Tourism Agency

Student:Galeata Bianca-Floriana

**Group:30235**

Table of Contents

Table of Contents 2

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 4

2. Use-Case Model 5

3. System Architectural Design 6

4. UML Sequence Diagrams 9

5. Class Design 9

6. Data Model 10

7. System Testing 12

8. Bibliography 12

1. Requirements Analysis

# Assignment Specification

The objective of this project is to design and implement an application for a tourism agency. The application is going to facilitate the way that a tourism agent handles his clients, reservations and all the data regarding them.

Beside the agent, the application will have another user, named administrator. This will have the possibility to manage the agents’ information.

The application should help agents and their supervisors (administrator) to manage the data more easily and to make the company that will use it more efficiently and organized.

# Functional Requirements

Using this application, an agent can do:

* Add/update/view client information
* Add/update/view/delete a holiday reservation for a client
* 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

All the data will be stored in a database, and the next information will be needed about:

*Client:*

* name,
* identity card number,
* personal numerical code,
* address
* any other information that may be important.

*Reservation:*

* destination,
* hotel name,
* number of persons who are going on holiday,
* details about each member going on holiday,
* total price,
* final payment date,
* any other information that may be important.

The administrator can perform CRUD operation on agents’ information and generate reports containing the activities performed by an agent in a particular period of time.

Both users have to use a username and a password in order to access the application.

# Non-functional Requirements

* *Availability* - This attribute counts the time the application is functional.

This project will be projected as a desktop application, and a result the application can be accessed anytime. But, there is a necessity to have an internet connection in order to have access the data from the database and to store new data in it.

* *Performance* - indicates how responsive an application is when an action and an interval of time are given.

As a desktop application the performance can be reported to the system that is used and to the database connection.

On a decent system, the performance will be on its best behavior and the time response will be small. But all the responsiveness will be affected by the connection to the database. If the connection does not give a response in a short amount of time, this will influent the performance.

* *Security* - The capability of a system to prevent malicious or accidental actions outside of the designed usage.

Any person that have the application can access it. But in order to use the application and to have access to the database you have to login with a username and a password. This date must be correct or else the access will not be provided. In this manner just the agents who have an account or the administrator can use the application.

* *Testability* - This measure show how easy is to create test for a system and how quickly a failure can be demonstrated.

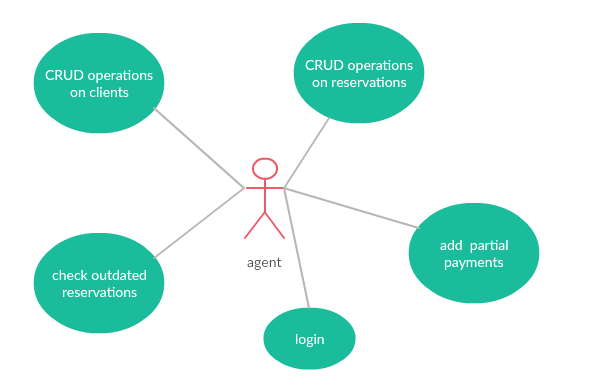
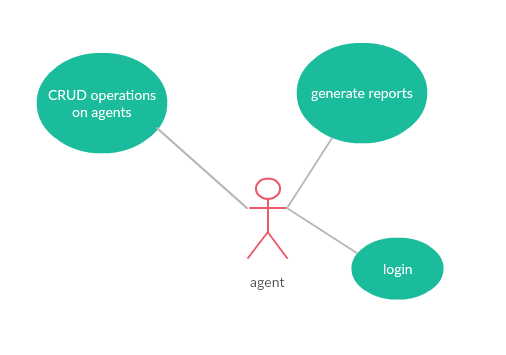
The application has been tested in the implementation process and after this process, as a system. Each component has been tasted separately and in connection with others. With each new module, a small test was made in order to find the error. Each greeted problem has been solved.

* *Usability* - This quality measure how intuitive, easy and simple is the user’s interaction with the application.

The UI is simple and intuitive. There are tables to show different date, each column having the name of the concept it represents. The buttons have suggestive names to indicate their role. There are messages that are shown in order to indicate if an action has been done wrong or it has success.

2. Use-Case Model

* *Use cases:*



* *Use case detailed:*

*Use case:* add a client

*Level:* user-goal level

*Primary actor:* agent

*Main success scenario:* The first step is to access the application. Once it is open, a window will be seen. There are 2 text field where the username and password must be inserted. Let’s suppose that those are correct. Once the button “login” is clicked, a new window will appear. There is a table with the clients and buttons that help to make CRUD operations on them. In order to add a new client, the button “new Row” mush be clicked to create an empty row. In that new line of the table, all the data needed about the client must be inserted. Supposing that the data are correct, to save them the button “Add/Update” must be clicked.

*Extensions:* The first derivation of the initial scenario would be if the name or password are wrong. If the button “Login” is click an error message will appear, which will say “Wrong password/username!”. He will log with success if will write the correct name and password.

3. System Architectural Design

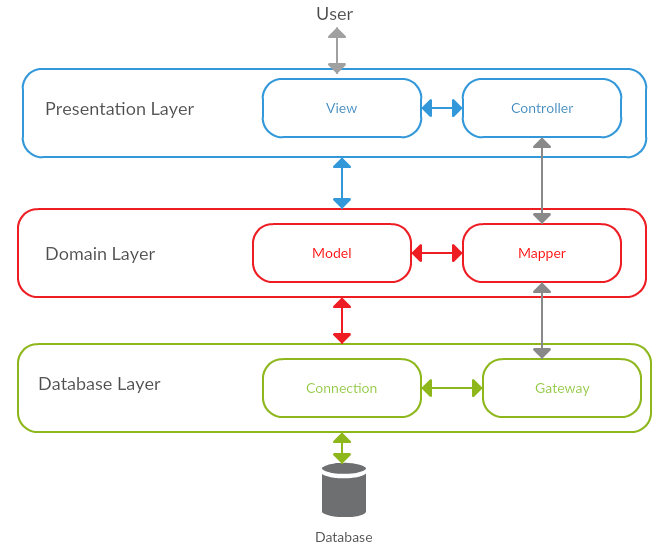
**3.1 Architectural Pattern Description**

The used Architectural Pattern is named Layered Architecture. The concept is simple explain by the division of the program in three layers: presentation (UI), domain logic (or business logic), and data access.

The presentation layer is responsible to handle the user interface, the domain logic layer has to execute specific business rules associated with a request and the last one has to handle the access to the database.

**3.2 Diagrams**

* Conceptual architecture:



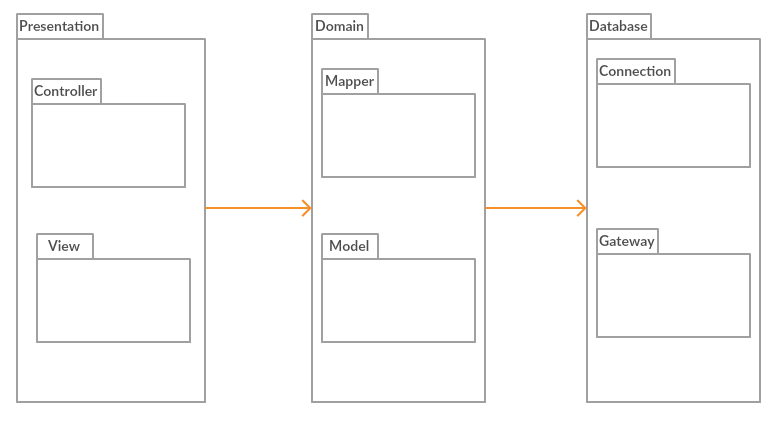
This diagram shows how the application is structured and how it’s components communicate. As it can be seen, there are three layers: Presentation Layer, Domain Layer and Database Layer.

The Presentation Layer has components as View classes which compose the UI and their controllers which connects the logic domain and the UI.

The Domain Layer has classes which represent the model which represent the data manipulated and their mapper. The classes which are mappers are the link between this layer and Database layer.

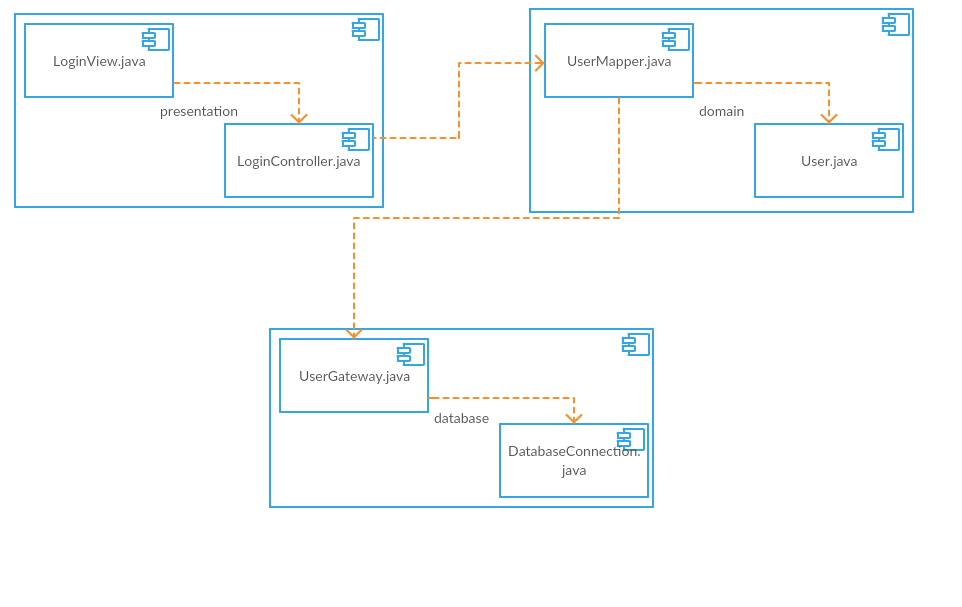
The database Layer has a class for connection to database and classes named gateway to map the action that can made to the database (insert, delete etc.)

* Packages diagram



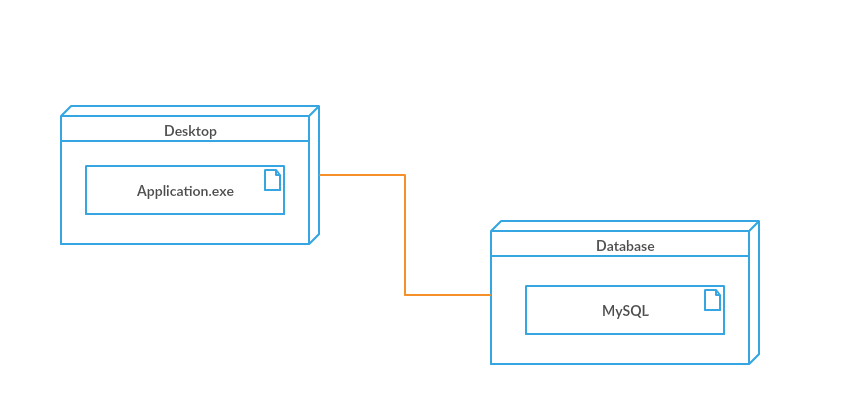
Here is presented the packages diagram. The structure of packages highlights the layered Architecture whit 3 main layers and their also structured on layers inside.

* Component diagram



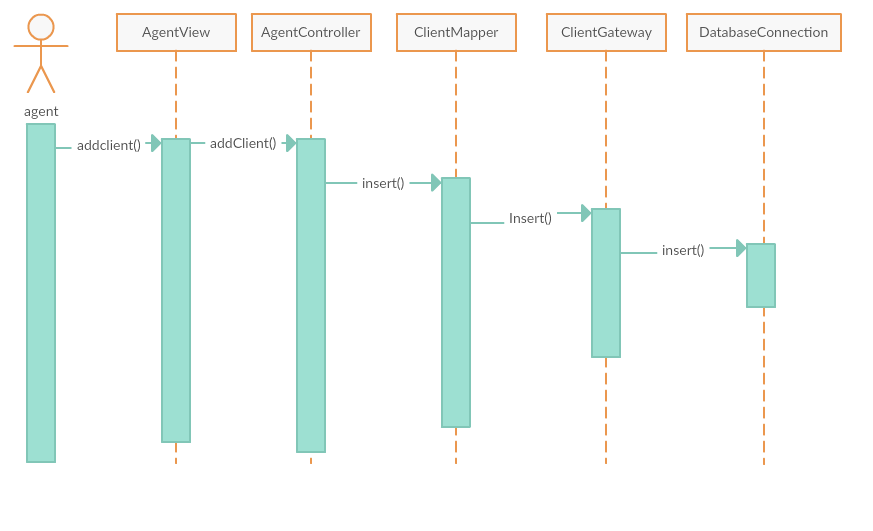
This component diagram shows how classes communicate in order to accomplish the login action. As is can be seen the view communicates with its controller and from there the controller communicates with a mapper from the domain layer. The mapper has a connection to the model and to a Gateway from database layer. The Gateway has the connection to the database. The connections are in both direction, the date being transmitted from the database to the UI and inverse.

* Deployment diagram



This diagram present how the application is deployed. As it can be seen the application has access to a database, but the main program runs on users’ computers.

4. UML Sequence Diagrams

**

This sequence diagram shows how the components connect in order to insert some data in the database. The request leaves from user to the controller by the view. From there it is transmitted to the mapper and then to the gateway. Finally, from the gateway through the connection the data is saved in the database.

5. Class Design

**5.1 Design Patterns Description**

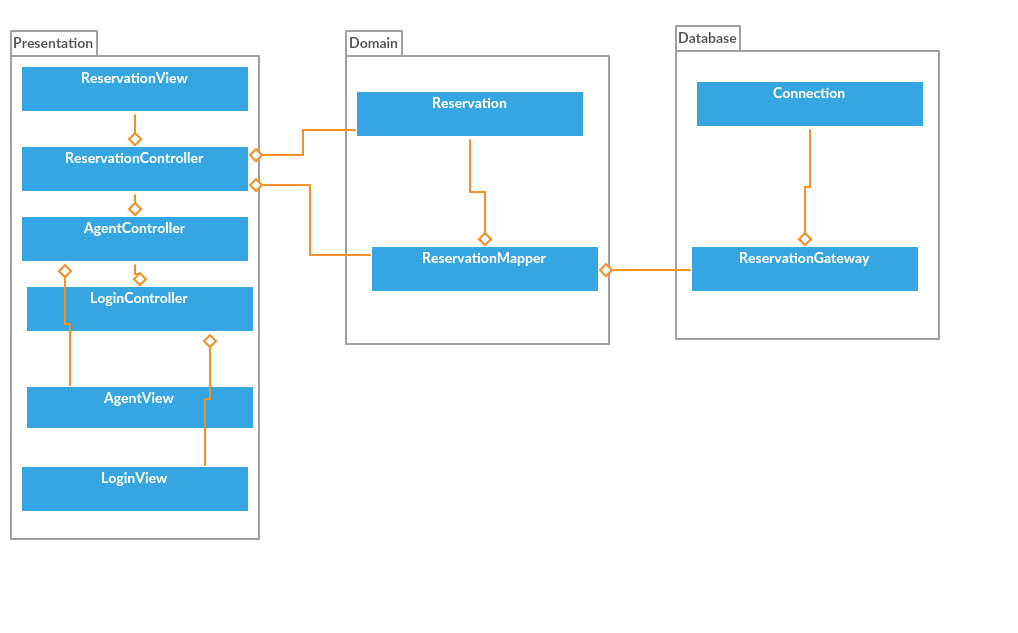
The main design patterns that have been used are: *Domain Model, Data mapper, Table Data Gateway.*

The *Domain Model* is a pattern for the domain logic. It represents the model of the domain that has behavior and data.

The *Data Mapper* separates the in-memory objects from the database, isolating one from the other. It is a data source pure pattern.

The *Table* *Data Gateway* handles all the logic for database access (insert, delete etc.) and all the rows from a table. It is a data source pure pattern.

**5.2 UML Class Diagram**



In this section is presented a class diagram. In the diagram are not included all the classes because the connection between them are similar. The classes are divided on layers, and implements some patterns.

The Gateway class represent the Table Gateway Pattern and has the operations needed to access the database. This patter is integrated whit the Data Mapper Pattern, which is a mapper for the model, and has the operation to manipulate the database but it does not have access to it. (ReservationGateway class in this case)

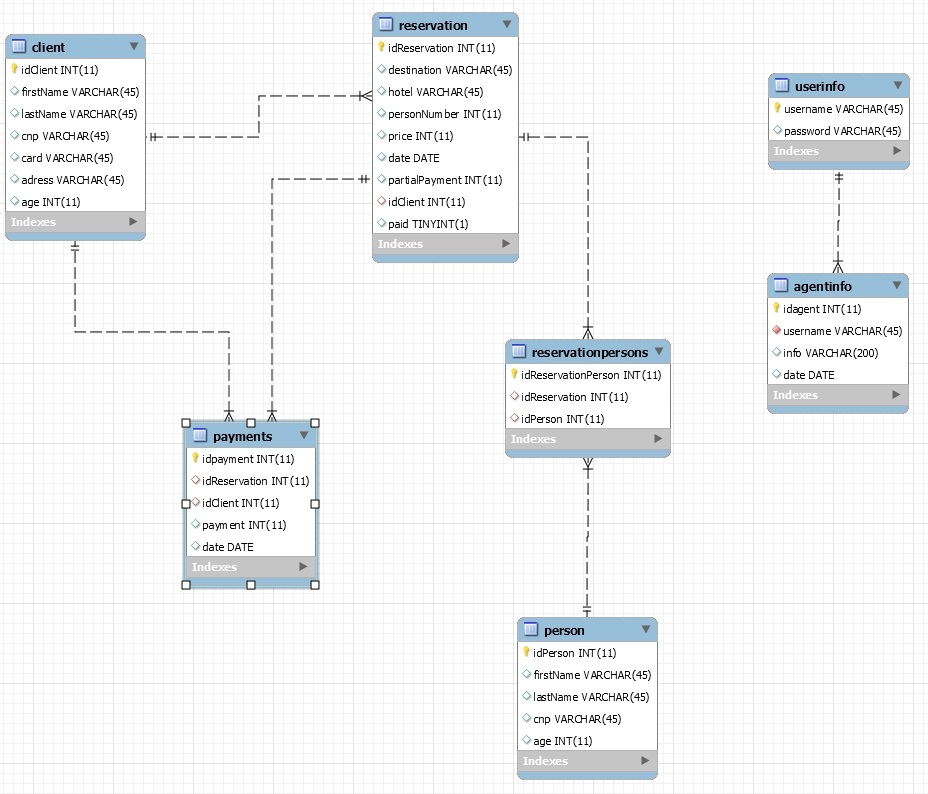
The Data Mapper use the gateway classes to insert data, in this way there is not a direct connection between domain layer and the database. (ReservationMapper class in this case)

The Domain Model pattern is also used, the logic and data being in the same class. (Reservation class in this case)

6. Data Model

The user Data Model in this project is *Relational model*. Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values.

The next diagram shows how the database for this project looks:



Database diagram

*Table Client:* Here are stored information about clients. Each client has an id, first name etc. The primary key is “idClient”.

*Table Reservation:* Information about the reservations are stored. The table has “idReservation” as primary key. It also has a foreign key named “idClient” which refers to table Client.

*Table Payments*: Here is stored a history of each payment made. The primary key is “idpayment” and the foreign keys are “idClient” and “idReservation”.

*Table Persons:* Information about persons that are going on a vacations but are not clients, are introduced here. The primary key is “idPersons”.

*Table reservationPersons:* Map the many to many relationships between Reservation and Persons. The primary key is “idreservationPersons” and has foreign keys “idPerson” and “isReservation”.

*Table userinfo: The login information can be found here. The primary key is “username”.*

*Table agentInfo: The information about agents’ activity is stored here. The primary key is “idAgent” and is has a foreign key “username”.*

7. System Testing

In each of the strategies that will be presented the data-flow method has been used. Even if it started from a user interface or from a class test that has triggered a series of event, information has been printed in order to detect if the data have anomalies.

* *Unit testing*

Each class has been tasted once implemented. Tests have been made for to the database layer, for the domain layer and for the presentation layer and theirs classes.

The database’s classes have been tested, making inserts, deletes and updates into the database using a test class for different tables.

The domain’s classes have been tested by printing the result and manually verify it.

The presentation’s classes have been tested by printing the result and manually verify it.

* *Integration testing*

The modules have been tested in a group in order to obtain the functionality required. First classes from database layer have been integrated with those in the domain layer, and then the ones from presentations layer have been added.

* *Validation testing*

This testing strategy has been made by users by testing the functionality provided by the application.

8. Bibliography

[1] *Presentation Domain Data Layering*, from <https://martinfowler.com/bliki/PresentationDomainDataLayering.html> [2018-03-29]

[2] *Chapter 1. Layered Architecture*, from <https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html> [2018-03-09]

[3] *Domain Model pattern example,* from <https://stackoverflow.com/questions/41335249/domain-model-pattern-example> [2018-03-29]

[4] *Implementing the Data Mapper Pattern,* from <http://richard.jp.leguen.ca/tutoring/soen343-f2010/tutorials/implementing-data-mapper/> [2018-03-29]

[5] *Data Mapper*, from <https://martinfowler.com/eaaCatalog/dataMapper.html> [2018-03-29]

[6] *Data model,* from <https://en.wikipedia.org/wiki/Data_model#Types_of_data_models> [2018-03-29]

[7] *Everything You Need to Know About Software Testing Methods,* from <https://www.thebalance.com/all-you-need-to-know-about-software-testing-methods-4019921> [2018-03-29]