

FlightBoardSystem

A full-stack real-time flight board management system built with React + TypeScript frontend and ASP.NET Core Web API backend.

Features

- Real-time flight board with automatic refresh every 2 minutes
- Add, delete, and filter flights
- Flight status calculation based on departure time
- RESTful API with validation
- Unit tests for services and controllers

Prerequisites

Before you begin, ensure you have the following installed:

- [.NET SDK 8.0](#) or higher
- [Node.js 14.0](#) or higher
- [Git](#)
- [Visual Studio 2022](#) or [Visual Studio Code](#) (optional)
- [Postman](#) (for API testing)

Getting Started

1. Clone the Repository

```
bash
```

```
git clone https://github.com/hillelskolnik/FlightBoard.git  
cd FlightBoard
```

2. Project Structure

```
FlightBoard/
├─ FlightBoardSystem/           # Backend (ASP.NET Core Web API)
│  ├─ Controllers/
│  ├─ Models/
│  ├─ Services/
│  └─ Program.cs
├─ FlightBoardSystem.Tests/     # Unit Tests
└─ flight-board-client/        # Frontend (React + TypeScript)
   └─ src/
```

Backend Setup (ASP.NET Core Web API)

1. Navigate to the Backend Directory

```
bash

cd FlightBoardSystem
```

2. Restore Dependencies

```
bash

dotnet restore
```

3. Check the Server Port

Open `Properties/launchSettings.json` and note the `applicationUrl`. By default, it should be:

- HTTPS: `https://localhost:7001`
- HTTP: `http://localhost:5001`

4. Run the Backend

```
bash

dotnet run
```

The server will start and you should see:

```
Now listening on: https://localhost:7001
Now listening on: http://localhost:5001
```

5. Verify the Backend

Open your browser and navigate to:

- Swagger UI: `https://localhost:7001/swagger`

Frontend Setup (React + TypeScript)

1. Open a New Terminal

Keep the backend terminal running and open a new terminal window.

2. Navigate to the Frontend Directory

```
bash
cd flight-board-client
```

3. Install Dependencies

```
bash
npm install
```

4. Configure the Backend URL

Open `src/services/flightService.ts` and update the API base URL to match your backend port:

```
typescript
const API_BASE_URL = 'https://localhost:7001/api'; // Update this to match your backend port
```

5. Run the Frontend

```
bash
npm start
```


The application will open automatically at `http://localhost:3000`

Configuration

Backend CORS Configuration

The backend is configured to accept requests from the React app. If you need to change the frontend port, update `Program.cs`:

```
csharp
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowReactApp",
        builder =>
        {
            builder.WithOrigins("http://localhost:3000") // Update if React runs on different p
                .AllowAnyMethod()
                .AllowAnyHeader();
        });
});
```



Frontend API Configuration

If your backend runs on a different port, update `src/services/flightService.ts`:

```
typescript
const API_BASE_URL = 'https://localhost:YOUR_PORT/api';
```

API Testing with Postman

1. Get All Flights

GET `https://localhost:7001/api/flights`

2. Get Filtered Flights

GET `https://localhost:7001/api/flights?status=Scheduled&destination=London`

3. Add a New Flight

```
POST https://localhost:7001/api/flights
```

```
Content-Type: application/json
```

```
{
  "flightNumber": "LY101",
  "destination": "New York",
  "departureTime": "2025-12-25T14:30:00",
  "gate": "A1"
}
```

4. Delete a Flight

```
DELETE https://localhost:7001/api/flights/{id}
```

Running Unit Tests

1. Navigate to the Test Project

```
bash
```

```
cd FlightBoardSystem.Tests
```

2. Run All Tests

```
bash
```

```
dotnet test
```

3. Run Tests with Details

```
bash
```

```
dotnet test --verbosity normal
```

4. Run Specific Test Categories

```
bash
```

```
# Run only Service tests
```

```
dotnet test --filter "FullyQualifiedName~Services"
```

```
# Run only Controller tests
```

```
dotnet test --filter "FullyQualifiedName~Controllers"
```

Troubleshooting

Common Issues

1. CORS Error

- Ensure the backend is running
- Verify the port numbers match in both frontend and backend configurations
- Check that CORS is properly configured in `Program.cs`

2. SSL Certificate Error

```
bash
```

```
dotnet dev-certs https --trust
```

3. Port Already in Use

- Backend: Change the port in `Properties/launchSettings.json`
- Frontend: Run with `PORT=3001 npm start`

4. Cannot Connect to Backend

- Verify the backend is running
- Check the console for any error messages
- Ensure the API_BASE_URL in `flightService.ts` matches your backend URL

Verify Everything is Working

1. Backend health check: Navigate to `https://localhost:7001/swagger`
2. Frontend health check: Navigate to `http://localhost:3000`
3. Create a test flight using the form
4. Verify it appears in the table
5. Try filtering and deleting flights

Technologies Used

Backend

- ASP.NET Core 8.0 Web API
- C# 12
- In-memory data storage
- xUnit for testing
- Swagger for API documentation

Frontend

- React 18
- TypeScript
- Axios for HTTP requests
- CSS for styling

Flight Status Logic

The flight status is calculated based on departure time:

- **Scheduled:** More than 30 minutes until departure
- **Boarding:** 10-30 minutes until departure
- **Departed:** Up to 60 minutes after scheduled departure
- **Landed:** More than 60 minutes after departure
- **Delayed:** 15+ minutes after scheduled departure (if not departed)

Contributing

1. Fork the repository
2. Create a feature branch
3. Commit your changes
4. Push to the branch
5. Create a Pull Request

License

This project is licensed under the MIT License.