Software Requirements & Design Specification

For

MinIO Data Search

Project Category : Data Search

Team Name : Celeron

Team Members:

Hari Krishnan UM	-	2021202022
Manu Gupta	-	2021202025
Shaon Dasgupta	-	2021201068

Table Of Contents

Software Requirements

1. Introduction

The goal of this project is to design and implement a proof of concept (POC) of a Data Search Engine that exposes a Data Search API using ElasticSearch over Healthcare, Mobility and some other miscellaneous datasets and their corresponding metadata stored in the MinIO data storage. One of the core components of our design would be to create relevant and appropriate data classes for the different metadata types that would allow useful and efficient search capabilities. Along with this, our goal is also to provide some additional functionalities to upload and manage these datasets on the MinIO storage. The outcome of this project would be a streamlined process that reduces the time and effort required to manage and access large amounts of data.
The API exposed by our application will be documented using Swagger that would provide the necessary details of how to use them to retrieve the required data using search queries.

2. Overall Description

The Data Search Engine is a proof of concept project aimed at streamlining the process of managing and accessing large amounts of healthcare, mobility, and miscellaneous data. The project will expose a Data Search API using ElasticSearch, which will allow users to search and filter datasets stored in the MinIO data storage. Additionally, the project will include functionalities for uploading and managing datasets, as well as tracking the status of the upload and indexing process.

The Data Search Engine will be built using a combination of Python and Node.js, and will make use of MinIO, RabbitMQ, and Elasticsearch to provide the necessary storage, messaging, and search capabilities. The project will be deployed using Docker to ensure that the environment is easily scalable and manageable.

The building blocks of the project include modules for user management, dataset upload, search and filter API, backend indexing, MinIO integration, RabbitMQ integration, and Elasticsearch integration. These building blocks will be developed and tested independently to ensure a modular and scalable design.

Overall, the Data Search Engine aims to provide a streamlined and efficient process for managing and accessing large amounts of data, making it easier for users to find the data they need and reducing the time and effort required to manage and access this data.

3. System Features

- **Data Upload**: The system provides an easy-to-use interface for uploading datasets to MinIO.

- **Metadata Indexing**: The associated metadata of the uploaded datasets is automatically indexed in Elasticsearch for efficient data retrieval.

- **Search and Retrieval**: The system provides an API that allows users to search and retrieve their data using Elasticsearch.

4. Non Functional Requirements

- **Performance**: The system should be able to handle large amounts of data and perform efficiently, with quick response times for data searches and retrievals.

- **Scalability**: The system should be able to scale to meet the changing needs of its users, including the ability to store increasing amounts of data.

- **Security**: The system should implement appropriate security measures to protect users' data, such as encryption and secure access controls.

- **Availability**: The system should have high availability, ensuring that users can access their data whenever they need it.

- **Interoperability**: The system should be able to integrate with other software solutions, allowing users to access their data in a variety of ways.

- **User-friendly**: The system should be easy to use, with a user-friendly interface and intuitive functionality.

- **Reliability**: The system should be reliable, with minimal downtime and a high degree of stability.

- **Compatibility**: The system should be compatible with various data formats, allowing users to upload and access their data in a variety of formats.

- **Maintainability**: The system should be easy to maintain and upgrade, with clear documentation and a robust architecture.

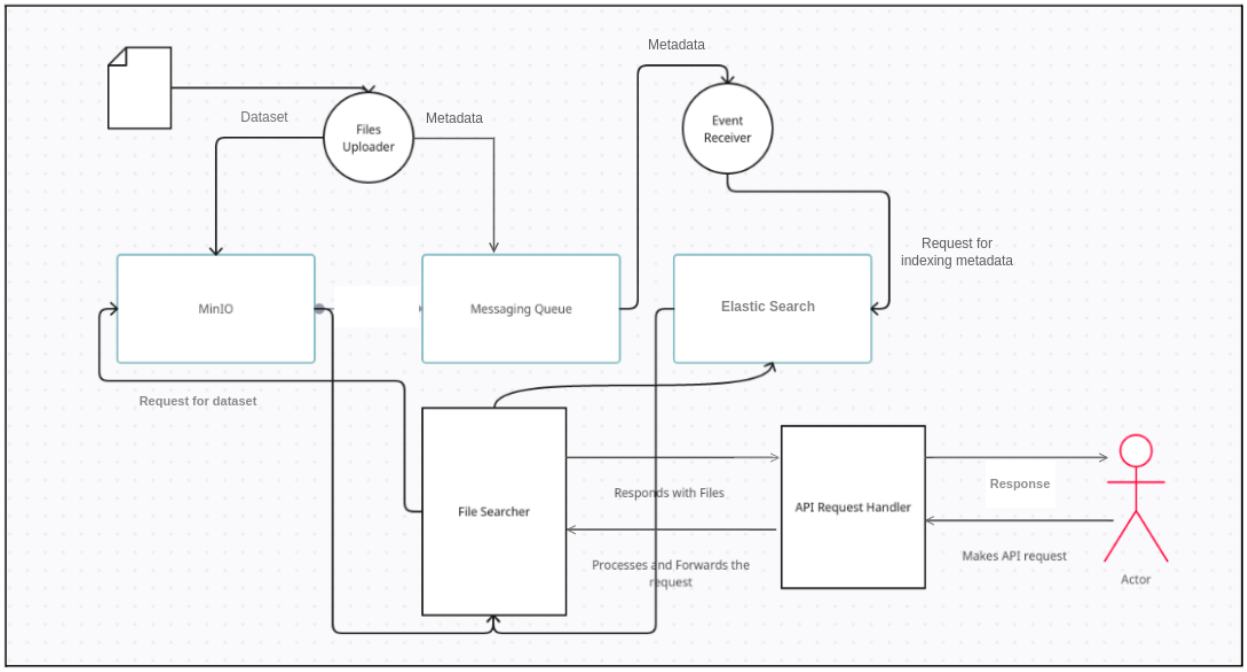
Design Specification

1. High Level Design

On a high level the design consists of mainly 2 workflows:

- Datasets Upload: Uploader will upload data to MinIO, as data is uploaded, an event will be triggered and added to a Messaging Queue, There will be an event listener which will then take data from the MQ and add it to Elasticsearch service for future queries.

- When an API call is made by the user, it will first go to API handler where the processing of query will take place, it will then forward the request to File Searcher which will then communicate with Elastic Search to get the Object IDs, which can then be used to fetch data from MinIO. It will then return the result from MinIO to the API handler and finally to the end user.



2. Proposed Design

2.1 Project Workflow

- Information Gathering: Team needs to meet with various stakeholders and aggregate various collected data and metadata from them, these objects will then be used to create relevant classes.

- Dataset upload: Users can upload datasets to MinIO using the Upload Manager SDK. The SDK will write metadata information to a queue.

- Backend indexing: The backend, acting as the consumer of the queue, will start the indexing process in Elasticsearch. The backend will also update the status of the dataset in the database.

- Search and filter API: The backend will provide APIs for searching and filtering the datasets. Users can use these APIs to retrieve the datasets and data they need.

2.2 Building Blocks

- Dataset upload: This module will handle the upload of datasets to MinIO, including the management of metadata information and the triggering of the indexing process.

- Search and filter API: This module will provide APIs for searching and filtering datasets stored in Elasticsearch.

- Backend indexing: This module will handle the indexing of datasets in Elasticsearch, including the management of the queue and the updating of the database with status information.

- MinIO integration: This module will handle the integration with MinIO, including the storage and retrieval of datasets and metadata information.

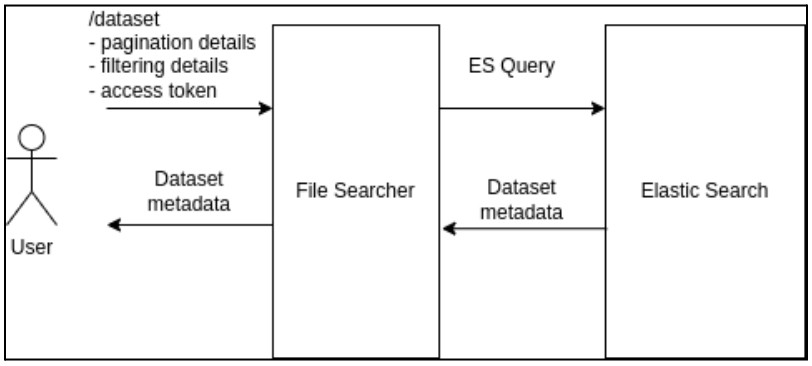
- RabbitMQ integration: This module will handle the integration with RabbitMQ, including the management of the queue used for triggering the indexing process.

- Elasticsearch integration: This module will handle the integration with Elasticsearch, including the indexing and searching of datasets.

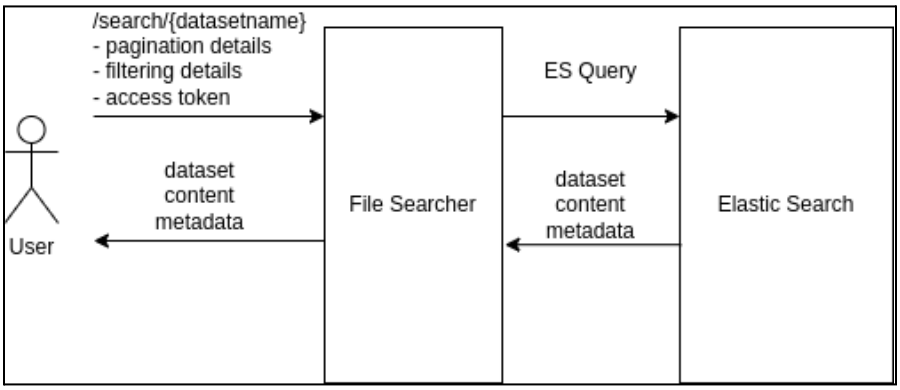
3. High Level API Design

The API interface will consist of the following endpoints:

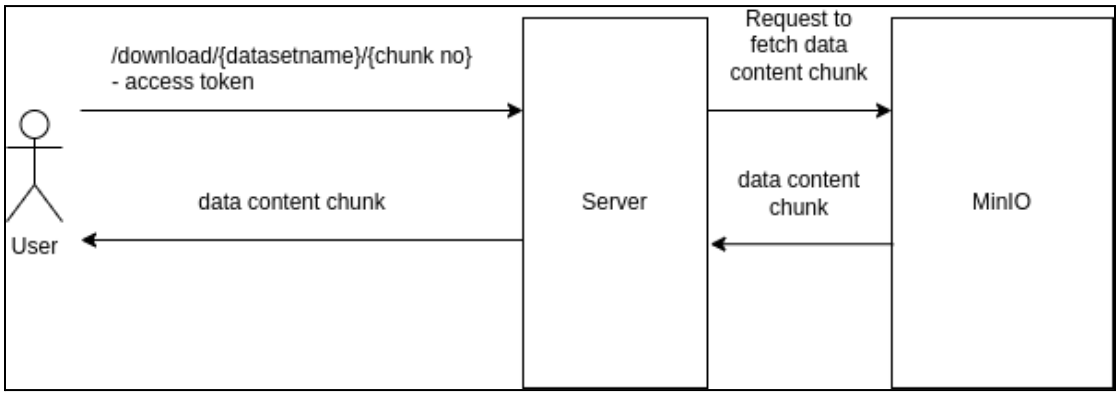
- /datasets : The body of the query should consist of access token (if required), information about the category of images, fine grained filters (if any), any other misc. information required to process the query.



- /search/{datasetname} : This query can be used to search and filter inside a particular dataset. This would handle pagination and filtering along with querying using a search text.



- /download/{datasetname}/{chunkno} : This query can be used to download the dataset content chunk wise.



4. Database Models

As the data is stored in MinIO and Elasticsearch is used for indexing and search, a traditional database may not be required for this project. However, for storing additional information related to the datasets, such as dataset information, a database can be used to store this information.

- Table : Datasets
    - id: unique identifier for each dataset
    - user_id: identifier of the user who uploaded the dataset
    - name: name of the dataset
    - description: description of the dataset
    - date_uploaded: date and time the dataset was uploaded
    - status: status of the dataset (e.g. uploaded, indexed, failed)

5. Tools/Libraries to be used

- Upload Manager SDK:
    - Python: The Upload Manager SDK will be developed using the Python
    - MinIO (Python SDK): The MinIO Python SDK will be used to interact with MinIO and perform operations such as uploading datasets.
    - RabbitMQ (Python SDK): The RabbitMQ Python SDK will be used to interact with RabbitMQ and write metadata information to a queue.
- Backend:
    - Node.js: The backend will be developed using the Node.js
    - MinIO (Node.js SDK): The MinIO Node.js SDK will be used to interact with MinIO and retrieve datasets.
    - RabbitMQ (Node.js SDK): The RabbitMQ Node.js SDK will be used to interact with RabbitMQ and receive metadata information from the Upload Manager SDK.
    - Elasticsearch (Node.js): The Elasticsearch Node.js library will be used to interact with Elasticsearch and perform indexing and search operations.
- Server Side:
    - Docker: Docker will be used to deploy and run the Minio, RabbitMQ and Elasticsearch on a server. Docker containers will be used to ensure that the components are isolated and have the required dependencies.

Overall, the project will use a combination of Python and Node.js for the development of the Upload Manager SDK and the Backend respectively. MinIO, RabbitMQ, and Elasticsearch will be used for data storage, messaging, and indexing respectively. Docker will be used to deploy and run the components on a server.

6. Deployment Models

- Upload Manager SDK (Python Module): The Upload Manager SDK will be used to upload datasets to MinIO. It will write metadata information to a queue once the upload is completed. The backend (node.js) will act as the consumer of the queue and start the Elasticsearch indexing process. The Upload Manager SDK will also provide status information (pending/done/failed) to the backend.

- Backend (Node.js): The backend will provide search and filter APIs to users. It will act as the consumer of the queue triggered by the Upload Manager SDK. The backend will start the Elasticsearch indexing process and provide status information (upload and indexing status) to the users.

Overall, the deployment architecture will follow a pipeline approach where data is first uploaded to MinIO using the Upload Manager SDK and then indexed in Elasticsearch by the backend. The backend will also provide the APIs for search and filter and keep track of the status of the upload and indexing process.

7. References:

https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1232049&dswid=3885
https://www.elastic.co/guide/index.html
https://min.io/docs/minio/linux/index.html