Managementul unei agentii de turism

Student: Bratean Larisa

**Group: 30235**

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 4

2.1 Use case Agent 4

2.2 Use case Admin 5

3. System Architectural Design 5

3.1 Architectural Pattern Description 5

3.2 Diagrams 6

4. Class Design 6

4.1 Design Patterns Description 6

4.2 UML Class Diagram 7

5. Data Model 7

6. System Testing 8

7. Bibliography 12

1. Requirements Analysis

# Specificari tema

Avem o agentie de turism, care are doua tipuri de user si anume, unul obisnuit care este un agent de turism si un admin care se ocupa de intreaga agentie. Ei trebuie sa se logheze cand deschid aplicatia, printr-un nume de user si o parola. Ambii utilizatori pot sa faca anumite operatii, agentii pe baza clientilor si a rezervarilor lor, iar administratorul pe baza datelor clientilor.

# Functional Requirements

Agentul de turism poate sa faca urmatoarele operatii:

* Sa adauge sau sa vada informatiile clientilor de care se ocupa ( nume, numar de identitate, pasaport, adresa);
* Sa adauge sau sa vada rezervarile pentru un client ( destinatia, numele hotelului, numarul de persoane care merg, pretul final, si data pana cand trebuie sa plateasca);
* Sa accepte plata partiala de la un client inainte de data finala cand trebuie sa plateasca;
* Sa vada toti clientii care nu au platit pana la data finala.

Admin-ul poate sa faca urmatoarele operatii:

* Sa faca operatii de adaugare, sterge, modificare sau vizualizare agenti;
* Sa genereze un report pe o anumita perioada a activitatilor unui agent.

# Non-functional Requirements

* + 1. **Disponibilitate**

Sistemul e disponibil 100% pentru utilizatorii aplicatiei, 24 ore pe zi, cat timp aplicatia Java ruleaza.

* + 1. **Performanta**

Aplicatia este dezvoltata sa functioneaze foarte rapid, sa raspunda repede cand agentul sau administratorul incearca sa faca o operatie, indiferent daca este de adaugare, vizualizare, stergere etc..

* + 1. **Securitate**

Aplicatia este sigura, deoarece si agentii cat si administratori trebuie sa se logheze inainte sa foloseasca aplicatia, cu un nume de utilizator si o parola, care este criptata cu caracterul special \*.

* + 1. **Usability**

Aceasta sectiune contine toate cererile pentru a nu afecta aplicatia.

Sistemul permite utilizatorilor sa acceze aplicatia folosing Java Application, deci este o aplicatie Desktop. Nu e necesar training, deoarece rulam aplicatia si se deschide interfata, unde e necesara logarea. Sistemul e prietenos si simplu.

2. Use-Case Model

2.1 **Agent**

Agentul are scopul sa se logheze cu success in aplicatie, si sa faca operatii pe datele clientelor, cat si pe a rezarvarilor. El poate sa adauge cu success rezervari noi.

Are un rol important pentru gestionarea rezervarilor in agentia de turism, fara el, administratorul ar trebui sa se ocupa atat de administrarea paginii, cat si de realizarea tuturor comenzilor.

Un scenariu de success ar fi: agentul incearca sa se logheze in aplicatie, si reuseste. Doreste sa realizeze o noua rezervare pentru un client. Introduce datele necesare cererii clientului, si se realizeaza rezervarea. Clientul trebuie sa plateasca pana la data introdusa ca ultima zi de plata, si de asemenea poate sa plateasca partial.

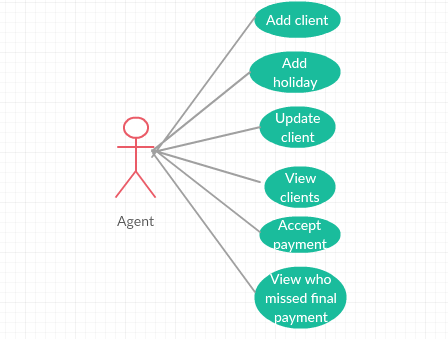
Un scenariu de succes-esec ar fi :

* Preconditii: agentul incearca sa se logheze pentru a vedea cine nu a platit pana la data finala.
* Postconditii:

Succes: agentul se logheaza cu success, si vede cine a ratat ultima de zi plata.

Esec: agentul are eroare la logare, a gresit username-ul sau parola.

Use case:



2.2 **Admin**

Administratorul are scopul sa se logheze cu success in aplicatie, si sa faca diferite operatii pe datele agentilor, si sa vada toate activitatile lor intr-o anumita perioada.

Are un rol important pentru gestionarea agentilor, cat si a activitatilor lor.

Un scenariu de success ar fi: administratorul se logheaza in aplicatie, si doreste sa stearga un agent, care nu mai lucreaza in agentie. Introduce id-ul angajatului, si datele lui se sterg din baza de date.

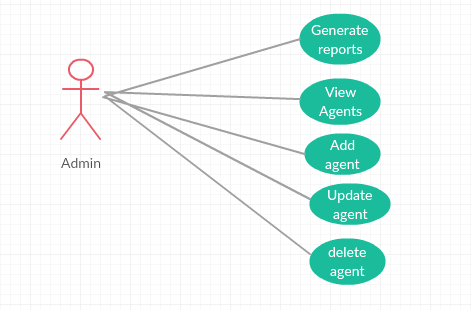
Un scenariu de success-esec ar fi:

* Preconditii: administratorul incearca sa se logheze pentru a vedea lista activitatilor unui anumit agent.
* Preconditii:

Succes: administratorul se logheaza cu success, si vede lista cu toate activitatile.

Esec: administratorul are eroare la logare, la gresirea username-ului sau a parolei.

Use case:



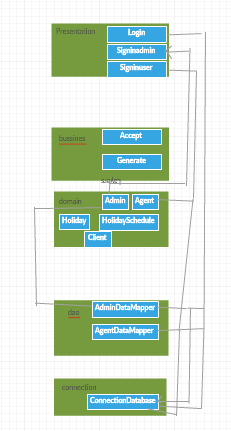
3. System Architectural Design

**3.1 Architectural Pattern Description**

Am folosit design-ul architectural Layers. Este o grupa de clasa, in cazul meu grupata pe pachete, care are dependete de module cu alte module. Este un grup de componente reutilizabile in circumstante similar. E impartit intr-o ierarhie de trei.Fiecare layer are un rol specific.

In cazul meu, in acest proiect, am avut cinci layer-e, si anume: bussines, connection, dao, domain si presentation. Layer-ul presentation este responsabil pentru gestionarea interfetei, crearea ei. Nu e responsabil pentru cum sa preia date din orice clasa model, doar afiseaza informatia in interfata. Bussines layer e responsabil pentru a executa anumite reguli associate unei reguli. Connection layer (Database layer) e folosit pentru a face conexiunea cu baza de date. In layer-ul dao, facem toate operatiile care au legatura directa cu baza de date, cum ar fi insert, delete, afisare etc. Iar in Domain, avem create clasele model pentru fiecare tabela din baza de date, a caror atribute sunt aceleasi cu coloanele din tabelele respective.

**3.2 Diagrams**

**

5. Class Design

**5.1 Design Patterns Description**

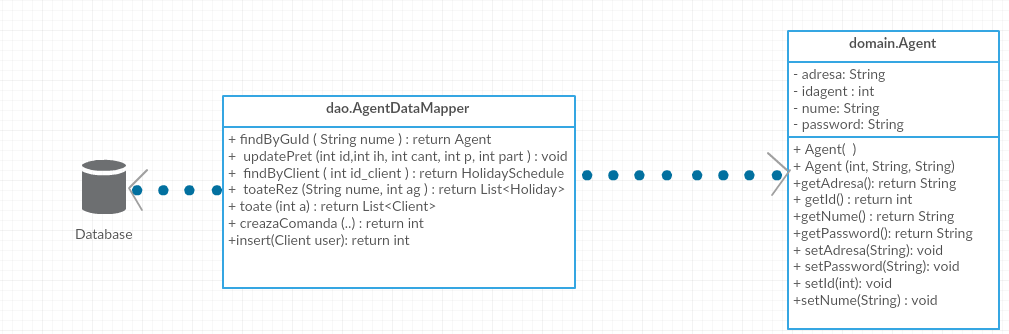
Design-urile arhitecturale pe care le-am folosit sunt Data Mapper, care oarecum include si Active Data Record cat si Domain Model.

Acest tip de design separate obiectele de baza de date. Transfera date intre cele doua, si in acelasi timp le izoleaza una de alta. Obiectele nici nu trebuie sa stie daca este o baza de date prezenta, nu au nevoie de cod SQL, sau sa stie despre schema din baza de date.

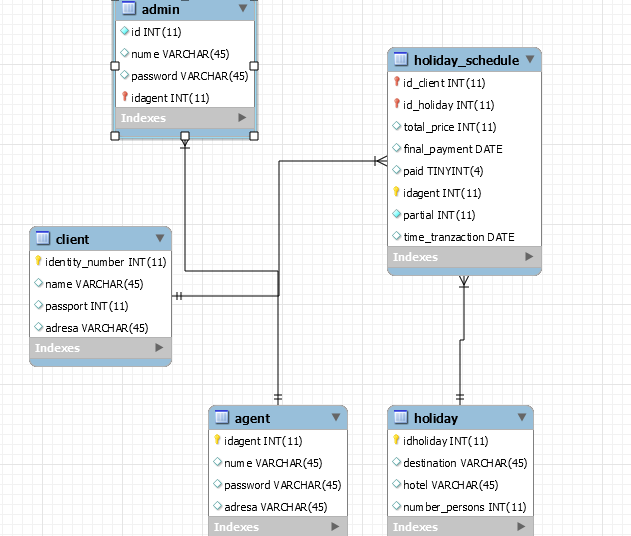
Acest design pattern imparte Active Record in doua clase, una model ( Domain Model) si una mapper. Cea mapper are operatiile pentru citire din baza de date si pentru a crea instante pentru model.

**5.2 UML Class Diagram**

Am ilustrat pentru o anumita clasa cum e folosit Design Pattern-ul Data Mapper pentru o clasa Agent:



6. Data Model

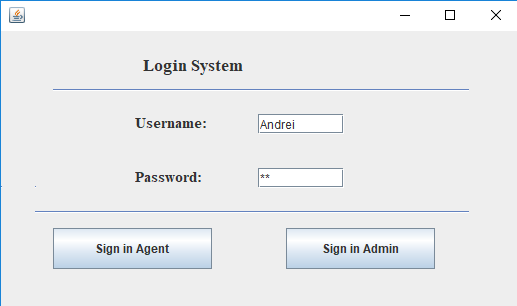
**

7. System Testing

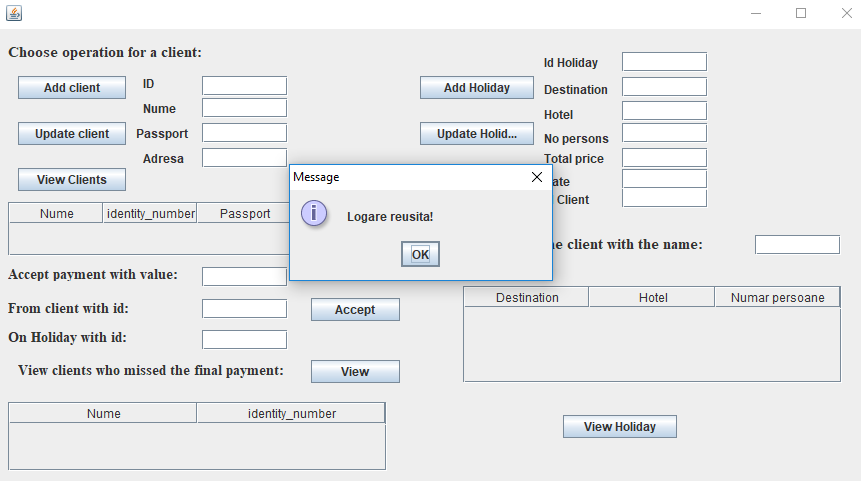
Am ilustrat cateva scenarii pentru testare:

**7.1 Login Agent**

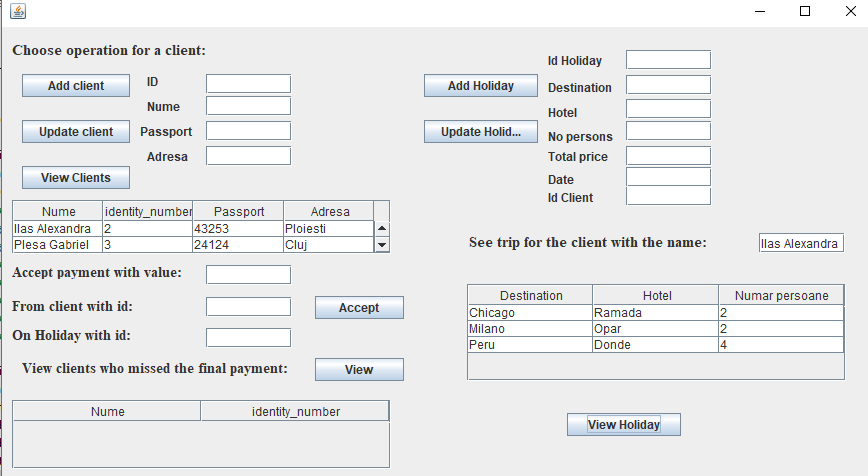
Introducere date:



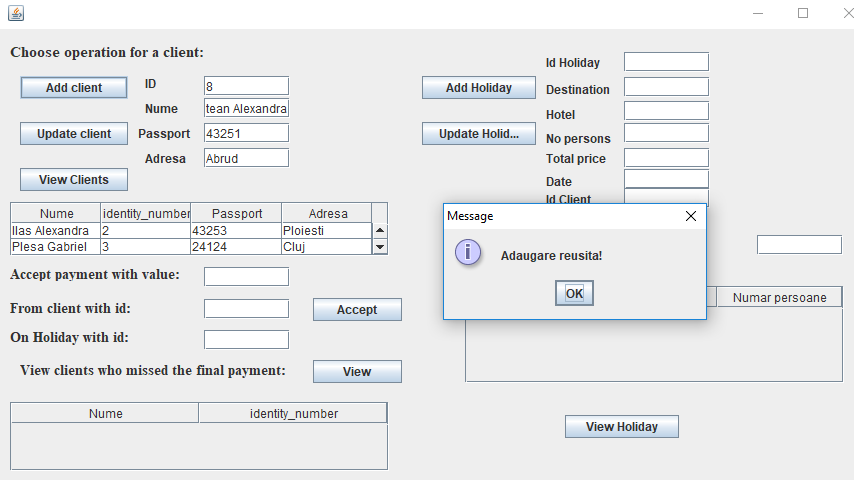
Afisare interfata agent dupa logare reusita:

****

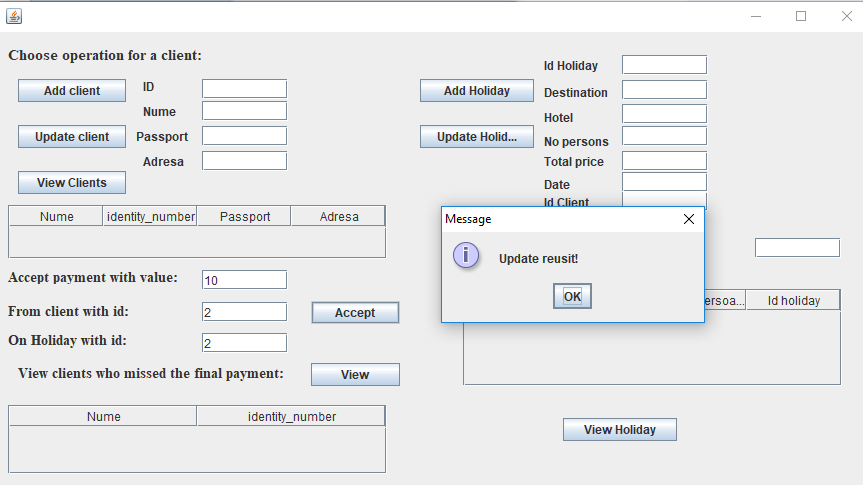
**7.2 Afisare clienti ai agentului cat si afisarea rezervarilor pentru un client cu un anumit nume:**

****

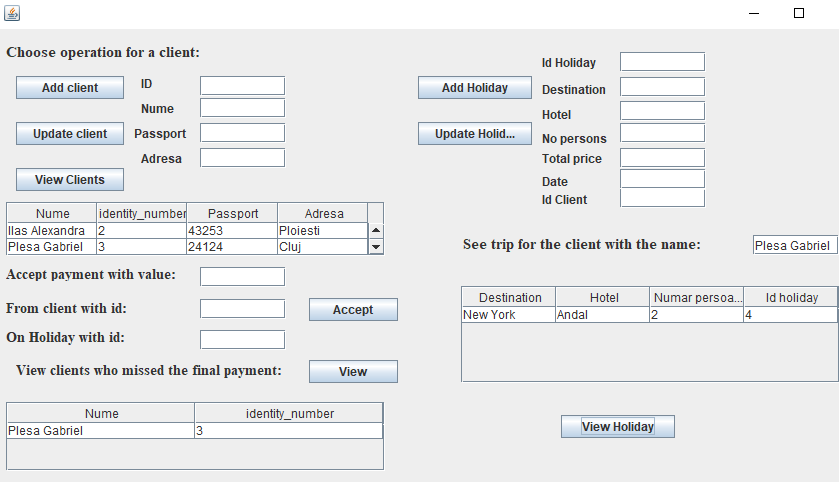
**7.3 Introducere client:**



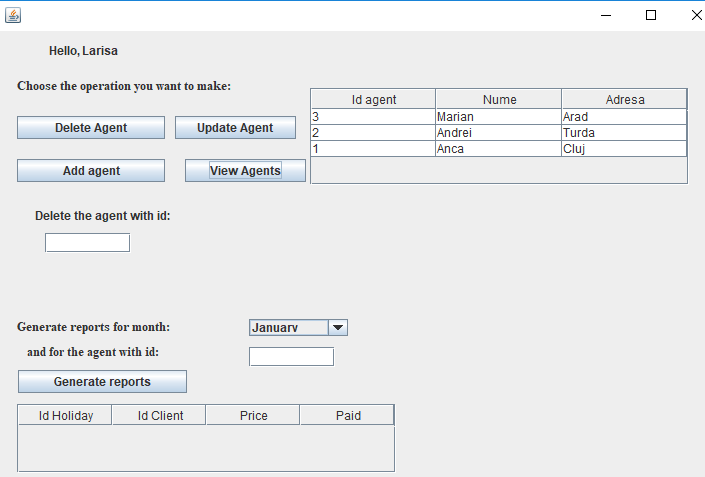
**7.4 Update pret la un client la o anumita rezervare:**

****

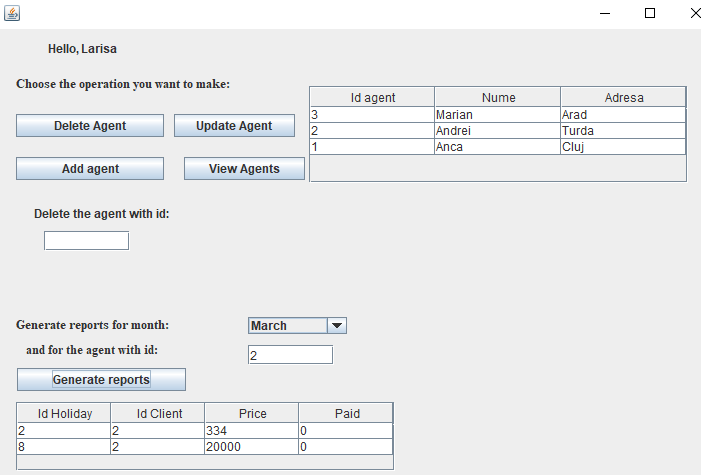
**7.5 Vizualizare clienti care nu au platit la timp:**

****

**7.6 Interfata Admin dupa logare si agenti:**

****

**7.9 Generare raportului pe o anumita luna pe care o selectam, pentru un anumit agent:**

****

8. Bibliography

<https://martinfowler.com/eaaCatalog/dataMapper.html>

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

<http://richard.jp.leguen.ca/tutoring/soen343-f2010/tutorials/implementing-data-mapper/>

Google

Youtube

MySQL zoo