# Storage microservice

#### Functional Requirements:

1. **File Upload**: Ability to upload files of any type and store them.
2. **File Download**: Ability to download stored files using a unique identifier.
3. **File Deletion**: Ability to delete stored files using a unique identifier.
4. **File Metadata Retrieval**: Ability to retrieve metadata (e.g., size, upload date) of stored files.
5. **Authentication and Authorization**: Secure access to the microservice using JWT-based authentication.
6. **Error Handling**: Provide appropriate error messages and status codes for different failure scenarios. (we can use middleware for this if needed)

#### Nonfunctional Requirements:

1. **Scalability**: Ability to handle a large number of file uploads and downloads efficiently.
2. **Performance**: Quick response times for file upload, download, and metadata retrieval.
3. **Security**: Secure file storage and transmission using HTTPS and JWT authentication.
4. **Reliability**: Ensure high availability and reliability of the microservice.
5. **Maintainability**: Clean and maintainable codebase using Clean architecture and separated implementation for each storage provider.
6. **Extensibility**: Easy to extend the microservice with additional features in the future like another storage provider , now I just implement Local and Azure provider.

### High-Level and Low-Level Design

#### High-Level Design:

* **Project Architecture:** designed by clean architecture to be maintainable , extendable and scalable application using high level abstraction dependency Injection pattern .
* **Client**: Any service or application that consumes the storage microservice.
* **API Gateway**: Optional component for routing requests to the microservice.
* **Storage Microservice**: Handles file upload, download, deletion based on storage type.
* **Authentication**: Handles JWT token generation and validation.
* **Database (SQL)** : Stores metadata of the files.
* **File Storage**: Physical storage (local or cloud) where files are stored.

#### Low-Level Design:

* **Controllers**:
  + FileController: Handles API endpoints for file operations.
* **Services**:
  + IStorageService: Interface for storage operations.
  + LocalStorageService: Implementation of IStorageService for local storage.
  + AzureBlobStorageService: Implementation of IStorageService for azure blob storage.
  + IFileManagerService: interface for communicate with storage service and db operations.
  + FileManagerService : Implementation of IFileManagerService for file metadata actions: upload, download, deletion, and metadata retrieval
  + IUnitOfwork : interface for centralisation of database transaction with different repositories
  + UnitOfwork : Implementation of IUnitOfwork that interact with database operation with different repositories.
* **Models**:
  + FileMetadata: Represents metadata for stored files.
  + FileMetadataResponse: the responseof retrive API
* **Repositories**:
  + FileMetadataRepository: Manages file metadata in the database.
* **Middleware**:
  + ExceptionHandlingMiddleware : Handles exceptions and provides consistent error responses.

### Type of Storage and File Handling

#### Storage Type:

* **Database**: SQL Server for storing file metadata.
* **File System**: Multiple implementation for this service depending on setting parameter
  + Local file system for storing actual files.
  + Cloud storage: Azure Blob Storage for scalability this can be extended to like AWS.

#### File Saving and Retrieval:

* **Saving Files**:
  1. Receive file upload request.
  2. Save the file to the local file system or cloud storage.
  3. Store metadata (e.g., file path, size, upload date) in the database.
  4. Return a unique identifier for the file.
* **Retrieving Files**:
  1. Receive file download request with unique identifier.
  2. Retrieve file metadata from the database.
  3. Fetch the file from the storage location.
  4. Return the file to the client.

### Communication with Other Microservices

#### Communication Method:

* **HTTP/HTTPS**: Other microservices will communicate with the storage microservice using RESTful HTTP/HTTPS endpoints.

#### Example Endpoints:

* **POST /api/files**: Upload a new file.
* **GET /api/files/{filename}**: Download a file by unique file name.
* **DELETE /api/files/{filename}**: Delete a file by unique file name.
* **GET /api/files/{filename}/metadata**: Retrieve metadata for a file by unique file name.

#### Authentication:

* **JWT Token**: Other microservices will include a JWT token in the Authorization header when making requests to the storage microservice.
  + **For demo**: Test API added to simulate JWT token generator.