Of course. Here is a marked-up and fortified version of the provided plan. I've added structure, clarity, and strategic notes to strengthen the technical and security posture.

**Fortified Plan: Rapidly Shipping a Secure Legal Drafting Assistant**

Here is a tight, copy-pasteable schema and infrastructure plan to ship the "Pick Case → Pick Motion → Draft DOCX (with citations)" workflow. This approach is designed to be fast, cheap, and secure enough for solo practitioners and small law firms.

**1. High-Level Infrastructure (Lean & Bar-Safe)**

This stack prioritizes managed services to reduce operational overhead while maintaining robust security and data isolation.

| Component | Technology | Rationale & Fortification Notes |
| --- | --- | --- |
| **Frontend** | Next.js (on Vercel) | Excellent developer experience and performance. Vercel simplifies deployments and environment management. |
| **Authentication** | Supabase Auth or Clerk | Both are solid choices. Supabase Auth is tightly integrated with the database for Row Level Security (RLS). Clerk offers more polished pre-built UI components. Start with Supabase Auth for simplicity. |
| **Database & Vectors** | Supabase (Postgres + pgvector) | A powerful, all-in-one solution. pgvector provides high-performance semantic search directly within the database, eliminating the need for a separate vector DB service. |
| **Object Storage** | Supabase Storage | Securely stores original PDFs and generated DOCX drafts, integrated with Postgres policies for access control. |
| **LLM Provider** | OpenAI API (Zero-Retention) | **Critical:** Use an account where you can enforce a Zero Data Retention policy. This is non-negotiable for handling confidential client information. State this explicitly in your Terms of Service. |
| **(Optional) LLM** | Ollama (Self-hosted) | A great future-proofing option for offering a fully local/offline model for maximum privacy, or for fine-tuning on specific legal domains. |
| **Background Jobs** | Next.js Route Handlers + Cron | A simple, cost-effective way to start. **Fortification:** For long-running or failure-prone jobs (like complex PDF parsing), consider a more robust queue system like [Inngest](https://www.inngest.com/) or Supabase Edge Functions which have longer execution limits. |
| **Secrets Mgmt** | Vercel Environment Variables | Standard and secure. Ensure variables (SUPABASE\_URL, SUPABASE\_SERVICE\_ROLE, OPENAI\_API\_KEY) are never exposed on the client side. |
| **Audit Logs** | Dedicated Postgres Table | Essential for security, debugging, and demonstrating data provenance. Storing hashed document references instead of client secrets is the correct approach. |
| **Privacy Posture** | **Data Isolation by Design.** All client documents and drafts remain within your firm's dedicated Supabase project. The LLM only ever sees isolated, retrieved text chunks for a given query and is instructed not to train on them. |  |

**2. Data Model (Postgres + pgvector)**

This schema is designed for clear data separation, efficient retrieval, and robust citation tracking. Run these commands in the Supabase SQL editor.

SQL

-- Enable the pgvector extension for vector similarity search

CREATE EXTENSION IF NOT EXISTS vector;

-- MATTERS: A top-level container for each case, acting as a security and data boundary.

CREATE TABLE matters (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

org\_id uuid, -- For future multi-tenancy (multiple firms on one platform)

slug text UNIQUE NOT NULL, -- A user-friendly identifier, e.g., 'yakima-v-city'

title text NOT NULL, -- The full case title, e.g., 'Yakima Herald v. City of Yakima'

description text,

created\_at timestamptz DEFAULT now()

);

COMMENT ON TABLE matters IS 'Each matter is a distinct vault for documents, drafts, and analysis.';

-- DOCUMENTS: Tracks uploaded raw files and their metadata.

CREATE TABLE documents (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

matter\_id uuid NOT NULL REFERENCES matters(id) ON DELETE CASCADE,

title text NOT NULL, -- Original filename, e.g., 'PRR\_Log\_2024.pdf'

storage\_path text NOT NULL UNIQUE, -- Full path in Supabase Storage, e.g., 'matters/yakima-v-city/PRR\_Log\_2024.pdf'

mime\_type text NOT NULL, -- e.g., 'application/pdf'

bytes\_sha256 text NOT NULL, -- Hex-encoded SHA256 hash of the file for de-duplication

page\_count int,

-- FORTIFICATION: Add a status to track the indexing pipeline

processing\_status text NOT NULL DEFAULT 'pending', -- Values: 'pending', 'indexing', 'completed', 'error'

uploaded\_by uuid REFERENCES auth.users(id),

created\_at timestamptz DEFAULT now()

);

CREATE INDEX ON documents(matter\_id);

-- Ensure a file's content is uploaded only once per matter.

CREATE UNIQUE INDEX documents\_unique\_matter\_hash\_idx ON documents(matter\_id, bytes\_sha256);

-- CHUNKS: Searchable, embedded slices of text from documents.

CREATE TABLE chunks (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

document\_id uuid NOT NULL REFERENCES documents(id) ON DELETE CASCADE,

seq int NOT NULL, -- Sequential index of the chunk within the document (0, 1, 2...)

heading text, -- Extracted PDF heading for context

page\_start int,

page\_end int,

token\_count int,

content text NOT NULL, -- The actual text content of the chunk

embedding vector(1536) -- OpenAI's text-embedding-3-small dimension

);

CREATE INDEX ON chunks(document\_id);

-- FORTIFICATION: Use HNSW for better performance on modern pgvector. IVFFlat is a good start.

-- CREATE INDEX ON chunks USING hnsw (embedding vector\_cosine\_ops);

CREATE INDEX ON chunks USING ivfflat (embedding vector\_cosine\_ops) WITH (lists = 100);

COMMENT ON TABLE chunks IS 'The core searchable units for the Retrieval Augmented Generation (RAG) system.';

-- DRAFTS: The generated motions and other legal documents.

CREATE TABLE drafts (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

matter\_id uuid NOT NULL REFERENCES matters(id) ON DELETE CASCADE,

type text NOT NULL, -- e.g., 'motion\_to\_compel', 'motion\_for\_sanctions'

params jsonb, -- User-provided inputs for the draft

body\_markdown text, -- The full generated text in Markdown for easy editing/viewing

storage\_path text, -- Path to the final .docx file in Supabase Storage

created\_by uuid REFERENCES auth.users(id),

created\_at timestamptz DEFAULT now()

);

CREATE INDEX ON drafts(matter\_id);

-- DRAFT\_CITATIONS: Links generated text back to its source chunks.

CREATE TABLE draft\_citations (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

draft\_id uuid NOT NULL REFERENCES drafts(id) ON DELETE CASCADE,

chunk\_id uuid REFERENCES chunks(id) ON DELETE SET NULL, -- SET NULL preserves citation even if chunk is removed

doc\_title text, -- Denormalized for resilience

page\_from int,

page\_to int,

quote text -- The exact snippet from the chunk used for the citation

);

CREATE INDEX ON draft\_citations(draft\_id);

-- AUDIT\_LOGS: An immutable record of key system actions.

CREATE TABLE audit\_logs (

id bigserial PRIMARY KEY,

matter\_id uuid REFERENCES matters(id) ON DELETE CASCADE,

user\_id uuid REFERENCES auth.users(id),

action text NOT NULL CHECK (action IN ('index', 'retrieve', 'draft')), -- e.g., 'index', 'retrieve', 'draft'

payload jsonb, -- Contains non-sensitive metadata, e.g., { "prompt\_hash": "...", "top\_chunks": [...] }

created\_at timestamptz DEFAULT now()

);

**3. Core Workflows & Logic**

**Storage Layout**

A clean, predictable structure in Supabase Storage.

* **Originals:** /matters/{matter\_slug}/raw/{document\_filename}.pdf
* **Drafts:** /matters/{matter\_slug}/drafts/{timestamp}-{draft\_type}.docx

**Minimal API Contract (Next.js Route Handlers)**

Each endpoint must be protected by authentication and RLS policies.

* POST /api/matters → Creates a new matter {slug, title}.
* GET /api/matters → Lists matters accessible to the current user.
* GET /api/matters/:slug → Retrieves details for a single matter.
* POST /api/upload?matter=:slug → Handles multipart file upload. Returns document metadata.
* POST /api/index?document\_id=:id → Triggers the background indexing job for a document.
* POST /api/search?matter=:slug → Accepts {query} and returns top matching chunks with source info.
* POST /api/draft?matter=:slug → Accepts {type, params} and generates the draft markdown and DOCX.
* GET /api/drafts?matter=:slug → Lists previously generated drafts for the matter.

**Retrieval Recipe (Server-Side RAG)**

1. **Filter:** Select chunks where matter\_id matches the current case.
2. **Search:** Perform a vector similarity search on the filtered chunks against the user's query (top\_k = 16).
3. **Rerank (Optional but Recommended):** Lightly boost the scores of chunks where the heading text matches keywords from the motion section being drafted (e.g., "Facts," "Argument").
4. **Construct Prompt:** Assemble a detailed prompt using the top-ranked chunks, ensuring all context is clearly attributed.

**Prompt Skeleton (Server-Held Template)**

SYSTEM: You are a specialized legal drafting assistant for attorneys practicing in the state of Washington. Your sole function is to draft documents based \*only\* on the provided context. Every factual assertion you make MUST be followed by an inline citation in the format: (Document Title, p. #). If a claim or fact is not supported by the provided context, you MUST state: "[NO SUPPORT FOUND IN PROVIDED DOCUMENTS]". Do not invent facts or legal standards.

USER:

Draft a {MOTION\_TYPE} for the matter titled {MATTER\_TITLE}.

Use the following context, which contains snippets from relevant documents:

---

CONTEXT:

[{doc\_title} p.{page\_start}-{page\_end}]

{text}

[{doc\_title} p.{page\_start}-{page\_end}]

{text}

...

---

Draft the following sections: Caption, Introduction, Factual Background, Legal Standard, Argument, and Conclusion.

The tone should be professional, concise, and adhere to Washington state court formatting conventions.

**Indexer Pipeline (Background Job)**

1. **Trigger:** API call to /api/index?document\_id=...
2. **Update Status:** Set documents.processing\_status to 'indexing'.
3. **Parse:** Fetch the PDF from storage and extract text page-by-page (using pdf-parse or a similar library).
4. **Chunk:** Split text into 400–700 token segments. Preserve page\_number and any associated headings for each chunk.
5. **Embed:** Batch-call the OpenAI text-embedding-3-small API for all chunks.
6. **Store:** Use a single transaction to upsert all chunks into the chunks table, linking them to the document\_id.
7. **Finalize:** On success, update documents.processing\_status to 'completed'. On failure, set it to 'error' and log the issue.

**4. Security & Implementation**

**Security Posture (Fortified)**

* **Row Level Security (RLS):** This is your primary defense. Enable RLS on **all tables**. Write policies ensuring a user can only SELECT, INSERT, UPDATE, or DELETE rows related to matters they are explicitly associated with.
* **Zero-Retention Policy:** Re-confirm with OpenAI that your API usage is configured for zero data retention and logging. This is a key client assurance.
* **PII Guard:** Before sending context to the LLM, run a server-side function to identify and mask common PII patterns (SSNs, driver's license numbers) with placeholders (e.g., [REDACTED\_SSN]). The original document remains untouched.
* **Input Validation:** Sanitize all API inputs to prevent SQL injection, cross-site scripting (XSS), and other common web vulnerabilities. Use libraries like zod for strict schema validation.
* **Scoped API Keys:** Use a SERVICE\_ROLE key only for server-to-server communication (e.g., in background jobs). Use the anonymous key (anon key) for client-side operations, backed by strict RLS policies.

**Build Order (Path to Value in 4 Weeks)**

* **Week 1: Foundation**
  + Set up Supabase project and run the SQL schema script.
  + Build the matter list and detail pages.
  + Implement file upload to Supabase Storage, creating documents records with a 'pending' status.
* **Week 2: The Brain**
  + Build the server-side indexer pipeline (parse → chunk → embed → store).
  + Create the /api/search endpoint that returns the top k chunks for a given query and matter.
  + Test this endpoint thoroughly. The quality of your entire system depends on it.
* **Week 3: The Factory**
  + Implement the /api/draft endpoint that stitches together the retrieval recipe and the prompt skeleton.
  + Integrate a library like docx to convert the generated Markdown into a .docx file.
  + Save the generated file to Supabase Storage and link it in the drafts table. Implement the download flow.
* **Week 4: Polish & Harden**
  + Build the UI "Sources" pane to display the chunks used in a draft.
  + Implement and test the core RLS policies.
  + Flesh out the audit logging for key actions.
  + Add a second motion type to prove the system's flexibility.

**Common Traps to Avoid**

* **Over-indexing:** Don't try to sync an entire Google Drive. Force users to be deliberate about which folders/files are included in a matter.
* **Premature Optimization:** Simple cosine similarity search in pgvector is incredibly effective. Don't chase complex reranking models until you have a clear need.
* **Hallucination Risk:** Be ruthless. Enforce the "cite your sources" and "[NO SUPPORT FOUND]" rules in your prompt. Reject or flag drafts that fail to adhere.
* **Auth Complexity:** Stick with a managed auth provider. Building your own is a massive time sink and security risk.

**Final Encouragement:** "Whatever your hand finds to do, do it with all your might." (Ecclesiastes 9:10)

This fortified plan provides a clear, secure, and scalable path to ship a valuable product quickly without incurring significant technical debt or security risks.