Objective

Develop a full-stack real-time flight board management system using a React + TypeScript frontend and an ASP.NET Core Web API backend. The system should support live updates, flight management, validation, and user-friendly UI with real-time feedback.

Technologies

- Back-end: ASP.NET Core Web API (C#)
- Front-end: React + TypeScript
- Styling: CSS / Tailwind / styled-components / Material UI

Core Features

1. Real-Time Flight Board (Frontend)

- Automatically refreshes flight data every 2 minutes
- Displays flights in a table with the following columns:
- Flight Number
- Destination
- Departure Time
- Gate
- Status (calculated client-side)
- Status updates every 2 minutes
- Status should animate visually on change

2. Flight Management via Backend API

Required API Endpoints

- GET /api/flights - Return all current flights
- POST /api/flights - Add a new flight (with validation)
- DELETE /api/flights/{id} - Delete a flight by ID
- GET /api/flights?status={status}&destination={destination} - Return filtered or searched flights by status and/or destination (query parameters should be optional and combinable)

Server-Side Validation

- All fields are required:
- Flight number (must be unique)

- Destination

- Gate

- Departure time (must be in the future)

- Return appropriate HTTP error codes for invalid inputs

Data Storage

- You may store flights using:

- In-memory list (minimum requirement)

- OR a simple database like SQLite using EF Core

3. Auto Status Calculation (Client-Side)

Use this logic to calculate and display the flight status based on departure time:

```
function getFlightStatus(departureTime: Date): FlightStatus {
const now = new Date();
const diff = (departureTime.getTime() - now.getTime()) / 60000;

if (diff > 30) return "Scheduled";
if (diff > 10) return "Boarding";
if (diff >= -60) return "Departed";
if (diff < -60) return "Landed";
if (diff < -15) return "Delayed";
return "Scheduled";
}
```

4. Add Flight

- Form should include:

- Flight Number

- Destination

- Departure Time (future only)

- Gate

- Validate inputs on both client and server

5. Delete Flight

- Each row includes a delete button

- Deletion is performed via the API

6. Filter / Search Flights

- Backend must support filtering by query parameters: status and destination

- Frontend must include:

- A search form with input fields or dropdowns to filter by status and/or destination

- A Search button that triggers the filtered request

- Optionally, a "Clear Filters" button to reset the table

Bonus Features

Frontend Animation Bonuses

- New row animation: Fade-in or slide-in when a new flight is added

- Status change animation: Background color transition or effect

- Highlight updated rows: Glow or border effect for a few seconds

Backend Extras (Optional, But Encouraged)

- Implement logging

- Log actions like "flight added", "flight deleted", etc. (console or file)

- Use persistent database

- Store flights in SQLite or SQL Server LocalDB using EF Core

- Write unit tests

- For controllers and services using xUnit / NUnit

- Add WebSocket/SignalR support

- Implement a real-time update channel:

- Broadcast new flights and deletions to connected clients

- Use SignalR to push updates instead of polling

Submission Guidelines

- Provide clean, modular code with full TypeScript and C# typings

- Include unit tests if implemented

- Add a README.md file with:

- Setup instructions for backend and frontend

- Example API requests (e.g. Postman or cURL)

- List of any third-party libraries used
- (Optional) Include a short screen recording of the running app