Flight Management Company

2-Day Hands-On Project (Layered Architecture, Repository Pattern, EF Core, LINQ)

Project Summary (one sentence)

Build a **Flight Management System (FMS)** backend for a company that manages flights, aircraft, crews, bookings, passengers and airports — implemented using **layered architecture**, **EF Core**, **Repository pattern**, and **LINQ**.

Learning Goals

- Design and document an ERD and relational schema.
- Implement EF Core models with annotations and relationships.
- Implement layered architecture: **Presentation / Service / Repository / Data (DbContext)**.
- Implement per-entity repositories exposing essential CRUD + query methods.
- Write LINQ queries: joins, groupings, aggregations, partitioning, projection to DTOs, hierarchical queries

Business Domain & ERD (entities + relationships)

Entities (core):

Airport

- AirportId (int PK)
- IATA (string, 3, unique)
- Name (string)
- City (string)
- Country (string)
- TimeZone (string)

Aircraft

- AircraftId (int PK)
- TailNumber (string, unique)
- Model (string)
- Capacity (int)

CrewMember

- CrewId (int PK)
- FullName (string)
- Role (enum/string) — Pilot/CoPilot/FlightAttendant
- LicenseNo (string, nullable)

Route

- RouteId (int PK)
- DistanceKm (int)

Flight

- FlightId (int PK)
- FlightNumber (string) — e.g., "FM101"
- DepartureUtc (DateTime)
- ArrivalUtc (DateTime)
- Status (string/enum)
- Note: add unique constraint on (FlightNumber, DepartureUtc.Date)

Passenger

- PassengerId (int PK)
- FullName (string)
- PassportNo (string, unique)
- Nationality (string)
- DOB (DateTime)

Booking

- BookingId (int PK)
- BookingRef (string, unique)
- BookingDate (DateTime)
- Status (string)

Ticket

- TicketId (int PK)
- SeatNumber (string)
- Fare (decimal)
- CheckedIn (bool)

FlightCrew ➔ relationship attributes on many to many

- RoleOnFlight (string)
- Primary Key (FlightId, CrewId)

Baggage

- BaggageId (int PK)
- TicketId (FK → Ticket)
- WeightKg (decimal)
- TagNumber (string)

AircraftMaintenance

- MaintenanceId (int PK)
- MaintenanceDate (DateTime)
- Type (string)
- Notes (string)

Relationships (high level):

- Airport 1..* Route (as origin and destination) — Route has two FKs to Airport.
- Route 1..* Flight
- Aircraft 1..* Flight
- Flight .. CrewMember via FlightCrew (many-to-many)
- Passenger 1..* Booking
- Booking 1..* Ticket (one booking could book multiple segments/flights)
- Flight 1..* Ticket
- Ticket 1..* Baggage
- Aircraft 1..* AircraftMaintenance

Required Implementations & Constraints (must-haves)

1. **EF Core Models**: DataAnnotations for PKs, FKs, required fields, column types (e.g., decimal precision).
2. **DbContext**: FlightContext with DbSets and Fluent API config for composite keys and unique constraints.
3. **Migrations**: Create initial migration and apply to local DB (use SQL Server or SQLite).
4. **Seed Data**: At least **10 rows** per table (airport, aircraft, crews, routes, flights, passengers, bookings, tickets, baggage, maintenance). Ensure referential integrity and distributed realistic timestamps.
5. **Repositories**: One repository class per entity each exposes:
    - GetAll(), GetById(int id)
    - Add(entity), Update(entity), Delete(id)
    - A few entity-specific helpers (see below per entity).
6. **Service Layer**: FlightService that uses repositories and exposes business operations and **LINQ**

7. **DTOs**: For all projections — e.g., FlightScheduleDto, CrewAssignmentDto, RevenueByRouteDto.
8. **Program** : Console app demonstrating each service method

---

Repository extra helper methods (suggested)

- FlightRepository: GetFlightsByDateRange(DateTime from, DateTime to), GetFlightsByRoute(int routeId)
- TicketRepository: GetTicketsByBooking(string bookingRef), GetTicketsByPassenger(int passengerId)
- CrewRepository: GetCrewByRole(string role), GetAvailableCrew(DateTime dep)
- AircraftRepository: GetAircraftDueForMaintenance(DateTime beforeDate)
- BookingRepository: GetBookingsByDateRange(DateTime from, DateTime to)

LINQ Service (must implement)

Implement these methods in FlightService using **repository methods and LINQ** (all return DTOs or collections):

1. **Daily Flight Manifest**
    - Input: date (local or UTC)
    - Output DTO: FlightNumber, DepartureUtc, ArrivalUtc, OriginIATA, DestIATA, AircraftTail, PassengerCount, CrewList (names + roles), TotalBaggageWeight.
    - Complexity: multiple joins, grouping, GroupJoin for crew.
2. **Top Routes by Revenue**
    - For a date range, compute revenue per route (sum of ticket fares), ordered descending; include number of seats sold and average fare.
    - Use GroupBy and projection.
3. **On-Time Performance**
    - For flights in a range, compute percentage on-time (ArrivalUtc within X mins of schedule) per airline/company or per route.
4. **Seat Occupancy Heatmap**
    - For each flight, compute occupancy rate = tickets sold / aircraft capacity. Return flights with occupancy > 80% or top N.
5. **Find Available Seats for a Flight**
    - Given flightId, return list of available seat numbers (assume seat map can be derived from capacity and booked seats). Use Except set operation.
6. **Crew Scheduling Conflicts**
    - Detect crew members assigned to flights that overlap in time (time overlap check) — return conflict DTO: CrewId, CrewName, FlightA, FlightB.

7. **Passengers with Connections**
    - Find passengers who have bookings with connecting flights (same booking, sequential flights within X hours), return itinerary DTO.
8. **Frequent Fliers**
    - Top N passengers by number of flights or total distance flown (sum of route distances via tickets).
9. **Maintenance Alert**
    - Aircraft with cumulative flight hours > threshold or last maintenance older than Y days — requires computing sum of distances or flights per aircraft (simulate hours = distance / avg speed).
10. **Baggage Overweight Alerts**
    - Tickets whose total baggage weight exceeds threshold (e.g., 30kg per passenger). Use GroupBy ticket and sum baggage weights.
11. **Complex Set/Partitioning Examples**
    - Use Union to combine two passenger lists (VIP + FrequentFliers), Intersect for passengers present in both, Except for passengers who canceled.
    - Partition flights into pages (Skip, Take) for an admin UI.
12. **Conversion Operators Demonstration**
    - Return ToDictionary of flights keyed by FlightNumber, ToArray of top 10 routes, AsEnumerable to switch to in-memory calculations for heavy operations, OfType example from mixed object collections.
13. **Window-like Operation (running totals)**
    - For each day, compute cumulative revenue (running sum). Implement via OrderBy and Select with aggregate accumulation.
14. **Forecasting (simple)**
    - Using historical bookings, project expected bookings for next week by simple average or growth rate (exercise in LINQ + basic math).

DTO Examples (suggested)

- FlightManifestDto { string FlightNumber; string Origin; string Destination; DateTime DepUtc; DateTime ArrUtc; string AircraftTail; int PassengerCount; decimal TotalBaggageKg; List<CrewDto> Crew }
- RouteRevenueDto { int RouteId; string Origin; string Destination; decimal Revenue; int SeatsSold; decimal AvgFare }
- CrewConflictDto { int CrewId; string CrewName; int FlightAId; int FlightBId; DateTime FlightADep; DateTime FlightBDep }
- PassengerItineraryDto { int PassengerId; string PassengerName; List<ItinSegmentDto> Segments }

Seed Data Requirements

- **At least 10 airports**, ensure several international and domestic.
- **At least 10 aircrafts** with varying capacity (100–300).
- **At least 20 crew members** distributed across roles and bases.
- **At least 20 routes** (many duplicate origin/destination pairs).
- **At least 30 flights** covering a 30–60 day window (mix scheduled, departed, landed).
- **At least 50 passengers**, mix of nationalities.
- **At least 100 bookings** and **200 tickets** covering single and connecting itineraries.
- **At least 150 pieces of baggage** associated to tickets.
- **At least 15 maintenance records.**

Seed generation should ensure:

- Tickets reference existing flights and bookings.
- Seat numbers are generated in a consistent scheme (e.g., "12A", "12B").
- Baggage weights realistic and some over threshold.

Grading Rubric / Evaluation Checklist

- ERD correctness & normalization — 10%
- EF Core models & migrations — 10%
- Repositories implemented with CRUD + helpers — 10%
- Service layer correctness & DI — 10%
- Seed completeness & referential integrity — 10%
- Coverage of advanced LINQ tasks (each task 3–5%) — 30%
- Code quality, DTO usage, tests, documentation — 10%