MRK Airport Management System - Frontend Documentation

Project Overview

This is a React-based frontend application for an airport management system that interfaces with a REST API backend.

Key Components

1. Core Features

- \*\*Flight Management\*\*: View, add, update, and delete flight information

- \*\*Airport Management\*\*: Manage airport details and track flights

- \*\*Aircraft Management\*\*: Handle aircraft fleet information

- \*\*Real-time Updates\*\*: Live flight status tracking with subscription capabilities

2. Main Pages

- `Home.jsx`: Landing page with navigation to main features

- `FlightList.jsx`: Displays and filters flight information

- `AirportList.jsx`: Shows airport details and associated flights

- `Admin.jsx`: Administrative interface for managing system data

3. Authentication

- Basic authentication system implemented in `Auth.jsx`

- Login overlay component for admin access

4. API Services

Located in `/services` directory:

```javascript

- airportService.js // Airport-related API calls

- flightService.js // Flight management operations

- aircraftService.js // Aircraft fleet management

- notificationService.js // User notifications

```

Setup Instructions

1. \*\*Installation\*\*

```bash

npm install

2. \*\*Running the Application\*\*

```bash

npm start

```

The application will run on `http://localhost:3000`

3. \*\*Testing\*\*

```bash

npm test

```

Key Features

Flight Management

- Real-time flight tracking

- Flight status updates

- Passenger notifications

- Flight filtering capabilities

Airport Information

- Airport details and locations

- Flight schedules

- Gate information

- Terminal details

Administrative Functions

- CRUD operations for flights

- Airport management

- Aircraft fleet management

- User notification system

Technical Dependencies

```json

{

"react": "^18.3.1",

"react-router-dom": "^6.28.0",

"axios": "^1.7.8"

}

```

Testing

- Jest for unit testing

- React Testing Library for component testing

- GitHub Actions integration for CI/CD

Assumptions and Requirements

- Backend API running on `localhost:8081`

- MySQL database connection required

- Modern browser support (see browserslist in package.json)

Flight Management System - Backend Documentation

Project Overview

This is a Spring Boot-based REST API for managing flights, airports, and aircraft. The system is built using Java and implements a comprehensive flight management solution.

Technical Stack

- Java 21

- Spring Boot 3.1.4

- MySQL Database

- JPA/Hibernate

- Maven

Core Components

1. Entity Models

The system consists of three main entities:

- \*\*Aircraft\*\* (`Aircraft.java`)

- Properties: id, model, manufacturer, aircraftId, capacity, flightRange, speed

- Relationships: One-to-Many with Flights

- \*\*Airport\*\* (`Airport.java`)

- Properties: id, name, iataCode, location

- Relationships: One-to-Many with Flights (departures and arrivals)

- \*\*Flight\*\* (`Flight.java`)

- Properties: id, flightNumber, airline, scheduledDepartureTime, scheduledArrivalTime, status, gate, terminal

- Relationships: Many-to-One with Aircraft and Airport

2. REST Controllers

- \*\*AircraftController\*\* (`/api/aircrafts`)

- GET, POST, PUT, DELETE operations for aircraft management

- Aircraft-specific flight queries

- \*\*AirportController\*\* (`/api/airports`)

- CRUD operations for airports

- Departure and arrival flight queries

- \*\*FlightController\*\* (`/api/flights`)

- Complete flight management operations

- Status updates and flight scheduling

3. Services

Each entity has corresponding service classes that implement business logic:

- `AircraftService`

- `AirportService`

- `FlightService`

Database Configuration

```properties

spring.datasource.url=jdbc:mysql://localhost:3306/FlightData\_DB

spring.datasource.username=root

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

```

Testing

The project includes comprehensive test coverage:

- Controller tests

- Service layer tests

- Mock email service tests

**API Endpoints Examples**

Aircraft Endpoints

```

GET /api/aircrafts

GET /api/aircrafts/{aircraftId}

POST /api/aircrafts

PUT /api/aircrafts/{id}

DELETE /api/aircrafts/{id}

```

Airport Endpoints

```

GET /api/airports

GET /api/airports/{iataCode}

GET /api/airports/{iataCode}/departures

GET /api/airports/{iataCode}/arrivals

POST /api/airports

PUT /api/airports/{iataCode}

DELETE /api/airports/{iataCode}

```

Flight Endpoints

```

GET /api/flights

GET /api/flights/{flightNumber}

GET /api/flights/status/{status}

POST /api/flights

PUT /api/flights/{flightNumber}

DELETE /api/flights/{flightNumber}

Data Seeding

The system includes a `SeedController` that populates the database with initial test data, including airports, aircraft, and flights.

Running the Application

1. Ensure MySQL is running

2. Configure application.properties

3. Run `MainApplication.java`

4. Access the API at `http://localhost:8081`

This documentation provides a comprehensive overview of the backend system's architecture and functionality.