MOBISEC TECHNOLOGIES PVT LTD

Practical Coding Evaluation (Backend)

Duration: 1.5 Hours
Tools: Node.js, Express.js, MongoDB, JWT, Multer (for file uploads)
Objective: Assess the ability to create a backend system with secure APIs, handle file uploads, and retrieve files.

Problem Statement

You are tasked with creating a backend API for a **"Document Management System"**. This system allows users to register, log in, and securely upload and manage their documents.

Requirements

1. User Authentication

1. **Register User**
   - Create a `POST /api/register` endpoint.
   - Accept the following fields: `name, email, password`.
   - Hash the password before storing it in MongoDB.
   - Validate that the `email` is unique.
   - Return a success message upon successful registration.
2. **Login User**
   - Create a `POST /api/login` endpoint.
   - Accept `email` and `password`.
   - Verify the credentials and return a JWT token if valid.
   - Return an error if the credentials are invalid.

2. Document Management

1. **Upload Document**
   - Create a `POST /api/documents` endpoint.
   - Implement functionality to upload files using the `multer` middleware to handle multipart/form-data.
   - Accept a JWT token in the `Authorization` header.
   - Accept a `file` (document) and `metadata` fields in the request body (e.g., `title, description`).
   - Validate that the file is in PDF or DOC/DOCX format.
   - Save the file to the server's local file system or a folder like `uploads/`.
   - Store the file's metadata (e.g., `title, description, upload date, file path`) in MongoDB, linked to the authenticated user.

2. **Retrieve All Documents**
    - Create a `GET /api/documents` endpoint.
    - Accept a JWT token in the `Authorization` header.
    - Return all documents and their metadata for the authenticated user, excluding file content.
3. **Download Document**
    - Create a `GET /api/documents/:id` endpoint.
    - Accept a JWT token in the `Authorization` header.
    - Retrieve the document by its `id` and allow the user to download it.
    - Validate that the document belongs to the authenticated user.
4. **Delete Document**
    - Create a `DELETE /api/documents/:id` endpoint.
    - Accept a JWT token in the `Authorization` header.
    - Delete the document from both the server (file system) and MongoDB, if it belongs to the authenticated user.

Setup Instructions

1. Use **Node.js** and **Express.js** to create the server.
2. Use **MongoDB** as the database. You may use an online service like MongoDB Atlas.
3. Use **JWT** for securing routes.
4. Use **Multer** for handling file uploads.
5. Ensure proper error handling (e.g., invalid input, unauthorized access, etc.).

Evaluation Criteria

1. **Functionality**: All endpoints should work as specified.
2. **File Handling**: Proper implementation of file upload, storage, and retrieval.
3. **Code Quality**: Clean, readable, and well-structured code.
4. **Security**: Proper use of JWT for authentication and authorization, and secure file handling.
5. **Validation**: Input validation, including file type and size checks, and error handling.
6. **Database Design**: Logical schema for users and documents, storing metadata and file paths efficiently.
7. **Testing**: Test the endpoints using Postman or similar tools.

Demonstration & Submission

Demonstrate your solution and send the zip file containing your sources to `contact@mobisec.in`