Assignment A1

Analysis and Design Document

Student: Cocoş Tudor-Adrian

**Group:** **e30433**

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 API to design and implement the API part of a web 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 (Pay) utilities bills (by inserting bill information).

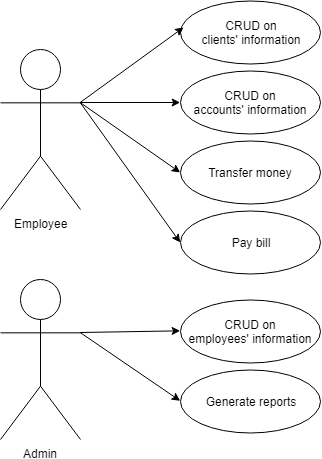
The administrator user can perform the following operations:

- CRUD on employees’ information.

- Generate two types of reports files (one in pdf and one in csv format) for a period containing the activities performed by an employee.

# Non-functional Requirements

A login system that checks the credentials with the database instead of the current credentials that are defined in the *application.properties* file.

**

2. Use-Case Model

Use case: transfer money

Level: user-goal level

Primary actor: employee

Main success scenario: check for existence of both origin and

destination accounts, check for the origin account to have at least the amount required for the transfer, then transfer the sum by subtracting the sum from the origin account’s balance and adding that to the balance of the destination account

Extensions: other scenarios include non-existing accounts or

origin account not having enough funds

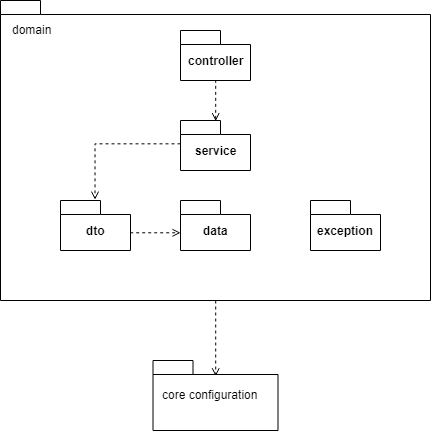
3. System Architectural Design

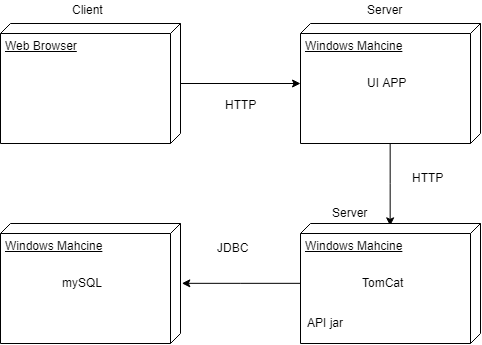
**3.1 Architectural Pattern Description**

*Layered Architecture*

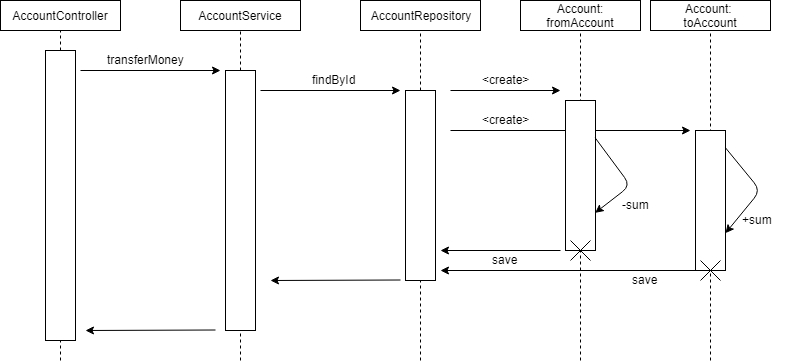
“Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database. In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components.”[1]

**3.2 Diagrams**

**

**

4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

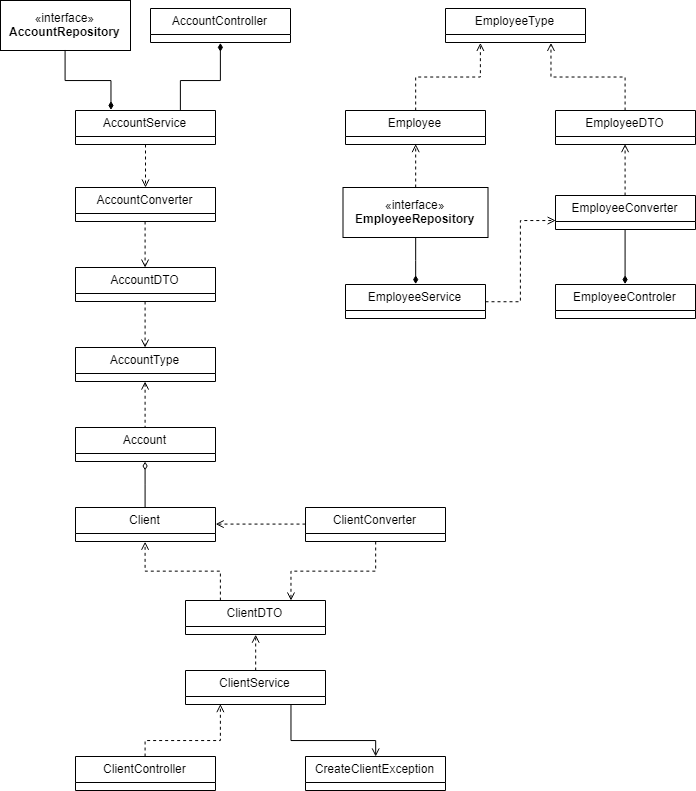
*Factory method design pattern*

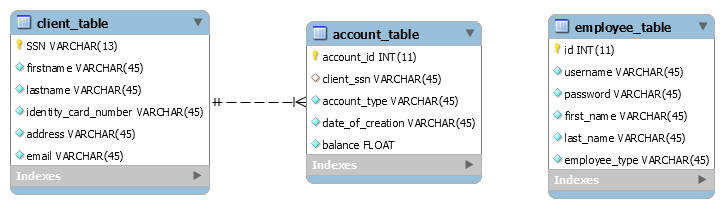
“In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.”[2]

*Singleton pattern*

“In software engineering, the singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system. The term comes from the mathematical concept of a singleton. Critics consider the singleton to be an anti-pattern in that it is frequently used in scenarios where it is not beneficial, introduces unnecessary restrictions in situations where a sole instance of a class is not actually required, and introduces global state into an application.”[3]

**5.2 UML Class Diagram**



**6. Data Model

7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

8. Bibliography

1. <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

2. <https://en.wikipedia.org/wiki/Factory_method_pattern>

3. <https://en.wikipedia.org/wiki/Singleton_pattern>