**Flight Ticket Booking Web Application Documentation**

**Table of Contents**

1. [**Introduction**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#introduction)
2. [**Technologies Used**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#technologies-used)
3. [**System Architecture**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#system-architecture)
4. [**Database Schema**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#database-schema)
5. [**API Documentation**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#api-documentation)
   * [**MainController**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#main-controller)
   * [**Aircraft API**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#aircraft-api)
   * [**Airport API**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#airport-api)
   * [**Flight API**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#flight-api)
   * [**Passenger API**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#passenger-api)
   * [**User API**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#user-api)
   * [**Role API**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#role-api)
6. [**User Access Levels**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#user-access-levels)
7. [**Front-end Design**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#front-end-design)
8. [**Testing**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#testing)
9. [**Conclusion**](https://chat.openai.com/c/92cffd60-d085-4741-bbbf-43dedb688351#conclusion)

**1. Introduction**

This documentation provides a comprehensive overview of the Flight Ticket Booking web application, detailing its design, APIs, and operations. The application facilitates CRUD operations for flight ticket booking and airport data management, catering to three different user access levels.

**2. Technologies Used**

* Back-end:
  + Java
  + Spring Boot Framework
* Database:
  + MariaDB
* Front-end:
  + Thymeleaf
  + HTML5
  + CSS
* **Testing:**
  + Mockito
  + MockBean
  + Selenium IDE

**3. System Architecture**

**Design**

**1. System Architecture**

The Flight Ticket Booking web application follows the Model-View-Controller (MVC) architectural pattern. This design ensures a clear separation of concerns and enhances maintainability and scalability.

**Model**

* **Aircraft Class:** Represents details about an aircraft, including its manufacturer, model, and the number of seats. It is associated with flights through a one-to-many relationship.
* **Airport Class:** Holds information about an airport, such as its code, name, city, state, and country. Similar to Aircraft, it is linked to flights through a one-to-many relationship.
* **Flight Class:** Represents a flight with attributes like flight number, departure, and destination airports, departure and arrival dates, times, and the flight charge. It establishes many-to-one relationships with Aircraft, Airport (departure and destination), and a one-to-many relationship with Passengers.
* **Passenger Class:** Represents an individual passenger with details like first and last names, phone number, passport number, email, and address. It is associated with a specific flight through a many-to-one relationship.
* **User Class:** Holds user information, including first and last names, username, email, and password. It establishes a many-to-many relationship with the Role class.
* **Role Class:** Defines user roles, such as admin, agent, or guest. It establishes a many-to-many relationship with the User class.

**View**

The view component is responsible for rendering the user interface. Thymeleaf, HTML5, and CSS are utilized to create a responsive and visually appealing front-end. The views are designed to support easy navigation and interaction for users with varying access levels.

**Controller**

* **MainController Class:** Manages the user input, interacts with the model, and returns the appropriate view. It acts as the entry point for user requests and delegates the business logic to the model.

**2. Database Schema**

The database schema is designed to ensure data integrity, eliminate redundancy, and support efficient queries. Key relationships are established through foreign keys, enforcing referential integrity.

**Tables**

1. **Aircraft Table:**
   * aircraftId (Primary Key)
   * manufacturer
   * model
   * numberOfSeats
2. **Airport Table:**
   * airportId (Primary Key)
   * airportCode
   * airportName
   * city
   * state
   * country
3. **Flight Table:**
   * flightId (Primary Key)
   * flightNumber
   * departureAirport (Foreign Key referencing Airport)
   * destinationAirport (Foreign Key referencing Airport)
   * departureDate
   * arrivalDate
   * departureTime
   * arrivalTime
   * flightCharge
   * aircraft (Foreign Key referencing Aircraft)
4. **Passenger Table:**
   * passengerId (Primary Key)
   * firstName
   * lastName
   * phoneNumber
   * passportNumber
   * email
   * address
   * flight (Foreign Key referencing Flight)
5. **User Table:**
   * id (Primary Key)
   * firstname
   * middlename
   * lastname
   * username
   * email
   * password
6. **Role Table:**
   * id (Primary Key)
   * name
7. **UserRole Table:**
   * user\_id (Foreign Key referencing User)
   * role\_id (Foreign Key referencing Role)

**Relationships**

* Aircraft and Flight: One-to-Many
* Airport and Flight (Departure): One-to-Many
* Airport and Flight (Destination): One-to-Many
* Flight and Passenger: One-to-Many
* User and Role: Many-to-Many

**4. Database Schema**

The database schema is designed to ensure data integrity and avoid redundancy.

**Relational Database:**

**Aircraft Entity:**

* Key: aircraftId (Primary Key)
* Functional Dependencies:
  + aircraftId → manufacturer, model, numberOfSeats
* Normalization Level: 3NF

**Airport Entity:**

* Key: airportId (Primary Key)
* Functional Dependencies:
  + airportId → airportCode, airportName, city, state, country
* Normalization Level: 3NF

**Flight Entity:**

* Key: flightId (Primary Key)
* Functional Dependencies:
  + flightId → flightNumber, departureAirport, destinationAirport, flightCharge, departureDate, arrivalDate, departureTime, arrivalTime
* Normalization Level: 3NF

**Flight and Aircraft Relationship:**

* Functional Dependencies:
  + flightId → aircraftId (via Aircraft association in Flight entity)
* Normalization Level: 3NF (Relational linking table)

**Flight and Airport Relationship:**

* Functional Dependencies:
  + flightId → departureAirport, destinationAirport (via Airport associations in Flight entity)
* Normalization Level: 3NF (Relational linking table)

**Passenger Entity:**

* Key: passengerId (Primary Key)
* Functional Dependencies:
  + passengerId → firstName, lastName, phoneNumber, passportNumber, email, address
* Normalization Level: 3NF

**Passenger and Flight Relationship:**

* Functional Dependencies:
  + passengerId → flightId (via Flight association in Passenger entity)
* Normalization Level: 3NF (Relational linking table)

**Role Entity:**

* Key: id (Primary Key)
* Functional Dependencies:
  + id → name
* Normalization Level: 3NF

**User Entity:**

* Key: id (Primary Key)
* Functional Dependencies:
  + id → firstname, middlename, lastname, username, email, password
* Normalization Level: 3NF

**User and Role Relationship:**

* Functional Dependencies:
  + id → roleId (via Role associations in User entity)
* Normalization Level: 3NF (Relational linking table)

**5. API Documentation**

MainController

GET /

* Displays the home page.

*Similar structure for other main functionalities.*

Aircraft API <a name="aircraft-api"></a>

*CRUD operations for Aircraft.*

Airport API <a name="airport-api"></a>

*CRUD operations for Airport.*

Flight API <a name="flight-api"></a>

*CRUD operations for Flight.*

Passenger API <a name="passenger-api"></a>

*CRUD operations for Passenger.*

User API <a name="user-api"></a>

*CRUD operations for User.*

Role API <a name="role-api"></a>

*CRUD operations for Role.*

**6. User Access Levels**

The application supports three types of users:

1. Admin:
   * Full access to all functionalities.
   * Can perform CRUD operations on aircraft, airports, flights, passengers, roles, and users.
2. Agent:
   * Limited access.
   * Can perform CRUD operations on flights and passengers.
3. Guest:
   * Read-only access.
   * Can view details of flights and airports.

**7. Front-end Design**

The front-end is designed using Thymeleaf, HTML5, and CSS, providing an intuitive and user-friendly interface for performing CRUD operations.

A screenshot of a computer

Description automatically generatedA white rectangular object with a black border

Description automatically generatedA white rectangular object with a black border

Description automatically generatedA white rectangular object with a black border

Description automatically generatedA screenshot of a computer

Description automatically generatedA white rectangular object with a black border

Description automatically generatedA white rectangular object with a black border

Description automatically generatedA white rectangular object with a black border

Description automatically generatedA white rectangular object with a black border

Description automatically generated

**8. Testing**

The application is thoroughly tested using unit tests, integration tests, and system tests. Mockito and MockBean are used for back-end testing, while Selenium IDE is employed for front-end testing.

**9. Conclusion**

The Flight Ticket Booking web application is a robust and user-friendly system, developed using Java, Spring Boot, MariaDB, Thymeleaf, HTML5, and CSS. Its MVC architecture ensures a clear separation of concerns, and the application has been thoroughly tested to ensure reliability and stability.