SYSTEM OVERVIEW

File management / Visualization by ai technology

A File Management and Visualization System that allows users to register, login, upload/download files, visualize data, and leverage AI-powered analysis such as embedding search, anomaly detection, and recommendations.

Key Components:

1. **UI Layer (Frontend):**
    - **Technologies:** React.js (or Next.js)
    - **Features:**
        - **Login/Registration:** User authentication with JWT or OAuth (Google).
        - **Dashboard:** Display file analytics, charts, visualizations.
        - **Upload/Download Files:** Interface for file management.
        - **Visualization:** Charts for file analytics using OpenSearch Dashboard or other visualization tools.
        - **Search:** Search interface that allows users to search indexed files by metadata or embeddings.
2. **Node.js API (Backend 1):**
    - **Technologies:** Node.js (Express.js), MongoDB
    - **Responsibilities:**
        - **User Authentication:** Handle login, registration, and JWT generation.
        - **File Metadata Storage:** Store and manage metadata (e.g., file name, type, size, user permissions) in MongoDB.
        - **File Handling API:** Handle file uploads/downloads using cloud storage (e.g., AWS S3, GCP Cloud Storage).
        - **Search Requests:** Pass search requests (metadata-based or embedding-based) to the OpenSearch backend.
3. **Flask API (Backend 2):**
    - **Technologies:** Python, TensorFlow, Transformers, Keras, Numpy
    - **Responsibilities:**

        - **Feature Extraction:** Extract embeddings from uploaded files (e.g., using Transformers or TensorFlow/Keras for image or document files).
        - **Machine Learning Tasks:** Process data for embedding generation, anomaly detection, and AI-driven recommendations.
        - **Communication:** Interface with Node.js API for tasks requiring machine learning (e.g., running TensorFlow models).

4. **Storage Layer:**
    - **Technologies:** Cloud-based storage (AWS S3, Google Cloud Storage, or Azure Blob Storage)
    - **Responsibilities:**
        - **File Storage:** Store all uploaded files securely.
        - **Versioning & Backup:** Ensure file versioning and backups for integrity.
        - **Access Control:** Manage access based on user roles or permissions via signed URLs.
5. **OpenSearch (Search and Indexing):**
    - **Technologies:** OpenSearch (formerly Elasticsearch)
    - **Responsibilities:**
        - **Indexing:** Index files for fast retrieval using metadata (filename, tags, etc.) and embeddings (for content-based searches).
        - **Embedding Search:** Allow vector-based searches for similar content using the embeddings generated by the Flask API.
        - **Analytics:** Perform various analytics such as aggregations and custom queries for data insights.
        - **Visualization:** Use **OpenSearch Dashboards (Kibana)** for real-time data visualizations and analytics.
6. **OpenSearch Dashboard (Visualization and AI Assistance):**
    - **Technologies:** OpenSearch Dashboards (Kibana), Machine Learning Modules
    - **Responsibilities:**
        - **Visualization:** Provide real-time visualizations of search queries, file statistics, and analytics.
        - **Anomaly Detection:** Detect unusual patterns in file usage or file content (based on embeddings or other metadata).
        - **Recommendation Engine:** Use AI-driven models (via Flask) to suggest relevant content to users based on their file interactions.
        - **Chat Assistant:** Integrate a chat-based interface powered by OpenSearch for answering user queries or interacting with data.

System Workflow:

1. **User Interaction (UI Layer)**
    - The user logs in or registers using the **UI Layer** (React.js or Next.js) connected to the **Node.js API**.
    - The dashboard shows file analytics, visualizations, and search options, providing the ability to upload/download files.
2. **File Upload and Processing**
    - Upon file upload, the **Node.js API** handles the metadata storage in **MongoDB** and the actual file storage in **Cloud Storage** (e.g., AWS S3).
    - The file is sent to the **Flask API** for feature extraction. This API uses **TensorFlow**, **Keras**, and **Transformers** to extract embeddings from files (images, PDFs, etc.).
3. **Indexing Files in OpenSearch**
    - The **Flask API** generates embeddings and returns them to the **Node.js API**.
    - The **Node.js API** sends the metadata and embeddings to **OpenSearch** for indexing.
    - OpenSearch indexes both the metadata and the embeddings for fast retrieval and similarity searches.
4. **Search and Visualization**
    - Users can search files by metadata (filename, tags, etc.) or by embeddings (content similarity).
    - **OpenSearch Dashboards** provides visualizations of the indexed data, allowing users to view search results and file analytics.
    - Anomaly detection and recommendations are also managed via **OpenSearch Dashboards**, powered by the **Flask API** for more complex AI tasks.
5. **Real-time Recommendations and Anomaly Detection**
    - Based on file usage patterns and embedding searches, the system can recommend files or detect anomalies.
    - These insights are visualized in the **OpenSearch Dashboard**, providing real-time feedback to users.

Technology Stack Summary:

- **Frontend (UI Layer):**
    - **React.js** or **Next.js** for user interface
    - **JWT/OAuth** for authentication

    - **Chart.js** or **D3.js** for additional custom visualizations
- **Backend 1 (Node.js API):**
    - **Node.js** (Express.js) for handling file metadata, file uploads/downloads, user authentication, and search queries.
    - **MongoDB** for metadata storage.
    - **AWS S3/GCS** for file storage.
- **Backend 2 (Flask API):**
    - **Flask** for handling machine learning tasks such as feature extraction.
    - **TensorFlow**, **Transformers**, **Keras** for ML models (e.g., embeddings generation, anomaly detection).
- **Search and Analytics:**
    - **OpenSearch** for indexing and embedding search.
    - **OpenSearch Dashboards (Kibana)** for data visualization, anomaly detection, and real-time analytics.
- **File Storage:**
    - **AWS S3**, **Google Cloud Storage**, or **Azure Blob Storage** for storing user-uploaded files with signed URLs for secure access.

<u>Additional Considerations:</u>

1. **Scalability:** Ensure both the Flask API (for ML tasks) and OpenSearch can scale horizontally to handle large volumes of data and concurrent searches.
2. **Security:** Implement proper authentication and authorization for file uploads/downloads and restrict access to OpenSearch.
3. **Performance Optimization:** Cache frequently accessed files and use batch processing for embedding generation.
4. **Monitoring and Logging:** Use **OpenSearch Dashboards** for monitoring search queries and system performance.