Please design a storage microservice, that can be used to store files from any extensions and act as storage hub for all the other microservices. And please indicate the following:

1. The function and nonfunctional requirement for this microservice
2. The high level and low-level design for the microservice
3. The type of the storage that will be used and why, and how the files will be saved and retrieved
4. How the other microservice will communicate with the new designed one?

### **1. Functional and Non-Functional Requirements**

#### **Functional Requirements:**

Each file to save consist of the file itself and it’s details or metadata.

1. **File Upload**: for upload the files with any type and save it’s metadata.
2. **File Download**: for retrieve the files with it’s metadata.
3. **File Deletion**: for delete the files with it’s metadata.
4. **File Update**: for Update the files with it’s metadata.
5. **Authentication and Authorization**: make sure the user authenticated and authorized for previous operation.

#### **Non-Functional Requirements:**

1. **Scalability**: The service can handle a large volume of file uploads and downloads.
2. **Reliability**: Ensure high availability and durability of stored files.
3. **Performance**: Optimize for fast file uploads and downloads.
4. **Security**: Secure the file by encrypting it and secure the operation of upload and retrieve the file.

### **2. High-Level and Low-Level Design**

#### **High-Level Design:**

1. **API Gateway**: when we need to make gateway or one entry point for all requests.
2. **Storage Service**: Handles the core functionalities (upload, download, update and delete).
3. **Authentication**: Manages authentication and authorization.
4. **Database**: Stores the data about the files (it’s metadata).
5. **File Storage**: Actual storage system for the files (database or folder or cloud).

#### **Low-Level Design:**

* **Endpoints**:
  + /upload: Accepts file data and details, stores the file, and returns a file ID.
  + /download/{fileId}: Retrieves the file using the file ID.
  + /delete/{fileId}: Deletes the file using the file ID.
  + /details/{fileId}: Retrieves details for a given file ID.
  + /update/{fileId,Filedata}: Update the file and file with it’s metadata.
* **Service Components**:
  + **File Controller**: Handle CRUD services.
  + **File Service**: Business logic for file operations (upload, download, delete,update).
  + **Storage**: service for save the file itself like (database, folder, Cloud Storage).
  + **Metadata Service**: Manages metadata operations, interfaces with the database.
  + **Authentication and authorization**: Ensures requests are authenticated and authorized by using auth middle ware.

### **3. Storage Type and File Handling**

#### Storage Types:

1. **Database Storage**: Store files directly in a relational database.
2. **File System Storage**: Save files in a directory on the server.
3. **Cloud storage**: Using cloud-based storage solutions like AWS and Azure Cloud Storage

#### File Saving and Retrieval:

**Saving Files**:

* + **Database**: The file is stored in a relational database Metadata is stored in a separate table.
  + **File System**: The file is saved to a specified directory with a unique filename. Metadata including the file path is stored in the database.
  + **Cloud**: Files are uploaded through the API the file is then stored in the object storage, and metadata is stored in the database.

**Retrieving Files**:

* **Database**: The file is retrieved from the database using its unique identifier.
* **File System**: The file path is retrieved from the database, and the file is read from the directory.
* **Cloud**: Using the file ID (name) the service fetches the file from the object storage and returns it to the requesting service.

### **4. Communication with Other Microservices**

* **REST API**: RESTful APIs. make HTTP requests.
* **Authentication**: Use token-based authentication (like JWT) to ensure secure access.

**Difference between save file in folder and database**

1. **Using Folders for Storage:**
   * **Simplicity:** Storing files in folders is straightforward and simple to implement.
   * **Direct:** Files can be accessed directly via file paths.
   * **Scalability:** Can be scalable depend on folder and the device .
   * **Performance:** Generally good performance.

**Considerations:**

* + **File Management:** Requires challenge for tasks like backup, deletion, and organization.
  + **Security:** security on the folder and the storage device system.
  + **Limited Metadata:** Metadata save in database far away from the file itself.

1. **Using a Database for Storage:**
   * **Centralized Management:** Files and metadata are stored in same place.
   * **Querying and Filtering:** Easier to query and filter files based on metadata because they are not separated.
   * **Transactions:** Supports transactional operations.
   * **Security:** Database security.

**Considerations:**

* + **Complexity:** Requires setting up a database schema and handling binary large objects for file storage.
  + **Performance:** Retrieving large files can be slower compared to direct file system access.
  + **Scalability:** Database performance can degrade with a large number of files unless managed properly.