Tourism Agency

Analysis and Design Document

Student: Prodan Andreea

**Group: 30235**

Table of Contents

Contents

[1. Requirements Analysis 3](#_Toc510109945)

[Assignment Specification 3](#_Toc510109946)

[Functional Requirements 3](#_Toc510109947)

[Non-functional Requirements 3](#_Toc510109948)

[2. Use-Case Model 4](#_Toc510109949)

[3. System Architectural Design 5](#_Toc510109950)

[3.1 Architectural Pattern Description 5](#_Toc510109951)

[3.2 Diagrams 5](#_Toc510109952)

[Packages 7](#_Toc510109953)

[Components 8](#_Toc510109954)

[Deployment 8](#_Toc510109955)

[4. UML Sequence Diagrams 9](#_Toc510109956)

[5. Class Design 9](#_Toc510109957)

[5.1 Design Patterns Description 9](#_Toc510109958)

[5.2 UML Class Diagram 10](#_Toc510109959)

[6. Data Model 11](#_Toc510109960)

[7. System Testing 12](#_Toc510109961)

[8. Bibliography 12](#_Toc510109962)

# 1. Requirements Analysis

## Assignment Specification

The application is designed for a tourism agency which has two types of users: a regular user represented by the regular tourism agent and an administrator user. An agent has the option to perform operations which permit them to hold informations about clients and their reservations. An administrator has the ability to perform operations on the agents information and to track their performed activities.

## Functional Requirements

**Interface requirements**

The PNC needs to be a number consisting of 13 digits.

The data needs to have the format YYY-MM-DD.

The e-mail must have the right format.

The phone number, ids, and the price must be a number.

The identity card number must consist of 2 letters.

**Regulatory/Compliance Requirements**

Clicking the Sign In button sends the user to its specific page depending on the type it is (agent or administrator).

Each time an agent performs an action, a data containing the agent and the action will be inserted into the history table.

When the agent selects a row from the clients table, the data from the row is put in the specific fields.

**Security Requirements**

The agent can’t access the data about itself or the history with its activities.

## Non-functional Requirements

**Access Security**

The application can be accesed just by authorized users, using their username and password.

**Accessibility**

Any person should be able to use the application after reading the user manual.

**Response time**

Each action must last less than 1.5 seconds in 90 precent of cases and less than 5 seconds in rest.

The data from tables should load in 2 seconds or less.

**Availability**

The application should be available allways between 6am and 6pm, the updates being applied just outside this hours.

**Reliability**

Probability of failure on demand (POFOD) shall be 0.0001 (1 out of 10000 plays) when an agent requests to see the some data.

# 2. Use-Case Model

Agent

Sign In using a username and a password

View all the clients who missed the final payment deadline and have the possibility to cancel their holiday

Accept partial payments from a client before final payment date

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

Add/update/view client information

Admin

CRUD on agents’ information.

Generate reports for a particular period containing the activities performed by an agent

Sign In using a username and a password

Use case: Sign In

Level: user goal

Primary actor: agent

Main success scenario: The agent enters its username and password correctly and presses the Sign In button. The agent is signed in and send to the agent’s page successfully

Extensions: If one of the introduced input is incorrect or the username is not found in the database, a message is displayed

Use case: update a client’s data

Level: user goal

Primary actor: agent

Main success scenario: The agent introduces all the data in the specific fields or selects the row with the client whoes data wants to change and makes the changes on the data and clicks the Update button. The data is updated successfully.

Extensions: If one of the data is incorrect an error message is shown and no changes are made.

Use case: see the actions performed by an agent during a period of time

Level: user goal

Primary actor: administrator

Main success scenario: The administrator introduces the user id and the dates between which period to show the data. The show button is clicked. The data are shown in the History table.

Extensions: If the id is not a valid id, or the date is incorrect, a message will be shown.

# 3. System Architectural Design

## 3.1 Architectural Pattern Description

The layer architectural pattern is used to organize the application. The layers are: presentation (views), bussiness (controllers, models and services),persistence (dao – Data Access Only) and database.

Each layer has a specific role and responsibility within the application. The presentation layer is responsible for handling all user interface communication. The business layer is responsible for executing specific business rules associated with the request. The persistence layer is responsible for communicating with the database through SQL querying. The database layer represents the actual database.

All the layers are closed, therefore 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 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

Users (agent / admin)

UI Components

Presentation Layer

Business Layer

Controller Components

Model components

Business Components

Persistence Layer

DAO Components

DB Layer

DB Components

The controller acts as a wall between the UI and the rest of the application, deceding which server to call to further execute specific operations for the request. For example if a request for updating a client’s information is received, the controller will chose to call the ClientService for the update and HistoryService for saving the agent’s action.

## Packages

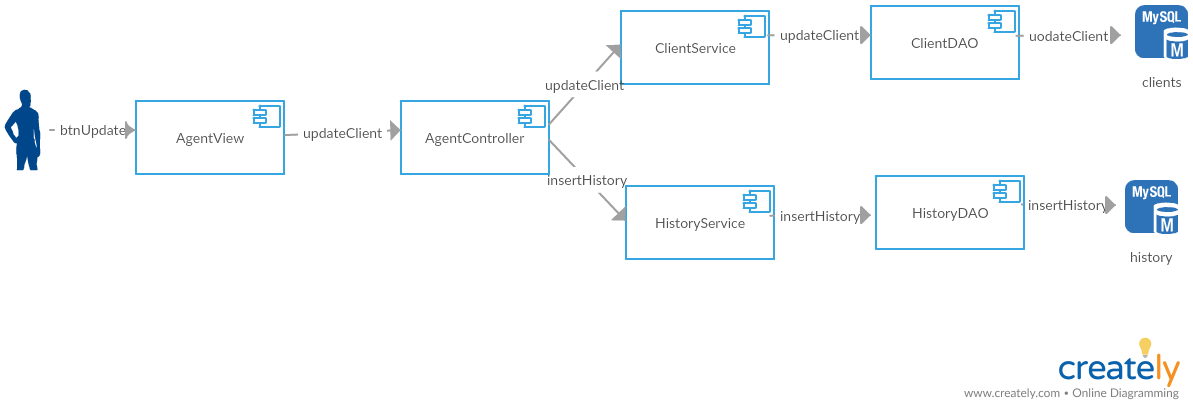


src

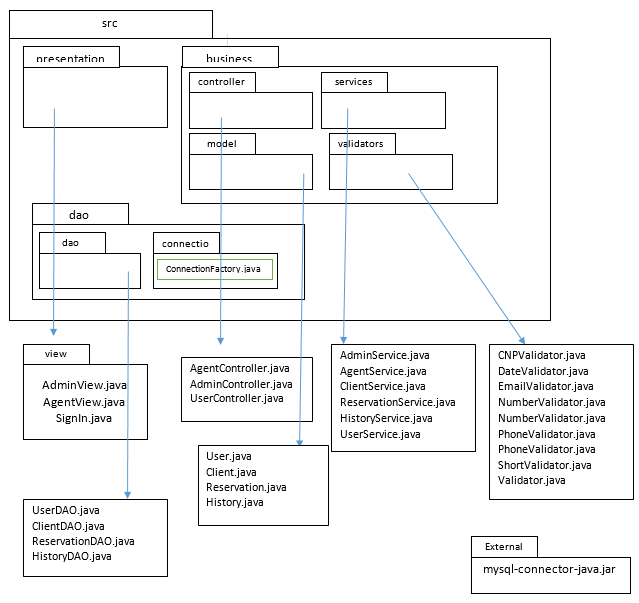


test

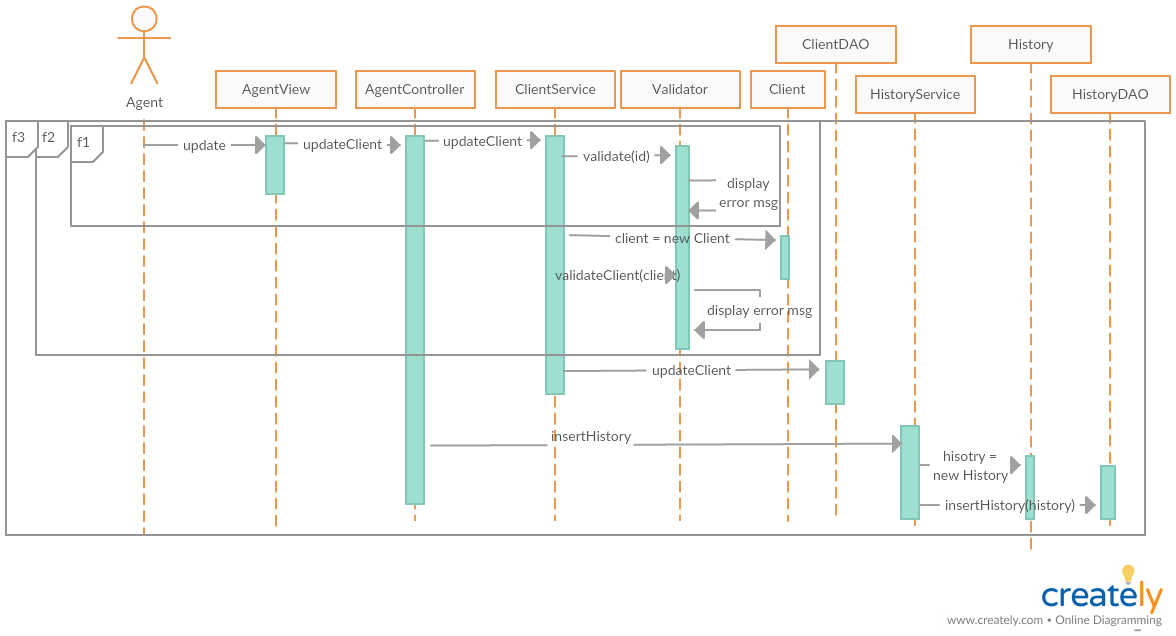
## Components



## Deployment



# 4. UML Sequence Diagrams



f1 – when validate(id) generates an error message

f2 – when validate(client) generate an error message

f3 – when no error is generated

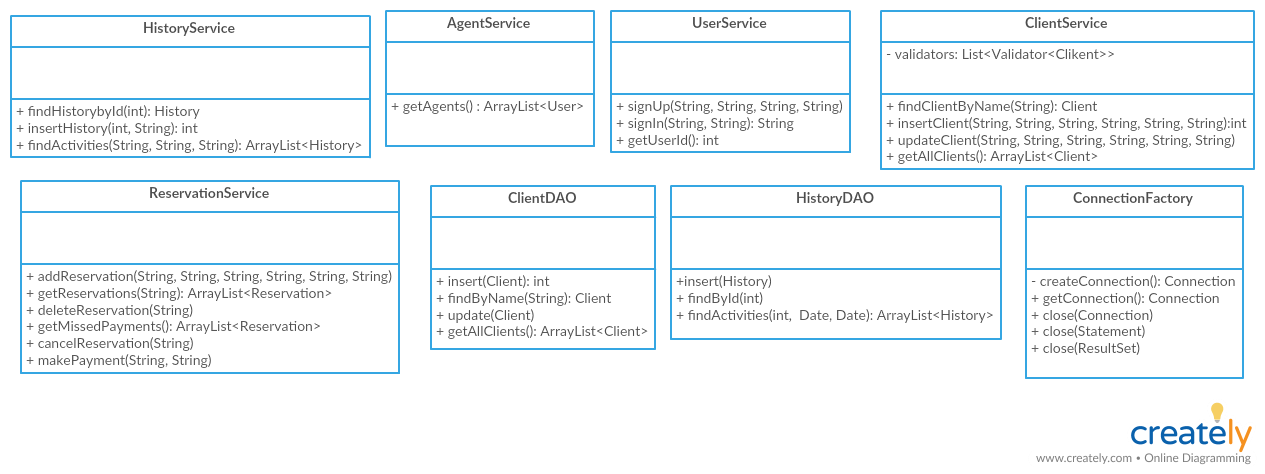
# 5. Class Design

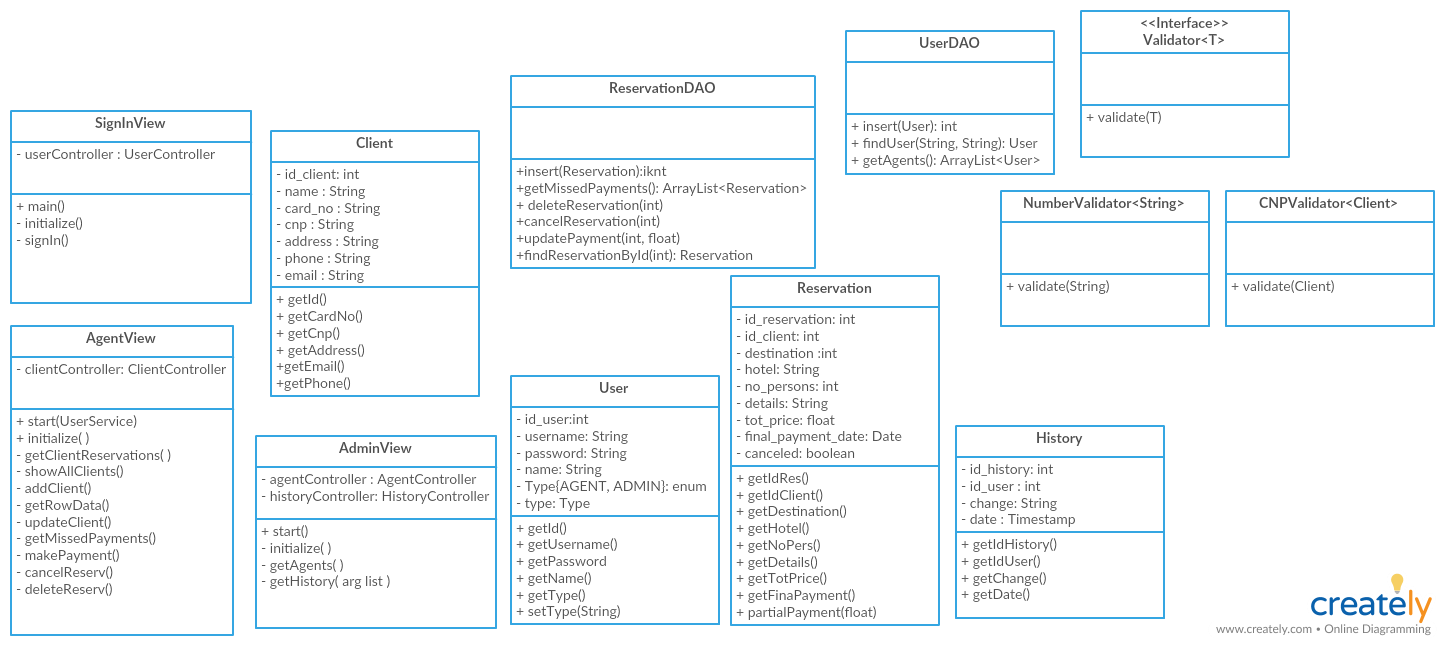
## 5.1 Design Patterns Description

Domain model is used as a domain logic pattern. There is a class to represent a table from the data base, which contains attributes, setters, getters and domain logic methods. Though most of the classes have just setters and getters this pattern has been chosen so further features can be added later.

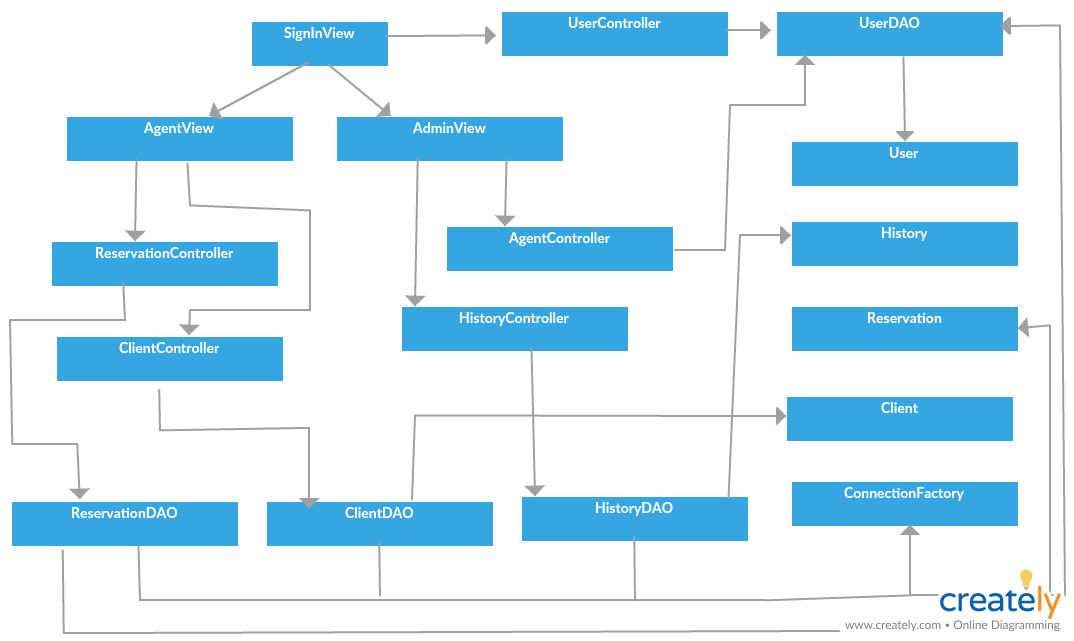
As data source pure pattern, the table data gateaway is used. ClientDAO, UserDAO, HistoryDAO and ReservationDAO represent the Gateaway classes which contain methods for SQL operations, each one for a specific table. StudentMapper class is represented by Services classes, having no QSL query, just methods for processing the data. Aside from this clasess, validator classes are used by the services to be able to validate the data befor making any oprations on it.

## 5.2 UML Class Diagram





# 6. Data Model



# 7. System Testing

The strategy used for testing was unit testing, using Junit Test. There is a Junit Test Case for each table.

Also testing was made directly on the application, verifying the validations for each field.

# 8. Bibliography

<https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

<http://richard.jp.leguen.ca/tutoring/soen343-f2010/tutorials/implementing-table-data-gateway/>

