DMS_2025 Protocol - Intermediate Hand-In

1. App Architecture

1.1 Solution Structure & Responsibilities

The solution ***DMS_2025.sln*** is split into the following projects:

- **DMS_2025.REST**: ASP.NET Core 8 REST API exposing document CRUD, search with pagination, and an upload endpoint that publishes to a queue.
- **DMS_2025.DAL**: Entity Framework Core with ***DmsDbContext*** and a repository abstraction (***IDocumentRepository*** / ***DocumentRepository***).
- **DMS_2025.Models**: Domain entities (e.g., ***Document***).
- **DMS_2025.Services.Worker**: .NET Generic Host background worker that consumes messages from RabbitMQ.
- **DMS_2025.UI**: Minimal static web UI (***index.html***, ***app.js***) consuming the REST API.
- **DMS_2025.Tests**: NUnit unit tests for the DAL.

1.2 Dependency Injection

- The API registers ***DmsDbContext*** and the repository in the default ASP.NET Core DI container.
- The Worker uses ***Host.CreateDefaultBuilder*** and DI to configure logging and RabbitMQ channel/services.

1.3 Logging & Libraries

- **Logging:** Serilog configured via ***appsettings*.json***, output to console.
- **Validation:** DTOs with a FluentValidation-based action filter returning RFC7807 ProblemDetails on 400 responses.
- **Configuration:** Environment-based configuration; connection strings supplied via ***appsettings*** and docker-compose.

2. Use Cases

- **Manage Documents**
  Create, read, update, and delete documents.
- **Search & Pagination**
  Filter by query term (q) and navigate via ***page*** and ***pageSize***.
- **Upload & Queueing**
  Upload endpoint accepts payloads and publishes a message; the Worker consumes messages from RabbitMQ.

3. Design Patterns & Key Decisions

- **Repository Pattern** to isolate EF Core and enable unit testing.
- **DTOs + Validation Filter** to keep controllers thin and responses consistent.
- **Message Queue Boundary** between API (ingest) and Worker (processing) using RabbitMQ.

4. Testing

- **Unit Tests (NUnit):** DAL repository tests covering CRUD and query behavior.
- **Mocking** using ***Moq.***

5. DevOps & Deployment

- **Containerization:** API has a Dockerfile.
- **Orchestration (docker-compose):**
  - API service
  - PostgreSQL database
  - RabbitMQ broker (with management UI)
- **Developer Experience:** Swagger UI enabled in Development.

6. API Overview

Base Route: *`api/v1/documents`*

- ***`GET /api/v1/documents?q={term}&page={n}&pageSize={m}`***: list + search + pagination
- ***`GET /api/v1/documents/{id}`***: get by id
- ***`POST /api/v1/documents`***: create document (validated)
- ***`PUT /api/v1/documents/{id}`***: update
- ***`DELETE /api/v1/documents/{id}`***: delete
- ***`POST /api/v1/documents/upload`***: upload payload (enqueued for async processing)

Conventions

- Request/response payloads use DTOs.
- Validation errors return ProblemDetails (HTTP 400).

7. UI

- Static assets under **DMS_2025.UI/wwwroot** (***index.html, app.js***).
- Supports listing with search/pagination, creation, and deletion.
- Basic inline status/feedback on requests.