Assignment 1

Analysis and Design Document

Student: Vereș Adela

**Group: 30235**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 4

2. Use-Case Model 4

3. System Architectural Design 5

4. UML Sequence Diagrams 7

5. Class Design 8

6. Data Model 9

7. System Testing 10

8. Bibliography 10

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

* Ability to authenticate either as an employee, or as an admin, using a username and a password, via the login page.
* Ability to do the following, while logged in as an employee:

- access only the login page and the clients page

- view the list of existing registered clients, and their details

- update a selected client

- delete an existing client

- create a new client - only possible if introducing a personal numerical code different from the existing ones

- view a client’s existing accounts

- update a client’s existing account – by depositing in or withdrawing a sum of money from the account

- delete a client’s existing account

- add a new account to a specific client

- transfer money between the selected accounts of a specific client

* Ability to do the following, while logged in as an admin:

- access only the login page and the employee page

- view the list of existing active employees, and their details

- update a selected employee

- delete a selected employee

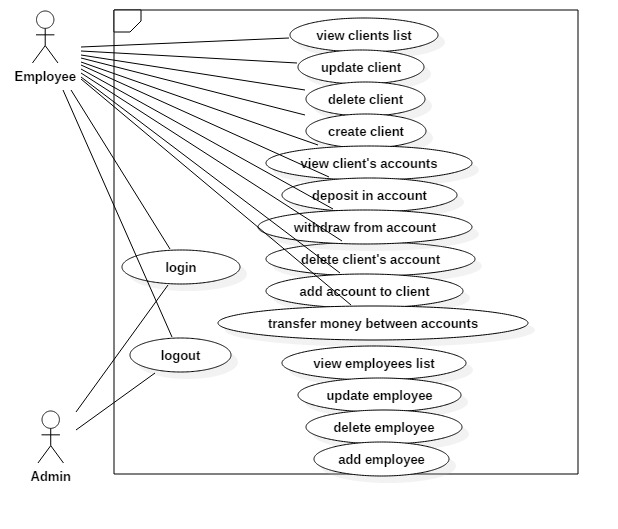
- add a new employee - only possible if introducing a personal numerical code different from the existing ones.

# Non-functional Requirements

The system presents the following quality attributes:

* *Usability*: the software product can be used by specific consumers – employees and the administrator – in order to facilitate easy, effective access to the application. This is enabled due to the use of user-friendly, intuitive graphical user interface. The layout of the webpage is clearly designed in order to support all operations, while the server side implementation renders qualitative response time.
* In terms of *performance*, the system is defined by a satisfactory *response time,* low *latency,* satisfactory utilisation of *computing resources.*
* *Reusability* and *extensibility* are supported due to the use of *low coupling* and separated functionalities that comes with the implementation of the *Layers* architectural pattern.
* *Scalability* is supported as well, being possible to handle large amounts of data – high number of clients, accounts, employees.

2. Use-Case Model



Use case: deposit in account

Level: user-goal level

Primary actor: employee

Main success scenario:

* login as an employee
* on client’s page, view the clients, locate the client searched for
* activate the accounts button, next to the client
* in the accounts section, locate the desired account from the list of accounts
* next to the account enter the sum to deposit in the adequate field
* activate the ‘Deposit’ button
* click the ‘Refresh’ button, in order to see the changes
* click the ‘Done’ button, in order to close the Accounts sections, and return to the client’s page

3. System Architectural Design

**3.1 Architectural Pattern Description**

The application is a web application that uses the Layered architectural pattern. The main layers and their included packages are as follows:

● ***Web (Presentation)*** ***Layer***

This layer comprises is the uppermost layer of the application that interracts with the user, processing user input and returning correct responses. The web layer also handles the exceptions thrown by other layers, and models the first line of authentication. Here is the starting point of the application, as well, the class containing the “main” method, responsible for running the application.

Included packages:

• **controller**

• **util**

● ***Service (Bussiness) Layer***

The service layer mediates between the user and the application data, providing the needed processing logic. This layer contains both application and infrastructure services. The *application services* provide the public API of the service layer, also acting as a transaction boundary. The *infrastructure services* are the ones that communicate with the external resources, in this case, with the database.

Included packages:

• **service**

● ***Repository (Data) Layer***

The repository layer is responsible with providing access to the data storage.

Included packages:

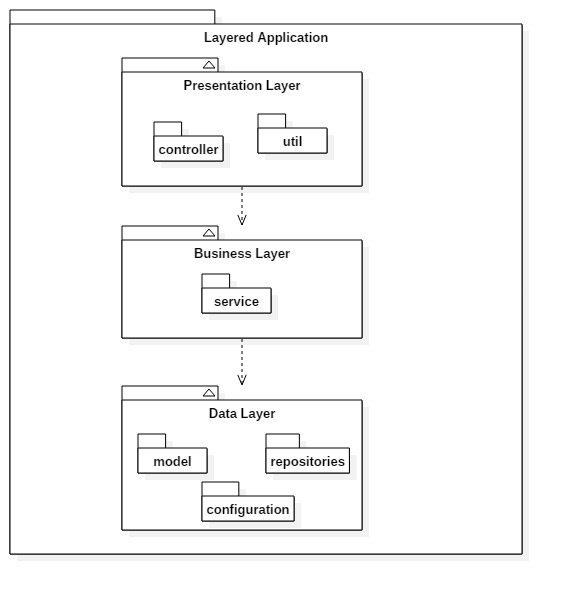
• **model**

• **repositories**

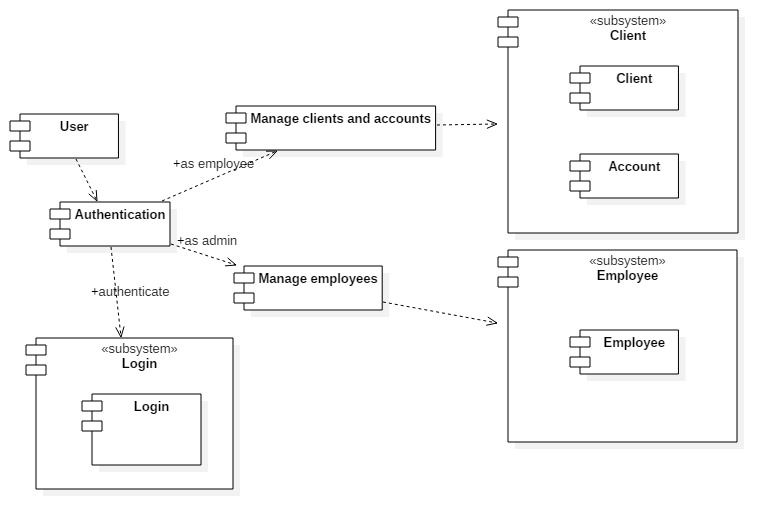
• **configuration**

**3.2 Diagrams**

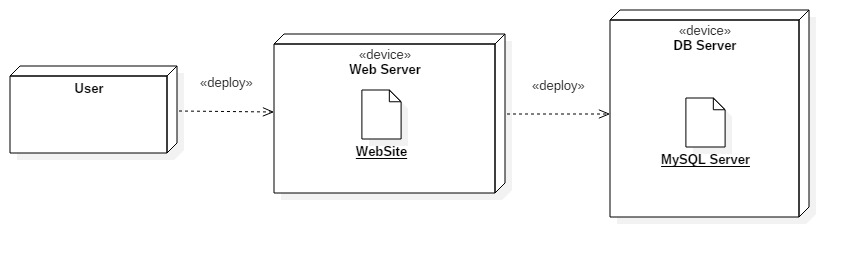
● Package diagram



● Component diagram

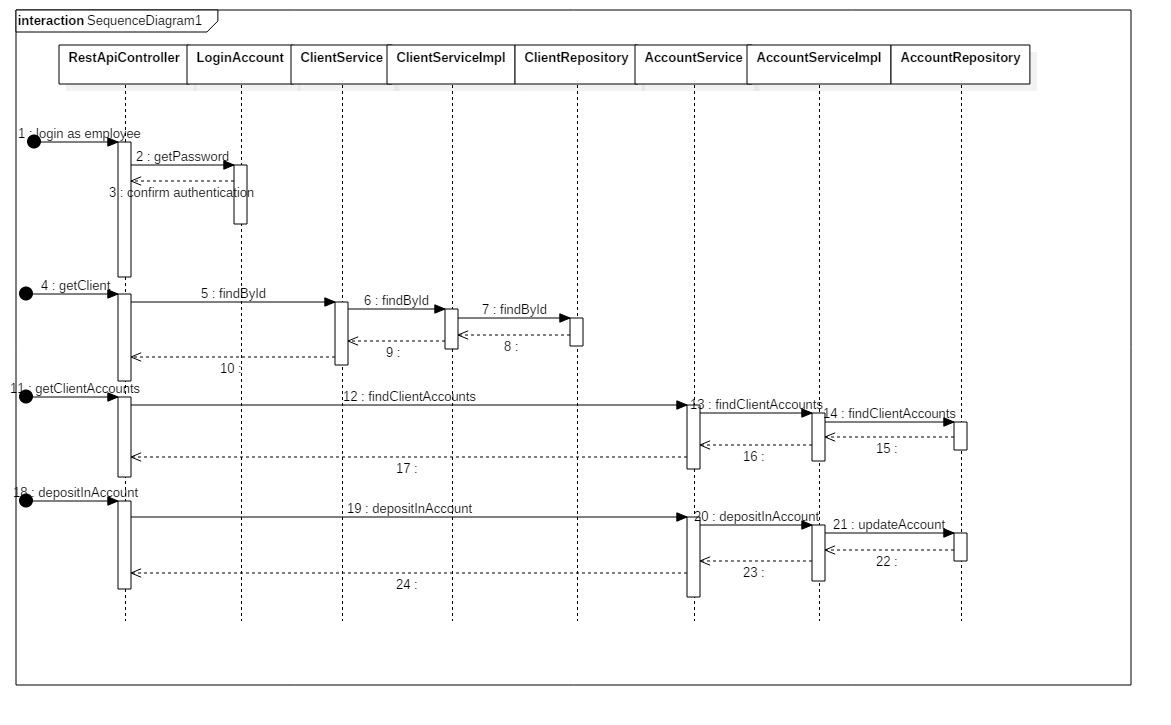


● Deployment diagram



4. UML Sequence Diagrams

Sequence Diagram for “deposit in account” use case:



5. Class Design

**5.1 Design Patterns Description**

The application uses Spring Boot Framework, which has an embeded Tomcat server.

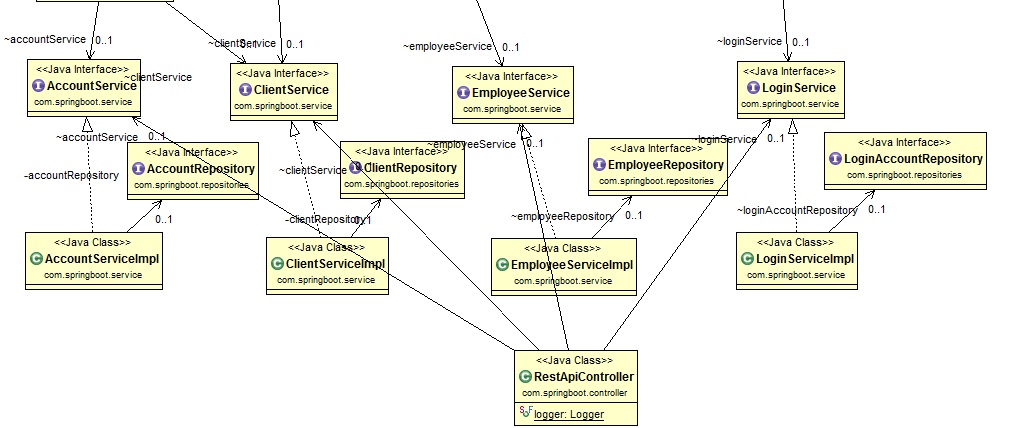
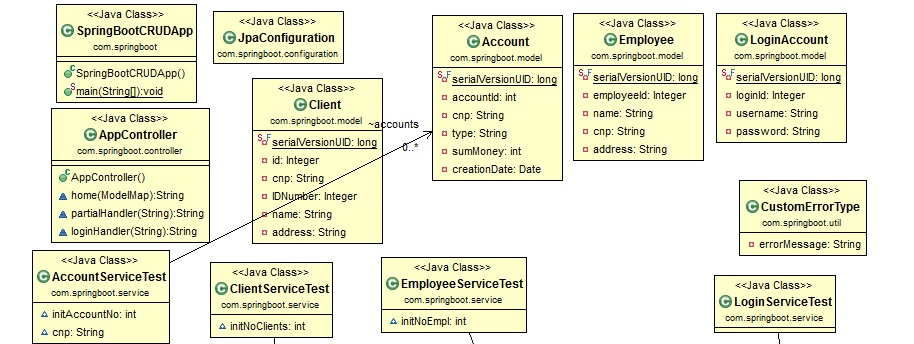
The Hibernate object-relational mapping tool provides a framework for mapping the object-oriented domain model to the relational database. Java classes are mapped to database tables, thus using the **Table Module** pattern, in relation with the **Table Data Gateway,** used in the data layer**.** Thedata layer also uses the **Data Mapper** pattern, represented by the repository classes.

The **Service layer** provides a facade containig some domain logic as well, built around client use-cases.

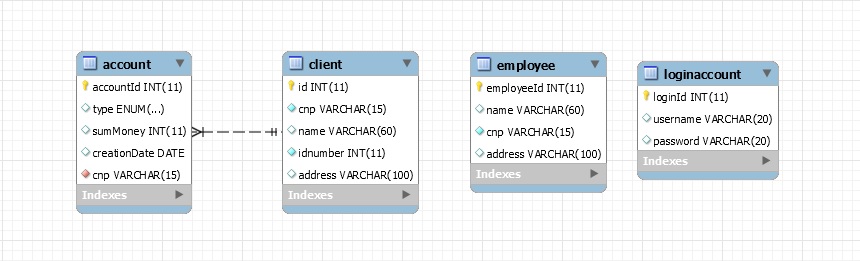
The AngularJS JavaScript framework is used to model the dynamic behaviour of the webpage, as well as bind the application data to html format. The interface is modelled using Html, Css and Bootstrap elements.

Maven is used for dependency management.

**5.2 UML Class Diagram**



6. Data Model



The application uses four main tables:

* Client table: stores client related information such as: Personal Numerical Code, Name, Identity Card Number, Address, while uniquely identifying clients by an id.
* Account table: stores each and every client’s accounts, relating to the client owning the account through a correspondent Personal Numerical Code (cnp in the picture), acting as a foreign key identifier. Account related information, such as: the type – saving or spending account – the amount of money stored, the creation date, are also registered. Accounts are uniquely identified using the accountId identifier.
* Employee table: stores employee related information such as: Name, Personal Numerical Code, Address, being uniquely identified by the employeeId identifier.
* LoginAccount table: stores the accounts available to users, (admin and employee accounts), by registering the usernames and their related passwords. This table is used for registering purposes. The loginId identifier ensures that the login accounts are uniquely identified.

7. System Testing

The system includes unit testing classes for unit testing. The Junit4 framework is used, and tests are methods encapsulated in a single class for each of the corresponding services classes. Thus we have: ClientServiceTest, AccountServiceTest, EmployeeServiceTest, and LoginServiceTest – classes testing each method provided by the correspondent service classes. Methods such as ‘findById, createClient, deleteEmploye, etc, are tested, thus testing in fact the CRUD operations’ well-functioning.

8. Bibliography

1. Spring Framework:  
<https://en.wikipedia.org/wiki/Spring_Framework>

2. Hibernate Framework:  
<https://en.wikipedia.org/wiki/Hibernate_(framework)>

3. Apache Maven:  
<https://en.wikipedia.org/wiki/Apache_Maven>

# 4. AngularJS tutorial: <https://www.w3schools.com/angular/default.asp>

5. Martin Fowler, Patterns of Enterprise Application Architecture:  
<http://disi.unal.edu.co/dacursci/sistemasycomputacion/docs/SWEBOK/Systems%20Engineering%20-%20EAA%20-%20Patterns%20of%20Enterprise%20Application%20Architecture%20-%20Addison%20Wesley.pdf>

# 6. Spring Boot, AngularJS, Spring Data, JPA CRUD App Example:

<http://websystique.com/spring-boot/spring-boot-angularjs-spring-data-jpa-crud-app-example/>

# 7. SQL reference: <https://www.w3schools.com/sql/sql_datatypes.asp>

# 8. Http status codes: <https://httpstatuses.com/>