Tourism Agency
Analysis and Design Document

Student: Bumbuc Ioana
Group: 30431

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 6

4. UML Sequence Diagrams 9

5. Class Design 9

6. Data Model 10

7. System Testing 11

8. Bibliography 12

1. Requirements Analysis

1.1Assignment Specification

The Travel Agency Manager System is a client-server application designed to manage the activity of a tourism agency. The system is primarily used by agency employees responsible for managing and booking vacations for clients. The system allows employees to add, modify, and delete client information as well as reserve vacations for clients.

Vacations can be reserved in the country and abroad, with three main types of vacations available: cruises, tours, and stays. Within a stay, clients can choose to go on one or more sightseeing trips. The system stores information about clients and vacations in a database, which is updated periodically according to information provided by operators that collaborate with the agency in XML files. The system should be able to read and validate these files to ensure the accuracy of the information stored in the database. Additionally, the system should provide user authentication and access control to ensure the security of the system.

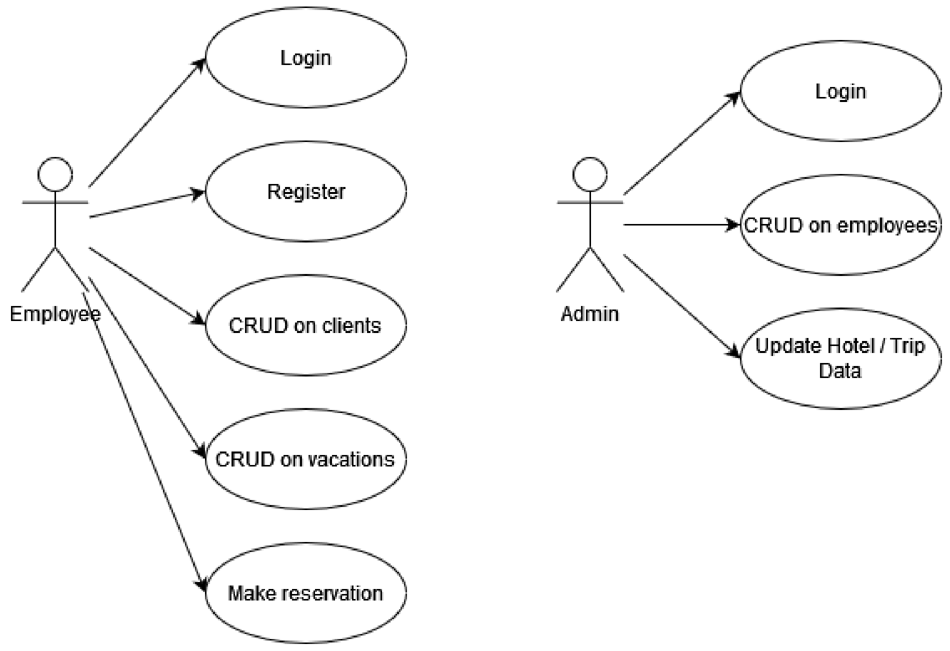
1.2Functional Requirements

- The application should allow the administrator to create, retrieve, update and delete employees.
- The application should allow employees to perform CRUD operations on vacations, clients, reservations and attractions.
- The application should allow both types of users (admin and employee) to login.

1.3Non-functional Requirements

- The application should be secure and protect sensitive information, such as passwords.
- Scalability: The application should be scalable to handle increasing amounts of data and users without sacrificing performance.
- Reliability: The application should be reliable and minimize the occurrence of errors, crashes, or data loss.

2. Use-Case Model



Use Case: Add a new vacation.

Level: User-goal level
Primary actor: Tourism agency employee
Main success scenario:

- The employee selects the "Add Vacation" option from the main menu.
- The system prompts the employee to enter the vacation details, including type, location, price, and availability.
- The employee enters the details and saves the vacation.
- The system confirms that the vacation has been added successfully.
- Extensions:
    - The vacation details are incomplete or invalid: The system prompts the employee to enter valid data.
    - The vacation conflicts with an existing vacation: The system prompts the employee to choose an alternative date or location.

Use Case: Make a reservation.

Level: User-goal level
Primary actor: Tourism agency employee
Main success scenario:

- The employee selects the "Reserve Vacation" option from the main menu.
- The system prompts the employee to select the client and the vacation.
- The employee selects the client and vacation and enters the reservation details, including dates, price, and payment information.
- The system confirms that the reservation has been made successfully.
- Extensions:
    - The client or vacation is not found in the system: The system prompts the employee to enter valid data.
    - The vacation is no longer available: The system prompts the employee to choose an alternative vacation.

Use case: Add a new employee.

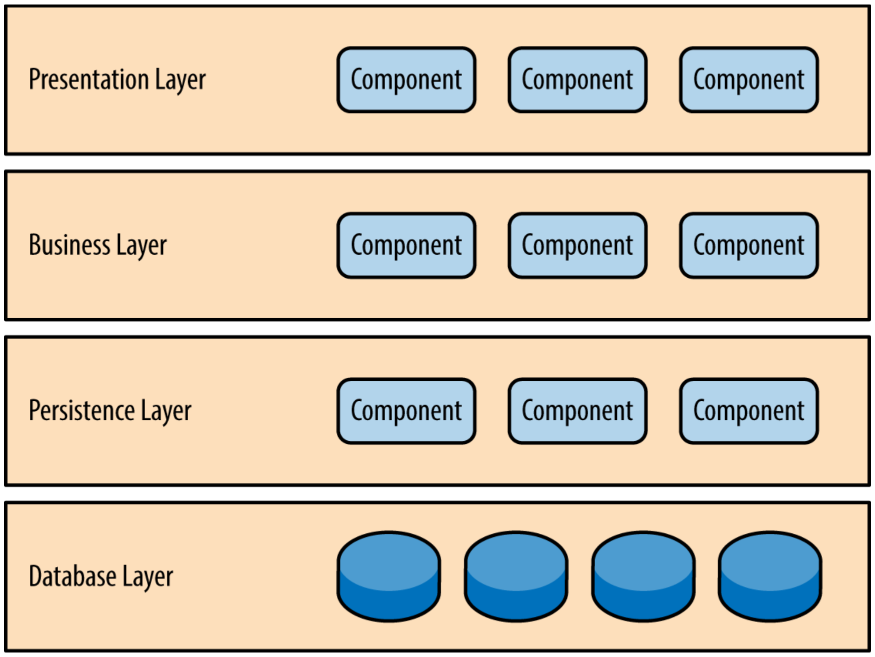
Level: Sub-function level
Primary actor: Admin
Main success scenario:

- The admin selects the "Add Employee" option.
- The system prompts the admin to enter the employee's personal information, including name, address, email, phone number.
- The admin enters the required information and saves it.
- The system confirms that the client has been added to the database.
- The employee can now view, modify or delete the client's information.
- Extensions:
    - If the employee tries to add a client with missing or invalid information, the system will prompt the employee to correct the errors before saving the information.
    - If the employee tries to add a client with a duplicate ID, the system will prompt the employee to either modify the existing client information or create a new ID for the new client.

3. System Architectural Design

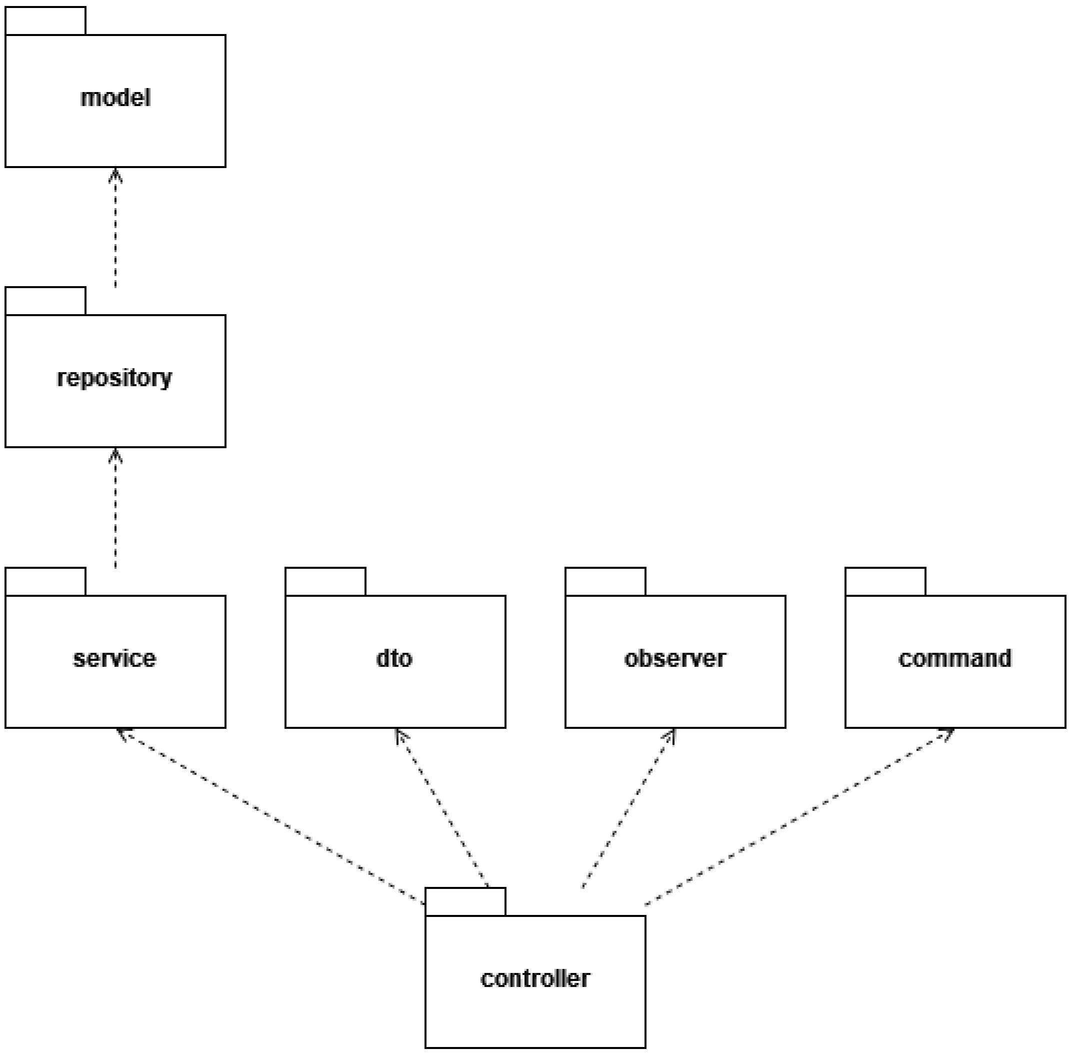
3.1 Layered Architectural Pattern

Layered architecture is a software architecture pattern that separates the concerns of a system into distinct logical layers, with each layer providing a well-defined set of services to the layer above it. The layers are organized in a hierarchical manner, with the lowest layer providing foundational services to the layers above it. This separation of concerns helps to promote modularity, maintainability, and scalability in large software systems. Typically, the layers in a layered architecture include the presentation layer, the application layer, the business logic layer, and the data access layer. Each layer has its own responsibilities and communicates with the layer above or below it through well-defined interfaces.

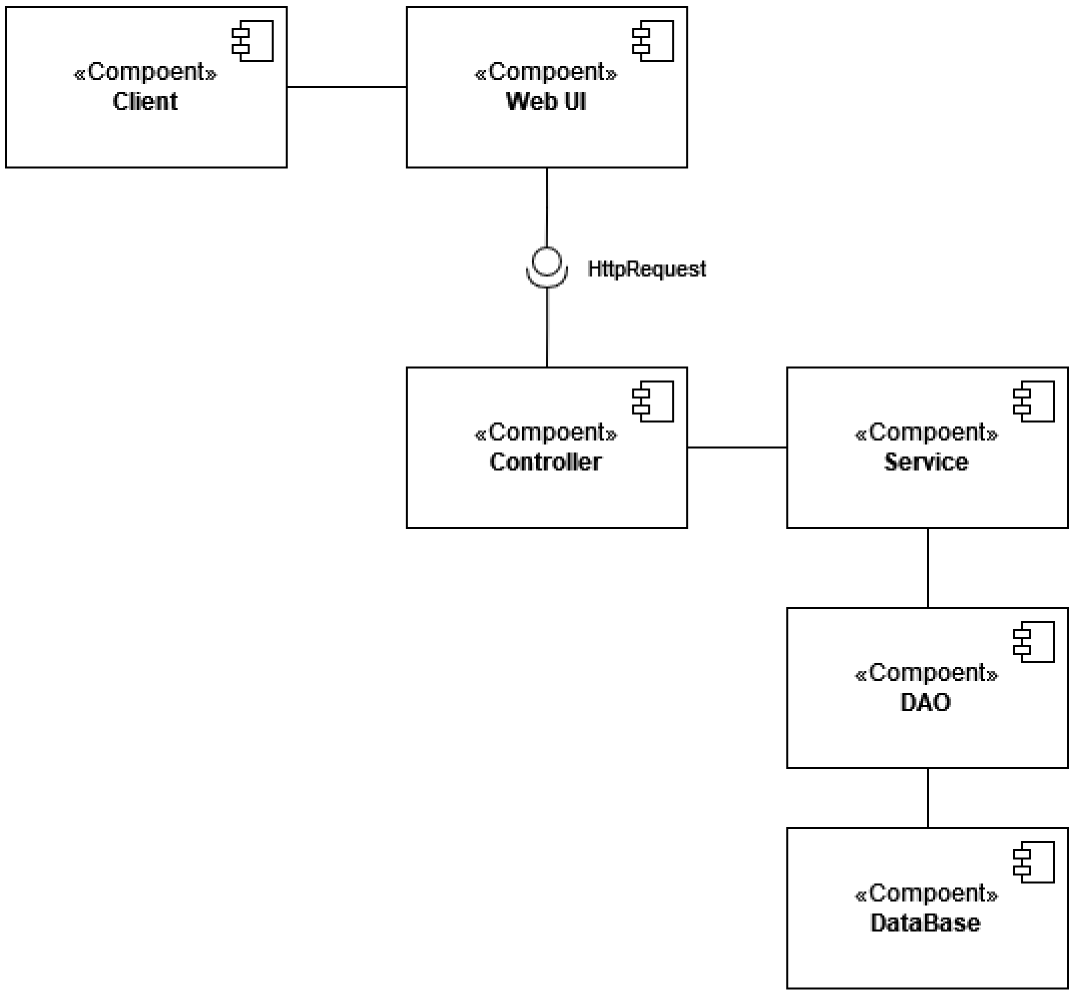


3.2 Diagrams

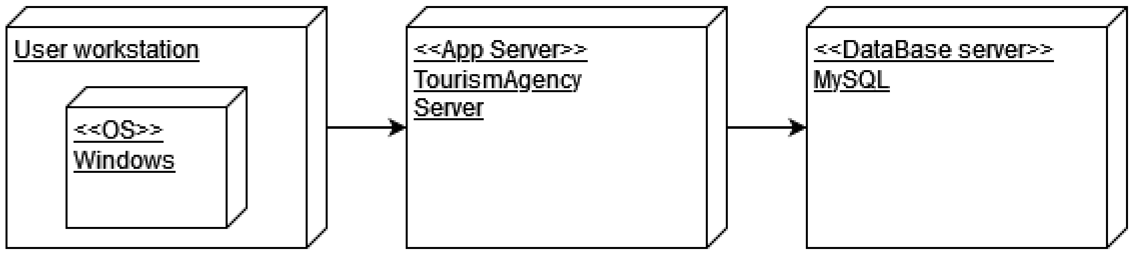
- Package diagram



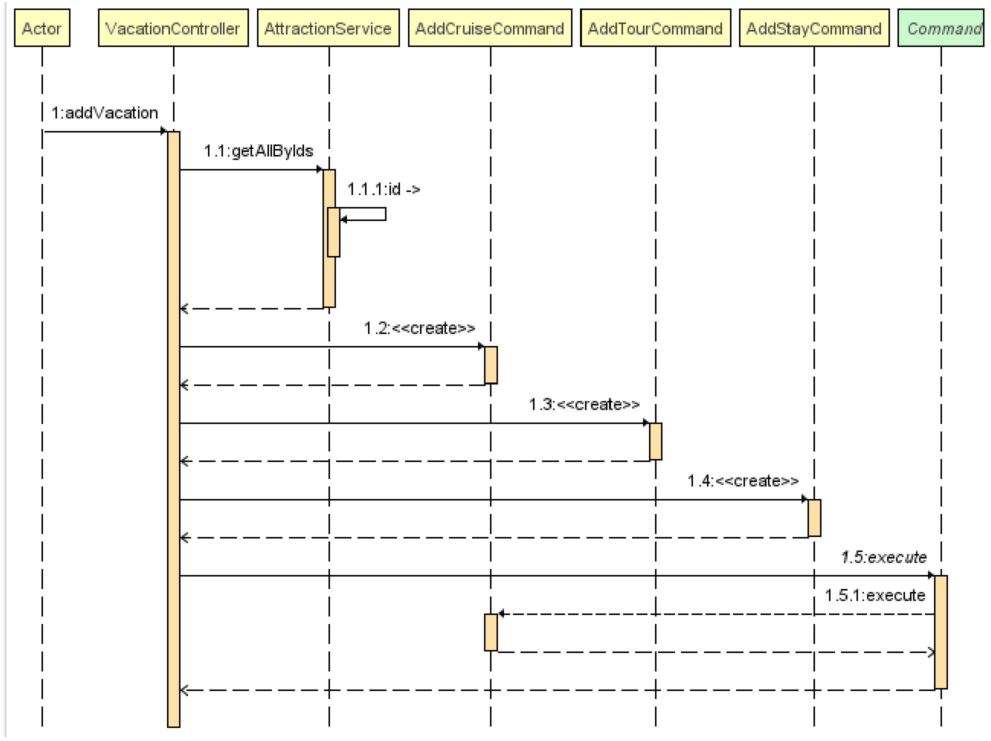
- Component diagram



- Deployment diagram


User workstation
<<OS>> Windows
<<App Server>> TourismAgency Server
<<DataBase server>> MySQL

4. UML Sequence Diagrams

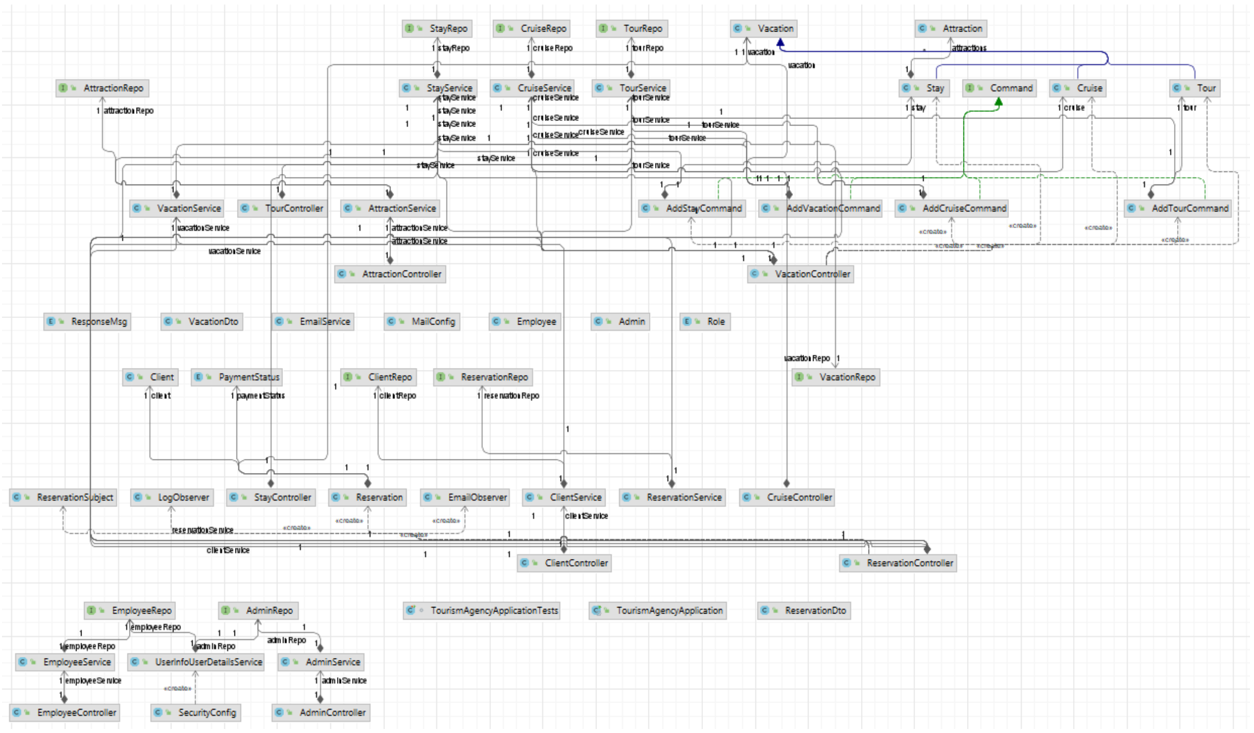


5. Class Design

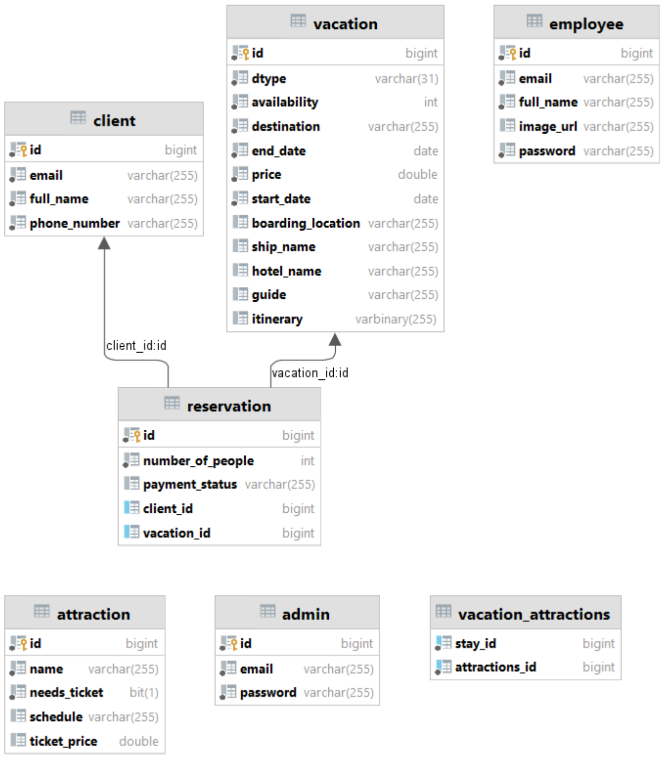
5.1 Observer Design Pattern Description

The Observer design pattern is a behavioral design pattern that allows objects to notify other objects when a change occurs in their state. In this pattern, one object, known as the subject or observable, maintains a list of its dependents, known as observers, and notifies them automatically of any state changes, usually by calling one of their methods. This allows for a loosely coupled design, where the subject and observer can vary independently, and enables multiple objects to be notified of changes to the same state. The Observer pattern is widely used in software development to implement event handling, message passing, and other reactive systems.

5.2 UML Class Diagram



6. Data Model



vacation

id	bigint
dtype	varchar(31)
availability	int
destination	varchar(255)
end_date	date
price	double
start_date	date
boarding_location	varchar(255)
ship_name	varchar(255)
hotel_name	varchar(255)
guide	varchar(255)
itinerary	varbinary(255)

employee

id	bigint
email	varchar(255)
full_name	varchar(255)
image_url	varchar(255)
password	varchar(255)

client

id	bigint
email	varchar(255)
full_name	varchar(255)
phone_number	varchar(255)

reservation

id	bigint
number_of_people	int
payment_status	varchar(255)
client_id	bigint
vacation_id	bigint

attraction

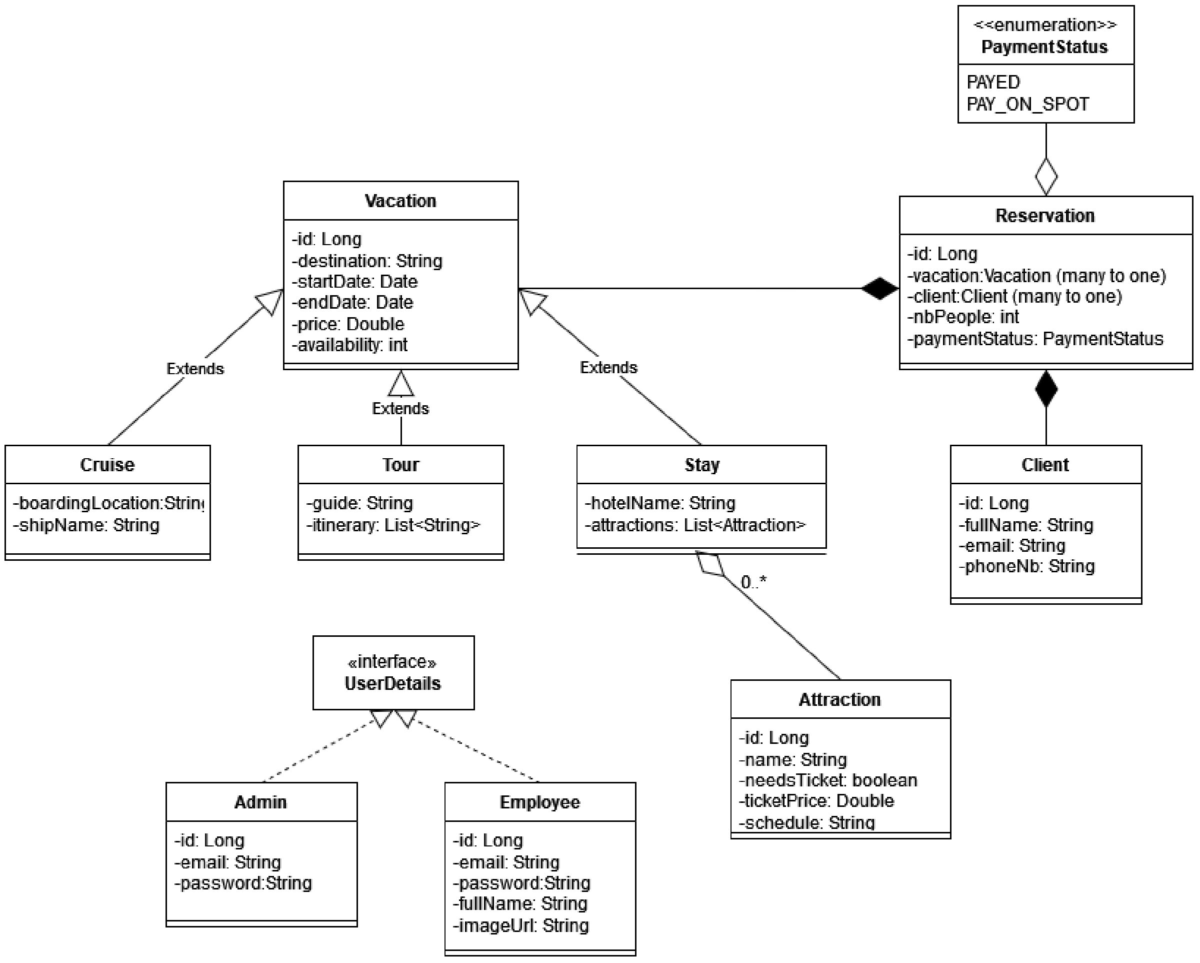
id	bigint
name	varchar(255)
needs_ticket	bit(1)
schedule	varchar(255)
ticket_price	double

admin

id	bigint
email	varchar(255)
password	varchar(255)

vacation_attractions

stay_id	bigint
attractions_id	bigint



7. System Testing

I wrote tests using Junit and Mockit. I used them to create mock versions of the service and repository classes, in order to not fill the database with a bunch of testing values.

I tested the methods for adding an employee and for adding a vacation. When adding an employee, I tested that the saved employee's password matches the encrypted original password.

I also tested the method addVacation in the VacationController class. That specific method reads from a "dtype" field the type of Vacation to be added (Tour, Cruise, Stay).

8. Bibliography

- https://www.section.io/engineering-education/mocking-with-junit-and-mockito-the-why-and-how/
- https://www.youtube.com/watch?v=R76S0tfv36w