ADSIDUOUS

Technical Assessment Task: Build a Multimedia Upload & Search

Task Overview

Build a scalable web app that allows users to:

1. Upload and preview multimedia files (images, videos, audio, PDFs) with secure authentication.
2. Search for uploaded files using keywords (e.g., file name, tags).
3. Rank search results by relevance (e.g., view count, upload date, tags).
4. Deploy the app with a live demo link.

Tech Stack Requirements

- Frontend: React.js (Hooks, Redux) + CSS3/SASS
- Backend: Node.js + Express.js
- Database: MongoDB (Atlas) for file metadata, Cloudinary for media storage
- Authentication: JWT (JSON Web Tokens)
- Tools: Git/GitHub, Postman, Swagger API docs

Deliverables

Submit a GitHub repository with:

1. Working Codebase:
   - Clean, modular code with proper folder structure.
   - Frontend (React) and backend (Node/Express) in separate folders.
2. API Documentation:
   - Swagger/OpenAPI spec for all endpoints.
   - Example: POST /upload, GET /search?query=video, GET /files/:id.
3. JWT Authentication:
   - User registration/login flow with secure token storage (HTTP-only cookies or localStorage).
   - Protected routes for uploads/search (only authenticated users).
4. Search & Ranking Logic:
   - Implement basic ranking (e.g., view count, upload date).
   - Bonus: Use relevance scoring (e.g., keyword matching in file names/tags).
5. Cloudinary Integration:
   - Upload files to Cloudinary and store metadata (URL, file type, size) in MongoDB.
   - Preview files directly from Cloudinary URLs.

6. Error Handling:
    - Graceful error messages for invalid uploads, authentication failures, and API errors.
7. README.md:
    - Instructions to run the app locally.
    - Deployment link (e.g., Vercel/Railway).

Evaluation Criteria

1. Code Quality (30%):
    - Readability, modularity, and adherence to best practices (e.g., DRY principles, proper naming).
    - Use of React Hooks/Redux and Express middleware correctly.
2. Functionality (30%):
    - All core features work as described.
    - File uploads/preview, search, and ranking logic are functional.
3. Security (20%):
    - Secure JWT implementation (e.g., token expiration, refresh tokens).
    - Input validation (e.g., file type/size limits).
4. Documentation (10%):
    - Clear README and Swagger API docs.
5. Testing & Deployment (10%):
    - Basic unit/integration tests (e.g., Jest/Mocha).
    - Deployed app with a working demo link.

Optional Stretch Goals (Bonus Points)

1. Real-Time Updates:
    - Notify users when a file is uploaded (e.g., WebSocket integration).
2. Advanced Search:
    - Implement filters (e.g., file type, date range) or fuzzy search.
3. Ranking Algorithm

Submission Guidelines

- Push code to a public GitHub repo and share the link.
- Include a live demo link (e.g., Vercel/Railway).
- Add notes in the README for any incomplete features or assumptions.
