<Administrare firma inchiriere masini>

Analysis and Design Document

Student: Dobrescu Bianca Maria

**Group: 30238**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <04/04/2019> | <1.0> | <details> | Dobrescu Bianca Maria |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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

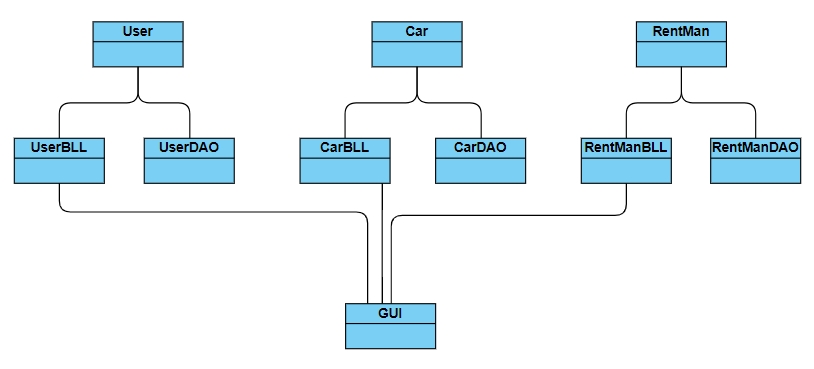
# Project Specification

# Aplicatia este utilizata pentru administrarea unei firme de inchirieri masini. Se vor efectua operatii de catre utilizator cum ar fi inchirierea unei masini, filtrarea lor pana cand gaseste una pe gustul lui dupa marca masinii sau anul in care a iesit. Acesta mai poate sa-si inchirieze masina proprie, insa administratorul trebuie sa aprobe acest lucru. Administratorul mai poate sa adauge sau sa stearga masini din baza de date, cat si sa stearga un utilizator daca ajunge la 3 abateri. O abatere consta in intarzierea returnarii unei masini imprumutate.

# Elaboration – Iteration 1.1

# Domain Model

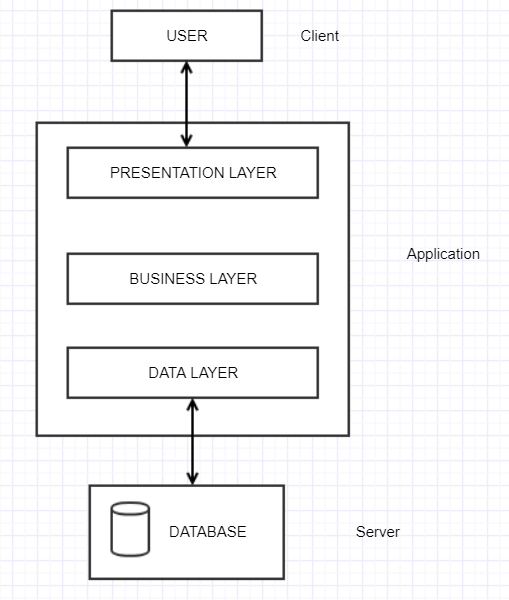
# Clasele UserDAO, CarDAO si RentManDAO extrag date din database pe care clasele din business layer (UserBLL, CarBLL, RentManBLL) le folosesc pentru a executa operatii CRUD de baza cum ar fi: inserare, stergere, updatare etc. Toate aceste date vor fi afisate in tabele folosind o interfata user-friendly descrisa in clasa GUI.



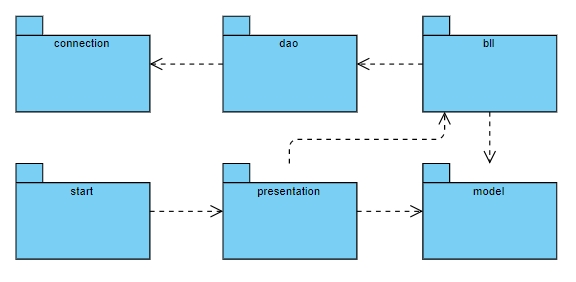
# Architectural Design

## Conceptual Architecture

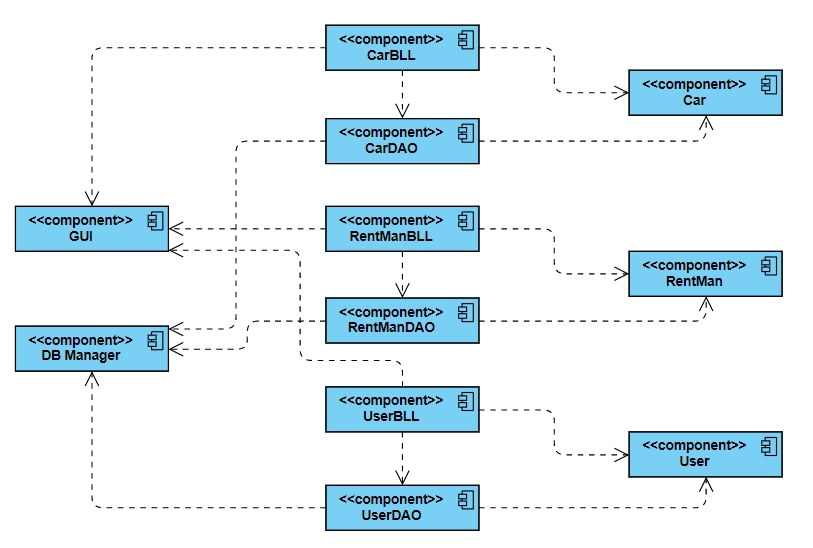
# Am folosit o arhitectura layered pe modelul MVC. MVC vine de la Model View Controller – acesta este un model folosit pentru software development si este popular pentru limbaje de programare ca Java, PHP etc. Modelul izoleaza logica de business de considerentele interfetei cu utilizatorul, rezultand in o aplicatie unde e mai usor de modificat interfata sau nivelele inferioare ale regulilor de business fara a afecta alte nivele. In cazul acestei aplicatii, principalele nivele ar fi Presentation Layer, Business Logic Layer si Data Acces Layer. In interiorul nivelului de Presentation avem clasele necesare pentru a realiza o interfata, adica renderizeaza modelele intr-o forma user-friendly pentru interactiunea cu utilizatorul. Business Logic Layer contine clase in care se realizeaza operatii logice asupra modelelor. In cele din urma, in Data Access Layer exista clasele cu metodele CRUD necesare pentru inserare, stergere si updatare.



## Package Design



## Component and Deployment Diagrams



# 

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

*[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]*

## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

# Data Model

*[Create the data model for the system.]*

# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

# Construction and Transition

# System Testing

# Pentru a testa metodele si clasele aplicatiei voi folosi JUnit. Testarea ajuta developerul sa verifice ca logica programului este corecta. Un test JUnit este o metoda aflata intr-o clasa care e folosita doar pentru testare. Aceasta clasa se numeste clasa Test. Pentru a defini ca o anumita metoda este o metoda de testare folosim @Test. Metoda assert oferita de JUnit verifica daca rezultatul asteptat coincide cu rezultatul dat.

# Future improvements

# Viitoare dezvoltari ale aplicatiei ar fi sa adaugam niste functionalitati noi cum ar fi: adaugarea unui noi tip de utilizator, angajat/operator, care poate aproba inchirierile masinilor si care poate primi un discount la firma in functie de cate “contracte” de inchiriere aproba etc. O alta dezvoltare a aplicatie ar fi sa o facem disponibila pe mobil sau web.

# Bibliography

<https://tallyfy.com/uml-diagram/>

<https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3564/85378_conceptual,l.html>

<https://online.visual-paradigm.com/features/uml-tool/>

<https://dzone.com/articles/layered-architecture-is-good>

<https://www.quora.com/What-are-the-advantages-of-using-layered-architecture-over-Model-View-Controller>