Document Management System (DMS)

To successfully complete this MERN stack project, we will develop a system with a set of features that integrate both relational and NoSQL databases, handling user authentication, workspace management, document handling, and more. Below is a detailed outline for each task and feature requirement of the project.

Project Overview

This project is built using the MERN stack (MongoDB, Express.js, React.js, Node.js) and incorporates both relational and NoSQL databases. The application will enable user registration and login, manage workspaces, and handle documents with various operations.

Task 1: Login & Registration (Relational Database)

- **Objective**: Implement user authentication using a relational database to store user information.
- **Requirements**:
  - **Database Setup**:
    - Use a relational database like PostgreSQL or MySQL to store user data.
    - Tables to include: `Users`, which will store email, password, National ID (NID), and personal information.
  - **Registration**:
    - Endpoint to register users by collecting email, password, NID, and personal information.
    - Ensure data validation and password hashing.
  - **Login**:
    - Endpoint to authenticate users via email and password.
    - Implement JWT or session-based authentication for session management.
  - **Security**:
    - Secure password storage using bcrypt.
    - Protect endpoints using middleware to check authentication tokens.

Task 2: Document Workspace Structure (NoSQL Database)

- **Objective**: Create a workspace directory structure using a NoSQL database for flexible data storage.
- **Requirements**:
  - **Database Setup**:
    - Use MongoDB to store workspace and document information.
    - Collections: `Workspaces`, `Documents`.
  - **Workspace Structure**:
    - Each registered user can create a workspace with a unique structure (e.g., School, Internship).

        - Link workspace structure to user profiles using the NID.
    - **APIs**:
        - Create endpoints to create, retrieve, update, and delete workspaces.
        - Ensure workspaces are uniquely tied to user accounts.

Task 3: Uploading Document (Multi-part API)

- **Objective**: Implement a multi-part API for document upload.
- **Requirements**:
    - **API Development**:
        - Create an endpoint to handle multi-part form data for document uploads.
        - Store documents in a file storage system or cloud storage (e.g., AWS S3).
    - **Data Handling**:
        - Save document metadata in the `Documents` collection.
        - Ensure metadata includes document name, type, owner, and workspace association.

Task 4: Download Document (Multi-part API)

- **Objective**: Enable document download functionality.
- **Requirements**:
    - **API Development**:
        - Create an endpoint to download documents based on document ID.
        - Ensure proper authentication and authorization checks.
    - **Integration**:
        - Retrieve documents from storage and serve them to authenticated users.

Task 5: Delete Document (Soft Deletion)

- **Objective**: Implement soft deletion of documents.
- **Requirements**:
    - **API Development**:
        - Create an endpoint to mark documents as deleted without removing them from storage.
        - Add a `deleted` flag in the document metadata.
    - **Data Integrity**:
        - Ensure soft-deleted documents are not visible in the user's active workspace.

Task 6: Preview Document (Base64 API)

- **Objective**: Provide a preview of documents as Base64-encoded data.

- **Requirements**:
  - **API Development**:
    - Create an endpoint to return document data as a Base64 string.
    - Optimize performance for common document types (e.g., PDF, images).
  - **Security**:
    - Ensure that only authorized users can preview documents.

Task 7: Document List

- **Objective**: List documents based on workspace and profile.
- **Requirements**:
  - **API Development**:
    - Create endpoints to list documents for a specific workspace.
    - Allow listing of documents based on the user profile.
  - **Filtering**:
    - Implement filters to sort and search documents by various criteria.

Task 8: Document Metadata

- **Objective**: Manage document metadata efficiently.
- **Requirements**:
  - **Metadata Structure**:
    - Ensure metadata includes document name, type, owner, versioning info, tags, and access controls.
  - **API Development**:
    - Create endpoints to update and retrieve document metadata.

Task 9: Document Search

- **Objective**: Implement a search feature for documents.
- **Requirements**:
  - **Search Functionality**:
    - Allow users to search for documents by name or type.
  - **Performance**:
    - Optimize search queries using MongoDB indexes.

Additional Features

Document Profiling

- Implement profiling to categorize documents based on user-defined attributes.

Document Versioning

- Enable version control for documents to track changes and restore previous versions.

Document Tags

- Allow users to tag documents with relevant keywords for easier categorization and searchability.

Document Indexing

- Index documents in MongoDB for improved search performance and quick access.

Document Access Control

- Implement granular access controls to restrict document visibility and actions based on user roles and permissions.

Implementation Notes

- **Frontend**: Use React.js to create an intuitive user interface with appropriate components for each feature.
- **Backend**: Develop APIs using Node.js and Express.js to handle requests and interact with databases.
- **Security**: Implement robust security measures, including input validation, data encryption, and secure authentication.
- **Testing**: Write unit and integration tests to ensure the reliability of each feature and endpoint.