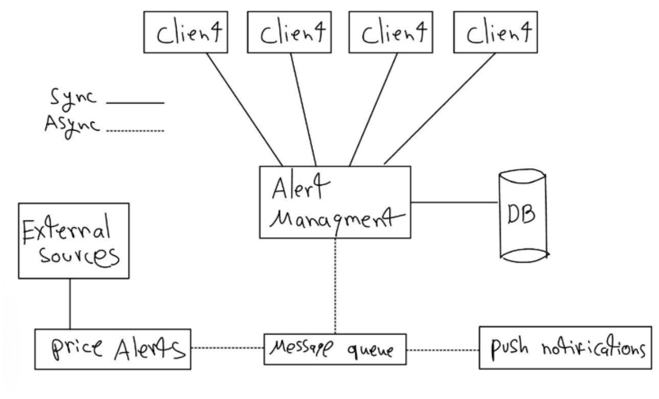
Architecture Diagram



Data Structures
Controllers

- **AlertManagementAPIController**
    - Uses the services: UserService, FlightService, UserFlightService.
    - Works with the DTO: ResponseDto.
    - Handles multiple endpoints for managing users, flights, and user-flight assignments.

Exceptions

- **Custom Exceptions**:
    - CreateUserWithInvalidFlightIdException
    - FlightNotFoundException
    - FlightsNotFoundForUserException
    - UserFlightNotFoundException
    - UserNotFoundException

Models

- **Flight**:
  Represents the schema for storing flight information.
- **User**:
  Represents the schema for storing user information.
- **UserFlight**:
  Represents the schema for linking users to flights with:
    - FlightId and UserId (foreign keys).
    - Navigation properties for related User and Flight objects.

DTOs

- **FlightDto**: Contains flight information for transfer over the web.
- **UserDto**: Contains user information for transfer over the web.
- **UserFlightsDto**: Includes user details and a list of their related flights.
- **PriceAlertDto**: Received by the consumer from a message queue (price alert notifications).
- **PushNotificationDto**: Sent to a message queue by the publisher for downstream consumption.
- **ResponseDto**: General response format used between the server and client.

RabbitMQ Components
Message Consumer

- **RabbitMQPriceAlertMessageConsumer**:
    - A background service that:
        - Listens for price alert messages from the PriceAlertQueue.
        - Deserializes messages and invokes the PriceService.

Message Sender

- **RabbitMQPushNotificationsMessageSender**:
    - Publishes messages (a list of PushNotificationDto) to the PushNotificationsQueue.
    - Ensures downstream services can consume and process push notifications.

Services

- **UserService**: Handles CRUD operations (CreateUser, GetUser, UpdateUser, DeleteUser).
  - Properties: Mapper, DbContext, ResponseDto.
- **FlightService**: Handles CRUD operations (CreateFlight, GetFlight, UpdateFlight, DeleteFlight).
  - Properties: Mapper, DbContext, ResponseDto.
- **UserFlightService**:
  - Manages operations like AssignFlightsToUser, GetFlightsForUser, and DeleteUserFlight.
  - Properties: Mapper, DbContext, ResponseDto.
- **PriceService**:
  - Updates flight prices in the database based on incoming messages.
  - Creates and sends PushNotificationDto objects to the message queue.

Mapping and Program

- **MappingConfig**:
  - Configures model-to-DTO and DTO-to-model mappings, with custom options.
- **Program**:
  - Serves as the app container, registering all components and dependencies.

Data Flow

1. Incoming Requests (API Level)

- **Controller: AlertManagementAPIController**
  - Receives HTTP requests (e.g., CreateUser, AssignFlightsToUser).
  - Validates and parses the data.
  - Delegates processing to the appropriate service (e.g., UserService).
  - Returns a ResponseDto to the client.

---

2. Processing Business Logic (Service Level)

- **Services: UserService, FlightService, UserFlightService, PriceService**
  - Perform CRUD operations and execute business logic.
  - Interact with the database using AppDbContext.
  - Use MappingConfig to convert models to DTOs (and vice versa).
  - Prepare ResponseDto for the controllers.

---

3. RabbitMQ Integration

- **Message Consumption (RabbitMQPriceAlertMessageConsumer)**
  - Listens for incoming PriceAlertDto messages from PriceAlertQueue.
  - Deserializes the message and calls PriceService.
- **Processing (PriceService)**
  - Updates flight prices in the database.
  - Creates PushNotificationDto objects.
  - Sends these objects to the PushNotificationsQueue using the sender.
- **Message Publishing (RabbitMQPushNotificationsMessageSender)**
  - Publishes messages to the PushNotificationsQueue.

---

4. Data Persistence (Database Layer)

- **Models: Flight, User, UserFlight**
  - Represent database schemas.
  - UserFlight links User and Flight via foreign keys, managing relationships.
- **DbContext: AppDbContext**
  - Handles database interaction using Entity Framework.
  - Executes operations to store, retrieve, update, or delete data.