Smart Documentation Hub – Backend Flow (for Backend Dev)

1. High-Level Idea

A web-based **Smart Documentation Hub** where users can: - Register & log in - Upload documents - View documents - Add inline comments/metadata - Manage access securely

Frontend is handled separately. Backend will expose **REST APIs** consumed by the UI.

---

2. Architecture Overview

- **Framework**: ASP.NET Core Web API
- **ORM**: Entity Framework Core
- **Database**: MS SQL Server
- **Auth**: JWT-based authentication
- **Storage**:
- Metadata → Database
- Files → Server storage / cloud (later)

---

3. Database → Entity Flow

Core Entities (Models are FIXED)

- **User** – user info & authentication
- **Document** – document metadata (no file properties here)
- **Comment** – inline comments on documents

Important: Models remain unchanged. Extra data is handled using DTOs.

---

4. DTO Layer (Very Important)

DTOs act as a **bridge** between frontend & models.

Example Responsibilities

- Combine data not present in models
- Hide internal DB structure
- Shape API responses cleanly

Example DTOs

- `DocumentUploadDto`
- IFormFile File

- int UserId

- `DocumentResponseDto`

- int DocumentId
- string FileName
- string FileUrl
- DateTime UploadedAt

- int UserId

- `CommentDto`

- int DocumentId
- string Text
- int UserId

---

5. API Flow (Step-by-Step)

1️⃣ Authentication

- `POST /api/auth/register`
- `POST /api/auth/login`
- Returns JWT token

2️⃣ Document Upload

- `POST /api/documents/upload`

Flow: 1. Receive `DocumentUploadDto` 2. Save file to storage 3. Save metadata in `Document` table 4. Return `DocumentResponseDto`

3️⃣ Get Documents

- `GET /api/documents/user/{userId}`

Flow: 1. Fetch documents by userId 2. Map entities → DTOs 3. Return list

---

4️⃣ Comments

- `POST /api/comments`
- `GET /api/comments/document/{documentId}`

Flow: 1. Validate document 2. Save comment 3. Return comment DTOs

---

6. Controller → Service → Repository Pattern

```
Controller
   ↓
Service (business logic)
   ↓
Repository (DB access)
```

Why? - Clean code - Easy testing - Easy scaling

---

7. Error Handling & Validation

- ModelState validation
- Proper HTTP status codes
- Central exception middleware

---

8. Security Notes

- JWT required for all document/comment APIs
- Users can only access **their own documents**
- Validate ownership before actions

---

9. Future Enhancements (Optional)

- Role-based access
- Cloud file storage
- Document versioning
- Full-text search

---

10. Summary (One-Liner)

Models stay fixed → DTOs shape data → Services handle logic → Controllers expose APIs

This flow should align frontend & backend cleanly without model changes.