Car Renting Application

Analysis and Design Document

Student: Olar Paul Dan

**Group: 30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <23/Apr/19> | <1.0> | First version of the document | Olar Paul Dan |
| <08/May/19> | <2.0> | Second version of the document | Olar Paul Dan |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 5

2.1 Conceptual Architecture 5

2.2 Package Design 6

2.3 Component and Deployment Diagrams 6

III. Elaboration – Iteration 1.2 6

1. Design Model 6

1.1 Dynamic Behavior 6

1.2 Class Design 7

2. Data Model 7

3. Unit Testing 7

IV. Elaboration – Iteration 2 7

1. Architectural Design Refinement 7

2. Design Model Refinement 7

V. Construction and Transition 7

1. System Testing 7

2. Future improvements 7

VI. Bibliography 7

# Project Specification

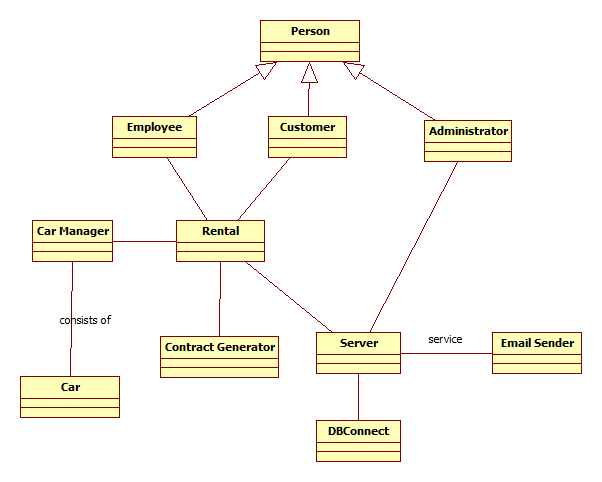
Main purpose of the application is to implement a client-server application for renting cars which can be used by customers and by the renting company’s employees. The application will allow a customer to search and view the available cars (detailed car information and pictures) for the selected date interval and from the specified source city. To rent a car, the customer must fill its personal data. After the customer has rented a car, an email is sent to him with the renting details.

When a customer that has rented a car comes to the renting company to pick up the car, an employee creates a contract which is then printed so that the client can sign it. When a car is brought back by a customer, the employee marks the car as returned.

The application must have also an administrator component for CRUD operations over cars, customers and employees.

# Elaboration – Iteration 1.1

# Domain Model



# The detailed class diagram to handle the rentals holds:

An even more detailed description of the most important event to handle in this application (the car rentals) is presented above.

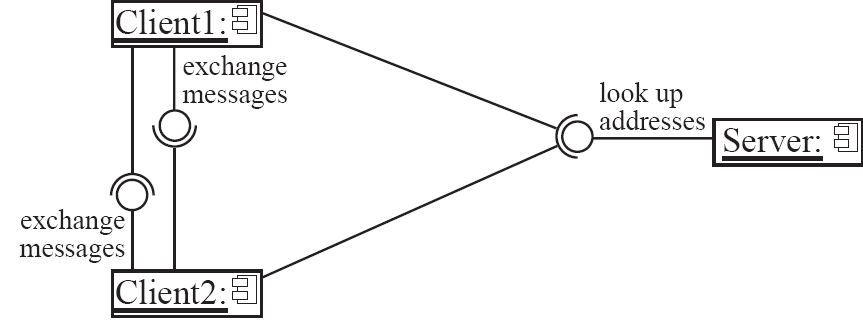
Please note that all of these might change at implementation time.

# Architectural Design

## Conceptual Architecture

Since we have to implement a client-server architecture in our design we should have into consideration the following aspects. There is at least one component that has the role of **server**, waiting for and then handling (or serving) connections. There is at least one component that has the role of **client**, initiating connections in order to obtain some service.

An important variant of the (above two-tier) client-server architecture is the **three-tier model**, which consists of three kinds of nodes: clients, application servers, and database servers. An application server behaves as a client when accessing a database server.



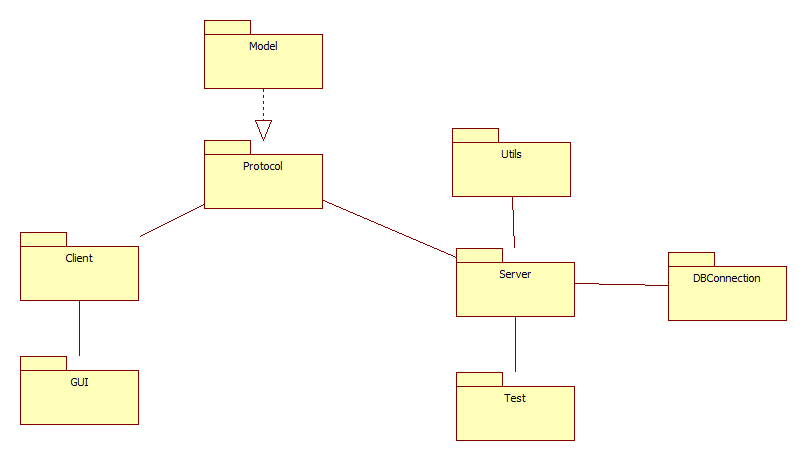
In our specific case of application the server will hold information about clients, cars and rentals. In order to access the server both the car renting company employees and the customers will need a username and password (issue already discussed in the previous documents). The administrator will be the one taking care of the server maintenance. Clients will exchange messages between them in the sense that employees and customers will establish contracts in the sense of car rentals.

The main benefits of the client/server architectural style are:

* **Higher security**. All data is stored on the server, which generally offers a greater control of security than client machines.
* **Centralized data access**. Because data is stored only on the server, access and updates to the data are far easier to administer than in other architectural styles.
* **Ease of maintenance**. Roles and responsibilities of a computing system are distributed among several servers that are known to each other through a network. This ensures that a client remains unaware and unaffected by a server repair, upgrade, or relocation.

The other patterns/architectures that will be used are: The Layered Architecture instead of MVC, the Singleton design pattern to efficiently establish the database connection. Other design patterns that could be used: Factory Method, Observer, Data Source Logic Patterns.

## Package Design



## Component and Deployment Diagrams

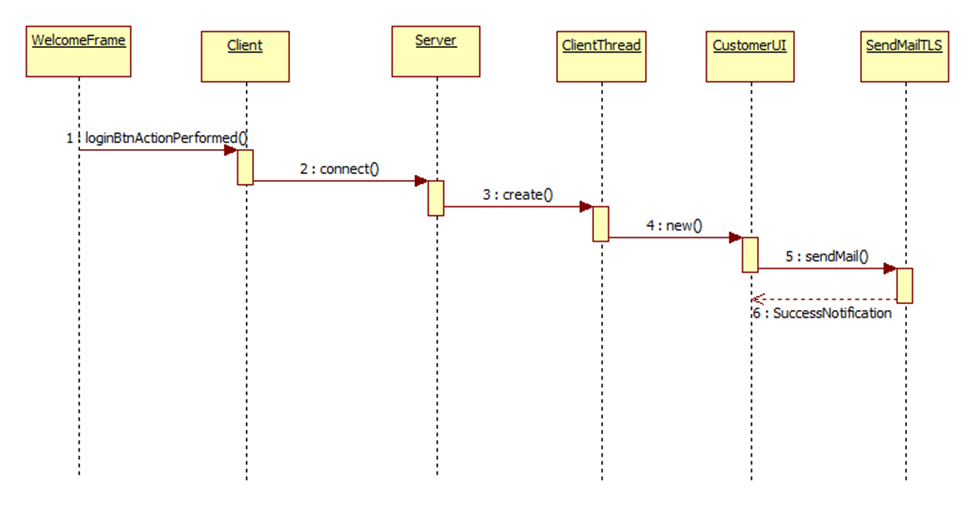
# 

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

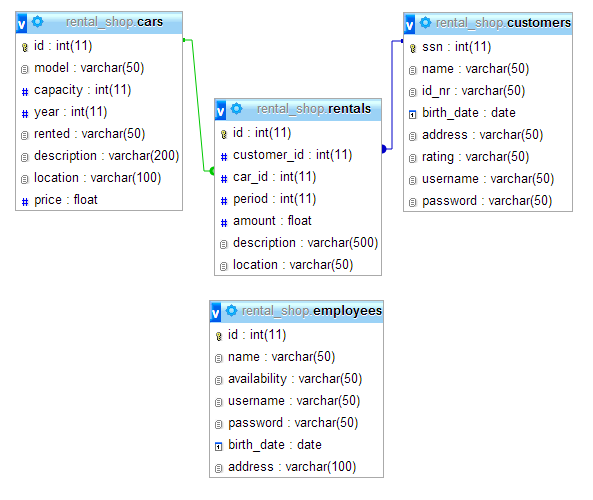
Sequence diagram for “send rental request via email message” use case

**

## Class Design

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

# Data Model



–

# Unit Testing

Unit tests are the fundamental tests in your app testing strategy. By creating and running unit tests against your code, you can easily verify that the logic of individual units is correct. Running unit tests after every build helps you to quickly catch and fix software regressions introduced by code changes to your app. For testing my application, I will create these types of automated unit tests:

**Local tests:** Unit tests that run on your local machine only. These tests are compiled to run locally on the Java Virtual Machine (JVM) to minimize execution time. For tests that depend on my own dependencies, I will use mock objects to emulate my dependencies' behavior.

**Instrumented tests:**  Unit tests that run on an Android device or emulator. These tests have access to instrumentation information, such as the “Context”  for the app under test. I will use this approach to run unit tests that have complex Android dependencies that require a more robust environment, such as Robolectric.

# 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

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*

# Bibliography