**Backend Team Lead Task**

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?

**Answers:**

**1)The function and nonfunctional requirement for this microservice:**

**Functional Requirements:**

1**. File Upload**: Allow users to upload files of any extension.

2. **File Retrieval**: Enable retrieval of files based on unique identifiers.

3. **File Deletion**: Provide the ability to delete files.

4. **File Metadata Management**: Store and retrieve metadata for each file, like name, size, type, and upload date.

5. **Access Control**: Implement permissions to control who can upload, retrieve, or delete files.

**Non-Functional Requirements:**

1. **Scalability**: The service should handle a growing number of files and requests without performance degradation.

2. **Availability**: Ensure high availability to allow other microservices to access files at all times.

3. **Performance**: Aim for quick upload and retrieval times, optimizing for low latency.

4. Security: Implement encryption for data at rest and in transit, along with authentication and authorization measures.

5. **Reliability**: Ensure data integrity and provide backup and recovery options to prevent data loss.

**2)The high level and low-level design for the microservice**

**High-Level Design:**

The storage microservice will consist of several components:

1. **API Gateway**: This handles incoming requests from other microservices and routes them to the appropriate endpoints.

2. **File Management Service**: This core component manages file uploads, retrievals, and deletions.

3. **Database**: A metadata database stores information about each file, such as its ID, name, and path.

4. **Storage Solution**: The actual files are stored in a cloud storage service, like Amazon S3 or Google Cloud Storage, which provides durability and scalability.

**Low-Level Design:**

**Endpoints:**

- POST /files: For uploading files.

- GET /files/{id}: For retrieving files.

- DELETE /files/{id}: For deleting files.

**Data Models:**

- File metadata model containing fields like ID, name, description, and creation date.

**3)The type of the storage that will be used and why, and how the files will be saved and retrieved.**

**Type of Storage:**

-**Cloud storage service**, like Amazon S3 or Google Cloud Storage

-**File storage systems**, including disk, network, shared disk, and database file systems

Using a cloud storage solution like Amazon S3 is ideal because it provides:

- Scalability: Automatically handles large volumes of data.

- Durability: Offers high redundancy and backup features.

- Cost-effectiveness: Pay only for what you use.

**File Storage and Retrieval:**

Files will be saved in the cloud storage after upload, with the service generating a unique identifier for each file. The file’s metadata will be stored in a database, linking it to its location in the cloud.

To retrieve a file, the service will look up the metadata in the database to find the file's location and then fetch it from the cloud storage.

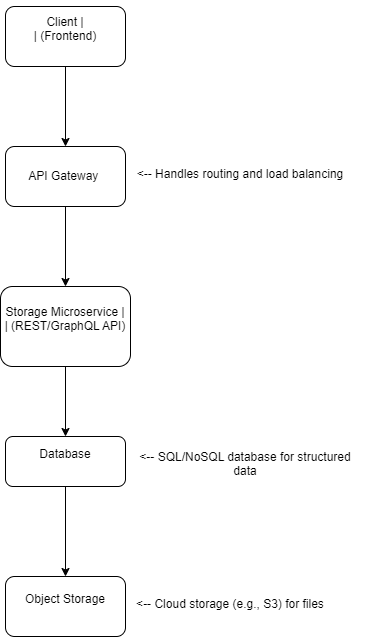
**4)How the other microservice will communicate with the new designed one?**

Other microservices will communicate with the storage microservice through RESTful API calls. They’ll send requests to the endpoints mentioned earlier to upload, retrieve, or delete files. Using a service discovery tool can help manage these interactions more effectively

**Diagrams:**

**1. Architecture Diagram**

This diagram shows the high-level components of your storage microservice and how they interact.



**2. API Endpoints Diagram**

The main API endpoints and their operations.

POST /items <-- Upload a new file

GET /items <-- Retrieve a list of files

GET /items/{id} <-- Retrieve a specific file

DELETE /items/{id} <-- Delete a specific file