<Tourism Agency>

Student:Anamaria Pali

**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 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

Use JAVA/C# API to design and implement an application for the agents of a tourism agency. The application should have two types of users (a regular user represented by the regular tourism agent and an administrator user) which have to provide a username and a password in order to use the application.

# Functional Requirements

The regular user can perform the following operations:

* Add/update/view client information (name, identity card number, personal numerical code, address, etc.).
* Add/update/view/delete a holiday reservation for a client (destination, hotel name, number of persons who are going on holiday, details about each member going on holiday, total price, final payment date)
* Accept partial payments from a client before final payment date
* View all the clients who missed the final payment deadline and have the possibility to cancel their holiday

The administrator user can perform the following operations:

* CRUD on agents’ information.
* Generate reports for a particular period containing the activities performed by an agent.

# Non-functional Requirements

The data will be stored in a database. Use the Layers architectural pattern to organize your application. Use a domain logic pattern (transaction script or domain model) / a data source hybrid pattern (table module, active record) and a data source pure pattern (table data gateway, row data gateway, data mapper) most suitable for the application.

All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.

2. Use-Case Model

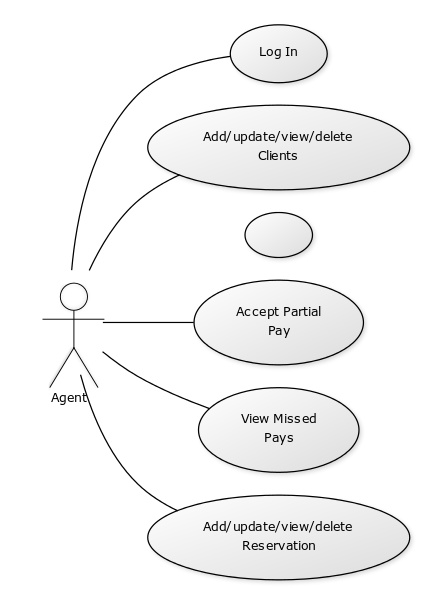
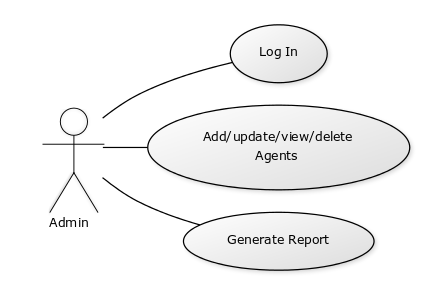
*Use case:* log in

*Level:* user-goal level

*Primary actor:* Regular user

*Main success scenario:* provide username and password and press the log in button in which case the actor will see a menu of the rest of actions he can perform

*Extensions:* the actor doesn’t have an account so he has to ask the administrator to create one



3. System Architectural Design

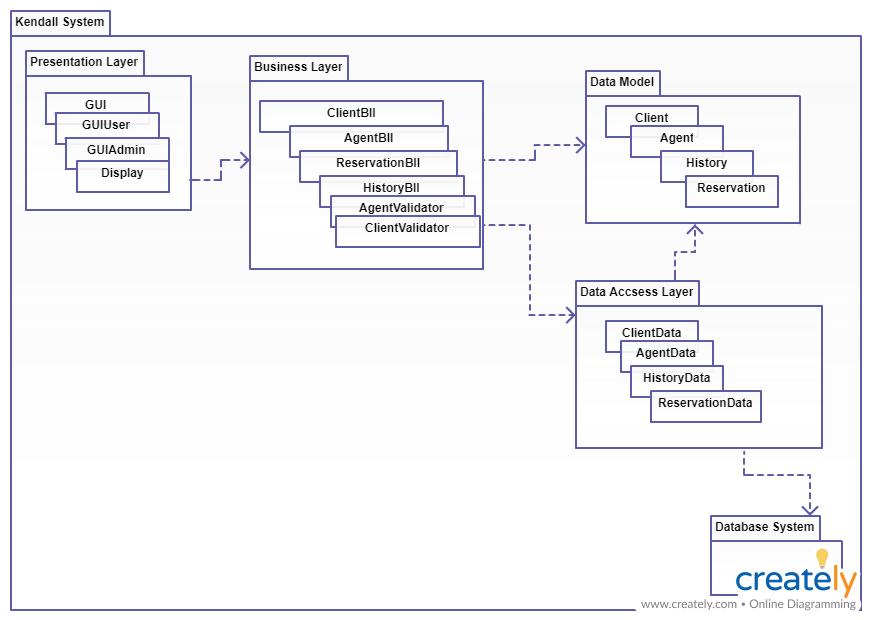
**3.1 Architectural Pattern Description**

The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. This pattern is the de facto standard for most Java EE applications and therefore is widely known by most architects, designers, and developers. The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.

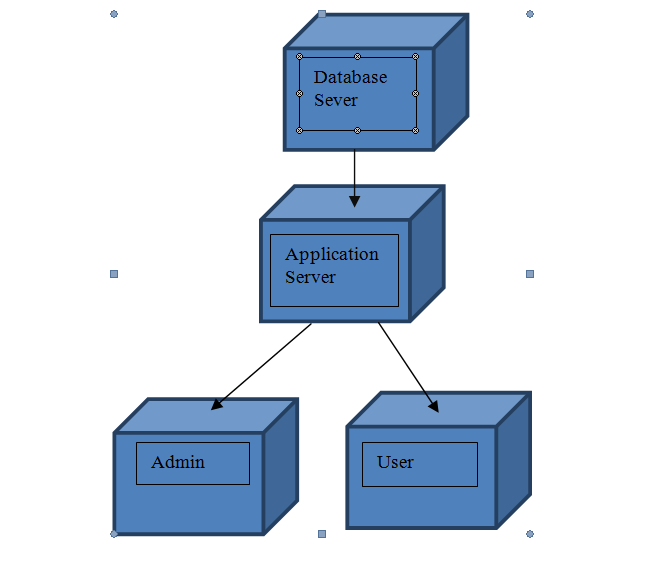
Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database. Each of the layers in the architecture is marked as being closed. This is a very important concept in the layered architecture pattern. A closed layer means that as a request moves from layer to layer, it must go through the layer right below it to get to the next layer below that one. For example, a request originating from the presentation layer must first go through the business layer and then to the persistence layer before finally hitting the database layer.

**3.2 Diagrams**

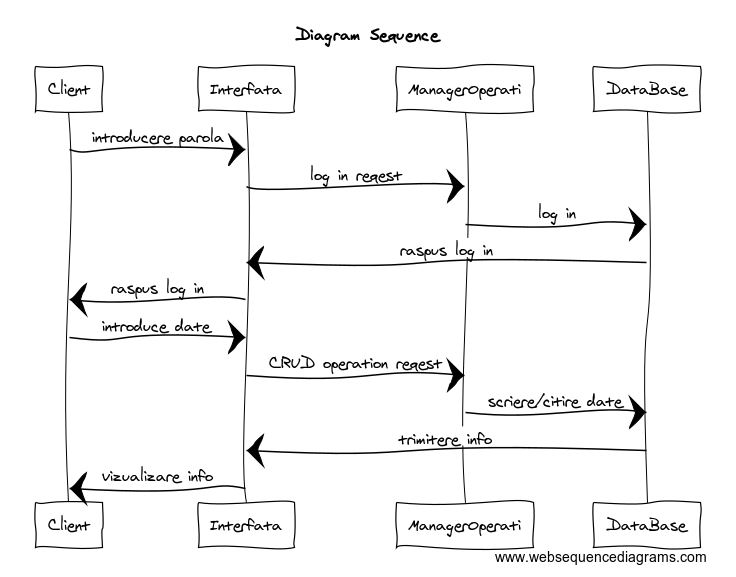
### 3.2.1Package Diagram



*3.2.2 Component and deployment diagrams*



4. UML Sequence Diagrams

**

5. Class Design

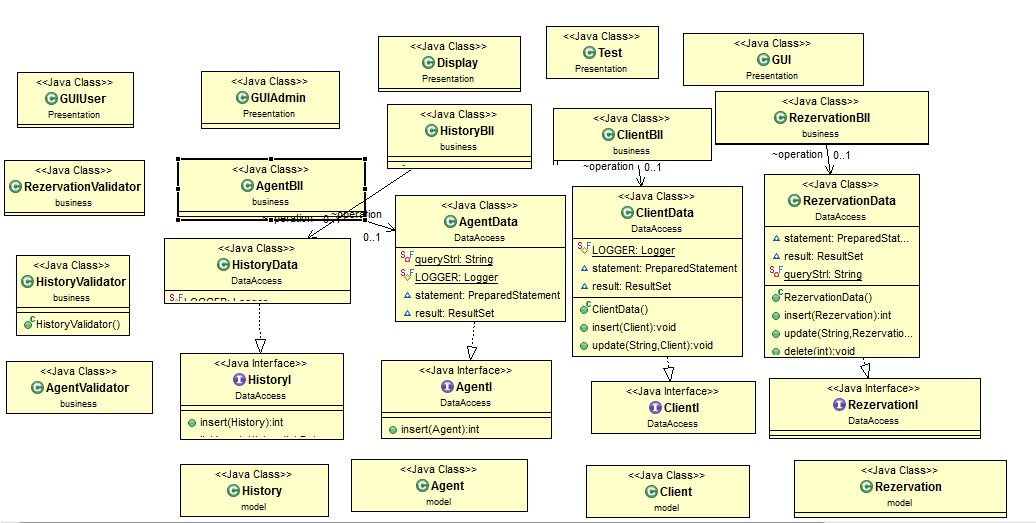
**5.1 Design Patterns Description**

**Transaction Script** (TS) is the simplest domain logic pattern we can find. It needs less work to implement than other domain logic patterns and therefore it’s perfect fit for smaller applications that doesn’t need big architecture behind them.

Transaction Script organizes all this logic primarily as a single procedure, making calls directly to the database or through a thin database wrapper. Each transaction will have its own Transaction Script, although common subtasks can be broken into subprocedures.

**Data Mapper** is a layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself. It is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and also to isolate them from each other. With Data Mapper the in-memory objects needn't know even that there's a database present; they need no SQL interface code, and certainly no knowledge of the database schema.

**5.2 UML Class Diagram**



6. Data Model

In my database I use four tables: clients,agents, reservation and history. For each table

we created a class that has as the instance variables the table rows.

7. System Testing

I used two methods for testing the data introduced in the graphical interface by the user: one of them is checking whether the CNP has 13 digits or 8 for companies and the other method is checking if the sum is < 0 when the user wants to take money from the account.

8. Bibliography

Martin Fowler et. al, Patterns of Enterprise Application Architecture