Rent a car

Analysis and Design Document

Student: Melniciuc Ioan-Alexandru

**Group: 30238**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 07-04-2017 | <1.0> | In acest document se vor descrie detalii structural ale aplicatie, cum ar fii arhitectura folosita sau design pattern-urile abordate in organizarea structurii si functionalitatii aplicatiei | Melniciuc Ioan Alexandru |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

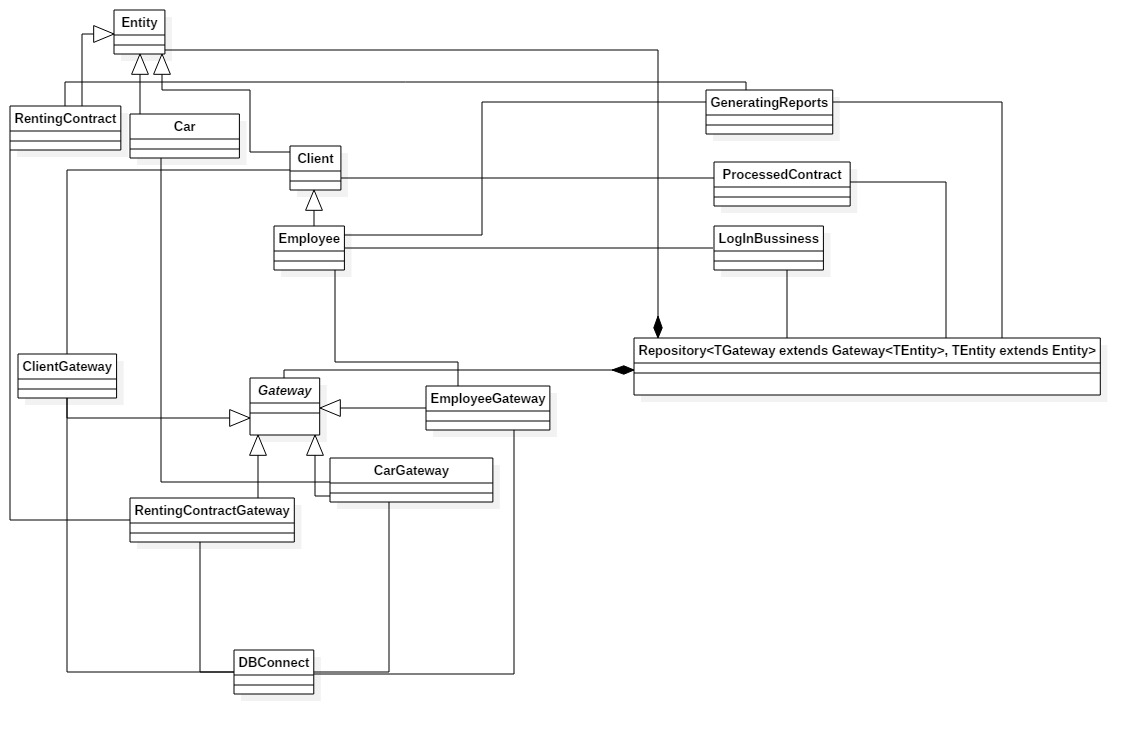
# Project Specification

Aplicatia “Rent a car” este o aplicatie care permite utilizatorilor ei sa vizualizeze masinile disponibile pentru inchiriat ale unei companii ce presteaza astfel de servicii, sa reserve o anumita masina pentru o durata stabilita, sa plateasca pentru serviciul de inchiriere si sa primeasca facture pentru plata depusa. Pe de alta parte, angajatii companiei de inchirieri auto pot sa gestioneze contractele de inchiriere si situatia masinilor din flota pentru inchiriere. Aceste operatiuni vor fi stocate intr-o baza de date si vor fi accesate prin operatiile specifice bazelor de date, CRUD.

# Elaboration – Iteration 1.1

# Domain Model

Datele vehiculate de logica aplicatiei vor fi reprezentate prin entitati reale carora li se asociaza cate o tabela( daca e vorba de baza de date), sau cate o clasa(cand vine vorba de reprezentarea in cod Java). Spre exemplu, in baza de date vor exista tabelele *Client*, *Car*, *RentingContract*, *Employee*, iar in reprezentarea in cod Java vor exista de asemenea clasele *Client*, *Car*, *RentingContract*, *Employee*. Fiecare clasa va contine attribute de tip *String* pentru reprezentarea numelor(clientilor, angajatilor, marca si modelul masinii, clauzele contractuale), de tip *BigInteger*( pentru reprezentarea cnp-ului clientilor sau al angajatilor, a seriei de sasiu a masinii, a codului contractului), de tip *Date* (pentru reprezentrarea datelor de incepere si incheiere a contractelor, a datei pentru care se rezerva o anumita masina, etc), de tip *Boolean* (pentru a semnala daca o masina este sau nu disponibila pentru inchiriere).

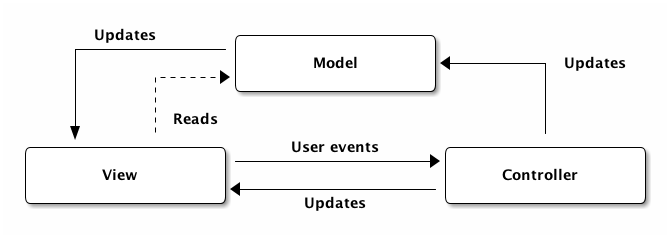
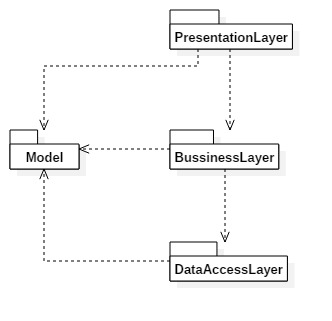


# Architectural Design

## Conceptual Architecture

Implementarea aplicatiei se va face pe baza unei “layered architecture” si pe baza “MVC” datorita necesitatii de a separa logica de extragere a datelor din baza de date pe structurile suport oferite de clasele din pachetul *Model* (*Client, Car, RentingContract, Employee…*). Un alt sablon arhitectural utilizat pentru extragerea datelor din baza de date este “Table Data Gateway”. Pentru realizarea interfetei cu utilizatorul se aborda structura *MVC*.

## Package Design



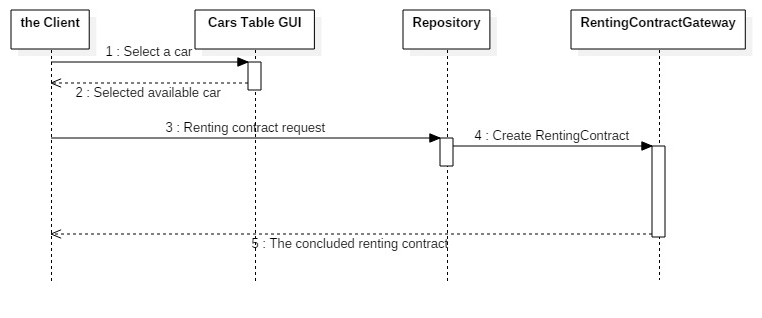
## Component and Deployment Diagrams

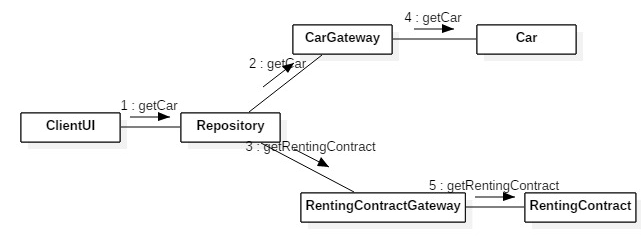
# 

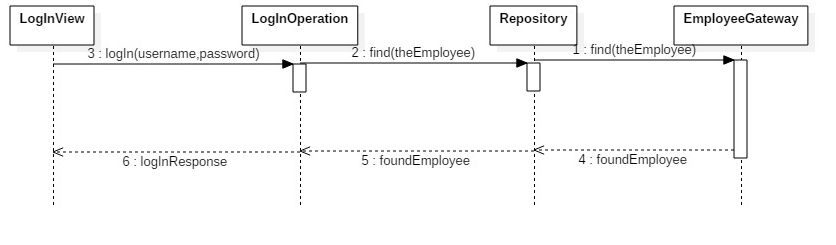
# Elaboration – Iteration 1.2

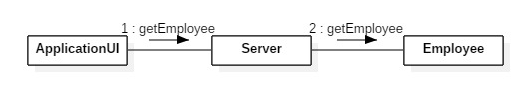
# Design Model

## Dynamic Behavior

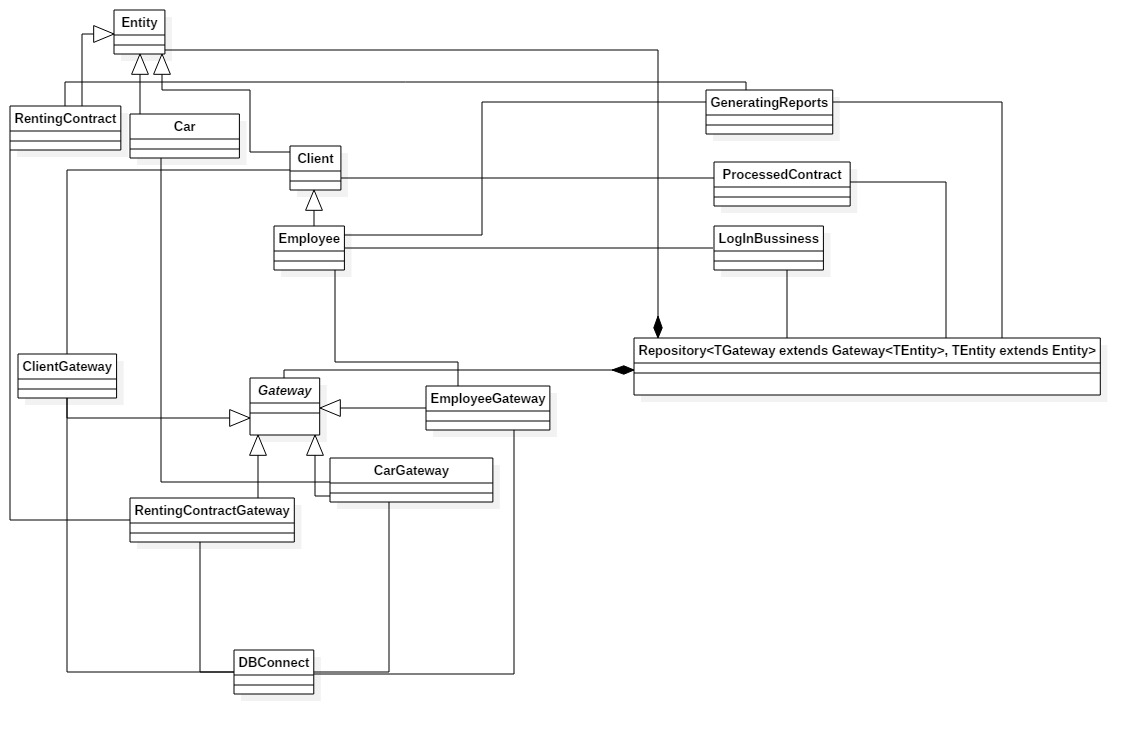
**

**

**

**

## Class Design



# Data Model

Datele vehiculate in logica aplicatiei *Rent a Car* sunt reprezentate in baza de date prin tabelele: custome*r, car, user, contract*. Aceste tabele sunt reprezentate structural in aplicatie prin clasele *Customer, Car, User, Contract*. Aceste clase au attribute de tip *String*(pentru stipularea numelui, a adresei sau a username-uli si parolei), sau de tip *BigInteger*(pentru stipularea cnp-ului clientilor, a id-ului contractelor).

# Unit Testing

Pentru testarea proiectului, s-a folosit JUnit4, implementandu-se metode de test pentru fiecare metoda considerata vitala in urma analizei de-alungul dezvoltarii si implementarii aplicatiei si

# Elaboration – Iteration 2

# Architectural Design Refinement

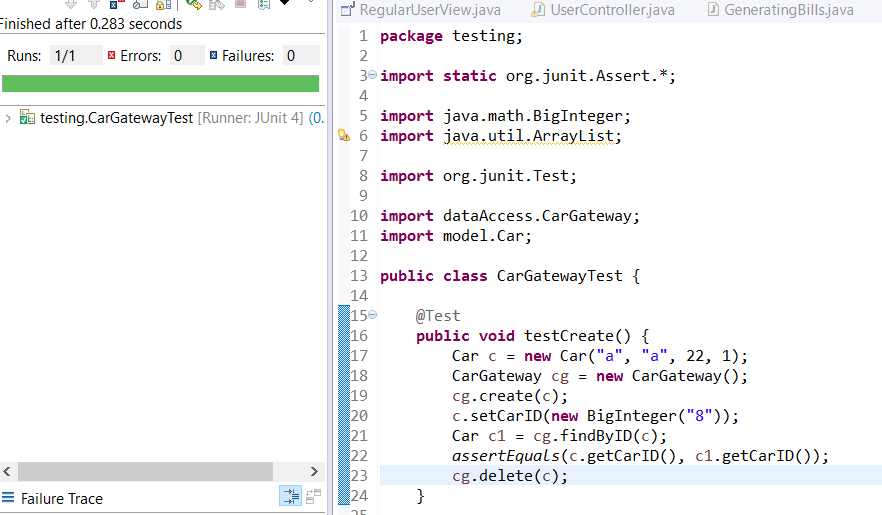
Daca initial s-a pornit cu dezvoltarea unei arhitecturi simpliste, de tip “layered”, pe parcursul dezvoltarii si implementarii aplicatiei s-a constatat necesitatea utilizarii principiilor MVC, pentru o mai buna delimitare intre partea de design a aplicatiei, interfata cu utilizatorul, si partea de back-end.

# Design Model Refinement

Pentru o structurare cat mai robusta a datelor, s-a incercat o delimitare eficienta a bazei de date in 4 tabele. Toate tabelele contin drept chei primare chei de tip BigInteger pentru o mai mare gama de stocare, iar aceste chei sunt setate pe autoincrementare, astfel incat cautarea in baza de date a unei anumite tuple sa fie cat mai eficienta din punct de vedere a timpului dar si a memoriei utilizate.

# Construction and Transition

# System Testing

******

# Future improvements

Pentru o utilizare mai facila, aplicatia s-ar putea extinde in domeniul mobile. Orice posibil utilizaor detine in ziua de astazi un telefon intelgent sau o tablet. De aceea, portarea aplicatiei si pe dispozitivele Android, IOS sau Microsoft Mobile ar ajuta utilizatorii sa expluateze mai facil functionalitatile aplicatiei.

# Bibliography