d SmartVault — AI-Optimized Personal Cloud Storage

> Project: end-to-end build plan, architecture, folder structure, API spec, MVP timeline, and starter boilerplate to ship a hiring-ready project.

---

1) Executive summary

SmartVault is a personal cloud storage app with AI features: auto-categorization, semantic search, OCR & summarization, duplicate detection, compression suggestions, and safe sharing. The goal: a production-like portfolio project demonstrating full-stack skills, cloud integration, and AI features.

MVP goals (what I will deliver first):

- User auth (email/password + JWT)
- File upload/download + metadata stored in MongoDB
- Files stored in S3 (or local for dev)
- Basic UI (Next.js + Tailwind)
- AI tagging (OpenAI embeddings + simple tag classifier)
- Semantic search over files
- Duplicate detection
- Deployment guide (Vercel for front, Render/Heroku/AWS for backend)

---

2) Default tech stack (changeable)

- Frontend: Next.js (TypeScript), Tailwind CSS, React Query
- Backend: Node.js + Express (TypeScript)
- Database: MongoDB Atlas
- File storage: AWS S3 (recommended) — dev: local filesystem or MinIO
- Auth: JWT + refresh tokens (option: Clerk or Auth0 if you prefer)
- AI: OpenAI (text embeddings for semantic search & GPT for summarization); Tesseract.js for client-side OCR fallback
- DevOps: Vercel (frontend), Render/AWS/Heroku (backend), GitHub Actions for CI

---

3) High-level architecture

1. Frontend (Next.js): upload UI, file listing, preview, semantic search box.
2. Backend (Express): REST API for auth, file metadata, search, AI jobs.
3. Storage (S3): binary file storage.
4. DB (MongoDB): user data, file metadata, embeddings index.

5. AI worker (same backend or separate service): generates embeddings, tags, summaries; stores them in DB.

---

4) Folder structure (monorepo style)

```
smartvault/
├─ apps/
│  ├─ web/                # Next.js app (frontend)
│  └─ api/                # Express app (backend)
├─ infra/                 # IaC scripts (optional)
├─ scripts/               # helper scripts (seed, maintenance)
├─ docker-compose.yml
├─ README.md
```

Frontend (web) structure

```
web/
├─ app/ (Next.js 13+ app router)
├─ components/
├─ lib/ (api client, auth helpers)
├─ styles/
├─ public/
└─ package.json
```

Backend (api) structure

```
api/
├─ src/
│  ├─ controllers/
│  ├─ services/ (s3, ai, search)
│  ├─ models/ (mongoose)
│  ├─ routes/
│  └─ workers/ (embeddings, ocr)
├─ Dockerfile
└─ package.json
```

---

5) Data models (simplified)

User

```
{ _id, email, passwordHash, name, createdAt }
```

File

```
{ _id, ownerId, filename, key, mimeType, size, folderId, tags:[], embeddings:
[], summary, createdAt, lastAccessed }
```

Folder (optional)

```
{ _id, ownerId, name, parentId }
```

---

6) Key APIs (MVP)

- `POST /api/auth/register` — create user
- `POST /api/auth/login` — returns JWT
- `POST /api/files/upload` — signed upload endpoint (multipart) -> returns file metadata
- `GET /api/files` — list files, accepts search query or filters
- `GET /api/files/:id/download` — redirect to S3 signed URL
- `POST /api/files/:id/analyze` — trigger AI (tagging, embeddings, summary)
- `GET /api/search?q=` — semantic search endpoint

---

7) AI workflows

1. On file upload, backend jobs will:
2. extract text (if PDF/Docx) via external lib / OCR
3. call OpenAI embeddings on the extracted text + filename
4. store embeddings in the file document
5. run a lightweight classifier (rules + few-shot) to assign tags
6. generate a short summary via GPT
7. Semantic search: receive query → get query embedding → cosine similarity against stored embeddings → return ranked results.

---

8) Security & privacy

- End-to-end auth + per-file owner access checks
- Signed S3 URLs for downloads/uploads
- Encrypt sensitive env vars (OpenAI key, S3 creds) in hosting platform
- Optional: client-side encryption (if you want strong privacy guarantees) — increases complexity

9) MVP timeline (detailed plan — 2 week sprint)

Week 1:

- Day 1: repo + monorepo setup, TypeScript configs, Dockerfile
- Day 2-3: Auth system, MongoDB connection, basic user flows
- Day 4-5: File upload API + S3 integration, frontend upload UI
- Day 6-7: File listing & previews, metadata CRUD

Week 2:

- Day 8-9: AI worker: text extraction, OpenAI embeddings, store embeddings
- Day 10: Semantic search API & frontend search UI
- Day 11: Duplicate detection & compression suggestions
- Day 12: Polish, tests, CI pipeline, deploy to Vercel + Render
- Day 13-14: README, demo video script, documentation for interview

---

10) Deliverables I will produce (if you want me to implement everything)

- Full repo scaffolding (monorepo) with working frontend and backend
- Auth, file upload/download (S3) + metadata stored in MongoDB
- AI analysis pipeline: embeddings + tags + summaries (OpenAI integration)
- Semantic search (embedding-based) with frontend search experience
- Duplicate detection logic and cleanup suggestions
- Deployment setup and README with how to run locally + deploy
- Demo script + GIFs/screenshots to include in your portfolio

---

11) What I need from you (decisions)

Pick defaults or tell me preferences for these so I can implement without stalling:

1. Use AWS S3 or use local storage/MinIO for dev and S3 for prod? (default: S3)
2. OpenAI account available? (default: I will use OpenAI API — you must provide a key for full functionality)
3. Auth: custom JWT or third-party (Clerk/Auth0)? (default: custom JWT)
4. Deployment preference: Vercel (frontend) + Render/AWS (backend)?

---

12) Next step (pick one)

- A — I scaffold the repo and return the code for the entire monorepo starter (auth + upload + basic UI).
- B — I implement a single vertical slice: full flow for upload → AI analyze → semantic search for a single user and return working code and instructions.
- C — I produce the complete project (all MVP features) in multiple messages/files until done.

Tell me which option and your preferences from section 11. Once you confirm, I will start implementing. Thank you!

in a one folder i want client part and one more server and from 11 i dont have aws