Assignment A1
Analysis and Design Document

Student: Bolba Raluca Maria
Group: 30235

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

Să se proiecteze și implementeze o aplicație pentru angajații unei bănci. Aplicația trebuie să aibe două tipuri de utilizatori: un utilizator obișnuit reprezentat de angajatul băncii și un utilizator administrator care trebuie să furnizeze un username și o parolă pentru a putea folosi aplicația. Datele trebuie păstrate într-o bază de date. Să se folosească pattern-ul arhitectural Layers pentru organizarea aplicației. Să se utilizeze un pattern *domain logic* (*transaction script* sau *domain model)* / un pattern *data source hybrid* (*table module* sau *active record)* și un pattern *data source pure* (*table data gateway, row data gateway* sau *data mapper)* cele mai potrivite pentru aplicație.

1.2 Functional Requirements

Utilizatorul obișnuit (angajatul) trebuie să poată efectua următoarele operații:

- Să adauge/actualizeze/vizualizeze informații despre clienți (informații client : nume, cod numeric personal, adresă etc.)
- Să efectueze operațiile CRUD pe conturile clienților (informații cont : identificator, tip, balanță cont, data creării)
- Să transfere bani între conturi
- Să efectueze plăți de facturi

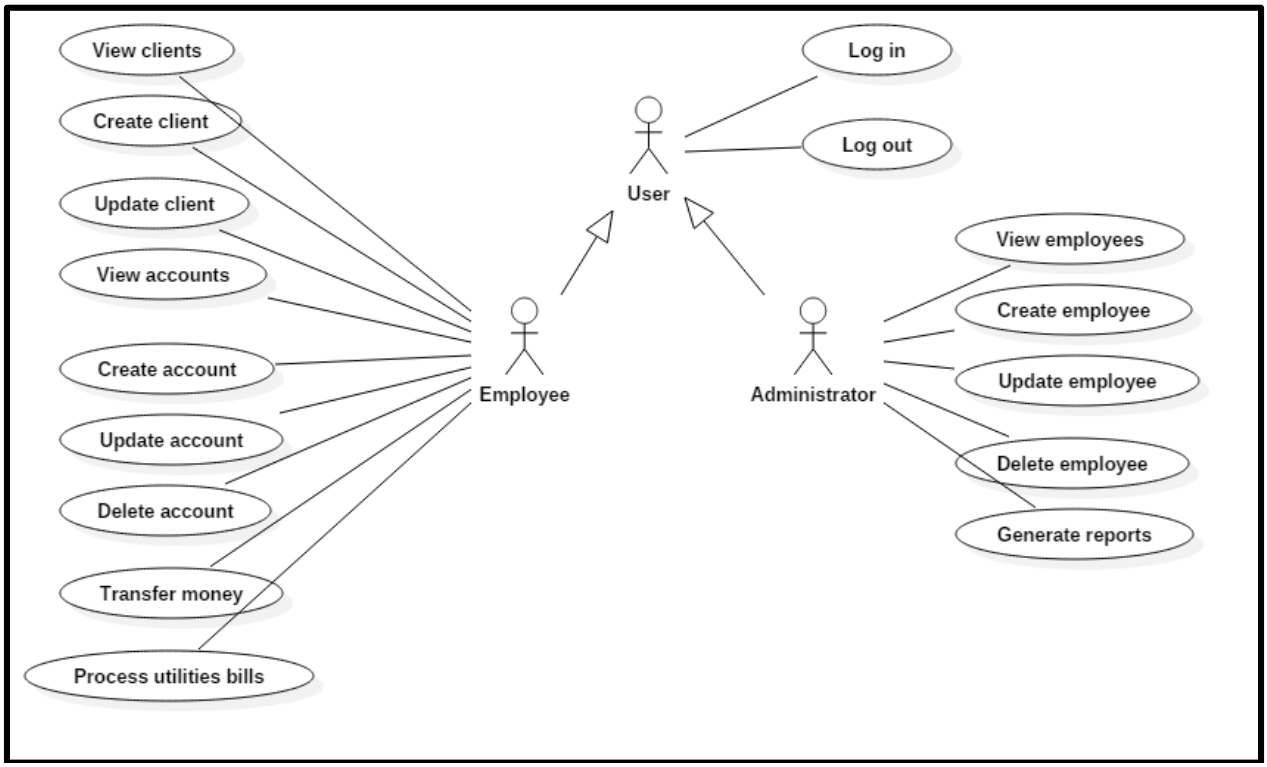
Administratorul trebuie să poată efectua următoarele operații:

- Să efectueze operațiile CRUD pe informațiile angajaților
- Să genereze rapoarte pentru o perioadă particulară care conține activitățile efectuate de un anumit anagajat

1.3 Non-functional Requirements

O parte din cerinţele funcţionale pe care ar trebui să le îndeplinească aplicaţia sunt : performanţa şi securitatea. Una din modalitățile de măsurare a performanței este timpul de răspuns, care cu cât este mai mic, cu atât aplicația poate fi considerată mai performantă. Timpul de răspuns poate fi măsurat pentru operațiile simple pe care le poate face utilizatorul aplicației, de exemplu: autentificarea la sistem, adăugarea unui client, generarea unui raport. În cazul acestei aplicații, toate operațiile au un timp de răspuns de cel mult 1 secundă. În cazul securității, această cerință este satisfăcută de aplicația proiectată prin faptul că nu permite utilizatorilor de niciun fel (angajat sau administrator) să poată obține informații despre parolele celorlalți utilizatori.

2. Use-Case Model

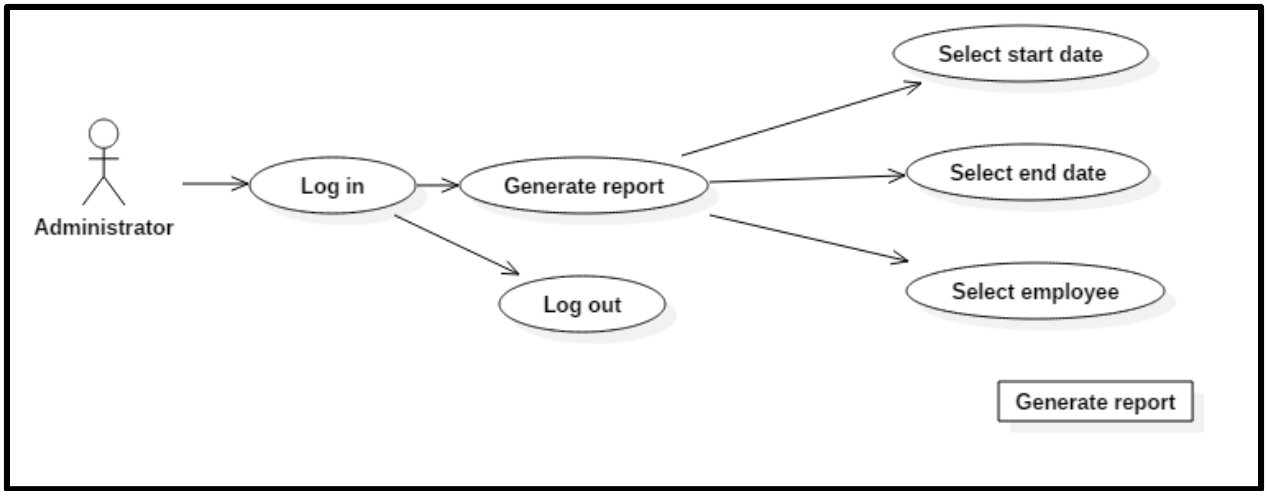


Use case: **Generare raport**
Level: User-goal
Primary actor: Administrator
Main success scenario:

1. Utilizatorul se autentifică la sistem
2. Administratorul alege să efectueze operația de generare raport
3. Administratorul alege perioada pentru raport; acesta alege data de start și data de sfârșit
4. Administratorul alege angajatul pentru care să se genereze raportul
5. Administratorul vizualizează raportul
6. Administratorul se deconectează

Extensions:

1a. Datele de autentificare sunt incorecte
    1a1.Se revine la pasul 1
3a. Data de sfârșit este mai mică decât cea de început
    3a1. Se revine la pasul 3
4a. Nu există niciun angajat
    4a1. Scenariul se termină

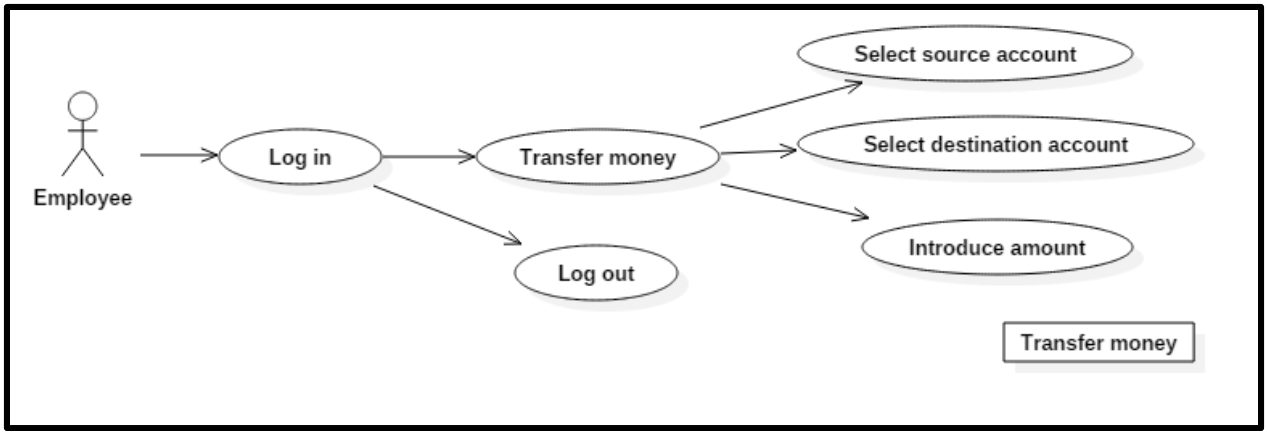


Use case: **Transfer bani**
Level: User-goal
Primary actor: Angajat
Main success scenario:

1. Angajatul se autentifică la sistem
2. Angajatul alege operația de transfer de bani
3. Angajatul selectează un cont sursă
4. Angajatul selectează un cont destinație
5. Angajatul introduce suma de bani de transferat
6. Angajatul transferă suma de bani
7. Angajatul se deconectează

Extensions:

1a. Datele de autentificare sunt incorecte
    1a1.Se revine la pasul 1
3a. Nu există conturi
    3a1. Scenariul se termină
4a. Contul destinație este același cu contul sursă
    4a1. Se revine la pasul 4
5a. Suma de bani introdusă este o valoare negativă
    5a1. Se revine la pasul 5
5b. Suma de bani introdusă nu este o valoare numerică
    5b1. Se revine la pasul 5
5c. Suma de bani introdusă este mai mare decât balanța contului sursă
    5c1. Se revine la pasul 5



3. System Architectural Design

3.1 Architectural Pattern Description

Pattern-ul architectural folosit pentru acest sistem este pattern-ul Layers. Acest pattern modelează arhitectura pe baza unor nivele astfel încât un nivel să se servească de funcționalitatea nivelelor inferioare. Nivele definesc interfețe, care ar trebui să rămână stabile și să încapsuleze responsabilități similare. De asemenea, utilizarea acestui pattern ar trebui să permită schimbarea anumitor componente, atâta timp cât acestea respect interfețele definite de nivele. În cazul acestui sistem, arhitectura este descompusă în trei nivele : *Presentation, Business și Data access.*

Nivelul de *Presentation Logic* este cel mai superior nivel și este cel responsabil de interfața grafică dintre utilizator și aplicație. Acesta preia date de la nivelul inferior, aici cel de *Business Logic* și le transmite la utilizator pentru vizualizare. Acesta pune la dispoziție utilizatorului o modalitate interactivă de a efectua operațiile pe aceste date și transmite mai departe modificările realizate de utilizator. În cazul acestei aplicații, nivelul de prezentare constă într-o singură clasă, *Window*, care reprezintă interfața grafică cu utilizatorul.

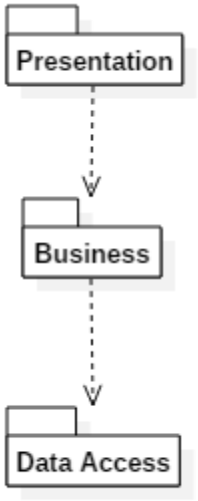
Nivelul de *Business Logic* este responsabil cu logica de business, în cazul acestei aplicații în special cu validarea datelor primite de la nivelul superior de prezentare și utilizarea lor pentru operații CRUD. De asemenea, acest nivel se folosește de nivelul inferior de *Data Access Logic* pentru a prelua date din baza de date și a face corelații între acestea, dar și pentru a introduce sau modifica anumite date. În cazul acesta, nivelul este compus din patru clase : *Account, Client, User și Activity*.

Nivelul de *Data Access Logic* este cel mai inferior nivel, și anume cel responsabil de accesul la baza de date. Acest layer nu folosește niciun alt nivel. Clase ce fac parte din acest nivel sunt : *AccountGateway, ClientGateway, UserGateway* și *ActivityGateway* și au rolul de a efectua operații precum citire, inserare, modificare sau ștergere de date. Datele pe care se fac operațiile sunt primite de la nivelul superior de business.

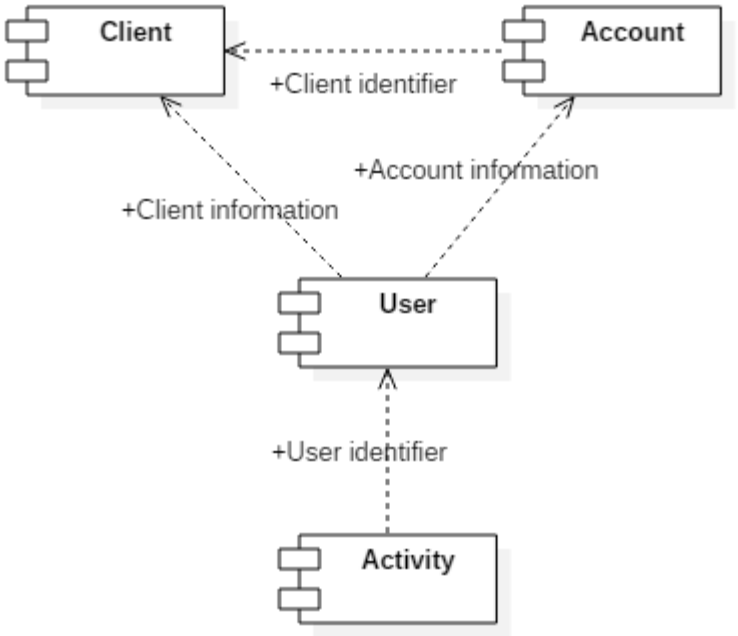
3.2 Diagrams

Package diagram

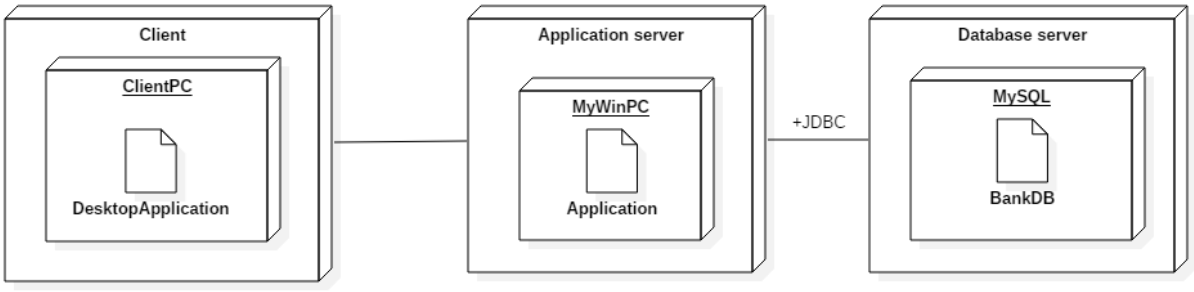
Pachetele corespund nivelelor descrise anterior pentru implementarea pattern-ului arhitectural Layers.



Component diagram

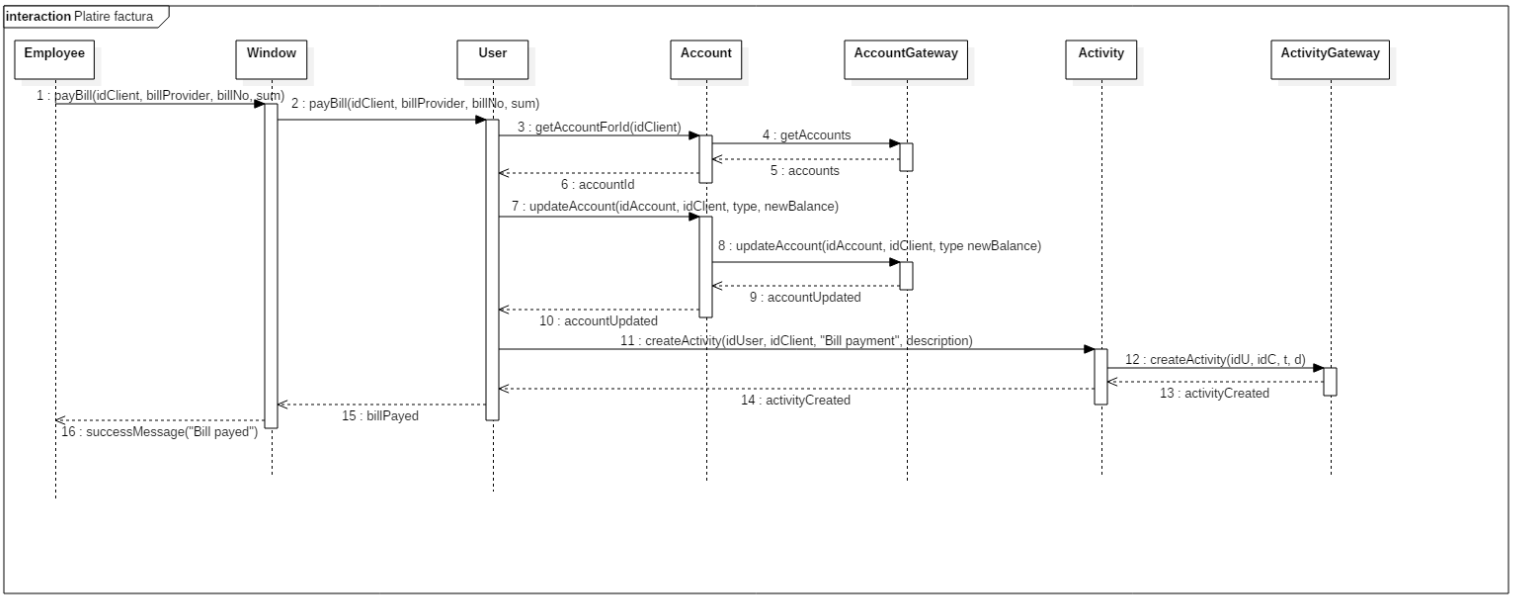


Deployment diagram



4. UML Sequence Diagrams

Diagrama de secvență pentru operația de *Plătire facture*:



5. Class Design

5.1 Design Patterns Description

Design pattern-urile folosite în cadrul acestei aplicații sunt: *Table Module* și *Table Data Gateway.*

Table Module este un design pattern care organizează logica domeniului cu o clasă pentru un tabel din baza de date, iar o singură instanță a unei clase va conține proceduri care vor opera pe datele tabelului respectiv. De obicei *Table Module* funcționează cu o structură de date tabulară similară cu cea folosită în baza de date. Pentru acest aspect am folosit clasa *CachedRowSetImpl* care păstrează datele returnate în urma unor interogări asupra bazei de date, însă acestea sunt păstrate în memorie, pentru a putea fi transmise către alte clase.

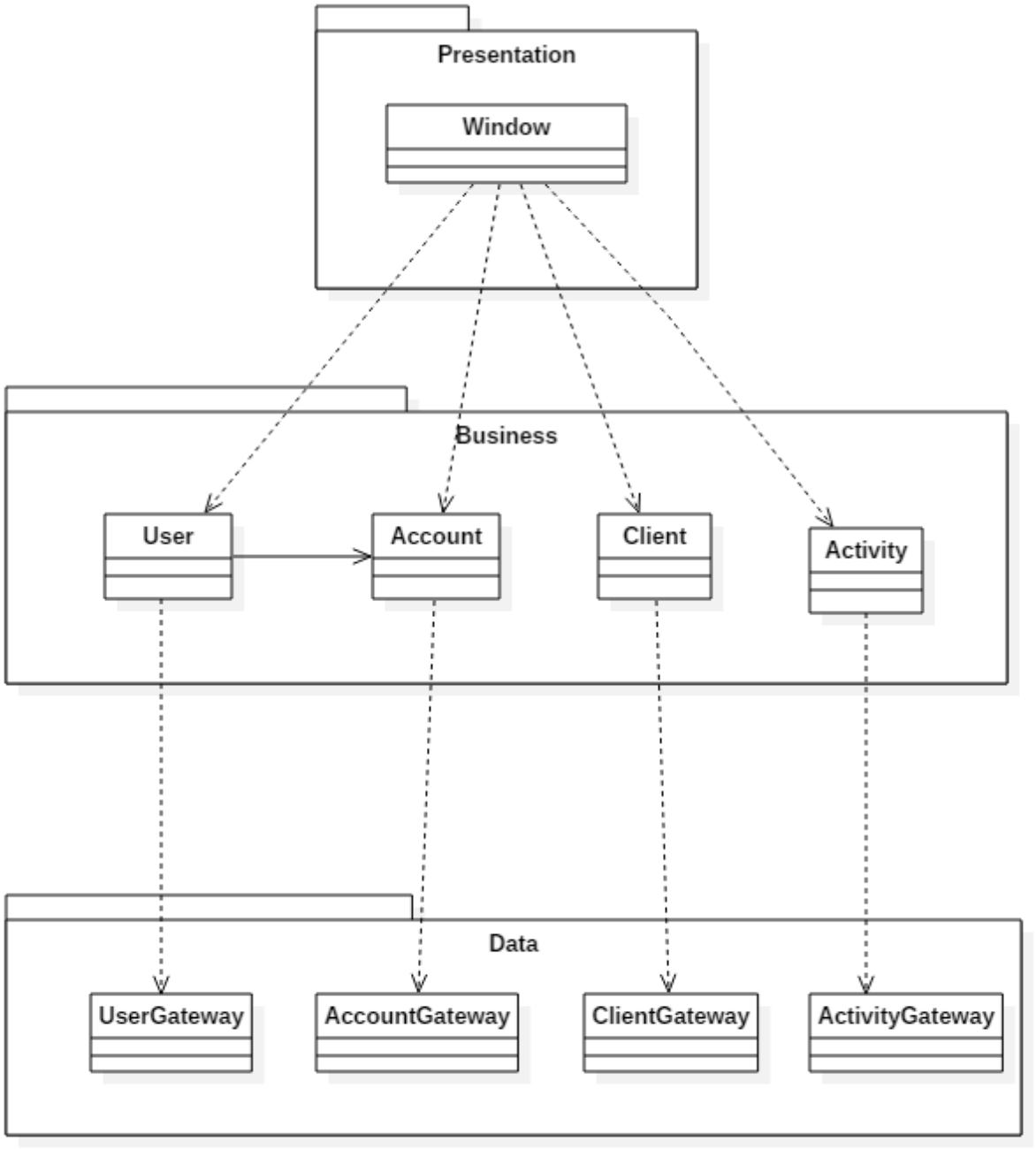
Table Data Gateway este un design pattern care conține toate interogările SQL pentru accesarea unui singur tabel din baza de date. O singură instanță controlează toate rândurile unui tabel. Metodele uzuale întânite în cadrul unei astfel de clase sunt de creare, citire, modificare sau ștergere din baza de date, dar și metode de căutare după anumite câmpuri din tabel. La fel ca *Table Module*, datele returnate din baza de date trebuie păstrate într-o structură tabulară. Și în acest caz am folosit *CachedRowSetImpl*.

5.2 UML Class Diagram

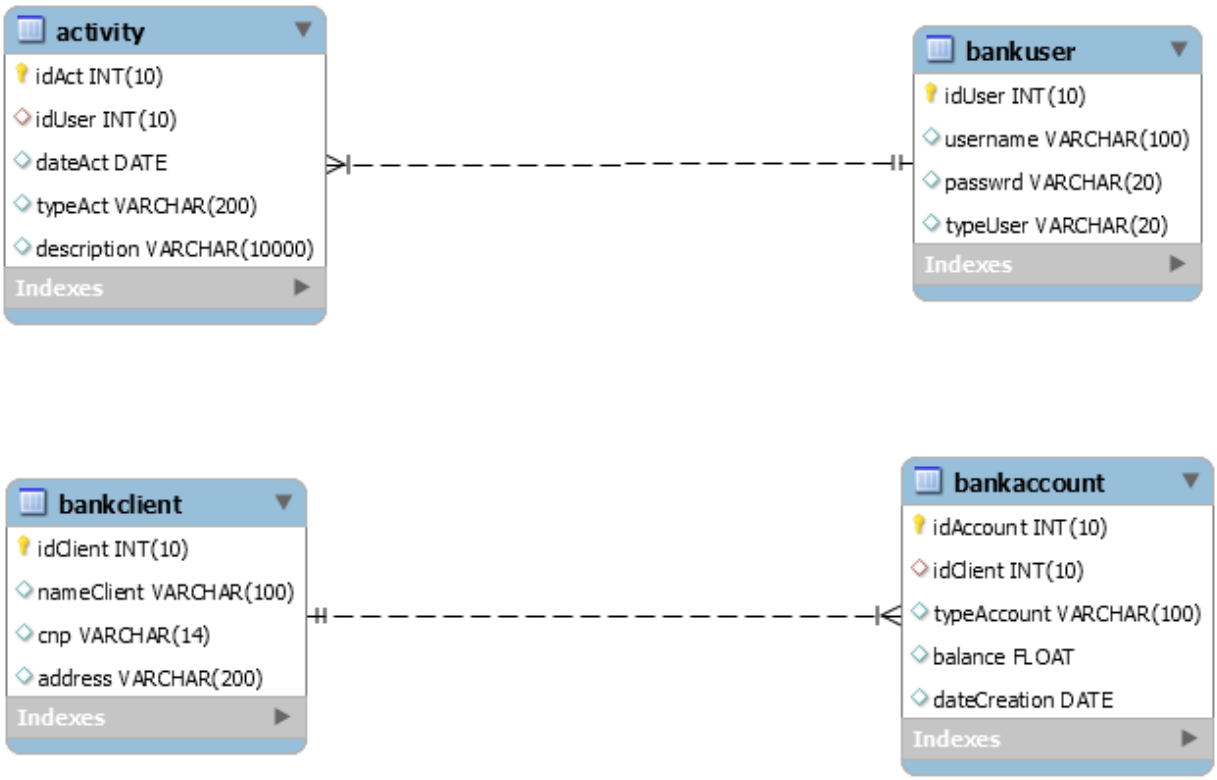
Clasele pentru care am aplicat pattern-ul *Table Module* sunt: *User, Account, Activity, Client* iar clasele pentru care am aplicat pattern-ul *Table Data Gateway* sunt: *UserGateway, AccountGateway, ActivityGateway, ClientGateway.*

Clasele pentru care am folosit *Table Module* controlează atât comportamentul tabelelor pe care le reprezintă, cât și logica de business asociată acestora. Din acest motiv, clasele se situează undeva între layer-ul de business și cel de data acces. Așadar am proiectat clasele ca făcând parte din layer-ul de business, unde acestea se ocupa de logica de business precum validarea datelor primite de la layer-ul de presentation, iar partea responsabilă de controlare a comportamentului se face în mod indirect prin instanțe la clasele *Gateway*. Din acest motiv o clasă de tip *Table Module* precum *User* are o instanță la clasa *UserGateway.* Pentru că o clasă controlează toate rândurile unui tabel corespondent din baza de date, majoritatea metodele sunt declarate statice.

Clasele de tip *Gateway* sunt responsabile de accesare a unui singur tabel din baza de date pentru preluarea sau adăugarea unor informații. Informațiile returnate din baza de date sunt transmise în formă tabulară către clasele de *Table Module*. Acestea nu conțin informații despre obiecte, ci doar metode de procesare a datelor din tabelul corespunzător din baza de date.


Presentation
Window
Business
User
Account
Client
Activity
Data
UserGateway
AccountGateway
ClientGateway
ActivityGateway

6. Data Model



Baza de date folosită pentru aplicație constă în patru tabele, respectiv: *Activity, BankUser, BankClient* și *BankAccount.*

Tabela *BankUser* conține date referitoare la utilizatorii aplicației și anume un identificator, date de autentificare (username și parolă) și tipul acestuia. Întrucât utilizatorul poate fi angajat sau administrator, am decis să păstrez această informație în câmpul *typeUser* în detrimentrul creării a două tabele, câte una pentru fiecare utilizator, întrucât datele ce trebuie păstrate pentru aceștia nu diferă.

Tabela *Activity* conține informații despre activitățile realizate de către un utilizator, motiv pentru care aceasta conține o cheie străină către tabela *BankUser*. Alte informații stocate pentru o activitate sunt: identificatorul activității, data la care a apărut activitatea, tipul activității (*„Create client", „Update client", „Bill payment" etc.)* şi o descriere mai în detaliu a acesteia.

Tabela *BankClient* conţine informaţii referitoare la clienţii băncii : identificator, numele clientului, codul numeric personal şi adresa acestuia.

Tabela *BankAccount* conţine date referitoare la conturile deţinute de clienţii băncii, aşadar are o cheie străină către aceasta tabelă de clienţi, un identificator, tipul acestuia (*Saving account* sau *Spending account),* suma existentă în cont şi data la care a fost creat.

7. System Testing

Pentru testarea aplicației aceasta s-a rulat și s-a verificat dacă operațiile sunt realizate în mod corect. De exemplu pentru operația de autentificare s-a verificat dacă datele introduse sunt cele prezente și în baza de date, iar în caz contrar s-a afișat un mesaj de eroare.

8. Bibliography

- http://www.informit.com/articles/article.aspx?p=1398617&seqNum=3
- http://www.informit.com/articles/article.aspx?p=1398617&seqNum=3
- https://docs.oracle.com/javase/7/docs/api/javax/sql/rowset/CachedRowSet.html