**Full Project Roadmap: Peer-to-Peer Academic Resource Sharing Platform (8-Day Lean MVP)**

**Project Goal:** A web platform where authenticated users can upload academic files, see their own files, and download files uploaded by others. This is a highly simplified "Google Drive" concept.

**Tech Stack (Node.js/Express Backend, React Frontend):**

* **Backend:** Node.js, Express.js, multer (for file uploads), bcryptjs (for password hashing), jsonwebtoken (for user tokens).
* **Frontend:** React.js (using a free template), axios (for API calls).
* **Data Storage (MVP $0 Budget):** User data in a local users.json file. Files stored directly on the server's file system.
* **Hosting:** Free tiers of services like Render/Railway (for backend) and Vercel/Netlify (for React frontend) – *Be mindful of their strict free tier limits.*

**Step 1: Backend Foundation (User Management & Server Setup)**

* **Target Completion:** Day 1-2 (Already detailed in previous response)
* **Key Tasks:**
  1. Project Initialization (npm init -y, install express, bcryptjs, jsonwebtoken).
  2. Basic Express server setup (server.js).
  3. File-based "DB" for users (data/users.json).
  4. User registration endpoint (/api/register) with password hashing.
  5. User login endpoint (/api/login) with password comparison and JWT generation.
  6. Basic JWT verification middleware (middleware/auth.js) for protected routes.
* **Expected Output:** A running Node.js server that can register and log in users, returning a JWT token on successful login.

**Step 2: File Upload Backend Logic & Storage**

* **Target Completion:** Day 2-3
* **Key Tasks:**
  1. **Install multer:** npm install multer
  2. **Configure Multer:**
     + Define upload directory (e.g., uploads/).
     + Set up Multer storage options (destination and filename generation). Consider making filenames unique (e.g., by prepending a timestamp or UUID) to avoid clashes.
  3. **Implement File Upload Endpoint (/api/upload):**
     + Create an Express POST route for file uploads.
     + Use your JWT verification middleware to protect this route (only logged-in users can upload).
     + Integrate multer to handle the file saving.
     + When a file is uploaded, **store metadata** (original filename, new filename, uploader's username/ID) in a separate data/files.json file. This is crucial for listing and associating files with users.
     + Send appropriate success/error responses.
* **Expected Output:** A protected backend endpoint that successfully receives files, saves them to a designated folder, and records their metadata to files.json.

**Step 3: Frontend Setup & User Authentication Integration (React)**

* **Target Completion:** Day 3-4
* **Key Tasks:**
  1. **React App Creation:** npx create-react-app client (or npx vite@latest client --template react for faster setup).
  2. **Install axios:** cd client && npm install axios
  3. **Choose/Integrate Free React Template:** Find a simple, clean, free React template (e.g., on GitHub, free theme sites). Copy its components/styles into your client directory. Focus on templates with basic layouts (navbar, forms, lists). *Do not spend too much time on design here; prioritize functionality.*
  4. **Backend Integration (CORS):** In your Node.js server.js, add cors middleware (npm install cors, then app.use(cors());) to allow your React app to make requests to the backend.
  5. **Login Component:** Create a React component (Login.js) with input fields for username and password.
     + On submit, use axios to POST data to /api/login.
     + If successful, store the received JWT token in localStorage.
     + Redirect the user to a "Dashboard" or "Home" page.
  6. **Register Component:** Create a Register.js component, similar to login, using axios to POST to /api/register. Handle success/error.
  7. **Authentication Context/State:** Implement a simple React Context or global state management (e.g., React's useState and useContext) to manage the user's logged-in status and JWT token across components. This avoids prop-drilling.
  8. **Protected Routes:** Use React Router (npm install react-router-dom) to define routes. Implement a simple protected route logic that checks for a valid token in localStorage before rendering private components.
* **Expected Output:** A basic React app with working login and registration forms. Successful login redirects to a protected page.

**Step 4: Frontend File Upload & User-Specific Listing**

* **Target Completion:** Day 5-6
* **Key Tasks:**
  1. **Protected Upload Component:** Create a React component (e.g., UploadForm.js) accessible only to logged-in users.
     + Include an <input type="file"> and a submit button.
     + Use axios to POST the file to your /api/upload endpoint. You'll need to send it as FormData.
     + Include the JWT token in the Authorization header of the request (Bearer <token>).
     + Display success/error messages after upload.
  2. **Backend Endpoint for User's Files (/api/my-files):**
     + Create a **protected** Express GET route.
     + Use the JWT verification middleware to get the current user's ID/username.
     + Read data/files.json and filter files to return only those uploaded by the current user.
  3. **My Files Component:** Create a React component (e.g., MyFiles.js) that:
     + Fetches the current user's files from /api/my-files using axios (with JWT in headers).
     + Displays them in a list with their original filenames.
* **Expected Output:** Authenticated users can upload files, and then see a list of files they have personally uploaded.

**Step 5: Frontend Global File Listing & Download**

* **Target Completion:** Day 6-7
* **Key Tasks:**
  1. **Backend Endpoint for All Files (/api/all-files):**
     + Create an Express GET route (can be public or protected, your choice for MVP).
     + Read data/files.json and return the entire list of file metadata.
  2. **Global Files Component:** Create a React component (e.g., AllFiles.js) to display all uploaded files.
     + Fetches data from /api/all-files.
     + Displays the files in a list.
  3. **Download Functionality:**
     + In both MyFiles.js and AllFiles.js, make each file name a clickable link.
     + The link should point to your backend download route (e.g., /download/:filename). When clicked, the browser will handle the download.
  4. **Navigation:** Ensure your React app has clear navigation (e.g., using react-router-dom) between Login/Register, Upload, My Files, and All Files pages.
* **Expected Output:** Users can view all uploaded files (global list) and download any file from either their own list or the global list.

**Step 6: Basic Styling, User Experience & Error Handling**

* **Target Completion:** Day 7
* **Key Tasks:**
  1. **Template Customization:** Integrate your chosen React template's CSS/components. Focus on making forms clear and lists readable.
  2. **UI/UX Refinements:**
     + Add clear labels and placeholders to all forms.
     + Provide user feedback (e.g., "Upload successful!" messages, loading spinners).
     + Ensure responsive design so it looks decent on mobile and desktop.
  3. **Frontend Error Handling:**
     + Display user-friendly error messages for failed API requests (e.g., "Registration failed," "File upload failed").
     + Handle network errors gracefully.
  4. **Cleanup & Comments:** Remove unnecessary boilerplate code from the template. Add comments to your custom React code and backend code for clarity.
* **Expected Output:** A functional application with a user-friendly interface that provides feedback and handles common errors gracefully.

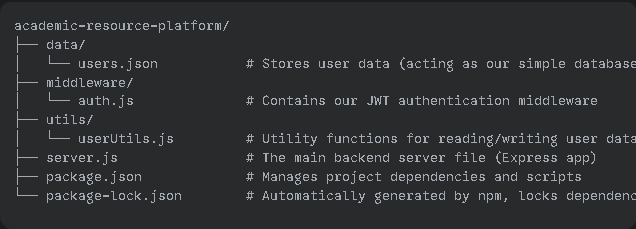
**Step 7: Deployment Preparation & Go-Live**

* **Target Completion:** Day 8
* **Key Tasks:**
  1. **Version Control:** Ensure all your code is in a Git repository (e.g., GitHub). This is crucial for deployment.
  2. **Backend Deployment (Node.js):**
     + Prepare your server.js for production (e.g., ensuring port is set by environment variable, using process.env.PORT).
     + Choose a free hosting provider for Node.js (e.g., Render.com, Railway.app – *check their free tier limits carefully as file storage can be an issue*).
     + Follow the provider's deployment guide to deploy your backend from your Git repository.
  3. **Frontend Deployment (React):**
     + Build your React app for production: npm run build in your client directory.
     + Choose a free static site hosting provider (e.g., Vercel, Netlify). These are excellent for React builds.
     + Follow their guide to deploy your build folder from your Git repository.
  4. **Environment Variables:** Configure environment variables for your backend (e.g., JWT\_SECRET) on your hosting platform, not directly in code.
  5. **CORS Update:** Update your backend's CORS configuration to explicitly allow requests from your deployed React frontend URL.
  6. **Final Testing:** Test the live application end-to-end (register, login, upload, view, download) in a production environment.
* **Expected Output:** A live, accessible web application deployed online, functional for user registration, login, file upload, viewing, and download.

This is a very condensed plan. You'll be moving fast, and encountering issues is part of the process. Remember to break down each day's tasks into even smaller, manageable chunks. Good luck!

**From Here we start the work**

Project File Structure



The JWT key is : If hate is real, then the pain you feel is justified. As emotions assures your humanity.

**Stories hidden within it’s roots**