<Tourism Agency>

Student: Plesa Gabriel

**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

3. System Architectural Design 5

4. UML Sequence Diagrams 6

5. Class Design 7

6. Data Model 8

7. System Testing 8

8. Bibliography 8

1. Requirements Analysis

# Assignment Specification

The goal of this assignment is to design and implement an application for the agents of a tourism agency. The application has 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, 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:

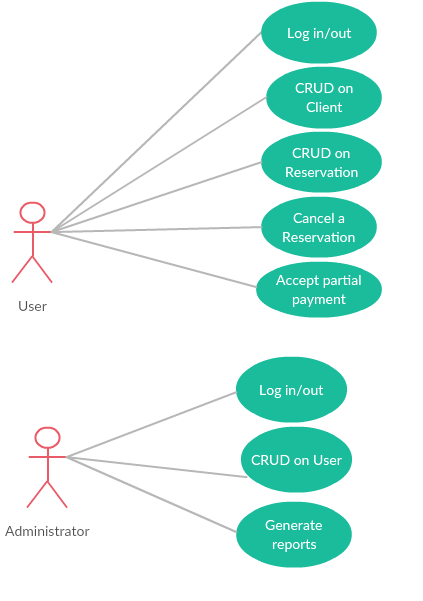
* CRUD on agents’ information.
* Generate reports for a particular period containing the activities performed by an agent.

# 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:** <Accept partial payment>

Level: <user-goal level >

Primary actor: <User>

Main success scenario: <The user successfully logins and then he enter the Client ID from who wants to accept the payment and press the accept button and register the payment>

Extensions: <the user wrongly introduce his username or password, an error message is shown, after that he finally login>

3. System Architectural Design

**3.1 Architectural Pattern Description**

An architectural style, sometimes called an architectural pattern, is a set of principles—a coarse grained pattern that provides an abstract framework for a family of systems. An architectural style improves partitioning and promotes design reuse by providing solutions to frequently recurring problems. You can think of architecture styles and patterns as sets of principles that shape an application

The Three Principal Layers of an architecture are three primary layers: presentation( Provision of services, display of information (e.g., in Windows or HTML, handling of user request, HTTP requests, command-line invocations, batch API), domain( Logic that is the real point of the system) and data source(Communication with databases, messaging systems, transaction, managers, other packages).

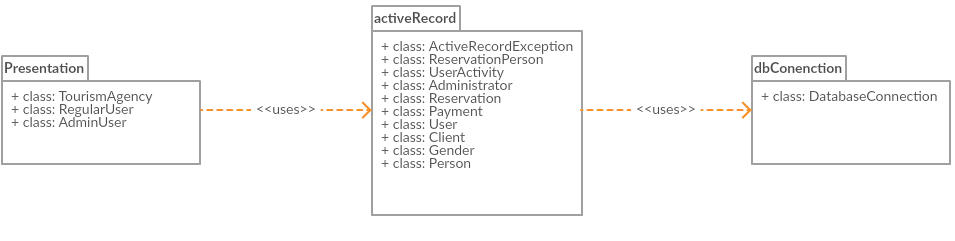
Presentation logic is about how to handle the interaction between the user and the software. This can be as simple as a command-line or text-based menu

Data logic is about communicating with other systems that carry out tasks on behalf of the application. These can be transaction monitors, other applications, messaging systems, and so forth. For most enterprise applications the biggest piece of data source logic is a database that is primarily responsible for storing persistent data.

The business logic is about the work that this application needs to do for the domain you’re working with. It involves calculations based on inputs and stored data, validation of any data that comes in from the presentation, and figuring out exactly what data source logic to dispatch, depending on commands received from the presentation.

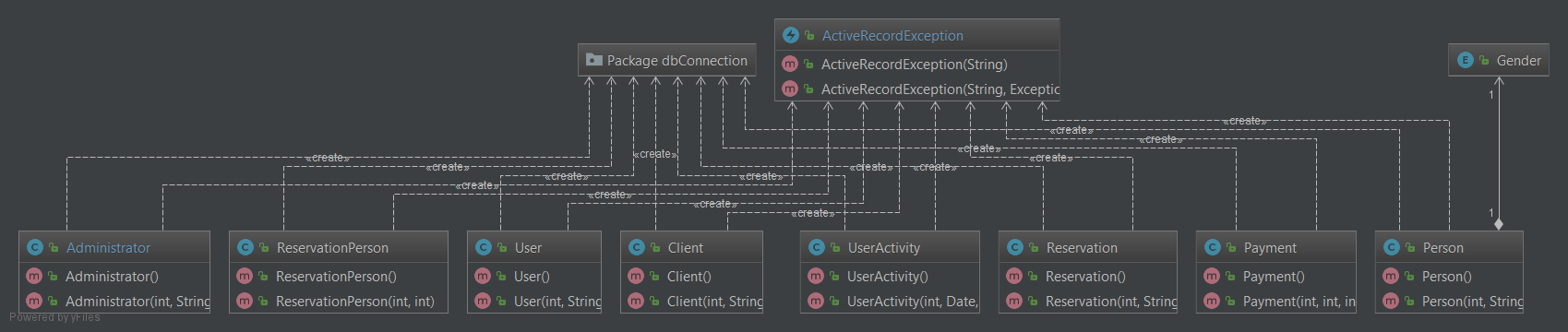
**3.2 Diagrams**

**Package Diagram**

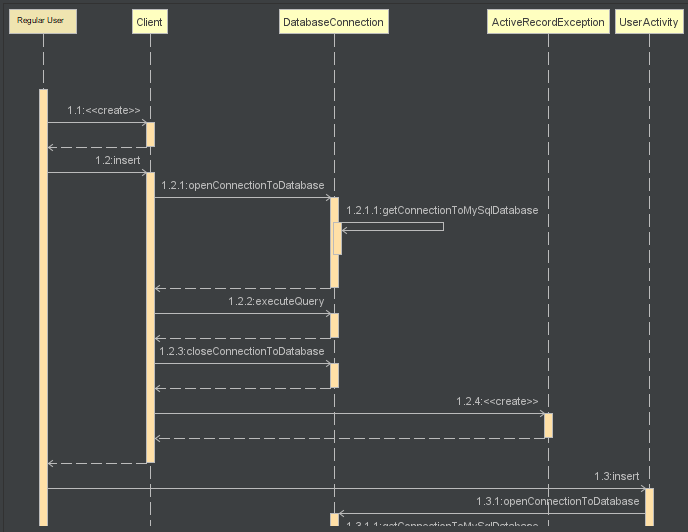


**Design patterns:**

**Active Record Pattern**- An object carries both data and behavior. Much of this data is persistent and needs to be stored in a database. Active Record uses the most obvious approach, putting data access logic in the domain object. This way all people know how to read and write their data to and from the database.



4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

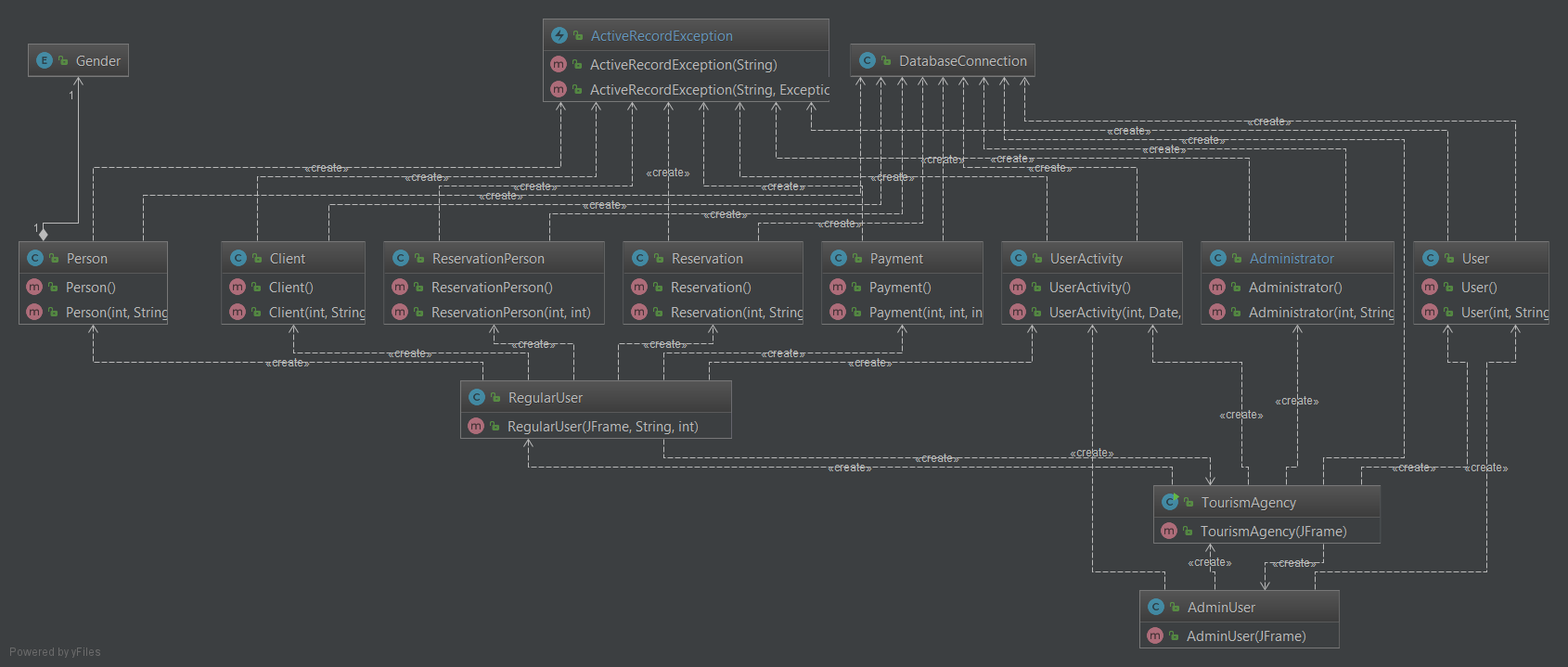
In “Active data record”, an object carries both data and behavior. Much of this data is persistent and needs to be stored in a database. Active Record uses the most obvious approach, putting data access logic in the domain object. This way all people know how to read and write their data to and from the database.

Active Record has a problem though; it has responsibilities which span both the Domain Layer and the Technical Services Layer:

* It represents a Domain Model Object, implementing both its behavior and its data attributes
* It deals with SQL and persists data to the database

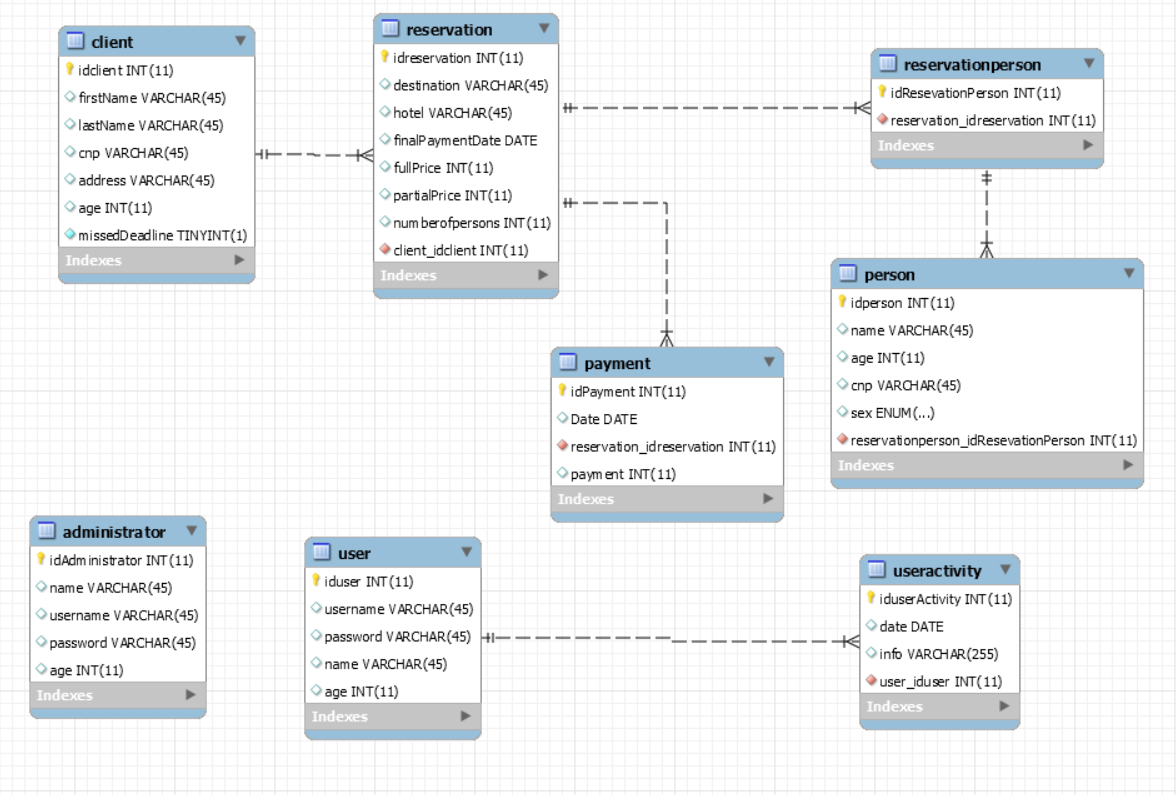
This is less than ideal, and results in the Active Record object residing in both the Domain Layer and the Technical Services Layer.

**5.2 UML Class Diagram**



6. Data Model

I choose eight entities for describe the functionality of my application: Client, Reservation, ReservationPerson, Person, Payment, Administrator, User, UserActivity. These entities are described in SQL Language as in the next image, which represents the database diagram:



7. System Testing

For the main operations the system supports tests: insert, delete, update, view, etc. If something is going wrong the application send an error message to inform the user. The information is not tested and validated before register it into the database.

8. Bibliography

[1] https://martinfowler.com/bliki/PresentationDomainDataLayering.html

[2] http://richard.jp.leguen.ca/tutoring/soen343-f2010/tutorials/implementing-active-record/

[3] https://stackoverflow.com

[4] https://www.google.ro