<Travel Agency>

Student:Deac Denisa

**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 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The Travel Agency Manager is a client-server application designed to manage the activity in a tourism agency. The application is used by the agency’s employees and allows them to add/modify/delete vacations, add/modify client information, and reserve vacations for clients. The vacations can be reserved in the country and abroad, and there are three types of vacations that can be reserved: cruises, tours, and stays. Within a stay, a client can choose to go on one or more sightseeing trips. The information about clients and vacations is stored in a database.

# Functional Requirements

The Travel Agency Employee should provide the following functionalities:

* Add, modify, and delete vacations.
* Add, modify, and delete client information.
* Make reservations for clients
* Send emails when a reservation is made.
* Create a receipt when a reservation is made.

The Administrator can:

* Add/modify/delete employee information.

The Client can:

* View vacations.

# Non-Functional Requirements

The non- functional requirements are the following:

* The application is responsive and performs tasks quickly and efficiently.
* The system provides clear and informative error messages for incorrect inputs or other errors.
* The application provides secure and encrypt passwords in the database with a one-way encryption algorithm.
* The application follows the Layers architectural pattern, is well-organized and maintainable.
* The application uses Observer design pattern and Strategy design pattern for a better reusability of the code and stronger OOP principles.
* The database schema respects 1st, 2nd, and 3rd normal forms.

2. Use-Case Model

The use-case model describes the system's functionality from the perspective of its users. The following is a use-case diagram for the Travel Agency Manager system:

Diagram

Description automatically generated

Use-case description:

Use case: Add Vacation

Level: User-goal level

Primary actor: Employee

Main success scenario:

* Employee sends request for "Add Vacation".
* Employee enters the vacation details (e.g. location, type, price).
* System validates the credentials of the employee
* System adds the vacation to the database.
* System confirms the addition of the vacation.

Extensions:

* Invalid vacation details are entered – error message.
* Credentials are wrong – error message.

Use case: Delete Vacation

Level: User-goal level

Primary actor: Employee

Main success scenario:

* + Employee sends request for "Delete Vacation" option.
  + Employee selects the vacation id to be deleted.
* System validates the credentials of the employee
  + System checks if the vacation exists then deletes it from the database.
  + System confirms the deletion of the vacation.

Extensions:

* 1. No vacations with the given id – error message.
  2. System fails – error message.
  3. Credentials are wrong – error message.

3. System Architectural Design

**3.1 Architectural Pattern Description**

The Presentation, Business, Model and Persistence layers make up the three layers that commonly make up the Layered pattern. The user interface and user interaction are handled by the Presentation layer, identified by the Controller package. Business logic and data processing are handled by the business and uses service classes. The Model layer contains all the entities needed for the application, and the persistence layer, represented by the Repository package, is responsible for interacting with the database through queries.

**3.2 Diagrams**

Diagram

Description automatically generated

**Package diagram:**

Diagram

Description automatically generated

**The business** package contains another 3 packages for a better oranization:

* Service: with all the service classes
* Notification: for the classes related to sending notification using observer pattern
* Strategy: with the classes related to the strategy design pattern.

**Model**: This package contains all the entity classes that represent the data model of your application.

**Controller**: This package contains the controllers for the student and teacher entities. The controllers are responsible for receiving HTTP requests from clients, processing the requests, and returning responses to the clients.

**Repository**: This package contains one repository for each entity in the model. A repository is responsible for handling data access for a specific entity.

## Component Diagram

*[Create the component and deployment diagrams.]*

The component diagram shows the different components that make up the system and their relationships, while the deployment diagram shows how these components are deployed on hardware or software infrastructure. In this case, the components include the Presentation layer, Business Logic layer, Data Access layer, and Database. The deployment diagram shows how these components are deployed on servers and clients, with the Presentation layer deployed on the client side and the other layers deployed on the server side.

Diagram

Description automatically generated

**Deployment Diagram**

A deployment diagram shows the physical architecture of a system and how the software components are deployed on hardware.

A picture containing diagram

Description automatically generated

4. UML Sequence Diagrams

*[Create a sequence diagram for a relevant scenario.]*

***SEQUENCE DIAGRAM*** *–* scenario for adding a new vacation of type Cruise by illustrating the Strategy design pattern used.

*Diagram

Description automatically generated*

Scenario for creating a new client with encrypted password:

*Diagram

Description automatically generated*

5. Class Design

The classes I design follow the architectural pattern in terms of separation into packages and their purpose.

The classes that do the logical manipulation of data are represented by the **Controllers**, one for teacher and one for student, where the services, methods specific for each use case, are called. The interaction with the user is done via htpp requests, through postman. Each method has a specific path with the necessary annotation for posting, getting, putting, or deleting.

The service classes are the ones doing the computations and the logical part of the CRUD operations. There is a service class for each entity with specific methods for them. Also, the application provides the implementation needed for a proper observer design pattern and strategy design patter. The repositories are instantiated and used here to retrieve the data from the database. They represent the business logic of the application.

The repositories are the persistence part of the project. I created a repository class for each entity, since these are interfaces that extend JBA Repository, making the connection with database through queries. Some basic query functions are already implemented, but for some entities I made a customized method.

**5.1 Design Patterns Description**

I use 2 design patterns: Observer and Strategy. The observer is used to send notifications to create a receipt and send an email when a reservation.

In this pattern, the notification system is modeled as the subject, while the log file writer and email sender is modeled as the observers. The subject maintains a list of its observers and notifies them automatically of any changes in its state.

When a reservation is made, the subject, being the ReservationService class, will notify the observers, which will then perform their respective tasks. The log file writer will write the details of the reservation to a log file, while the email sender will create an email containing the reservation details and send it to the client.

Using the Observer pattern helps decouple the notification system from the reservation-making process, making it easier to add new observers in the future without modifying the subject. It also allows multiple observers to be notified of the reservation, providing a flexible and extensible solution.

The Strategy design pattern is a behavioral pattern that allows you to define a family of algorithms, encapsulate each one, and make them interchangeable. The pattern lets the algorithm vary independently from clients that use it.

In my project, I have implemented the Strategy pattern to simplify the CRUD operations for different types of vacations, such as tours, cruises, and stays. By defining a common interface (VacationStrategy) for all vacation services and implementing separate concrete classes for each type of vacation (TourService, CruiseService, StayService), I could easily switch between different implementations of the same method without changing the client code.

For this approch I also used generic types for an easier manipulation of vacation types.

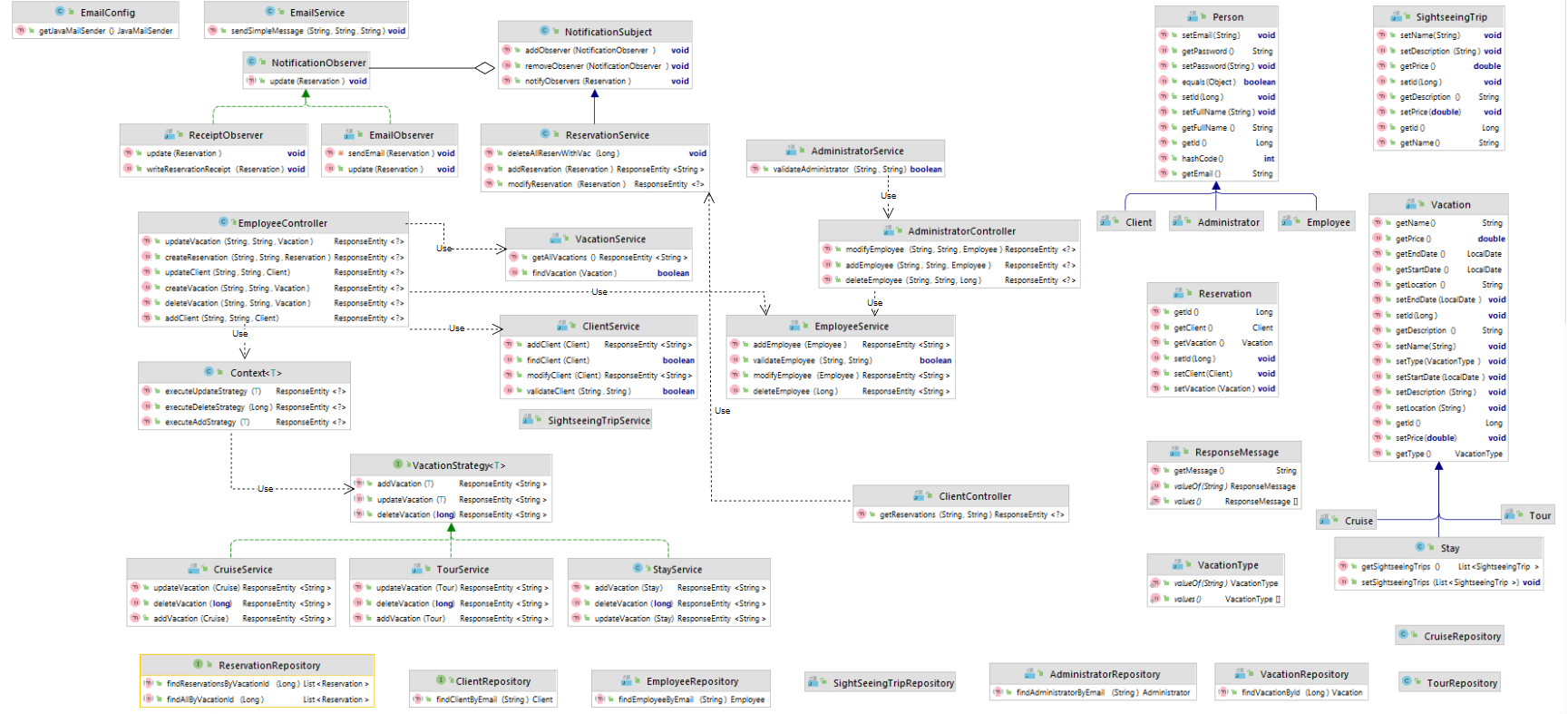
**5.2 UML Class Diagram**

*[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]*

The following class diagram represents the main dependencies between classes, along with the design patterns used. First, there is the observer pattern which is connected with the action of creating a reservation existent in ReservationService. This class is the subject of the notification, and it notifies the 2 observers, EmailObserver and ReceiptObserver. The first one sends an email to the client with the details of the reservation and the second generates a receipt in a txt format. The email is sent using JavaMailSender and all the needed configurations are set in the EmailConfig class.

The second design pattern used is the Strategy Design pattern. The VacationStrategy interface defines methods for adding, updating and deleting a vacation. The class has a generic type such that, when implemented, is easier to handle the vacation types.

The concrete strategies are represented by the service classes for the 3 types of vacations where the implementation of the methods are made. The context class handles the type and the calls for each strategy.

**

6. Data Model

*[Present the data models used in the system’s implementation.]*

The data model for this project consists of the following entities:

* Person: which can be of three types: Administrator, Employee and Client, all being users in the application and having stored their name and credentials.
* Vacation: which can be of three types: Cruise, Tour and Stay.
* Sightseeing\_trip: each Stay vacation has a list of sightseeing tips. This entity represents one of this trips and stores significant details.
* Vacation\_sightseeing\_trip: since a stay can have multiple sightseeing trips, there exists a one to many relationship between them. This relationship is illustrated in this table.
* Reservation: each client can book a vacation, representing the reservation entity having as foreign keys the id of the client and the id of a vacation.

Graphical user interface, application

Description automatically generated

7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

For testing the system, I used validation through postman application, but also 2 unit testing for 2 of the most complex cases. The code makes good use of response messages, all sored in an Enum class, used along with the correspondent Http response.

*Graphical user interface, text, application

Description automatically generated*

The test units are done for deleting a vacation and modifying a reservation.

Graphical user interface, application

Description automatically generated

8. Bibliography

* <https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm>
* https://www.tutorialspoint.com/design\_pattern/observer\_pattern.htm