Overview

Cloud based Media Files Storage Service – Detailed Project Specification - Web

> A full, practical spec you can hand to a team or use solo. It covers goals, scope, user stories, architecture, schema, APIs, security, deployment, and a milestone plan. Tailored for a Next.js frontend, Node/Express backend, PostgreSQL (via Supabase) and object storage (Supabase Storage or S3).

---

1) Executive Overview

A cloud file storage and sharing web app with auth, folders, file upload/download, search, and granular sharing. Think “Google Drive core” with a clean UI, strong access controls, and scalable storage. The MVP focuses on reliability and simplicity; advanced features (versioning, previews, activity logs) are staged for later sprints.

Primary stakeholders: end users (individuals/teams), admin (platform owner).

Non-goals (MVP): Office editors, real-time doc co-editing, complex org hierarchy, desktop sync client.

---

2) Feature Scope

2.1 Core (MVP)

- Email/OAuth authentication, sessions, and profile.
- Folder CRUD (create, rename, move, delete), hierarchical tree, breadcrumbs.
- File upload (drag & drop, picker), download, rename, move, delete.
- Share to users (Viewer/Editor), revoke, list who has access.
- Public share links with expiry (optional password).
- Search by name/type/owner, sort and filters.

- Recent files, Starred/Favorites. (Optional)
- Trash with retention window (e.g., 30 days) and restore.

2.2 Phase 2+ (Nice-to-have)

- Version history (keep current pointer + archived versions).
- File previews (images, PDFs, basic text) with thumbnails.
- Activity/audit log.
- Tags/labels, bulk actions, keyboard shortcuts.
- Advanced search (content indexing later), shared drives/teams.
- Quotas and usage dashboard.

---

3) User Roles & Permissions

- **Owner**: full control over their files/folders.
- **Editor**: upload, edit metadata, move, delete within shared items.
- **Viewer**: read/download only.
- **Public link holder**: constrained by link settings (view-only by default), optional password, expiry.

Permission checks run **server-side** for every operation, enforced at API and storage-layer policy.

---

4) User Stories (selected)

- As a user, I can sign up/sign in and stay logged in securely.
- As a user, I can upload files via drag & drop and see progress.
- As a user, I can create folders, move items, and navigate with breadcrumbs.
- As a user, I can share a folder with a teammate as Viewer/Editor.
- As a user, I can generate a public link with expiry to share a file.
- As a user, I can search across my files and sort by name/date/size.
- As a user, I can star important items and find them quickly.
- As a user, I can restore a deleted item from Trash within 30 days.

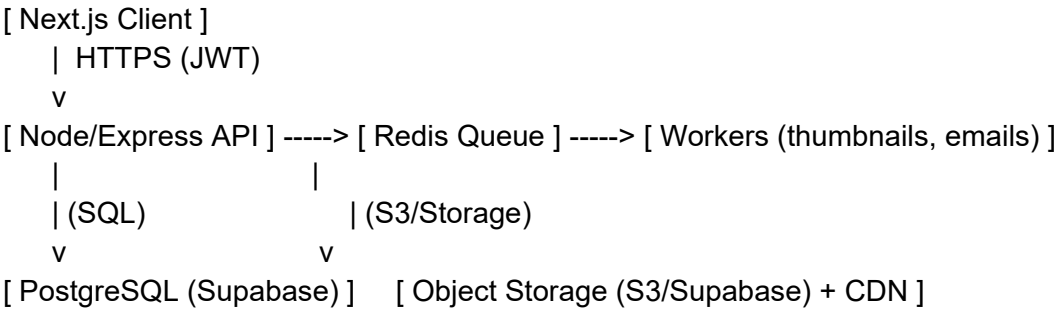
---

5) System Architecture

5.1 Recommended Stack

- **Frontend:** Next.js (App Router), React, Tailwind CSS, TanStack Query, React DnD, Upload library (Uppy/Tus or native + presigned URLs).
- **Backend:** Node.js + Express.js (REST), Multer (small uploads) or presigned URL flow for large uploads.
- **Database:** PostgreSQL (Supabase managed) for metadata, relations, ACL, and search indices.
- **Object Storage:** Supabase Storage (RLS policies) or Apprite or Convex or Firebase
- **Auth:** Supabase Auth or Clerk/NextAuth (JWT + refresh, httpOnly cookies).
- **CDN:** Supabase CDN or CloudFront for file delivery.
- **Background tasks:** lightweight queue (BullMQ + Redis) for thumbnails, virus scan (optional), email invites.

5.2 High-Level Diagram (ASCII)



5.3 Key Flows (Sequence)

Upload (presigned/multipart):

1. Client requests upload-init → API checks auth/ACL → creates DB placeholder row (status: "uploading"), returns key + presigned URLs (or upload token for Supabase Storage).
2. Client uploads parts directly to storage (progress tracked client-side).
3. Client calls upload-complete → API verifies parts/etag → finalizes DB row (status: "ready"), enqueues thumbnail job.

Download: Client requests → API checks ACL → returns short-lived signed URL → client downloads from CDN.

Share: Owner POST /shares → create ACL entry (user_id/role or link token with expiry/password hash).

Search: GET /search?q=...&filters=... → uses indexed columns and Postgres full-text/trigram for fast results.

---

6) Data Model (PostgreSQL ERD)

6.1 Tables

users

- id (uuid, pk)
- email (text, unique, indexed)
- name (text)
- image_url (text, nullable)
- created_at (timestamptz, default now())

folders

- id (uuid, pk)
- name (text, indexed)
- owner_id (uuid → users.id, indexed)
- parent_id (uuid → folders.id, nullable, indexed)
- is_deleted (bool, default false)
- created_at (timestamptz)
- updated_at (timestamptz)

files

- id (uuid, pk)
- name (text, indexed)

- mime_type (text)
- size_bytes (bigint)
- storage_key (text, unique) // path in bucket
- owner_id (uuid → users.id, indexed)
- folder_id (uuid → folders.id, nullable, indexed)
- version_id (uuid → file_versions.id, nullable) // pointer to current
- checksum (text, nullable) // md5/sha256
- is_deleted (bool, default false)
- created_at, updated_at (timestamptz)

file_versions

- id (uuid, pk)
- file_id (uuid → files.id, indexed)
- version_number (int)
- storage_key (text)
- size_bytes (bigint)
- checksum (text, nullable)
- created_at (timestamptz)

shares (per-user ACL)

- id (uuid, pk)
- resource_type (enum: 'file' | 'folder')

- resource_id (uuid)
- grantee_user_id (uuid → users.id)
- role (enum: 'viewer' | 'editor')
- created_by (uuid → users.id)
- created_at (timestamptz)
- unique(resource_type, resource_id, grantee_user_id)

link_shares (public links)

- id (uuid, pk)
- resource_type (enum)
- resource_id (uuid)
- token (text, unique, indexed)
- role (enum: 'viewer')
- password_hash (text, nullable)
- expires_at (timestamptz, nullable)
- created_by (uuid)
- created_at (timestamptz)

stars

- user_id (uuid)
- resource_type (enum)
- resource_id (uuid)
- pk (user_id, resource_type, resource_id)

activities

- id (uuid, pk)
- actor_id (uuid)
- action (enum: 'upload'|'rename'|'delete'|'restore'|'move'|'share'|'download')
- resource_type (enum)
- resource_id (uuid)
- context (jsonb)
- created_at (timestamptz, indexed)

indexes to add: files(name, owner_id), folders(name, owner_id), files using gin (to_tsvector('simple', name)), activities(created_at desc), shares(resource_type, resource_id), link_shares(token).

6.2 Hierarchy Notes

- Use **adjacency list** via `parent_id` for folders. For path/breadcrumbs, query recursively (CTE) or cache `path` string. Add **constraint** preventing cycles.
- Moving a folder updates `parent_id`. Consider a unique constraint `(owner_id, parent_id, name)` to avoid duplicates at the same level.

---

7) Storage Design

- **Key format:** `tenants/{owner_id}/folders/{folder_id}/files/{file_uuid}-{slug}.{ext}` (versions: append `v{n}`)
- **Multipart uploads:** S3 multipart (5MB+ parts) or Supabase's resumable via Tus/Uppy. Track `upload_id`, `part_numbers`, `etag` per part.
- **Validation:** allowlist mime-types (images/pdf/txt/docx/xlsx), max size per plan.

- **Integrity:** compute MD5/SHA-256 client-side (optional) to detect duplicates; store checksum.
- **Thumbnails/Previews:** background worker uses ImageMagick/PDFium to generate previews into `previews/` prefix.
- **Retention:** Trash = soft delete (`is_deleted = true`) + scheduled purge after N days.

---

8) Security & Compliance Checklist

- **Auth:** httpOnly cookies, short-lived access token + refresh rotation. OAuth providers (Google/GitHub) optional.
- **ACL:** enforce at API and storage (Supabase RLS policies or S3 presigned checks). Never expose raw storage keys.
- **Links:** signed URLs (short TTL). Public link tokens are long, random; optional password with rate limiting.
- **Validation:** Zod/Yup schema on all inputs. Filenames sanitized; no path traversal.
- **Rate limits:** IP + user-based (e.g., 100 req/5 min). Upload init stricter.
- **Headers:** CSP, X-Content-Type-Options, Referrer-Policy, CORS (allow only your origins), download `Content-Disposition`.
- **Secrets:** managed via environment (Vercel/Render), never in repo.
- **Logging:** auth events, share creations, admin actions; PII minimal.
- **Backups:** daily DB backups; lifecycle policies for storage.

---

9) API Design (REST, JSON)

Conventions: snake_case in DB, camelCase in JSON; `createdAt` ISO 8601; pagination via `limit`, `cursor`.

9.1 Auth

- `POST /api/auth/register` → { email, password, name }
- `POST /api/auth/login` → { email, password } → Set cookie tokens
- `POST /api/auth/logout`
- `GET /api/auth/me` → current session user

9.2 Folders

- `POST /api/folders` → { name, parentId|null }
- `GET /api/folders/:id` → { folder, children: { folders[], files[] }, path[] }
- `PATCH /api/folders/:id` → { name?, parentId? }
- `DELETE /api/folders/:id` → soft-delete

9.3 Files

- `POST /api/files/init` → { name, mimeType, sizeBytes, folderId|null } → { fileId, upload:{method:'multipart', parts:[{partNumber, url}]}, storageKey }
- `POST /api/files/complete` → { fileId, parts:[{partNumber, etag}] } → 200; background preview job
- `GET /api/files/:id` → { file, signedUrl }
- `PATCH /api/files/:id` → { name?, folderId? }
- `DELETE /api/files/:id` → soft-delete

9.4 Shares

- `POST /api/shares` → { resourceType, resourceId, granteeUserId, role }
- `GET /api/shares/:resourceType/:resourceId` → list shares
- `DELETE /api/shares/:id`

9.5 Public Links

- `POST /api/link-shares` → { resourceType, resourceId, expiresAt?, password? }
- `GET /api/link/:token` → resolves (with optional password)
- `DELETE /api/link-shares/:id`

9.6 Search & Stars & Trash

- `GET /api/search?q=&type=&owner=&starred=`
- `POST /api/stars` → { resourceType, resourceId }
- `DELETE /api/stars` → { resourceType, resourceId }
- `GET /api/trash` → list deleted
- `POST /api/trash/restore` → { resourceType, resourceId }
- Purge job: cron clears > N days

Error format: `{ error: { code: 'FORBIDDEN', message: '...' } }` with proper HTTP status codes.

---

10) Frontend UX Details

- **Layout:** sidebar (My Drive, Shared, Starred, Recent, Trash), toolbar (New, Upload, Share, Sort, Grid/List toggle), main grid list.
- **Components:** breadcrumb, upload dropzone with progress, context menus (right-click), modals for share/link, details panel (owner, size, activity).
- **State/Data:** TanStack Query for server cache; optimistic rename/move; skeleton loaders; infinite scroll.
- **Accessibility:** keyboard shortcuts (A11y), focus rings, ARIA on menus/modals, color contrast.
- **Internationalization:** key strings externalized for future i18n.

---

11) Search Strategy

- MVP: name + type + owner filters using B-Tree indexes.
- Enhanced: Postgres full-text (`tsvector(name)`), trigram (`pg_trgm`) for fuzzy matches; optional tags.

---

12) Versioning Model

- `files.version_id` points to the current row in `file_versions`.
- New upload to same logical file → create `file_versions` row, update pointer.
- UI exposes version list with createdAt and size; allow revert.

---

13) Activity Log

- Append-only `activities` with `actor_id`, `action`, `resource`, `context` (e.g., { oldName, newName }).
- Show recent activity in a side panel for selected item.

---

14) Performance & Scale

- Use CDN for downloads; caching headers on signed URLs.
- DB: add composite indexes, avoid N+1 via batch queries, use pagination with cursors.
- Background workers for heavy tasks (previews, virus scan, emails).
- Storage lifecycle: move cold previews to cheaper tier; set retention for Trash.

---

15) Deployment & DevOps

- **Environments:** dev, staging, prod (separate DBs/buckets).
- **CI/CD:** GitHub Actions to lint, test, build; preview deploys on PRs (Vercel). Run migrations on deploy.
- **Hosting:**
    - Frontend: Vercel.
    - Backend: Render/Fly.io/Heroku.
    - DB & Storage: Supabase (Postgres + Storage) or AWS (RDS + S3).
- **Monitoring:** Sentry (frontend/backend), Logtail/Datadog logs, Uptime checks.
- **Backups:** daily automated DB backups; restore drills quarterly.

---

16) Milestone Plan (4 Sprints)

Sprint 1 (Auth & Base UI)

- Auth (email/password, OAuth optional), protected routes, layout, folder tree, create folder.

Sprint 2 (Files Core)

- Upload init/multipart, progress, list, rename, move, delete, download via signed URLs.

Sprint 3 (Sharing & Search)

- Per-user shares, public links, stars, search and filters, recent.

Sprint 4 (Polish & Ops)

- Trash + restore, activity log, thumbnails/previews, rate limits, monitoring, docs.

---

17) Skills You Will Learn (Detailed)

- Designing relational schemas for hierarchical data and ACL.
- Implementing secure, resumable multipart uploads.
- Building REST APIs with robust validation, pagination, and error handling.
- Enforcing RLS/ACL both at DB and storage layer (Supabase policies or S3 presigned checks).
- Building a performant React UI with drag-drop, optimistic updates, and infinite scroll.
- Operating a small cloud app: CI/CD, logs, alerts, backups, cost control.

---

18) Cost Awareness (indicative)

- Supabase Starter (DB/Auth/Storage) for dev; scale to Pro as MAU/storage grows.
- CDN egress dominates cost at scale; prefer regional users close to storage.
- Thumbnails/previews increase compute/storage modestly; batch and cache.

---

19) Resources (Carefully Curated)

Frontend

- Next.js: https://nextjs.org/docs
- TanStack Query: https://tanstack.com/query/latest
- React DnD Kit: https://dndkit.com/
- Uppy (uploads): https://uppy.io/docs/

Backend & Validation

- Express: https://expressjs.com/
- Zod: https://zod.dev/

Database & Search

- Supabase Postgres: https://supabase.com/docs
- Postgres Full Text: https://www.postgresql.org/docs/current/textsearch.html
- pg_trgm (fuzzy): https://www.postgresql.org/docs/current/pgtrgm.html

Storage

- Supabase Storage: https://supabase.com/docs/guides/storage

- AWS S3 multipart: https://docs.aws.amazon.com/AmazonS3/latest/userguide/mpuoverview.html

Auth

- Supabase Auth: https://supabase.com/docs/guides/auth
- NextAuth: https://authjs.dev/

Security

- OWASP Cheat Sheets: https://cheatsheetseries.owasp.org/

DevOps

- GitHub Actions: https://docs.github.com/actions
- Render: https://render.com/docs
- Fly.io: https://fly.io/docs/

---

20) Appendix

20.1 Example .env (adjust per provider)

```
# Backend
PORT=8080
NODE_ENV=development
DATABASE_URL=postgres://USER:PASSWORD@HOST:5432/drive
JWT_SECRET=super-long-random
REFRESH_SECRET=another-long-random
CORS_ORIGIN=https://your-frontend-domain.com

# Supabase (if used)
SUPABASE_URL=...
SUPABASE_ANON_KEY=...
SUPABASE_SERVICE_ROLE_KEY=...
SUPABASE_STORAGE_BUCKET=drive
```

```
# AWS (if S3 used)
AWS_ACCESS_KEY_ID=...
AWS_SECRET_ACCESS_KEY=...
AWS_REGION=ap-south-1
S3_BUCKET=drive-bucket
```

20.2 Suggested Repo Structure (separate frontend/backend)

```
root/
  apps/
    web/           # Next.js app (App Router)
    api/           # Express app
  packages/
    ui/            # shared UI components (optional)
    config/        # eslint, tsconfig, prettier
  infra/           # IaC, docker, compose, migrations
  .github/workflows/
```

20.3 Sample SQL (Postgres)

```
create extension if not exists pg_trgm;

create table if not exists users (
  id uuid primary key default gen_random_uuid(),
  email text unique not null,
  name text,
  image_url text,
  created_at timestamptz default now()
);

create table folders (
  id uuid primary key default gen_random_uuid(),
  name text not null,
  owner_id uuid references users(id) on delete cascade,
  parent_id uuid references folders(id) on delete set null,
  is_deleted boolean default false,
  created_at timestamptz default now(),
  updated_at timestamptz default now()
);

create unique index on folders(owner_id, parent_id, name) where is_deleted = false;

create table files (
```

```
  id uuid primary key default gen_random_uuid(),
  name text not null,
  mime_type text,
  size_bytes bigint,
  storage_key text unique not null,
  owner_id uuid references users(id) on delete cascade,
  folder_id uuid references folders(id) on delete set null,
  version_id uuid, -- set after first version
  checksum text,
  is_deleted boolean default false,
  created_at timestamptz default now(),
  updated_at timestamptz default now()
);

create index on files(owner_id);
create index on files using gin (name gin_trgm_ops);

create table file_versions (
  id uuid primary key default gen_random_uuid(),
  file_id uuid references files(id) on delete cascade,
  version_number int not null,
  storage_key text not null,
  size_bytes bigint,
  checksum text,
  created_at timestamptz default now()
);

create table shares (
  id uuid primary key default gen_random_uuid(),
  resource_type text not null check (resource_type in ('file','folder')),
  resource_id uuid not null,
  grantee_user_id uuid references users(id) on delete cascade,
  role text not null check (role in ('viewer','editor')),
  created_by uuid references users(id) on delete set null,
  created_at timestamptz default now(),
  unique(resource_type, resource_id, grantee_user_id)
);

create table link_shares (
  id uuid primary key default gen_random_uuid(),
  resource_type text not null check (resource_type in ('file','folder')),
  resource_id uuid not null,
  token text not null unique,
  role text not null default 'viewer' check (role = 'viewer'),
```

```
  password_hash text,
  expires_at timestamptz,
  created_by uuid references users(id) on delete set null,
  created_at timestamptz default now()
);

create table stars (
  user_id uuid references users(id) on delete cascade,
  resource_type text not null check (resource_type in ('file','folder')),
  resource_id uuid not null,
  primary key (user_id, resource_type, resource_id)
);

create table activities (
  id uuid primary key default gen_random_uuid(),
  actor_id uuid references users(id) on delete set null,
  action text not null check (action in
('upload','rename','delete','restore','move','share','download')),
  resource_type text not null check (resource_type in ('file','folder')),
  resource_id uuid not null,
  context jsonb,
  created_at timestamptz default now()
);
```

20.4 Example RLS (Supabase) – idea sketch (not exhaustive)

- Bucket policy: objects are private by default.
- Row-Level Security on `files`, `folders`, `shares`, `link_shares` ensures:
    - Owners can read/write their items.
    - Grantees can read (and write if role=editor).
    - Link tokens resolve via RPC that validates token/expiry/password.

20.5 Testing Matrix (samples)

- Auth: register/login/logout, refresh rotation, role checks.
- Files: init/complete upload, rename/move, soft delete/restore, signed download.

- Shares: grant/revoke, link expiry, password, access via token.
- Search: by name, filters, pagination, case/fuzzy.
- Rate limits: burst tests.

---

21) How to Implement (Step-by-step)

1. **Scaffold** repos (frontend & backend), add linting/formatting, CI skeleton.
2. **Auth** with provider of choice; protect API routes; set cookies.
3. **Folders** CRUD + UI (tree + breadcrumbs), unique name per parent.
4. **Uploads**: implement init → multipart → complete; show progress.
5. **Listing**: folder contents (folders first, then files), sort & grid/list view.
6. **File ops**: rename, move (drag-drop), delete → Trash; restore.
7. **Shares**: user ACL + public links with expiry/password; share dialog in UI.
8. **Search** + Starred + Recent.
9. **Previews** + thumbnails (images/PDF), activity log.
10. **Hardening**: rate limits, CSP, logs, backups, docs; staging → prod.

---

22) What You'll Learn (Mapped to Tasks)

- **Schema & ACL** → designing `shares`, `link_shares`, and enforcing in API.

- **Uploads at scale** → resumable multipart + presigned URLs.
- **Optimistic UX** → fast rename/move with rollback on failure.
- **Caching** → TanStack Query patterns, pagination, and invalidation.
- **Observability** → tracing slow queries, error budgets, and alerts.

---

23) Glossary

- **ACL**: Access Control List.
- **RLS**: Row-Level Security (Postgres/Supabase).
- **Multipart**: breaking large files into parts to upload in parallel.
- **Signed URL**: time-limited URL granting access to an object.

Tech Stack

Tech Stack :

1 Frontend : React or its Framework - Nextjs, Vite Js, Astro Js, Tailwind CSS
2 Backend : Python(Flask/ FAST API), Javascript(Node js, Express js), Java(Spring Boot)
3 Data Base : Supabase

Note : Warning, Frontend and Backend in different Repo

Schedule

🗓️ Week 1: Backend Development & Basic Frontend Setup

Day	Task
Day 1 - Project Setup & Planning	✅ Define database schema (tables for users, files, folders, permissions) ✅ Set up a Node.js + Express backend with TypeScript/Javascript ✅ Connect PostgreSQL (or MongoDB) using Supabase/Firebase Storage
Day 2 - Authentication System	✅ Implement JWT-based authentication with bcrypt (signup, login, logout) ✅ Google OAuth integration ✅ Protect API routes using middleware
Day 3 - File Upload & Storage	✅ Set up Supabase S3 / Firebase Storage for file storage ✅ Implement API for file uploads (multer for handling files) ✅ Save file metadata (size, format, path, user) in the database
Day 4 - File Management APIs	✅ Create CRUD APIs for files & folders (upload, rename, delete) ✅ Implement a folder structure & hierarchy in the database ✅ Add Soft Delete (Trash feature)

Day 5 - Sharing & Permissions	✅ Implement file sharing via unique shareable links ✅ Add role-based permissions (view, edit, owner) ✅ Generate signed URLs for secure access
Day 6 - Search & Optimization	✅ Implement a search API using PostgreSQL full-text search ✅ Optimize database queries for performance ✅ Implement pagination & lazy loading for large datasets
Day 7 - Testing & Deployment of Backend	✅ Test all backend APIs using Postman ✅ Write unit tests (Jest, Supertest) ✅ Deploy backend to AWS/GCP/Vercel

---

🖥️ Week 2: Frontend Development & UI Enhancements

Day	Task
Day 8 - Frontend Setup	✅ Initialize a React + Vite or Next.js project ✅ Set up Tailwind CSS for styling ✅ Implement authentication pages (Login, Signup)
Day 9 - Dashboard UI & File Listing	✅ Design file explorer layout (Google Drive-style UI) ✅ Fetch and display user files & folders ✅ Implement folder navigation (breadcrumbs)

Day 10 - File Upload & Management	✅ Implement drag-and-drop file uploads ✅ Show upload progress & success/error notifications ✅ Add preview support for images, PDFs, and text files
Day 11 - Sharing & Permissions UI	✅ Implement UI for sharing files via email/links ✅ Allow users to modify permissions (viewer, editor) ✅ Display shareable links for public access
Day 12 - Search & Optimization	✅ Add a search bar with real-time filtering ✅ Implement sorting by name, size, date ✅ Optimize performance using caching & lazy loading
Day 13 - Trash, Versioning & Final Testing	✅ Implement the "Trash" section (restore & delete permanently) ✅ Add versioning to track file changes ✅ Test all features thoroughly
Day 14 - Deployment & Final Touches	✅ Deploy the frontend on Vercel ✅ Optimize for mobile responsiveness ✅ Final bug fixes & performance improvements

---

Bonus (If Time Permits)

🚀 **Payments Integration** – Implement Stripe for premium storage plans
🚀 **Real-time Collaboration** – Use WebSockets to enable live updates
🚀 **Mobile/Desktop App** – Wrap with React Native or Electron

---

This schedule ensures that you **build a working MVP in two weeks**. Do you want a GitHub boilerplate or more details on any step? 🚀

Resources

Resources

- **Frontend:**
    - React Docs: https://react.dev
    - Next.js Docs: https://nextjs.org/docs
    - Tailwind CSS: https://tailwindcss.com/docs
- **Backend:**
    - Node.js Docs: https://nodejs.org/en/docs/
    - Express Docs: https://expressjs.com/
- **Database:**
    - MongoDB University: https://learn.mongodb.com/
    - PostgreSQL Docs: https://www.postgresql.org/docs/
- **Cloud Storage:**
    - AWS S3 Docs: https://docs.aws.amazon.com/s3/
    - Firebase Storage Docs: https://firebase.google.com/docs/storage
    - Supabase Storage: https://supabase.com/docs/guides/storage
- **Authentication:**
    - JWT: https://jwt.io/introduction
    - Clerk Auth: https://clerk.com/docs

Resources : Build and Deploy a Full Stack Google Drive Clone with Next.js 15