<Assignment1>

Analysis and Design Document

Student:Ionel Carmen

**Group:30237**

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

Utilizati JAVA API pentru a implemeta o aplicatie pentru angajatii unei banci. Aplicatia trebuie sa aiba doua tipuri de utilizatori : un utilizator reprezentat de un angajat al bancii si un utilizator reprezentat de administrator. Utilizatorii trebuie sa furnizeze un username si o parola pentru a putea utiliza aplicatia.

# Functional Requirements

Pentru aceasta aplicatie, datele trebuie stocate intr-o baza de date.

Se va utilizadesign pattern-ul Layers pentru organizarea codului aplicatiei.

Utlizatorul reprezentat de catre angajatul bancii trebuie sa aiba la dispozitie urmatoarele operatii: adaugarea, stergerea si vizualizarea unui cont, crearea, actualizarea, stergerea si vizualizarea unui client, transfer intre conturi si platirea unei facturi.

Utilizatorul reprezentat de catre administratorul bancii trebuie sa dispuna de urmatoarele operatii: adaugarea, stergerea, actualizarea si izualizarea unui angajat si generearea de rapoarte pentru un angajat.

# Non-functional Requirements

# Availability: Reprezinta probabilitatea ca un sistem sa functioneze corect chiar si dupa folosirea lui un anumit timp. Atat datele cat si pagina web vor fi disponibile atat timp cat autorul sau administratorul nu le sterge.

## Performanta: Una dintre cele mai importante cerinte non-functionale dorite de catre client este un sistem cu o rata mare de performanta. Obiectivul de performanta este divizat in trei: timp, spatiu si receptivitate. Timpul este impartit in timp de raspuns si timp de procesare. Receptivitatea reprezinta cat de repede raspunde sistemul la interogari, iar spatiul reprezinta spatiul de stocare. Pentru o buna performanta, nu trebuie irosit spatiul.

## Security: Obiectivul de performanta este divizat in: confidentialitate, integritate si disponibilitate. Pentru a acoperi confidentialitatea este importanta autentificarea: utilizatorii trebuie sa se conecteze la sistem prin furnizarea unui username si a unei parole. Integritatea se refera la validarea datelor si asigura consistenta.

## Testability: Reprezinta gradul in care un sistem suporta testarea intr-un anumit context. Pentru a testa un sistem trebuie sa avem in vedere:

-proprietatile metodelor de test folosite

-cerintele trebuie sa fie consistente, complete, neambigue, realizabile

-componentele software trebuie sa fie: controlabile, observabile, heterogene, documentate.

## Usability: Orice sistem creat pentru a fi utilizat de catre oameni trebuie sa fie usor de folosit, usor de invatat, usor de amintit (instructiunile) si folositor utilizatorului.

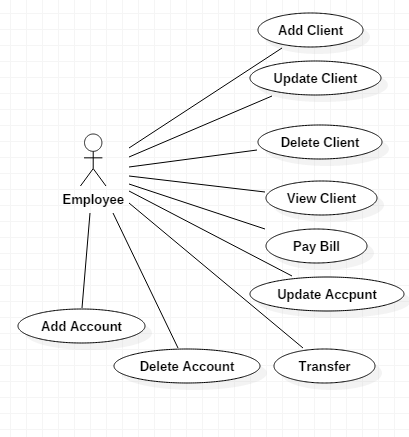
Principii:

-concentrat pe utilizatorii finali si pe nevoile acestora

-design iterative, acesta fiind imbunatatit in permanenta

Pentru asta. Aplicatia dispune de o interfata web, usor de utilizat si interactiva.

2. Use-Case Model

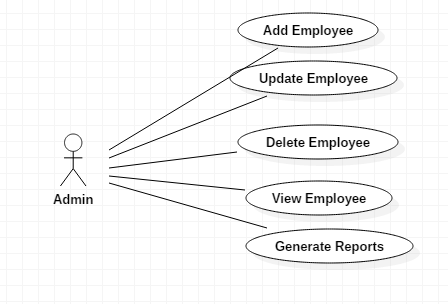


*Use case: <Add Client>*

*Primary actor: <Employee>*

*Main success scenario: <Angajatul introduce datele, apasa pe butonul de Add si clientul se salveaza in baza de date. >*

*Extensions: <Daca clientul nu are date valide se va afisa eroare >*



*Use case: <Add Employee>*

*Primary actor: <Admin>*

*Main success scenario: <Administratorul introduce datele, apasa pe butonul de Add si angajatul se salveaza in baza de date. >*

*Extensions: <Daca angajatul nu are date valide se va afisa eroare >*

3. System Architectural Design

**3.1 Architectural Pattern Description**

*Design patternul Layers:*

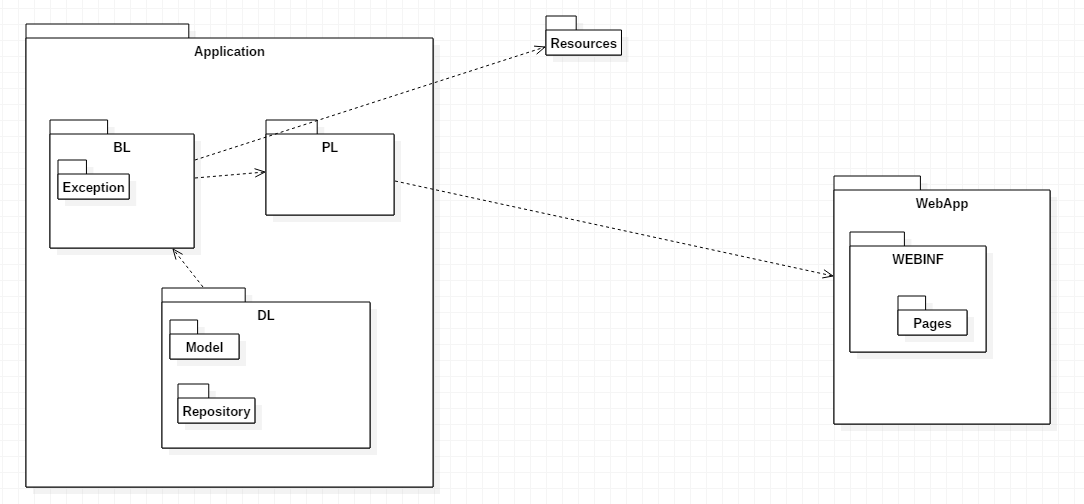
*Acest design pattern reprezinta o arhitectura client-server in care prezentarea (interfata), procesarea si managementul datelor sunt fizic separate. Cea mai utilizata este cel cu trei layer-e: Bussiness Layer, Processing Layer si Data Layer.*

*Design patternul MVC: este utilizat pentru implementarea interfetelor utilizator. Divide o aplicatie in trei parti interconectate cu scopul de a separa reprezentarea datelor interne de modul in care acestea sunt prezentate utilizatorului.*

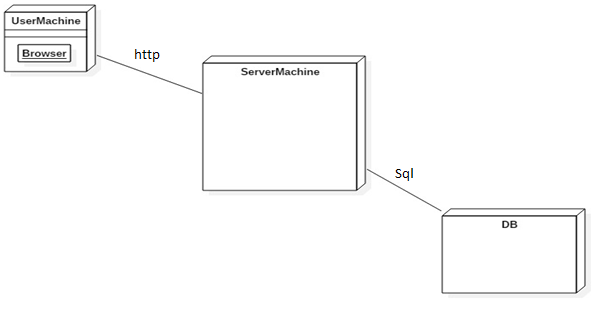
**3.2 Diagrams**

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

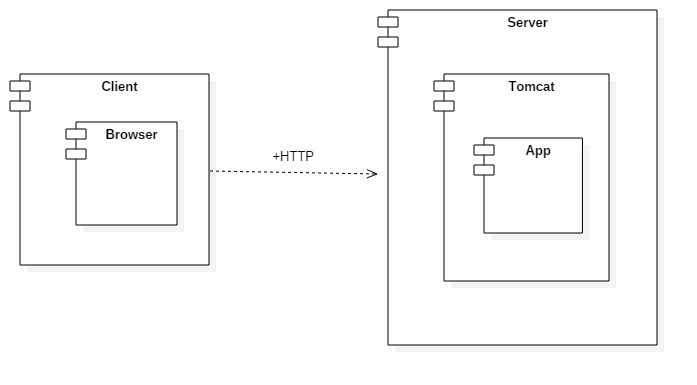
*Diagrama de pachete:*

**

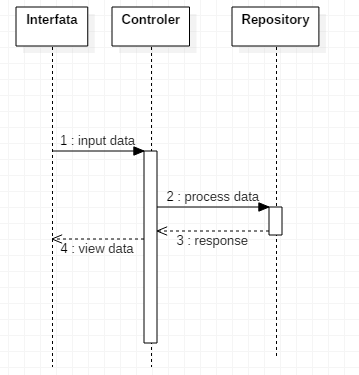
*Deployment diagram:*



*Component diagram:*



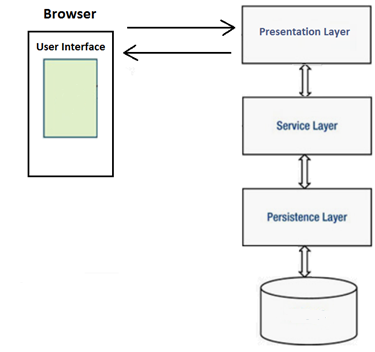
4. UML Sequence Diagrams

**

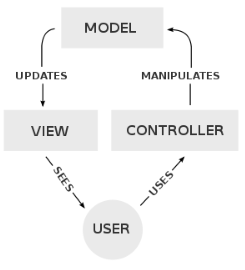
5. Class Design

**5.1 Design Patterns Description**

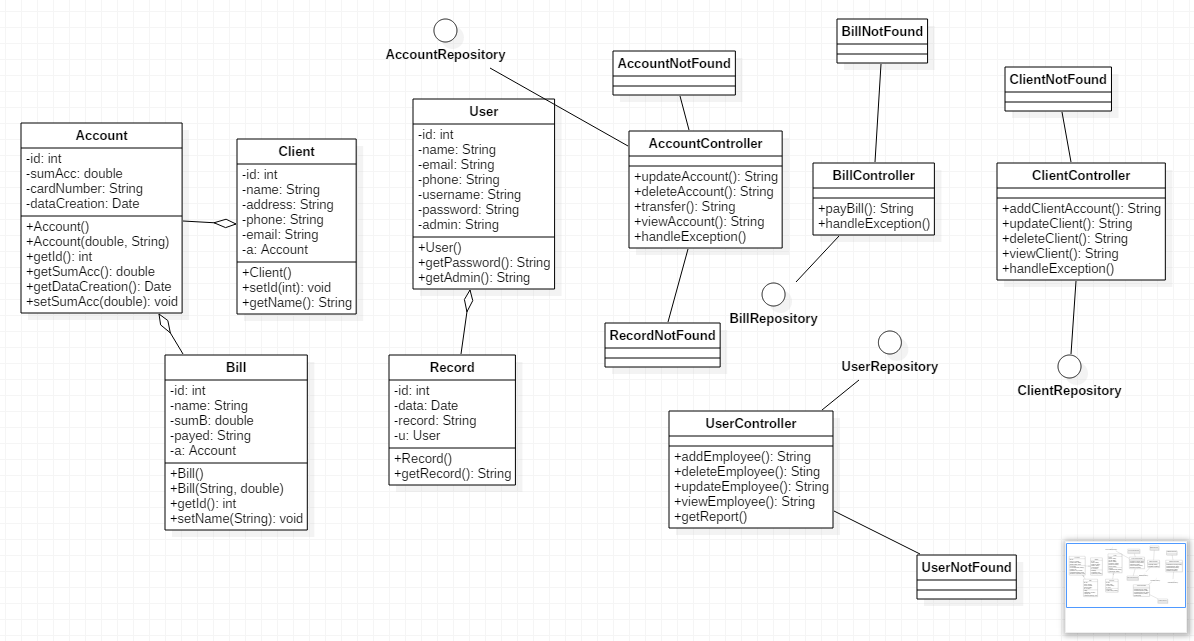
*Layers:*

**

MVC:

**

**5.2 UML Class Diagram**



6. Data Model

*Din specificatia problemei rezulta clasele importante ale aplicatiei: Record, Account, Client, User,Bill. Un user poate fi angajat sau admin. Un client are un cont si o factura. Clasa Recor pastreaza operatiile facute de fiecare user, pentru a putea fi interrogate de catre admin mai tarziu.*

7. System Testing

Aplicatia a fost testat introducand date din interfata web si verificand baza de date in paralel. Conexiunea cu baza de date a fost testat in prima faza a aplicatiei. In ceea ce priveste testarea metodelor prin unitTest, era necesara de o baza de date suplimentara, cea ce interfera cu baza de date actuala.

8. Bibliography

1. Curs M.Dansoreanu

2. Patterns of enterprise application architecture – Martin Fowler

3. Surse web

<http://docs.oracle.com/javase/tutorial/uiswing/>

<http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>

<https://msdn.microsoft.com/en-us/library/54xbah2z(v=vs.110).aspx>

<https://msdn.microsoft.com/en-us/library/e80y5yhx(v=vs.110).aspx>