sub-function>*

*Primary actor: <UserEmployee>*

*Main success scenario: <1. Open the application*

*2. Log in into your user account*

*3. Logic successfully*

*4. Go to view/edit/delete client*

*5. See the list of clients, chose one , the one you want to delete*

*6. Press delete button*

*7. Client is not in the table anymore: Succesfully deleted!*

*8. Log Out from the application>*

*Extensions: <2.Step 2 failed to log in, try again*

*3. The client you wanted to remove is not in the table, check again.*

*>*



3. System Architectural Design

**3.1 Architectural Pattern Description**

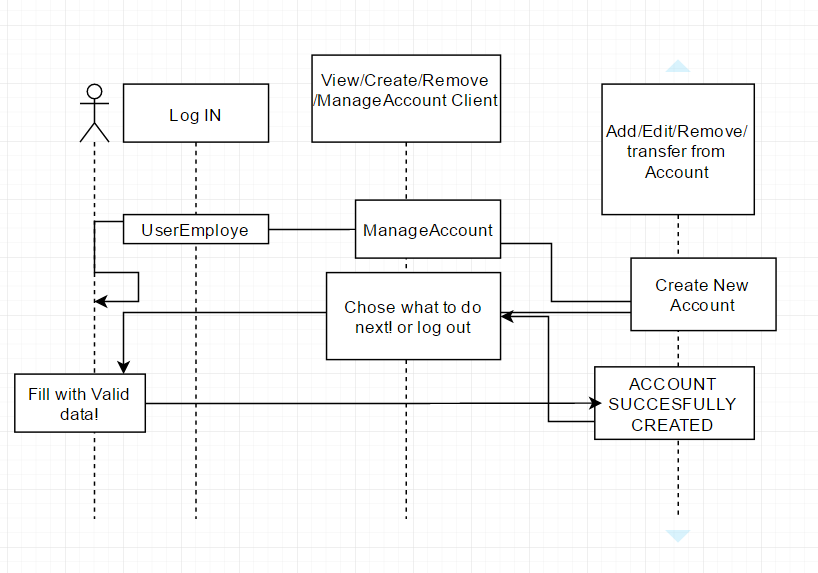
Used the Layers architectural pattern to organize my application. Common principles for designs that use the layered architectural style include:

* **Abstraction**. Layered architecture abstracts the view of the system as whole while providing enough detail to understand the roles and responsibilities of individual layers and the relationship between them.
* **Encapsulation**. No assumptions need to be made about data types, methods and properties, or implementation during design, as these features are not exposed at layer boundaries.
* **Clearly defined functional layers**. The separation between functionality in each layer is clear. Upper layers such as the presentation layer send commands to lower layers, such as the business and data layers, and may react to events in these layers, allowing data to flow both up and down between the layers.
* **High cohesion**. Well-defined responsibility boundaries for each layer, and ensuring that each layer contains functionality directly related to the tasks of that layer, will help to maximize cohesion within the layer.
* **Reusable**. Lower layers have no dependencies on higher layers, potentially allowing them to be reusable in other scenarios.
* **Loose coupling**. Communication between layers is based on abstraction and events to provide loose coupling between layers.

**3.2 Diagrams**

*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*

4. UML Sequence Diagrams



5. Class Design

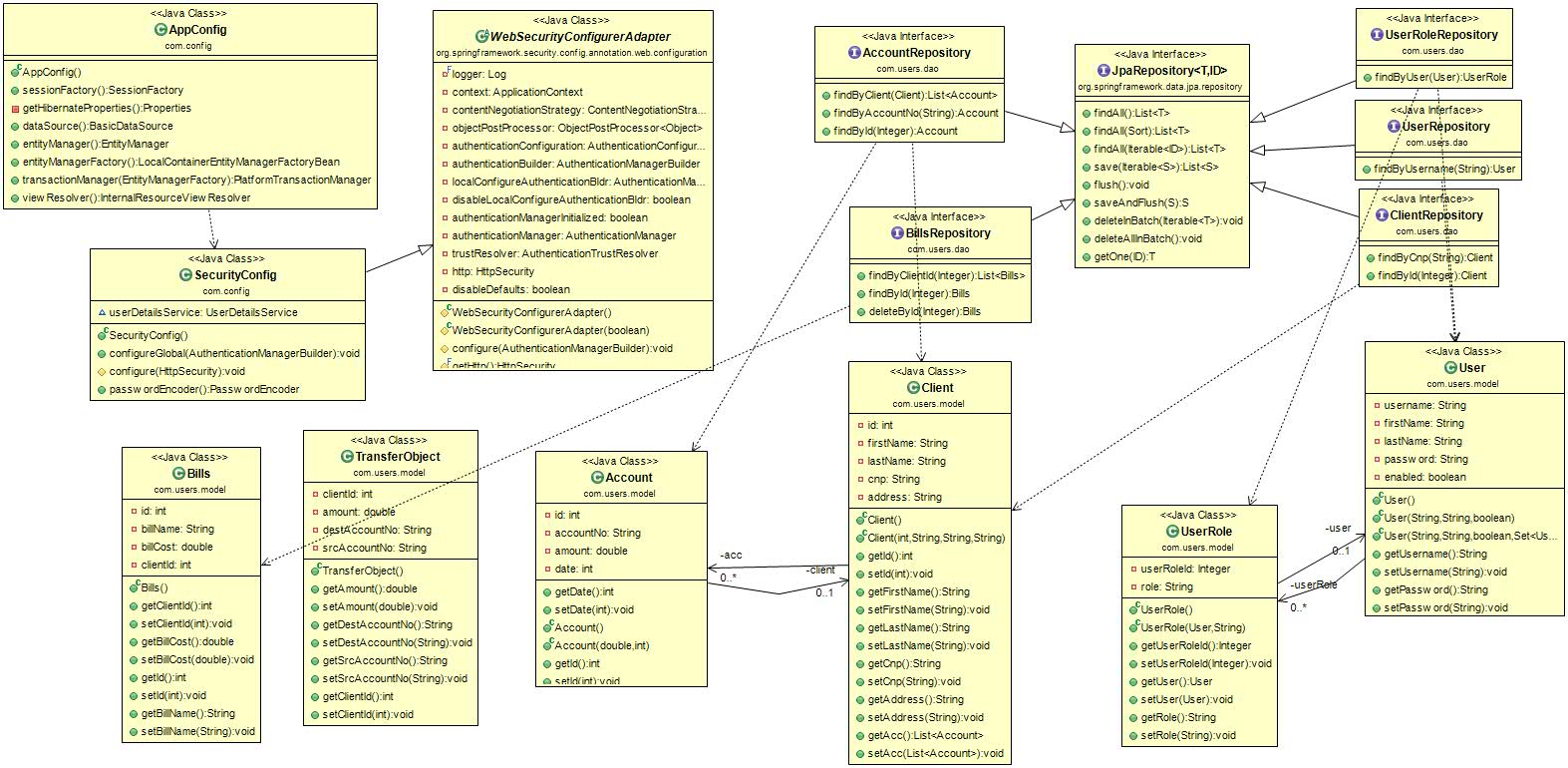
**5.1 Design Patterns Description**

*I did not use any explicit design pattern.*

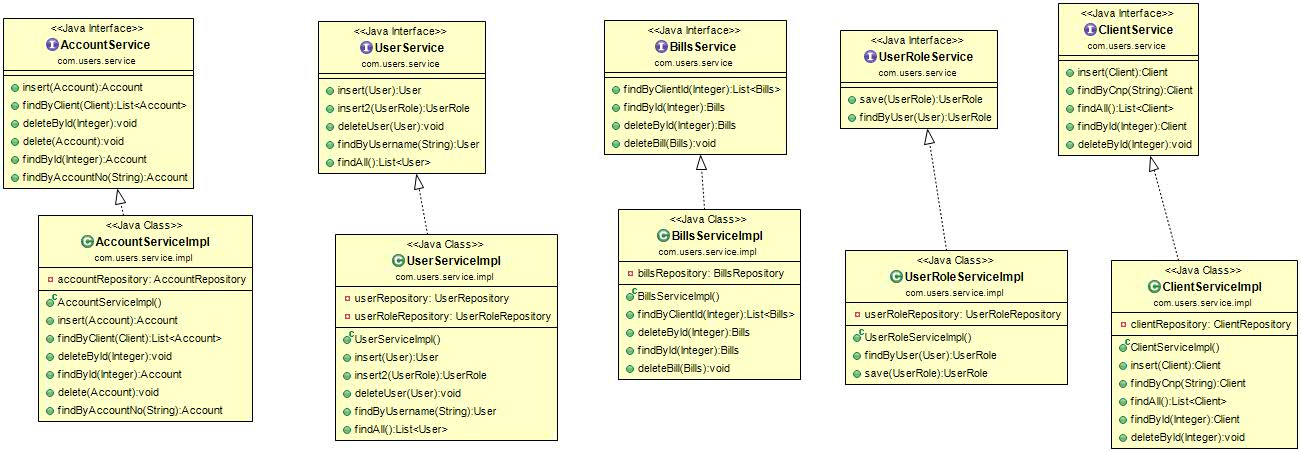
**5.2 UML Class Diagram**

*Com.dao and com.model represents the Data Layer, they work with db. Com.service and com.serviceImpl represents the Business layer where the methods are implements for inserting /deleting/ finding in the database. And the Interface Layer is formed by com.controller, and JSP pages where is created the graphical interface and the functionality. A small diagram of model.com classes.(which represent the entities)*

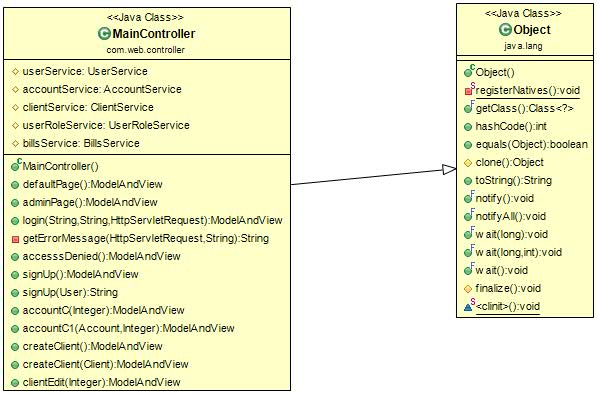
*DATALAYER-DIAGRAM:*



*BussinesLayer –Diagram*



*UI Layer Diagram: (Interface)+ jsps*



6. Data Model

*The data models are : Users, Client, Account, UserRole*

*User represents the employee who can perform operations like create add remove client or client account. User is of 2 types, employee or admin, this is set by the second entity UserRole, containing the set of roles: USER\_ROLE, ADMIN\_ROLE. Admin can perform a little bit more than the user like, create,edit ,remove or generate reports.*

*Client represents the customer of the bank, the one who can have an account or more than one account in the bank. They have as fields Firstname, Lastname, Cnp and Id, and a list of accounts for every client. Account is linked to client by cliendID field, an account can be represented only by 1 clientid but a client can have more accounts.*

7. System Testing

*The system is tested by JUnit methods, (create and remove). The field that must be correct as account no and CNP are validated before , same as username and password.*

8. Bibliography

<http://stackoverflow.com/questions/8669387/>

<http://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial>

http://www.tutorialspoint.com/jsp/