

Diagram

Description automatically generatedAirline company

Name:Ioneasa Cristina-Ioana

Group: 30236

Table of Contents

[Deliverable 1 2](#_Toc2055786399)

[Project Specification 3](#_Toc825512728)

[Functional Requirements 3](#_Toc362225657)

[Use Case Model 1 3](#_Toc324720183)

[Use Cases Identification 3](#_Toc159950011)

[UML Use Case Diagrams 3](#_Toc1353450407)

[Supplementary Specification 3](#_Toc501087280)

[Non-functional Requirements 3](#_Toc1600632703)

[Design Constraints 3](#_Toc729746755)

[Glossary 3](#_Toc178374120)

[Deliverable 2 3](#_Toc1123232848)

[Domain Model 3](#_Toc605147785)

[Architectural Design 3](#_Toc1115440048)

[Conceptual Architecture 3](#_Toc2005513277)

[Package Design 3](#_Toc596515525)

[Component and Deployment Diagram 4](#_Toc1771682286)

[Deliverable 3 4](#_Toc174956409)

[Design Model 4](#_Toc2136237097)

[Dynamic Behavior 4](#_Toc853475638)

[Class Diagram 4](#_Toc2045288274)

[Data Model 4](#_Toc1322833382)

[System Testing 4](#_Toc975007147)

[Future Improvements 4](#_Toc902595900)

[Conclusion 4](#_Toc1374315000)

[Bibliography 4](#_Toc2082771450)

# Deliverable 1

## Project Specification

The main goals of the project are: working with the Spring framework, creating a link to a MySQL database and a graphical interface using react and creating specific functionalities of the project theme. I chose to developed and implemented a web-based application for an aviation company that caters to two distinct types of users:

* Administrators, who possess the ability to add flights and modify the price of tickets
* Customers, who can purchase flight tickets

The backend part will be made in Java, using the Spring framework and the frontend part will be made in React, as I said above.

## Functional Requirements

The functional requirements in my project are devided in 3 categories: autentification, admin operations and client operations.

The authentication features are: register, login, logout and change the password. The first one allows a new user to create an account by entering his data (it only requires a username and a password). In the second one, the login, the user enters his credentials, and if they are correct, he is redirected to a specific page (depending on the user's role: client or admin) where he can perform certain operations depending on his role. At the end, the user can log out of the site.

Client -> the client is able to query the database, searching for airline tickets by date, departure/landing airport. He can buy a flight ticket and has the option to cancel if he changes his mind. An invoice is automatically generated in the customer's name when the ticket is purchased.

Admin -> similar to the client, the administrator can perform CRUD operations, more specifically, inserting new flights on database or update the price.

## Use Case Model 1

### Use Cases Identification

Use-Case: Buy ticket

Level: Client Goal

Primary Actor: Client

Main success scenario:Buy ticket with success

Succes scenario for client:

1. The client login. If he doesn’t have an account, he need to register first, after that to login
2. The client search an flight suitable for his date and destination
3. The client buy the ticket
4. The app generate a bill for ticket
5. The client logout

Alternative sequence:

1.At step 1 the client don’t remember his password

2.The client update his password.

3. Go back to step 2 in the success scenario.

Use-Case: Add flight

Level: Admin Goal

Primary Actor: Admin

Main success scenario:Add flight with success

Succes scenario for admin:

1. The admin login. If he doesn’t have an account, he need to register first, after that to login
2. The admin add a new flight
3. The admin logout.

### Diagram Description automatically generatedUML Use Case Diagrams

## 

Diagram

Description automatically generated

Diagram

Description automatically generated

## Supplementary Specification

### Non-functional Requirements

In addition to functional requirements, there are also non-functional requirements such as usability, maintainability, security and performance.

To discover usability we need to ask the question "Can the user use this application easily or is it difficult for them to find what they need to do?". Thanks to the simplicity of the graphical interface, the customer can use the application very easily, everything is suggestive with clear indications.

Mentability defines the time it takes for a solution or component to be repaired or modified. Due to the efficient division of classes into packages according to functionality (model, repository, service, controller) problems can be identified and solved. Also, improvements to increase sustainability can be made very easily by modifying only certain components, which are independent of the rest of the project.

Security: The application should be designed with robust security measures to protect against unauthorized access, data breaches, and other security threats.

Performance: The application should be able to handle a high volume of requests and transactions while maintaining good response times and efficient resource utilization.

### Design Constraints

The project is built on the MVC architecture, where each level is divided into several layers. For Model I have the packages: model, repository, service, for view is the package that contains the graphical interfaces created with React, and the controller has the package controllers

The access and transport of data is linear, from the level of extraction from the database to the level of display in the frontend. The linking between the tables in the database has been done using ORM (One-To-Many, Many-To-One) relationships.

## Glossary

CRUD: Create, Read, Update and Delete are the four basic operations of persistent storage.

MVC (Model-View-Controller): A model in software design commonly used to implement user interfaces, data and control logic.

SQL (Structured Query Language): is a standardised programming language that is used to manage relational databases and perform various operations on data

# Deliverable 2

## Domain Model

A picture containing graphical user interface

Description automatically generated

Flight: Represents a flight that can be booked by a user. Contains the following properties:

id (Long): Unique identifier for the flight.

number (String): The flight number.

departure (String): The departure airport.

arrival (String): The arrival airport.

date\_time (LocalDateTime): The date and time of departure.

date\_time\_arrival (LocalDateTime): The date and time of arrival.

duration (float): The duration of the flight in hours.

nr\_seats (int): The number of available seats on the flight.

price (int): The price of a ticket for the flight.

ticket (List<Ticket>): A list of tickets booked for the flight.

Ticket: Represents a ticket that has been booked for a flight. Contains the following properties:

noTicket(Long): Unique identifier for the ticket.

passengerName (String): The name of the passenger who booked the ticket.

seat (String): Number of seat

flightToTicket (Flight): The flight for which the ticket was booked.

userToTicket(User): The user for which the ticket was booked

User: Represents a user of the application. Contains the following properties:

id (Long): Unique identifier for the user.

name (String): The name of the user.

password (String): The password of the user.

email (String): The email address of the user.

bookedTickets (List<Ticket>): A list of tickets booked by the user.

## Architectural Design

### Conceptual Architecture

I use MVC pattern which helps to achieve a separation of concerns in my application, where each layer has a distinct responsibility and can be developed and tested independently.

MODEL: The Model layer represents the data and business logic of the application. It consists of the domain objects and their relationships, which are represented as classes in model package. The domain objects include classes such as Flight, Ticket, User.

VIEW: The View layer represents the user interface of the application. It is responsible for displaying the data to the user and collecting user input. In Angular frontend, the View layer consists of the HTML templates and TypeScript components that render the UI.

CONTROLLER: The Controller layer acts as an intermediary between the Model and the View layers. It is responsible for handling user input and updating the Model layer accordingly. In Spring Boot backend, the Controller layer consists of the RESTful endpoints that receive HTTP requests from the frontend and invoke the appropriate Service methods to perform the necessary business logic. The Controller also returns the response back to the frontend in a format that the View can understand.

Backend packages:

Model package: This package contains classes that represent the domain objects. These classes are responsible for encapsulating the data and behavior of the application and provide a consistent representation of data across different layers.

Repository package: This package contains classes responsible for data access and interaction with the database. The repository is responsible for storing and retrieving data from the database.

Service package: This package contains classes responsible for business logic and processing data. The service layer interacts with the repository to retrieve and store data and performs any required data validation.

Controller package: This package contains classes responsible for handling incoming requests from the client and returning responses. The controller receives data from the client, delegates the request to the service layer, and returns a response to the client.

### Package Design

A picture containing chart

Description automatically generated

### Component and Deployment Diagram

Diagram

Description automatically generated

# Deliverable 3

## Design Model

### Dynamic Behavior

Scenario 2: User books a flight

1. User selects a flight from the list of available flights
2. User select a seat
3. System verify availability of seats.
4. System validates the input and books the flight
5. System displays the booking confirmation to the user

Sequance diagram

Diagram

Description automatically generated

### Class Diagram

Timeline

Description automatically generated

I applied the MVC architecture.

Model:

The Model represents the domain objects in your application, such as Flight, Ticket, User, etc.

It includes the business logic and data access logic.

The Model interacts with the database through the Repository layer to perform CRUD (Create, Read, Update, Delete) operations on the data.

View:

The View represents the user interface of your application, such as HTML pages, forms, and templates.

It is responsible for presenting the data to the user and receiving input from the user.

The View sends user input to the Controller for processing and updates the display based on the data returned from the Controller.

Controller:

The Controller acts as an intermediary between the Model and the View.

It receives input from the View, processes it using the business logic in the Model, and updates the Model as necessary.

The Controller then selects the appropriate View and passes the updated data to it for display.

In your application, the Controller can be implemented using Spring MVC or Angular's component-based architecture.

## Data Model

Graphical user interface, application, Teams

Description automatically generated

# System Testing

Testing methods that I used for my project:

1. Unit Testing: This involves testing individual components or modules of the application. It is usually done by developers during the development process to ensure that each module works as expected.

Example Unit Tests:

Test the getAllFlights function to ensure that it returns the correct flights details.

Test the insertTicket() function to ensure that it correctly books a seat and updates the number of available seats.

1. System Testing: This involves testing the entire system as a whole to ensure that it meets the specified requirements.

Example System Tests:

Test the user registration and login system by creating a new user account and logging in with the newly created account details. Ensure that the user is redirected to the correct page after successful login.

Test the flight search system by entering a departure and arrival airport and ensure that the system returns a list of available flights that match the search criteria.

Test the booking system by selecting a flight and booking a seat. Ensure that the booking confirmation page displays the correct booking details and total cost.

# Future Improvements

Payment gateway integration: If your system involves online transactions, you may need to integrate with a payment gateway to handle credit card or other forms of electronic payments. There are many payment gateway providers, such as Stripe, PayPal, or Braintree, that offer APIs or plugins that can be integrated with your system.

# Conclusion

Based on the requirements and design specifications provided, the proposed flight ticket booking system is expected to provide a convenient and efficient way for customers to book flights and manage their bookings. The system utilizes a Model-View-Controller (MVC) architecture, which promotes separation of concerns and maintainability.

# Bibliography