**Use Cases**

* UserRegistration
* UserLogin
* WorkspaceCreation
* WorkspaceModification
* DocumentUpload
* DocumentDownload
* DocumentDeletion
* DocumentMetadataUpdate
* DocumentPreview

**Dividing Your Document Management System into Clean Architecture Layers**

**Understanding the Layers**

Before diving into the specifics, let's recap the four main layers in Clean Architecture:

* **Domain:** Contains core business rules and entities.
* **Application:** Defines application-specific use cases and orchestrates the domain.
* **Infrastructure:** Handles data access, external systems, and other technical details.
* **Presentation:** Handles the user interface and input/output.

**Applying to Your DMS Project**

**Domain Layer**

* **Entities:** User, Workspace, Directory, Document, DocumentMetadata, DirectoryStatus (enum)
* **Value Objects:** Consider using value objects like EmailAddress, DocumentSize if applicable.
* **Domain Services:** If complex business logic is required (e.g., calculating document usage), create domain services.

**Application Layer**

* **Use Cases:**
  + UserRegistration, UserLogin, WorkspaceCreation, DocumentUpload, DocumentDownload, DocumentDeletion, etc.
  + These use cases will orchestrate the domain layer and interact with the infrastructure layer.
* **DTOs (Data Transfer Objects):** Define input and output models for use cases.

**Infrastructure Layer**

* **Repositories:** UserRepository, WorkspaceRepository, DirectoryRepository, DocumentRepository
* **Data Access:** Entity Framework Core DbContext
* **External Services:** File storage (e.g., Azure Blob Storage)
* **Mappers:** Convert between domain entities and data transfer objects.

**Presentation Layer**

* **API Controllers:** Handle incoming requests and return responses.
* **Angular Components:** Build the user interface.
* **ViewModels:** Prepare data for the UI.

**Example Structure**

DocumentManagementSystem

├── Domain

│ ├── Entities

│ │ ├── User.cs

│ │ ├── Workspace.cs

│ │ ├── Directory.cs

│ │ ├── Document.cs

│ │ └── DocumentMetadata.cs

│ └── Services

│ ├── DocumentService.cs

│ └── ...

├── Application

│ ├── UseCases

│ │ ├── UserRegistrationUseCase.cs

│ │ ├── WorkspaceCreationUseCase.cs

│ │ └── ...

│ └── DTOs

│ ├── UserDto.cs

│ ├── WorkspaceDto.cs

│ └── ...

├── Infrastructure

│ ├── Persistence

│ │ ├── MyDbContext.cs

│ │ ├── UserRepository.cs

│ │ ├── WorkspaceRepository.cs

│ │ └── ...

│ ├── FileStorage

│ │ ├── FileStorageService.cs

│ └── Mappers

│ ├── UserMapper.cs

│ └── ...

├── API

│ ├── Controllers

│ │ ├── UserController.cs

│ │ ├── WorkspaceController.cs

│ │ └── ...

├── UI

│ ├── AngularApp

│ ├── ...

**Additional Considerations**

* **Dependency Injection:** Use a DI container to manage dependencies between layers.
* **Error Handling:** Implement proper error handling mechanisms in all layers.
* **Testing:** Write unit tests for domain and application layers, integration tests for infrastructure, and end-to-end tests for the entire system.
* **Security:** Implement appropriate security measures, including authentication, authorization, and data protection.
* **Performance Optimization:** Consider performance implications when designing the architecture.

By following this structure and applying the principles of Clean Architecture, you can create a well-organized and maintainable Document Management System.