Travel Agency Manager System
Analysis and Design Document

Student: Bumbuc Ioana
Group: 30431

Revision History

Date	Version	Description	Author
11/04/2023	1.0		Bumbuc Ioana

Table of Contents

I. Project 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.

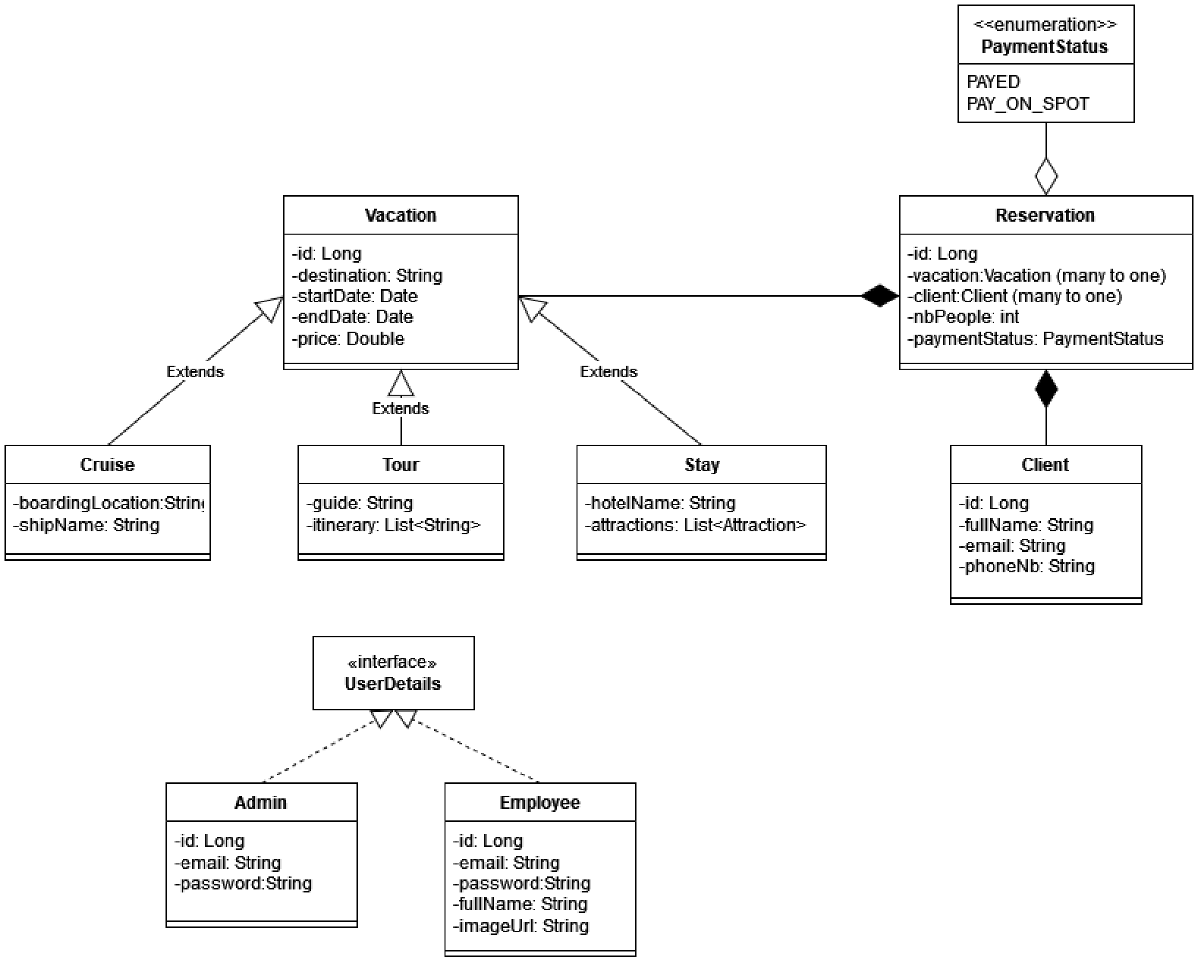
II. Elaboration – Iteration 1.1

1. Domain Model

The domain model consists of the various entities, attributes, relationships, and behavior within a specific domain or subject area. It represents the objects, concepts, and relationships within the domain.

The Travel Agency Manager has the following entities:

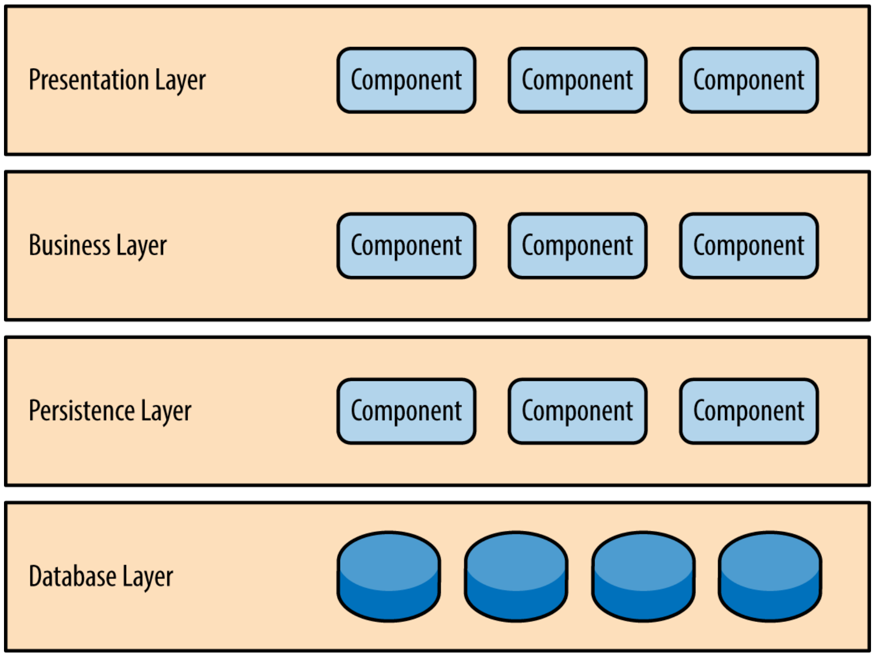
- Vacation – Represents a vacation that can be reserved, with attributes such as id, destination, start date, end date, price and availability. It can be of 3 types: Cruise, Tour and Stay.
- Cruise is a type of vacation. It has the attributes ship name and boarding location.
- Tour is another type of vacation. It has a tour guide and an itinerary, which is a list of locations to visit.
- The last type of vacation is the Stay. Its attributes are hotel name and a list of Attractions.
- Attraction – represents a touristic attraction that a client can book in the stay. It has the attributes id, name, needs ticket (true or false), ticket price (optional), schedule (also optional).
- Client – has as attributes the clients' data: id, full name, email, phone number.
- Reservation – links clients with vacations. The attributes are: id, vacation, client, number of people, payment status.
- Admin – type of user, has id, email and password.
- Employee – the main user of the application, has id, email, password, full name and profile picture URL.



2. Architectural Design

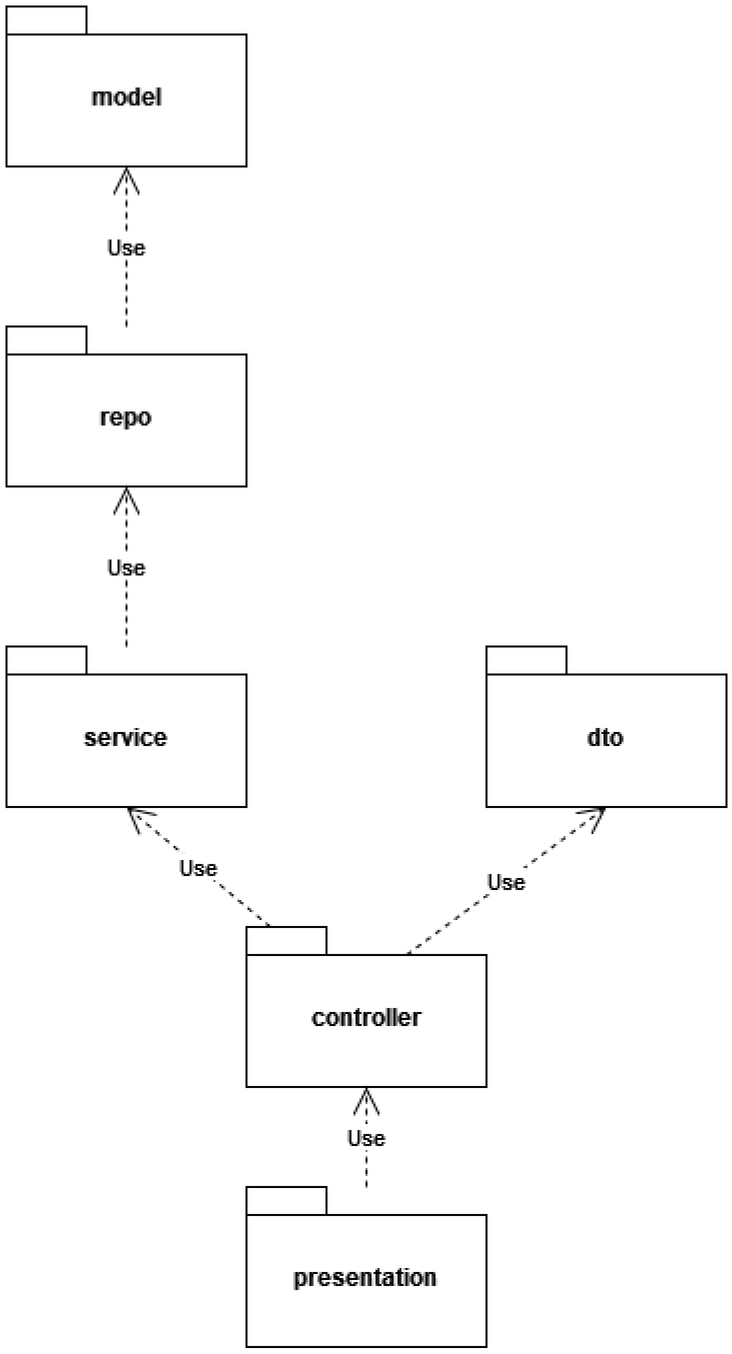
2.1 Conceptual Architecture

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.

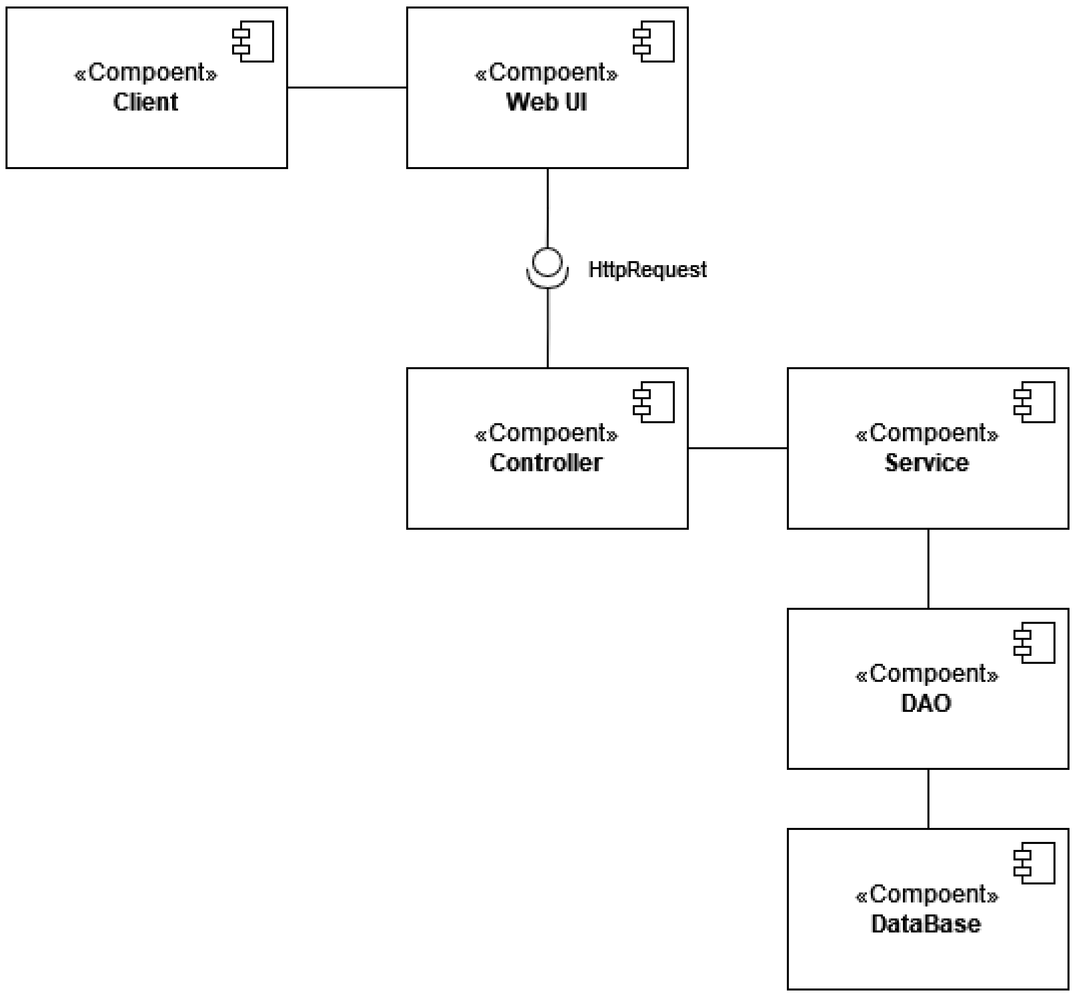


2.2 Package Design

The packages that will be used for implementing the application are named after the layers they represent. The packages will depend on one another only between consecutive layers. The most stable package is the model; all the others ultimately depend on it. Nothing depends on the presentation layer, because it is the one most likely to change, based on possible improvements. The package diagram is presented below.

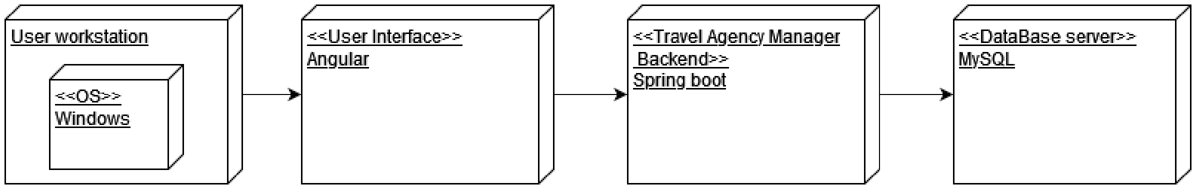

model
Use
repo
Use
service
dto
Use
Use
controller
Use
presentation

2.3 Component and Deployment Diagrams



The components are:

- The client, which interacts with the web app,
- The user interface, which needs to create http requests,
- The controller, which accepts and responds to the http requests,
- The service, that performs the necessary business logic,
- The Data Access Objects that are the medium between the data base and the program
- And finally, the data base itself.



The deployment diagram shows how the application will be running at very surface level. There is the user workstation, which is a computer running with an operating system, specifically windows. On the working station we see the Front End, which will be implemented using Angular. The back end, or the server, is the Travel Agency Manager application, implemented using Spring Boot. It needs the data base to store the data, that the back end retrieves, applies computations to, then sends it tot the front end via http requests. The database will run using MySQL.

III. Elaboration – Iteration 1.2

1. Design Model

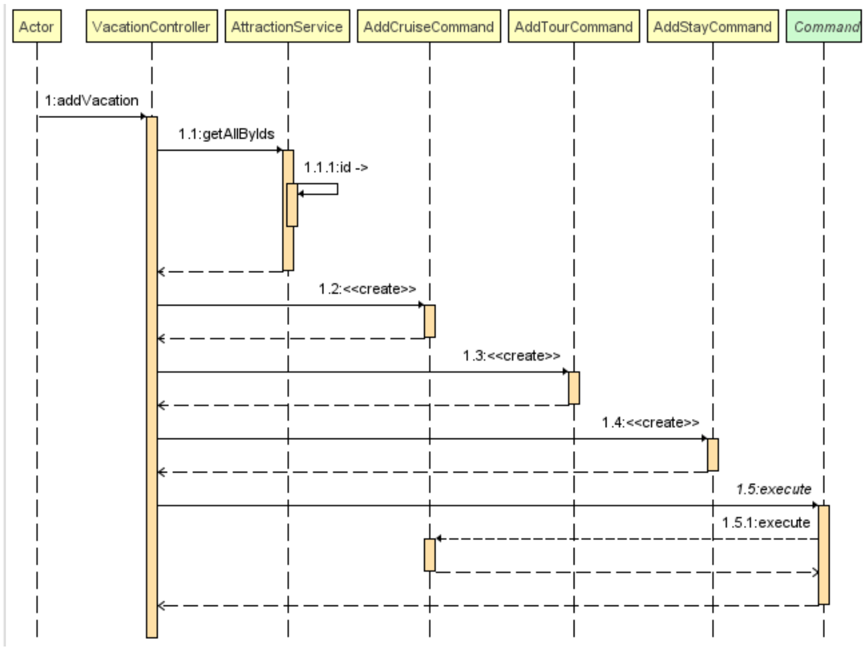
1.1 Dynamic Behavior

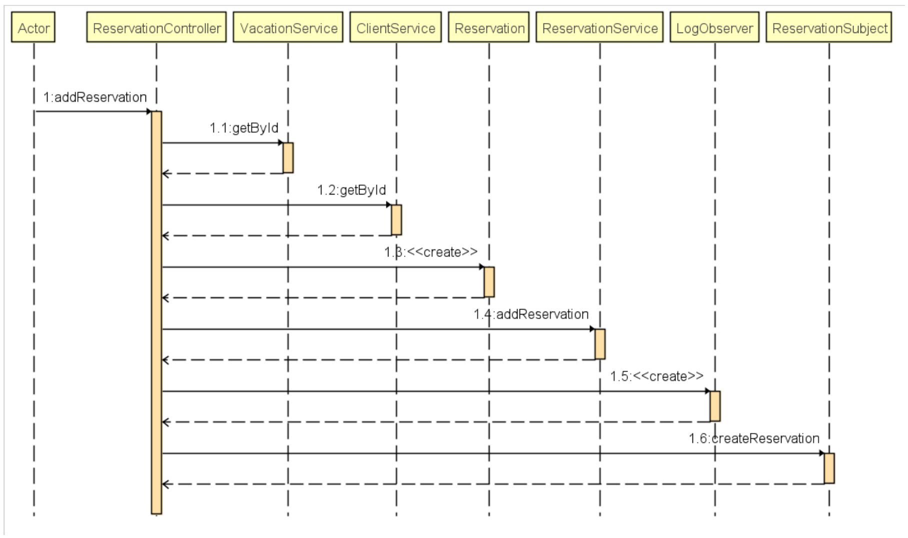
Adding a vacation

When the employee adds a new vacation, the controller identifies the specific command it needs to create. The 3 cases are seen in the diagram below. The employee enters in the "dtype" field the specific vacation to create: CRUISE, TOUR or STAY. Then, the Vacation Controller creates one instance of the corresponding command, cruiseCommand, tourCommand, stayCommand, which all extend the Command interface. Then, the execute method is called on the specific command.

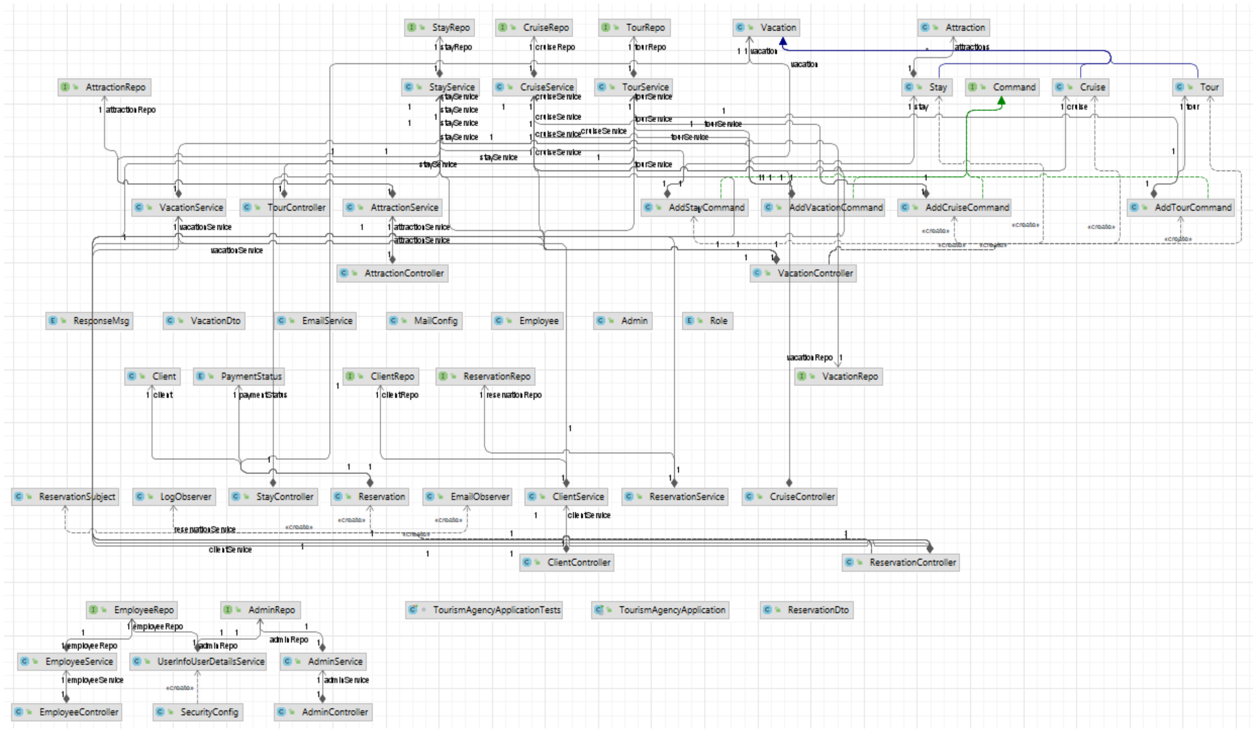
Creating a reservation

The employee can make a reservation by providing to the Reservation Controller the id of the vacation and the id of the client, as well as other relevant information. The Reservation Controller then

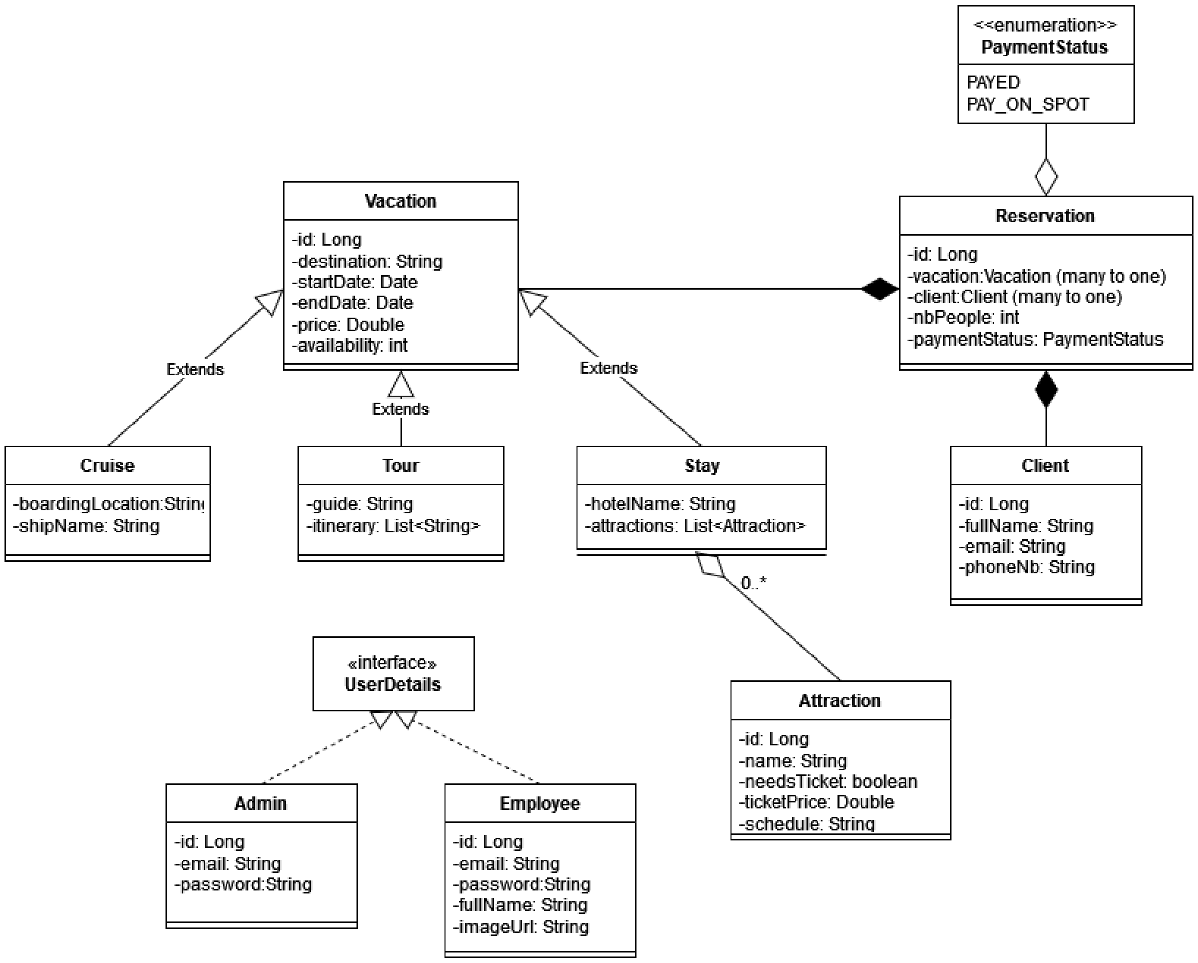

Actor
VacationController
AttractionService
AddCruiseCommand
AddTourCommand
AddStayCommand
Command
1:addVacation
1.1:getAllByIds
1.1.1:id ->
1.2:<<create>>
1.3:<<create>>
1.4:<<create>>
1.5:execute
1.5.1:execute


Actor
ReservationController
VacationService
ClientService
Reservation
ReservationService
LogObserver
ReservationSubject
1:addReservation
1.1:getById
1.2:getById
1.3:<<create>>
1.4:addReservation
1.5:<<create>>
1.6:createReservation

1.2 Class Design



2. Data Model



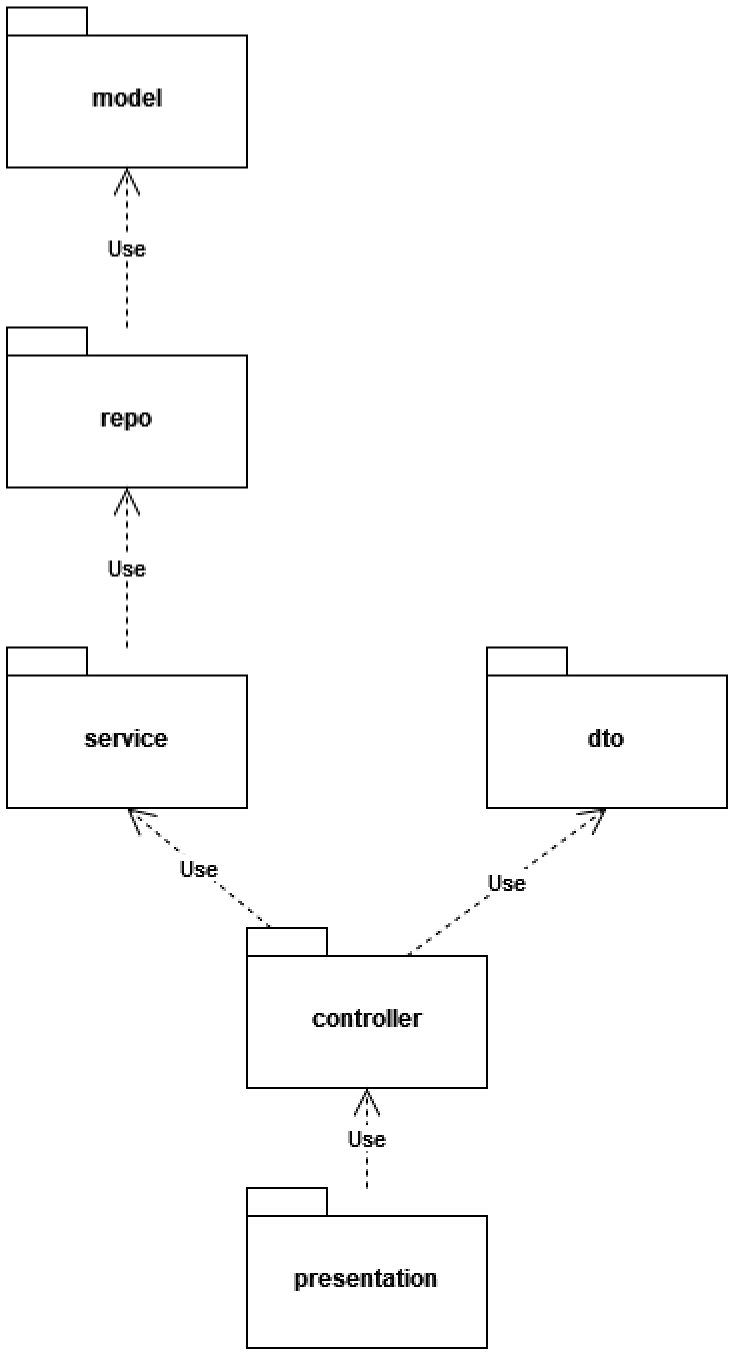
3. Unit Testing

In this project, unit testing was performed to ensure the correctness and robustness of the code. The tests were written using the JUnit framework and Mockito library for mocking dependencies. The following tools were used for testing:

- JUnit: provides annotations and assertions for writing and running tests.
- Mockito: Mockito is a mocking framework that allows the creation of mock objects for testing. It helps in isolating the code under test by providing mock implementations of dependencies.

IV. Elaboration – Iteration 2

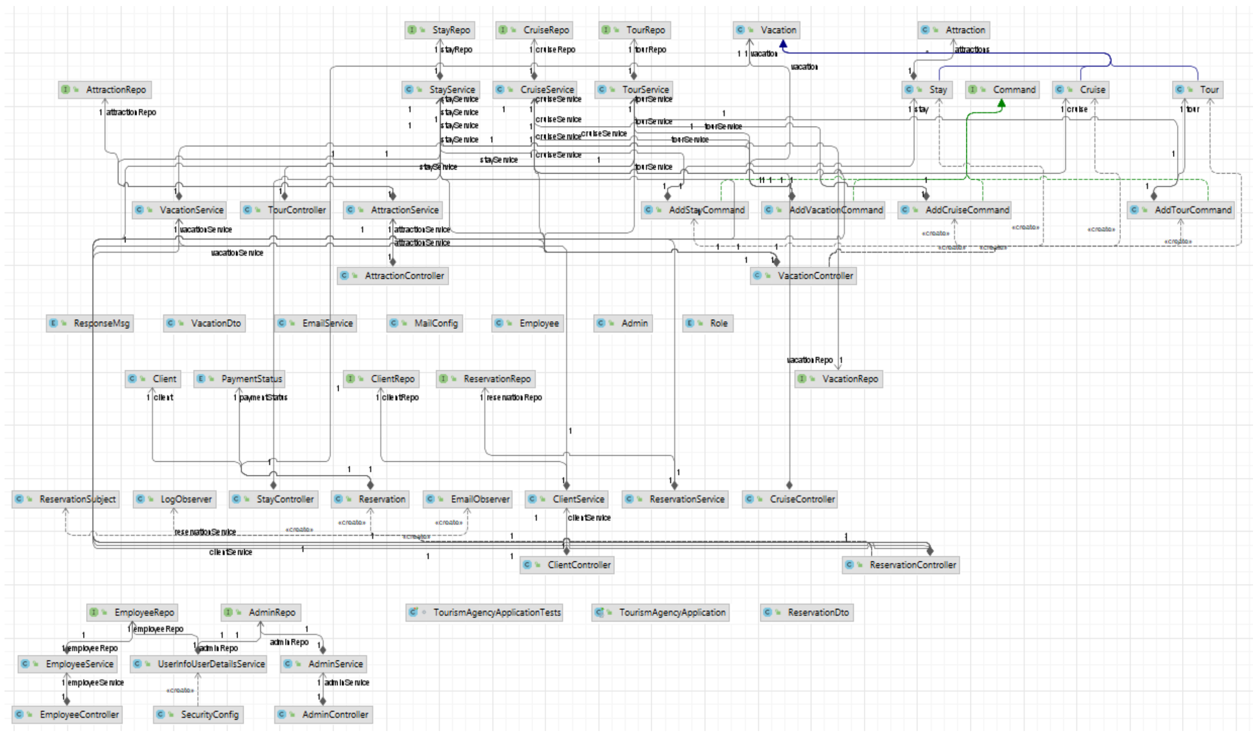
1. Architectural Design Refinement



The packages I used for the project are:

- Model, containing the entities: Admin, Attraction, Client, Cruise, Employee, Reservation, Stay, Tour, Vacation. It also contains enumeration for Payment Status (Paid, Pay on spot, Pending).
- Repo is the package that contains repository interfaces, that extend the JpaRepository framework. It is responsible for creating the queries necessary for CRUD operations on entities.
- Service package contains services for each entity. It calls the methods from the repo and its responsible for all business logic.
- Controller package is the one responsible for the interaction between the model and the presentation. It defines for each entity the endpoints that the GUI needs.
- DTO package contains the Data Transfer Object classes. They are needed for passing data between front end and back end.

2. Design Model Refinement



The project's UML class diagram is presented above. It contains all the classes from each package and the relationships between them.

V. Construction and Transition

1. System Testing

5.1.EmployeeControllerTest:

The EmployeeControllerTest class contains tests for the EmployeeController class. It verifies the behavior of the "addEmployee" method in different scenarios.

Test Cases:

- testAddEmployee: This test case verifies the behavior of the "addEmployee" method when a new employee is successfully added. It sets up the necessary objects and mocks the dependencies. The test ensures that the method returns the expected response status code (HttpStatus.CREATED) and the new employee object.

- testAddEmployee (negative case): This test case verifies the behavior of the "addEmployee" method when an employee with an existing email is added. It sets up the necessary objects and mocks the dependencies. The test ensures that the method returns the expected response status code (HttpStatus.BAD_REQUEST) and a response message indicating that the email already exists.

5.2.ReservationControllerTest:

The ReservationControllerTest class contains tests for the ReservationController class. It verifies the behavior of the "addReservation" method.

Test Cases:

- testAddReservation: This test case verifies the behavior of the "addReservation" method when a new reservation is successfully added. It sets up the necessary objects and mocks the dependencies. The test ensures that the method returns the expected response status code (HttpStatus.CREATED).

5.3.VacationControllerTest:

The VacationControllerTest class contains tests for the VacationController class. It verifies the behavior of the "addVacation" method in different scenarios.

Test Cases:

- testAddVacationCruise: This test case verifies the behavior of the "addVacation" method when a new cruise vacation is added. It sets up the necessary objects and mocks the dependencies. The test ensures that the method returns the expected response status code (HttpStatus.CREATED) and the cruise object. It also verifies that the "addCruise" method of the cruiseService is called exactly once.

- testAddVacationTour: This test case verifies the behavior of the "addVacation" method when a new tour vacation is added. It sets up the necessary objects and mocks the dependencies. The test ensures that the method returns the expected response status code (HttpStatus.CREATED) and the tour object. It also verifies that the "addTour" method of the tourService is called exactly once.

- testAddVacationStay: This test case verifies the behavior of the "addVacation" method when a new stay vacation is added. It sets up the necessary objects and mocks the dependencies. The test ensures that the method returns the expected response status code (HttpStatus.CREATED) and the stay object. It also verifies that the "getAllByIds" method of the attractionService and the "addStay" method of the stayService are called exactly once.

2. Future improvements

There are several potential areas for further improvements in the Travel Agency Manager System, like:

- Enhanced Security Measures: Strengthening the system's security measures is crucial to protect sensitive client data and ensure the integrity of the system. This can involve implementing additional authentication mechanisms, such as Spring Security and two-factor authentication.

- Mobile Application: Developing a mobile application can provide employees with the flexibility to access and manage the system while on the go. A mobile app can offer features such as push notifications for booking updates, offline access to essential information, and a user-friendly interface optimized for mobile devices.

- Analytics and Reporting: Implementing analytics and reporting capabilities can provide valuable insights into the system's usage, client preferences, and booking trends. This information can assist in making data-driven decisions, optimizing marketing strategies, and identifying areas for improvement or expansion.

VI. Bibliography

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