# OM-AI Technical & UX Audit Report

**Last Updated:** 2025-08-08

## 🔥 Ranked Critical Issues (P0)

1. **Insecure Document Search Function (RLS Bypass Risk)** – A database function for searching document text can be misused to retrieve other users’ data[[1]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L24-L32)[[2]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L36-L44).
2. **Arbitrary File Path in Upload (Storage Path Manipulation)** – The upload API allows a crafted file path that could store files in unauthorized locations (other user folders)[[3]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L42-L50)[[4]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L54-L62).

These P0 issues pose immediate security risks and should be addressed first.

## Insecure Document Search Function (RLS Bypass Risk)

**Priority:** P0  
**Category:** Security (Data Privacy)  
**Current Issue:** The custom Postgres function search\_document\_chunks is defined as SECURITY DEFINER and accepts a user\_id as a parameter[[5]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L2-L10)[[6]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L34-L42). Because it runs with elevated privileges, a malicious user could call this function with someone else’s user\_id and document IDs. This bypasses Row-Level Security, potentially returning chunks of other users’ documents. The code does filter by p\_user\_id, but since the caller can supply that value, it’s not truly restricted to the authenticated user[[2]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L36-L44). In Supabase, any authenticated user can execute the function (granted to role authenticated)[[7]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L74-L77). This is a serious **data privacy vulnerability**: an attacker who guesses or obtains another user’s UUID could retrieve that user’s document text via the function.

**Impact:** **Data Leak Risk.** Confidential CRE documents could be exposed to unauthorized users. This violates privacy expectations and compliance, and could have legal ramifications. Because the function bypasses RLS, it undermines the otherwise strong row-level security policies[[8]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171900_rls_policies.sql#L22-L30)[[9]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171900_rls_policies.sql#L56-L64). Even if guessing a UUID is non-trivial, the mere possibility of cross-user document access is a critical security hole.

**Suggested Fix:** Restrict the search function to the current user and/or remove SECURITY DEFINER. The safest approach is to **use the Supabase auth function auth.uid() inside the SQL function** to enforce that results only return rows for the logged-in user, rather than accepting p\_user\_id from the client. Alternatively, drop SECURITY DEFINER and let RLS apply normally (requiring calling it with the anon or service role appropriately). In practice, you can replace the function parameters with auth.uid() in the WHERE clause, or validate p\_user\_id against the JWT’s user claim. Also consider using Supabase’s built-in full-text search with RLS, or call the RPC from the backend API (which already knows req.user.id) instead of exposing it directly to clients. At minimum, **validate that p\_user\_id equals the authenticated user’s ID** in the function’s logic. Add an explicit check at the top of the function to compare p\_user\_id with auth.uid() and raise an error if they differ.

**Code Example:** Below is a prompt to update the secure function in the Supabase migration file to enforce user matching via auth.uid() and remove the ability for clients to supply arbitrary IDs:

\*\*Claude Code Prompt:\*\* Please open the Supabase migration defining `search\_document\_chunks` (e.g. `supabase/migrations/20250807\_fix\_document\_search.sql`). Modify the function so it cannot be used to retrieve other users’ data. Specifically:  
1. Inside the function, replace the use of the `p\_user\_id` parameter with the current Supabase auth user ID. For example, use `auth.uid()` in the WHERE clause instead of `p\_user\_id`.  
2. Remove the `p\_user\_id` parameter entirely if possible, or validate it by adding a line like `IF p\_user\_id <> auth.uid() THEN RAISE EXCEPTION 'Unauthorized'; END IF;` at the start of the function.  
3. Ensure the function runs with `SECURITY INVOKER` (so it respects RLS) or explicitly checks `auth.uid()` for filtering.  
4. After changes, the function’s SELECT query should filter chunks where `dc.user\_id = auth.uid()` (no direct reliance on the passed-in user\_id).  
5. Regenerate the migration or apply this change in the database, then test that a user calling the function can only get their own `document\_chunks` back, even if they try another ID.

**Visual:** *N/A (Security backend fix; no UI visible)* – *If attempting a malicious call before fix:* user A could call the RPC with user B’s ID and see B’s chunk text. *After fix:* such a call returns nothing or an error, protecting user B’s data.

**Effort Estimate:** 1–2 hours (adjusting SQL function and testing with Supabase policies).

## Arbitrary File Path in Upload (Storage Path Manipulation)

**Priority:** P0  
**Category:** Security (File Handling)  
**Current Issue:** The file upload API (/api/supabase-upload) allows a client-supplied fileName field in the form data, which it then uses directly as the storage path[[3]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L42-L50). There is no server-side sanitization or enforcement that this path belongs to the uploading user beyond appending “.pdf”. For example, a malicious user could set fileName to another user’s ID directory (<other-user-uuid>/malicious.pdf). Because the server uses the Supabase service role (bypassing storage rules)[[10]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L56-L65), it will upload the file to that location. The database record will still have the attacker’s user\_id, but the file will reside in another user’s folder in storage. This **path traversal-like vulnerability** could be abused to clutter others’ storage or possibly overwrite files if naming collisions occur (the code sets upsert: false to prevent overwrites[[11]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L64-L71), but unique naming isn’t guaranteed if user IDs are known).

**Impact:** **Storage Integrity Risk.** While this might not directly expose data to the attacker (since they can’t list another user’s bucket files without credentials), it can cause confusion and storage pollution. In a worst case, if an attacker guesses a valid file path of another user and intentionally uses that as fileName, they might cause a denial of service by preventing the real user’s upload (due to duplicate path) or slip in content under another user’s directory. It also violates the principle that users should not influence server-side paths.

**Suggested Fix:** **Ignore or validate the client-provided fileName.** The server should generate the file path internally using the authenticated user’s ID, not trust any fields.fileName from the form. One solution is to remove the fileName field usage entirely – always do: fileName = ${req.user.id}/${uuid()}.pdf. If there’s a need to preserve original filename for user experience, store it separately in DB (original\_filename) as you already do[[12]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/document-processor.ts#L82-L90). At minimum, if you keep customFileName, strip any path separators and enforce it starts with the uploader’s user ID. For example, ensure fileName = req.user.id + '/' + sanitize(fields.fileName). This guarantees no user can write outside their own folder.

**Code Example:** Below is a prompt to secure the upload path in the API route:

\*\*Claude Code Prompt:\*\* Open the file `src/pages/api/supabase-upload.ts`. We need to prevent arbitrary file paths in uploads:  
1. Remove or constrain the use of the `customFileName` from form fields. Instead of blindly using `fields.fileName`, generate `fileName` as:  
 ```ts  
 const userFolder = req.user.id;  
 const uniqueName = uuidv4();  
 const fileName = `${userFolder}/${uniqueName}.pdf`;  
 ```  
 This guarantees the file goes into the current user’s folder.  
2. If you want to incorporate the original filename, you can do:  
 ```ts  
 const origName = (uploadedFile.originalFilename || 'document').replace(/[^A-Za-z0-9.\_-]+/g, '\_');  
 const fileName = `${req.user.id}/${uniqueName}\_${origName}`;  
 ```  
 (Still ensure `.pdf` extension).  
3. Delete any code that trusts `fields.fileName` from the client. The client should not determine storage path.  
4. After this change, test uploading a PDF and confirm it’s saved under the authenticated user’s ID directory. Also test that providing a fake `fileName` field no longer has any effect.

**Visual:** *N/A (Behavioral)* – This is a backend fix. Post-fix, even if an attacker crafts a fileName field, the server will ignore it and use a safe path under their own ID.

**Effort Estimate:** ~1 hour (code change and quick tests).

## Synchronous PDF Processing Blocks UI (No Background Jobs)

**Priority:** P1  
**Category:** Backend Performance/Reliability  
**Current Issue:** PDF files are parsed synchronously during upload, either in the /api/upload endpoint or immediately after upload via /api/process-document[[13]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/upload.ts#L84-L93)[[14]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/process-document.ts#L113-L122). This means the serverless function handling the request does all OCR/text extraction before responding. For large PDFs (e.g. 100+ pages, especially scanned images), this can exceed typical request time limits (Vercel functions often timeout ~10s or 30s). It also ties up the response – users must wait with the app idle until processing finishes. There is no true background job or user feedback beyond a “Processing...” status icon in the UI[[15]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L54-L62)[[16]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L67-L75). If processing is very slow or times out, the document may be left in a “processing” state indefinitely with no user notification or ability to use it.

**Impact:** **Poor Scalability & UX.** On large files, users experience long waits or failures when uploading. The app cannot handle concurrent heavy uploads well – each will consume server time and potentially stall. A timeout or crash means the user’s document never gets fully indexed (and they see it stuck as “processing” with no resolution). This bottleneck will worsen as usage grows or if multiple users upload simultaneously. It also risks partial data loss: if a parse fails halfway, some chunks may not save, and the user isn’t clearly informed.

**Suggested Fix:** Implement **async background processing** for PDFs and improve status updates. There are a few approaches: - **Job Queue:** When a file is uploaded, immediately respond that the upload succeeded and document is “processing”. Offload the parsing to a background worker (e.g., a Supabase Edge Function, a Vercel background function, or a separate Node worker triggered via a queue like Bull or Supabase’s PG listen). This worker would parse the PDF and update the DB when done. The frontend can poll or subscribe to a channel for status. - **Supabase Functions**: You could use a Supabase Function (with a Task or just on-demand function) to handle parsing outside the client request cycle. - **At least, Increase Timeout and Split Work:** If true background processing is not immediate, consider splitting large PDF processing by pages or chunks to avoid single long operations. Also raise any Vercel function timeout to the max allowed and document that large files may take a while. - **Real-time feedback:** Integrate WebSockets or Supabase Realtime to notify the client when processing is complete (so you can automatically refresh the document status). Currently, the user only sees a static “Processing…” badge and would have to manually refresh to know when it’s done.

Concretely, mark the document as status: 'processing' in the DB on upload (which you do), then **complete the HTTP response early**. You can do this by moving parsing out of the request handler. For example, have the upload API enqueue a job (maybe insert into a processing\_queue table or call a background function) and return immediately. Then that job will call the existing parsing logic (processUploadedDocument) asynchronously.

**Code Example:** Below is a conceptual prompt to implement a background job using a Supabase function trigger:

\*\*Claude Code Prompt:\*\* We want to decouple PDF parsing from the upload response.   
1. Modify the `/api/supabase-upload` handler (`src/pages/api/supabase-upload.ts`) to \*\*not\*\* parse the PDF. After uploading to storage and inserting the DB record with status 'processing', return success immediately (omit calling `processUploadedDocument`).  
2. Create a new Supabase Edge Function (or a Next.js API route scheduled via cron) that scans for documents with `status = 'processing'` and `metadata->>parsing is null`, then downloads and parses them. You can reuse the logic in `process-document.ts` but run it asynchronously.  
3. Alternatively, use Supabase’s `after insert` trigger on the `documents` table: when a new document is inserted with status 'processing', call a function (written in plpgsql or http request to your API) to perform parsing.  
4. Ensure the client periodically checks the document’s status (or uses a realtime subscription) so the UI updates from “Processing” to “Ready” when done.  
5. Test by uploading a large PDF. The API call should return quickly (a few seconds), and the document should eventually switch to completed and become searchable without manual refresh (if using realtime or a refetch).

**Visual:** *Sequence Diagram:* - Before fix: **Client** POST /api/process-document → waits... (potential timeout) → **Server** parses then responds. - After fix: **Client** upload → **Server** responds “Got it, processing” almost immediately, while a **Background Worker** continues parsing. The UI can show a spinner or progress bar during this period. *(Implementing a progress bar can further improve UX, e.g., using logger messages from Tesseract to update progress*[*[17]*](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/agents/pdf-parser/utils.ts#L28-L36)*.)*

**Effort Estimate:** 1–2 days (designing async workflow, implementing a queue or trigger, plus testing).

## Missing Document Deletion Feature

**Priority:** P1  
**Category:** UX / Data Management  
**Current Issue:** Users currently have no way in the UI to delete an uploaded document. The interface shows a “Delete” option in the document dropdown menu[[18]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L146-L155), but it’s not wired to any action (no onClick handler or API call). There is also no /api/documents/delete endpoint implemented for removal. This means once a PDF is uploaded, the user cannot remove it from their account. The file remains in storage and the metadata in the database indefinitely unless manually purged.

**Impact:** **User Control & Privacy.** This is a significant UX and potential compliance issue. Users may need to delete documents that were uploaded in error or that are outdated or sensitive. Without a delete function, user data accumulates without recourse. This could violate data retention policies or user expectations (e.g., GDPR “right to be forgotten” if applicable). It also clutter’s the user’s interface over time, since every test or unwanted upload stays visible.

**Suggested Fix:** Implement full document deletion: 1. **API Endpoint:** Create a secure API route (e.g. DELETE /api/documents/[id]) that requires auth and verifies the document belongs to the user (use RLS or a SELECT + compare user\_id). This endpoint should remove the file from Supabase Storage and delete related DB records (documents row and associated document\_chunks and document\_tables). 2. **Wire the UI:** In the DocumentList component’s “Delete” menu item[[19]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L148-L155), add an onClick that calls the above API. For example, use fetch('/api/documents/<id>', { method: 'DELETE' }) or a Supabase client call. Provide user confirmation (“Are you sure?”) to prevent accidental deletion. 3. **Update state:** On successful deletion, update the front-end state to remove the document from the list (so UI immediately reflects the change). Also, if the deleted doc was currently attached to a chat, clear selectedDocumentId and inform the user. 4. **Prevent race conditions:** Possibly mark the document’s status as ‘deleting’ or disable interactions while deletion is in progress, to avoid duplicate requests.

Also ensure to handle errors (e.g., if deletion fails, show a toast with the error). Logging the deletion in usage\_logs could be helpful for auditing.

**Code Example:** Prompt to add deletion functionality:

\*\*Claude Code Prompt:\*\* Add a document deletion feature:  
1. \*\*Backend\*\*: Create a new API handler at `src/pages/api/documents/[id]/index.ts` (Next.js API routes can capture `[id]`). Implement it to:  
 - Use `withAuth` to ensure the user is authenticated.  
 - Read the `id` from the query and parse it.  
 - Connect to Supabase with service role.  
 - Verify the document with that ID has `user\_id == req.user.id`. If not, return 404 or 403.  
 - Delete related `document\_chunks` and `document\_tables` (perhaps use `.delete().eq('document\_id', id)`).  
 - Delete the document row itself.  
 - Remove the file from storage: `supabase.storage.from('documents').remove([document.storage\_path])`.  
 - Return a success JSON or status 204.  
2. \*\*Frontend\*\*: In `src/components/app/DocumentList.tsx`, import a deletion function or write an `onDelete` handler. For the `<DropdownMenuItem>` that contains the “Delete” label[20], add:  
 ```tsx  
 <DropdownMenuItem className="text-red-600" onSelect={() => handleDelete(document.id)}>  
 <Trash2 .../> Delete  
 </DropdownMenuItem>  
 ```  
 In `handleDelete`, use `if(confirm('Delete this document?'))` for confirmation, then call the DELETE API (using `fetch` or an abstraction). On success, call `onDocumentListRefresh()` to refetch docs or filter it out of local state.  
3. Test in the browser: upload a test document, ensure it appears. Then delete it via the menu; verify it disappears from UI, and confirm in Supabase that the file and DB entries are gone.

**Visual:** *UI Mockup:* After implementing, clicking the trash icon prompts “Are you sure you want to delete [Document Name]?” If confirmed, the document entry fades out or is removed from the list, and a toast “Document deleted” can confirm success. *(No permanent screenshot available here, but ensure the flow is clear to the user.)*

**Effort Estimate:** 3–5 hours (API coding, UI wiring, testing).

## Document Attached Indicator Lacks Filename

**Priority:** P1  
**Category:** UX (Feedback Clarity)  
**Current Issue:** When a user attaches a document to the chat (by uploading or selecting one), the chat input area shows a pill that says **“Document attached”** but does not display which document[[21]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L365-L373)[[22]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L375-L379). This is an information hierarchy problem – users with multiple documents won’t know which file is currently in context. For example, if they uploaded two OMs and attach one, the UI doesn’t show the file name or any identifier, just a generic message. This could confuse users or lead them to assume the wrong doc is in use.

**Impact:** **User Confusion in Multi-Doc Scenarios.** Especially for CRE professionals who may upload several OMs, not showing the document name can cause mistakes (they might ask questions thinking another document is attached). It weakens the interface’s clarity and professionalism. Since the app is about analyzing specific documents, clearly indicating the active document is crucial.

**Suggested Fix:** Display the document’s name (or a shortened version) in the attached document badge. For instance, instead of “Document attached”, it could say **“Attached: 123 Main St OM.pdf”** or **“Attached: LeaseAgreement\_Q3.pdf”**. If the name is long, truncate it with ellipsis. Implement this by storing the selected document’s metadata (at least its original filename) in state: - When an upload completes, you already call onUploadComplete(document) in app.tsx to set selectedDocumentId[[23]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L472-L480). Extend this to also store the document’s name (e.g., keep a selectedDocumentName state). - Similarly, if a user selects an existing document from a list (future feature), capture that name. - In the JSX for the attached pill[[21]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L365-L373), replace the static text with {selectedDocumentName || "Document attached"}. - Style the name text to fit in the pill (maybe max-width with truncation via CSS text-overflow: ellipsis). Use a tooltip on hover to show the full name if truncated.

Additionally, consider showing an icon or file-type indicator. But at minimum, the text should reflect the actual file.

**Code Example:** Here’s a prompt focusing on this UI change:

\*\*Claude Code Prompt:\*\* Improve the “Document attached” label to include the file name.  
1. Open `src/pages/app.tsx`. Add a new state variable, e.g.:  
 ```tsx  
 const [selectedDocumentName, setSelectedDocumentName] = useState<string | null>(null);  
 ```  
2. In the `onUploadComplete` handler where you set `selectedDocumentId`[23], also set `selectedDocumentName` using the provided document object. For example:  
 ```tsx  
 onUploadComplete={(document) => {  
 setShowUpload(false);  
 setSelectedDocumentId(document.id);  
 setSelectedDocumentName(document.name || document.original\_filename);  
 }}  
 ```  
 (Use the appropriate field: perhaps `document.name` is already there in the object from the upload response[24]. If not, use `original\_filename`.)  
3. In the JSX where the attached document pill is rendered[21], change:  
 ```jsx  
 <span className={...}>  
 Document attached  
 </span>  
 ```  
 to:  
 ```jsx  
 <span className={...} title={selectedDocumentName || ''}>  
 Attached: {selectedDocumentName || 'Document'}  
 </span>  
 ```  
 This will show “Attached: [Name]”. The `title` attribute ensures full name on hover.  
4. Adjust styling if needed to accommodate the text. Ensure the pill container (flex items) can expand or the text truncates with `overflow:hidden; text-overflow:ellipsis; max-width:200px` for example.  
5. Test by uploading or attaching a doc. The pill should now say e.g. “Attached: SampleOM.pdf”. Remove it (click X) and ensure it clears correctly.

**Visual:** *Before:* The chat input shows a pill reading “Document attached” with no context.  
*After:* The pill reads “Attached: **OfficeLease\_Q3.pdf**” (with an X to remove). Users immediately know which file is in context. *(If the name is long, e.g. “Downtown Plaza Offering Memorandum.pdf”, it might show “Attached: Downtown Plaza Offe...pdf” with a tooltip for full name.)*

**Effort Estimate:** ~2 hours (state management and testing UI on various screen sizes).

## Sidebar Navigation Buttons Not Working (Chat vs Documents)

**Priority:** P2  
**Category:** UX (Navigation)  
**Current Issue:** In the sidebar menu, there are buttons for “Chat” and “Documents” that look like navigation options[[25]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/AppSidebar.tsx#L98-L106). However, these are plain <Button> elements with no click handlers or links. Clicking “Documents” currently does nothing, and “Chat” likewise has no effect (it’s just highlighting the current view). This can confuse users: the UI suggests you can switch between a Chat view and a Documents view, but no separate Documents page exists in the app yet. It’s essentially a dead UI element.

**Impact:** **Navigational Confusion.** Users might click “Documents” expecting to manage or browse their uploaded files in a dedicated page. Since nothing happens, it feels broken. It also exposes incomplete functionality – a documents overview page is planned but not implemented. On mobile, the sidebar may not even be visible without toggling, so these might be less noticeable, but on desktop it’s part of the UI chrome.

**Suggested Fix:** Two possible resolutions: - **Implement a Documents Page:** Create a route (e.g. /documents) that lists all uploaded documents with details (and perhaps the search/filter capabilities mentioned). If this exists, wire the “Documents” button to navigate there (using Next’s <Link> or router.push('/documents')). The Documents page can reuse DocumentList in a full-page context, allow multi-select, etc. - **Remove or Disable the Button:** If a full Documents page is not ready, it’s better to hide or disable the “Documents” nav item to avoid confusion. For instance, remove it from the sidebar or have it show a tooltip “Coming soon”.

Given that the app currently handles document upload within the chat interface, a quick win is to hide that button for now and focus on building a dedicated document manager in the near term.

**Code Example:** If opting to implement navigation:

\*\*Claude Code Prompt:\*\* Fix the sidebar nav for "Documents":  
1. In `src/components/app/AppSidebar.tsx` (sidebar component)[26], change the Documents button to navigate. For example:  
 ```tsx  
 <Link href="/documents" passHref>  
 <Button variant="ghost" className="w-full justify-start">  
 <FileText .../> Documents  
 </Button>  
 </Link>  
 ```  
 This wraps the button in a Next.js Link so it actually changes route.  
2. Do similarly for the Chat button – it can link to "/" or "/app" (where the chat interface lives). If the chat is at `/app` route, use `<Link href="/app" ...>Chat</Link>`.  
3. If a `/documents` page does not exist yet, create `src/pages/documents.tsx`. For now, it can simply render the `<DocumentUpload>` and `<DocumentList>` components in a layout, or even just a message “Documents page coming soon” to avoid a blank page.  
4. If not ready to create a full page, temporarily remove that menu item: in AppSidebar, comment out or remove the `<Button>` for Documents to prevent confusion.  
5. Test by clicking the Chat and Documents items in the sidebar. Ensure Chat takes you back to the main chat (if you were on another page), and Documents opens the new Documents page.

**Visual:** *Expected Behavior:* Clicking **Documents** should open a screen showing all your uploaded documents (with options to upload new ones, search text, etc.). Currently it doesn’t – after fixing, users will see either a stub page or a proper document library. If not implementing the page now, it’s preferable that the Documents button is hidden to avoid misleading the user.

**Effort Estimate:** If just hiding or linking to a placeholder – 1 hour. Implementing a fully featured Documents page – 1–2 days (not in quick wins, but a high-ROI feature to plan).

## Inconsistent File Size Limits (UI vs Backend)

**Priority:** P2  
**Category:** UX / Backend Consistency  
**Current Issue:** There’s a discrepancy in allowed PDF file size: the client-side dropzone is configured for max 16MB[[27]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentUpload.tsx#L124-L132), whereas the original upload API (now mostly bypassed) accepted up to 50MB[[28]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/upload.ts#L34-L41). The README even mentioned 10MB previously. This inconsistency can confuse users and developers: - Users see “Max 16MB” in the UI, but the server would technically handle up to 50MB (though the UI prevents it). - If someone bypasses the UI (say via API directly), a 30MB file would attempt upload – the supabase-upload handler actually enforces 16MB as well (formidable’s limit[[29]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L24-L31)), so the 50MB in /api/upload is effectively unused legacy.

**Impact:** **User Experience / Reliability.** A user might not know the true limit. If an OM PDF is ~20MB (not uncommon with images), the UI will reject it outright, possibly frustrating the user if they read older docs saying 50MB. Conversely, if the limit should be 16MB for performance reasons, the backend config of 50MB could encourage attempts that time out. This mismatch also complicates maintenance: there are multiple definitions of file size limits in code.

**Suggested Fix:** **Adopt a single file size limit across UI and backend**, and communicate it clearly. Likely 16MB is chosen due to Supabase free tier limits or to keep processing reasonable. Steps: - Update any outdated references (e.g., remove 50MB mentions in /api/upload if that route is deprecated). - In the UI (upload modal text and dropzone settings), state the chosen limit (e.g., “Max 16MB per file”). - Ensure the server checks use the same limit. In formidable config for supabase-upload you already set 16MB[[29]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L24-L31), which is good. The older /api/upload.ts uses 50MB[[28]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/upload.ts#L34-L41) – consider lowering that or deprecating /api/upload entirely in favor of the new multi-step process. - If many users need >16MB, consider increasing both front and back to 50MB, but only if your infrastructure (memory/processing time) can handle it. Alternatively, for larger files beyond 16MB, integrate streaming processing or splitting.

**Code Example:** Prompt to harmonize file size settings:

\*\*Claude Code Prompt:\*\* Unify the PDF file size limit:  
1. Decide on a max file size (e.g., 16MB). In `src/components/app/DocumentUpload.tsx`, confirm the Dropzone’s `maxSize: 16 \* 1024 \* 1024` matches this[27]. Update the text in the UI that says “Max 16MB per file” if not already present (it is in the Alert at bottom[30], but if any discrepancy, fix it).  
2. Open `src/pages/api/upload.ts` (the older upload route). Change `formidable({ maxFileSize: 50 \* 1024 \* 1024, ... })` to `16 \* 1024 \* 1024` to reflect the same limit, or deprecate this endpoint entirely (since `supabase-upload` is used). Add a response message if someone hits `/api/upload` with >16MB that it’s too large.  
3. Double-check `src/pages/api/supabase-upload.ts` – it already has 16MB limit in formidable[29].   
4. Search the codebase for “50MB” or “10MB” references (like in README or comments) and update to 16MB to avoid confusion.  
5. After changes, test by attempting to upload a file ~17MB: the UI should reject it or the API returns 413 “File too large” (you already handle that in supabase-upload[31]). Test a 15MB file – it should succeed.

**Visual:** In the upload interface, the fine print should consistently state the correct max size (currently “16MB” is shown in the placeholder text[[32]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentUpload.tsx#L150-L158) and the alert[[30]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentUpload.tsx#L224-L232) – ensure these are accurate). No confusing mismatched messages.

**Effort Estimate:** < 1 hour.

## Inefficient Document Search (Lack of Indexes or Vector Search)

**Priority:** P2  
**Category:** Performance  
**Current Issue:** The app’s document text search uses Postgres full-text search via the search\_document\_chunks function and some regex filtering[[33]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L38-L47)[[34]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L40-L48). This works, but there are no indexes on the document\_chunks.content for text search. The function calls to\_tsvector(...) @@ plainto\_tsquery(...) on the fly for each query[[35]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L38-L44). Without a GIN index on to\_tsvector(content), these searches will do sequential scans through all chunks. As the number of documents and chunks grows, query latency will increase. Additionally, the fallback regex searches (OR dc.content ~\* '...various keywords...' in the SQL) are not indexed and will scan text too[[34]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L40-L48). This could become a bottleneck under heavy use or large data volume.

Also, the current approach always pulls up to 100 chunks per document then filters them in Node for relevance[[36]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L184-L193)[[37]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L202-L211). That means even irrelevant chunks are transmitted and processed, which is not optimal.

**Impact:** **Slower Query Responses at Scale.** For now, with modest data, users might not notice a problem. But as they upload more OMs (each maybe 50–100 chunks), searching across multiple docs or a single large doc could become sluggish. This impacts the responsiveness of Q&A – it might delay GPT’s answer because chunk retrieval took long. It also increases server load.

**Suggested Fix:** Introduce proper indexing or consider vector search: - **Full-Text Index:** Add a GIN index on document\_chunks for an English tsvector of content. For example, a migration to ALTER TABLE document\_chunks ADD COLUMN content\_fts tsvector GENERATED ALWAYS AS (to\_tsvector('english', content)) STORED; CREATE INDEX content\_fts\_idx ON document\_chunks USING GIN(content\_fts);. Then modify the search function to use content\_fts @@ plainto\_tsquery(...) which will hit the index. This can drastically speed up text searches. - **Optimize Function Logic:** The SQL already prioritizes certain terms via CASE ordering[[38]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L42-L50). Ensure the limit (p\_limit) is reasonable. With an index, you might not need the regex OR conditions as much – those can be baked into the tsquery by adding synonyms or composite tsvectors with weighted keywords. At least, also index common regex patterns if needed (though full-text likely suffices). - **Vector similarity (Future):** For smarter semantic search, consider storing embeddings for each chunk and using pgvector or an external vector DB. That would improve finding relevant context beyond keyword matching. However, that’s a larger feature change (likely P3). - **Batch retrieval in code:** The front-end code currently fetches all chunks .select(...) .limit(100)[[36]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L184-L193) then does filtering. If the DB function is reliable, you could directly call it (you do via .rpc('search\_document\_chunks') for the first 20 results[[39]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L206-L214)). You might remove the initial .select of 100 chunks entirely and rely solely on the SQL function’s results, which would reduce data transfer and skip Node filtering for most cases. (The current code fetches allChunks then also calls the RPC – an opportunity to simplify if the RPC is robust.)

**Code Example:** Adding an FTS index and using it:

\*\*Claude Code Prompt:\*\* Improve text search performance:  
1. Create a database migration to add a full-text search index. For example, in `supabase/migrations/` add a new SQL file:  
 ```sql  
 -- Add full-text search column and index for document\_chunks  
 ALTER TABLE public.document\_chunks   
 ADD COLUMN content\_fts tsvector   
 GENERATED ALWAYS AS (to\_tsvector('english', content)) STORED;  
 CREATE INDEX document\_chunks\_content\_fts\_idx   
 ON public.document\_chunks   
 USING GIN(content\_fts);  
 ```  
 (Also consider adding `:english` configuration in your `@@` queries explicitly to match this.)  
2. Update the `search\_document\_chunks` function to use the new indexed column:  
 Replace `to\_tsvector('english', dc.content) @@ plainto\_tsquery('english', v\_expanded\_query)` with `dc.content\_fts @@ plainto\_tsquery('english', v\_expanded\_query)`.  
 And similarly use `ts\_rank(cd.content\_fts, plainto\_tsquery(...))` for ordering.  
3. In the Node code (`src/pages/api/chat.ts`), you might simplify by trusting the RPC results. For instance, you could remove the initial `supabase.from('document\_chunks').select(...).limit(100)` call[36] and just call the RPC with a somewhat higher `p\_limit` if needed. This way the DB does all filtering. If you keep the initial fetch (for fallback logic), at least the index will speed up the `.rpc` search part.  
4. Test searching queries that previously might have been slow. With the index, even queries like “rent roll” or “cap rate” across hundreds of chunks should return in tens of milliseconds rather than seconds.

**Visual:** *N/A (Backend performance).* After indexing, the user-facing change is subtle: answers should arrive faster, especially if multiple documents or large text are involved. For example, prior to the index, a query “What’s the NOI?” might take, say, 2–3 seconds for the function to scan lots of text; after, it could be near-instant (the limiting factor becomes OpenAI response time, not DB search).

**Effort Estimate:** 4–6 hours (writing and applying migration, updating function, testing query accuracy).

## Limited Multi-Document Query Support

**Priority:** P2  
**Category:** Feature Gap / UX  
**Current Issue:** The platform advertises analyzing documents, but the UI only allows attaching one document at a time to a chat session. Internally, the API does support an array of documentIds in requests[[40]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L30-L39), and the code merges context if multiple IDs provided. However, the front-end has no UI to select multiple files for a single query. Users cannot easily ask comparative questions like “Compare the cap rates between Document A and Document B” because they can only attach one document (the state selectedDocumentId is a single value, not a list[[41]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L45-L48)). There’s also no interface to search across *all* documents at once (unless the user manually starts a chat without selecting a doc, which the system doesn’t explicitly support except maybe via commands).

**Impact:** **Reduced Analytical Power.** CRE professionals often compare multiple OMs or need portfolio-level insights. The lack of multi-doc or global search means they must query each document separately and manually compare answers, which is inefficient. It’s a missed opportunity for a high-value feature that could differentiate OM-AI (e.g., side-by-side document analysis or aggregate Q&A across docs).

**Suggested Fix:** Introduce UI mechanisms for multi-document queries: - **Multi-select UI:** In the document attachment flow, allow users to select more than one document. This could be a dropdown with checkboxes or a dedicated “Attach documents” modal listing all docs with multi-select. Update state to hold an array of selected IDs. - **Global Search/Ask:** Provide an option to query “All Documents”. For example, in the ChatWelcome screen, an option “Ask a question across all uploaded documents” that sets the document context to all user’s docs. - **Comparison Mode:** Eventually, implement a comparison view: user picks two documents and the app can produce a comparison report. This could leverage a specialized prompt or simply allow the question to mention both, since GPT has both contexts.

From a backend perspective, the infrastructure is already there (the /api/chat endpoint handles multiple IDs in documentContext.documentIds). It may need tweaking to fetch more chunks or smarter merging of context from multiple sources, but the heavy lift is UI/UX.

**Code Example:** Outline enabling multi-select:

\*\*Claude Code Prompt:\*\* Enable multi-document selection for queries:  
1. Modify state in `AppPage` (`src/pages/app.tsx`) to hold an array of selected document IDs instead of a single. E.g.:  
 ```tsx  
 const [selectedDocumentIds, setSelectedDocumentIds] = useState<string[]>([]);  
 ```  
 and maybe a parallel array or map for names if needed.  
2. Create a UI control for multi-selection. Perhaps reuse the sidebar’s document list: allow the user to check some boxes next to documents. For a quick approach, when the user clicks a document in the sidebar now, it calls `onSelectDocument(id)` which sets a single ID[42]. Instead, change it so Ctrl+click (or a checkbox in each item’s dropdown) toggles selection without closing the dropdown.  
 - You might add a checkbox in `DocumentList` items for multi-select mode.  
 - Or have an “Attach multiple” button that opens a modal with all docs and checkboxes.  
3. When `selectedDocumentIds` has multiple entries, update how you call `sendMessage`. In `useChatPersistent.sendMessage`, currently it sends one `documentId`[43]. Change that to:  
 ```js  
 documentContext: { documentIds: selectedDocumentIds }  
 ```  
 and adjust the payload accordingly.  
4. Also adjust the attached document indicator UI: if multiple, it could say “Attached: 3 documents” or list their names separated by comma (maybe truncated).  
5. Test by selecting 2 or more docs and asking a question like "Compare the NOI of these documents." Verify that `req.body.documentContext.documentIds` includes both IDs on the server (and that the backend returns a combined context in the answer).

**Visual:** *Example UI concept:* Instead of a single “Document attached” pill, you might see “Attached: Document A, Document B” or “Attached: 2 documents”. The sidebar or a modal would allow ticking checkboxes next to “Doc A” and “Doc B”. The result from GPT-4 would then reference both, e.g. “In Document A, the cap rate is X, whereas in Document B it is Y.”

**Effort Estimate:** 1–2 days for a robust UI (less if just doing a simple multi-select without fancy design). This is a medium complexity feature but high ROI for user value.

## Insufficient Rate Limiting & Abuse Protection

**Priority:** P2  
**Category:** Backend Security/Scaling  
**Current Issue:** The app implements a rudimentary in-memory rate limit in the chat API (withRateLimit(req.user.id, 20, 2, ...) allowing 20 requests burst, 2 per minute refill)[[44]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L113-L121). However, this is not distributed – each serverless instance has its own bucket map, meaning an attacker could bypass limits by invoking new instances (or after a restart). There’s also no rate limit on file uploads or other endpoints. No IP-based throttling either. Additionally, if an attacker scripts rapid requests, the in-memory map might not protect well at scale (and on Vercel, functions may cold-start without the previous memory, resetting the limit).

**Impact:** **Potential Abuse and Cost Spike.** Without robust rate limiting, a malicious or buggy client could spam the OpenAI API with requests, incurring high costs or saturating the system. Also, someone could attempt a denial-of-service by rapid uploads or chat messages. The current token bucket is per-user but doesn’t handle cross-user or overall system load.

**Suggested Fix:** Implement a more robust rate limiting and monitoring: - Use an external store for rate-limits (Redis or Supabase DB) to persist counts across instances. For example, maintain a small table of user\_id -> request\_count per period, and check/update it in transactions. Or use a Redis INCR key with expiry for IP addresses and user IDs. - Consider IP-based limits for unauthenticated routes (like login attempts or signup, if any) and for file upload to prevent flooding. - Additionally, enforce OpenAI usage limits per user according to their subscription. The code has SUBSCRIPTION\_LIMITS and usage tracking fields in DB[[45]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171800_initial_schema.sql#L12-L20)[[46]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171900_rls_policies.sql#L10-L18), but I did not see enforcement logic live. Implement checks before sending a chat: e.g., if user is on starter plan and already asked 10 questions today, deny further requests or ask them to upgrade. - Use built-in Vercel Edge Middleware or API middleware to throttle if needed. You might also integrate Sentry or logging to flag suspicious usage patterns.

By strengthening rate limits, you ensure one user can’t inadvertently or deliberately exhaust your OpenAI quota or degrade service for others.

**Code Example:** Since we can’t demonstrate Redis here, here’s a conceptual prompt to improve:

\*\*Claude Code Prompt:\*\* Improve rate limiting beyond in-memory:  
1. Install a lightweight KV store (like Upstash Redis or Supabase's key-value store if available).  
2. Create a utility (e.g. `lib/rate-limit.ts`) that, given a key (user ID or IP) and limit, will atomically increment a counter and return whether it’s within limit. For example, in Redis use `INCR` and set expiry of 1 minute for per-minute counts.   
3. Replace the `withRateLimit` usage in `src/lib/auth-middleware.ts` or specifically in `api/chat.ts`. Instead of the current Map logic[47][48], call the new rateLimit utility. For example:  
 ```ts  
 const allowed = await rateLimit.check(req.user.id, { perMinute: 2, perDay: 50 });  
 if (!allowed) return createApiError(res, ERROR\_CODES.RATE\_LIMIT\_EXCEEDED);  
 ```  
 This would check that the user hasn’t exceeded X per minute or day.  
4. Also implement a simple IP rate limit for file uploads in `/api/supabase-upload.ts`. You can use `req.headers['x-forwarded-for']` to get IP and similarly throttle IP to, say, 5 uploads per minute.  
5. Test by simulating more than the allowed calls via unit test or script, ensure the API starts returning 429 or your custom error after threshold. Also test normal usage is unaffected.

**Visual:** *N/A (system behavior)* – After this, if a user sends requests too fast, they might see an error like “Rate limit exceeded, please slow down,” rather than the assistant continuing to respond (or incurring huge hidden costs). This protects overall stability.

**Effort Estimate:** 4–6 hours for a basic Redis-based limit (plus infra setup if not already), slightly more if implementing detailed subscription-tier enforcement.

## Lack of Automated Testing for PDF Parsing and Edge Cases

**Priority:** P3  
**Category:** Code Quality / Maintainability  
**Current Issue:** Critical logic like PDF validation and parsing (PDFParserAgent, EnhancedPDFParser) has no unit or integration tests in the repository. There are tests for the chat API[[49]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/__tests__/chat.test.ts#L90-L98)[[50]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/__tests__/chat.test.ts#L104-L112), but none explicitly for uploading and parsing various PDFs. This means potential bugs (like parsing failures on certain PDF structures, OCR mis-triggering, etc.) can go unnoticed until a user encounters them. For example, edge cases like encrypted PDFs, malformed files, or very large scans might not be handled gracefully – without tests, it’s hard to verify improvements (like the table extraction accuracy or multi-page handling mentioned in scope).

**Impact:** **Uncertain Reliability & Regression Risk.** Lack of tests makes it risky to change the parsing logic for improvements, since you might inadvertently break something (e.g., text chunking). It also means new contributions (Phase 2 advanced analysis) could introduce bugs in old features. For users, this translates to occasional unexplained failures (document stuck in “error” state) that could have been caught in development.

**Suggested Fix:** **Add tests and sample files** to cover parsing: - Create a set of representative sample PDFs (small ones) and include them in a test assets folder. E.g., a 1-page text-only PDF, a PDF with tables, a scanned image PDF (to test OCR), an encrypted PDF with known password or expected failure. - Write unit tests for PDFValidator (feed it valid vs invalid data and assert it flags them properly). - Write integration tests for PDFParserAgent.parseBuffer and EnhancedPDFParser.parseBuffer. You can simulate a Buffer (read the sample PDF file in the test) and then assert that: - parseResult.success is true for normal PDFs, and false for corrupted ones. - The number of pages and chunks matches expectations. - Tables are detected (if sample PDF has a known table, ensure tables.length > 0 and headers match). - OCR kicks in only when expected (e.g., if you have a scanned PDF sample, set performOCR: true and assert the returned text contains an expected phrase from the image). - Also test the end-to-end of upload -> process -> chunks saved. Possibly using a mocked supabase client to verify that after calling processUploadedDocument, the database insert payload contains the right data (you can spy on supabase.from('documents').insert as done in chat tests with jest mocks).

**Code Example:**

\*\*Claude Code Prompt:\*\* Add parsing tests:  
1. In `src/lib/agents/pdf-parser/\_\_tests\_\_/PDFParserAgent.test.ts` (create this file), set up tests for parseBuffer. Use Node’s fs to read a small PDF file into a Buffer. For example:  
 ```ts  
 import { PDFParserAgent } from '@/lib/agents/pdf-parser';  
 import \* as fs from 'fs';  
 import \* as path from 'path';  
  
 describe('PDFParserAgent', () => {  
 let parser: PDFParserAgent;  
 beforeAll(() => { parser = new PDFParserAgent(); });  
  
 test('parses a simple text PDF successfully', async () => {  
 const pdfBuffer = fs.readFileSync(path.resolve(\_\_dirname, 'fixtures/simple-text.pdf'));  
 const result = await parser.parseBuffer(pdfBuffer);  
 expect(result.success).toBe(true);  
 expect(result.pages.length).toBe(1);  
 expect(result.fullText).toContain('Hello World'); // assuming that text is in the PDF  
 expect(result.tables.length).toBe(0);  
 });  
 });  
 ```  
 Provide similar tests for a PDF with a table, etc.  
2. Do likewise for EnhancedPDFParser if that is actively used.  
3. Also consider adding a test for the `/api/process-document` flow: simulate a call with a known Buffer and verify that `documents` and `document\_chunks` would be inserted. This might require refactoring to allow injecting a fake supabase client or using a test database.  
4. Run `npm test` to ensure these new tests pass. Commit a few sample PDFs under a `test/fixtures` directory (keep them small to not bloat repo).

**Visual:** *N/A* – This is an internal improvement. However, having tests means higher confidence; e.g., after making changes to OCR logic, you can run tests to ensure an image-based PDF still gets text extracted. Over time, this yields a more stable experience for users (fewer parsing bugs reaching production).

**Effort Estimate:** 1 day to write a handful of tests and set up sample files.

## Weak Typing and Documentation (Technical Debt)

**Priority:** P3  
**Category:** Code Maintainability  
**Current Issue:** The codebase has TypeScript “strict” mode off (strict: false noted in Project Status[[51]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/PROJECT_STATUS.md#L144-L152)). Many types are defined but often as optional or using any in places (for instance, PDFAnalyzer.detectTableStructure accepts an untyped array of textItems). This weak typing can mask bugs (e.g., a function might return undefined but callers don’t expect it). Additionally, documentation is mostly in README and some markdown files; inline code comments explaining complex logic (like chunk scoring algorithms in chat.ts[[52]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L222-L231)[[53]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L277-L285)) are sparse. New developers or contributors might struggle to grasp these without more comments or up-to-date docs.

**Impact:** **Developer Efficiency & Future Bugs.** It’s easier to introduce errors when types aren’t strict – e.g., a refactor might change a function signature and some usages won’t error due to any’s. Also, lack of comments on intricate parts (like the multi-pass chunk relevance filtering in chat) makes it harder to modify or optimize those sections confidently. While this doesn’t directly affect end-users right now, it slows down development and increases the chance of regressions slipping in.

**Suggested Fix:** Gradually enforce stricter typing and add documentation: - Turn on strict in tsconfig.json and fix resulting errors. This is a big task; you can do it module by module. Focus on critical modules like the parsing agents and API handlers to ensure types line up (e.g., define a proper type for Chunk that covers both PDFParser and EnhancedParser outputs, rather than using {...} as any or making fields optional). - Remove redundant type conversions. The Project Status notes a fix where TextChunk interface was adjusted to unify fields[[54]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/PROJECT_STATUS.md#L34-L42)[[55]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/PROJECT_STATUS.md#L44-L51) – continue that process for all overlapping types (for example, ensure ParsedPage and pages returned by EnhancedParser align on field names/types). - Add JSDoc comments or inline comments explaining non-obvious logic. For example, in chat.ts above the block that does financial keyword scoring[[52]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L222-L231), explain what the goal is (“// If the search function fails, manually score chunks by financial terms to prioritize key content”). - Update the README’s technical sections if any behavior changed (the README is quite good, just ensure any new limits or features are reflected).

This is more of a maintenance suggestion – perhaps set aside time each sprint to convert a portion of the code to strict typing.

**Code Example:** (Turning on strict mode yields many errors, but here’s how to start)

\*\*Claude Code Prompt:\*\* Improve TypeScript strictness:  
1. Open `tsconfig.json` at project root. Set `"strict": true` under compilerOptions. Also ensure `"noImplicitAny": true`, `"strictNullChecks": true`, etc.  
2. Run `npm run typecheck` to see errors. Triage them:  
 - A lot will be easy (explicitly define types for objects that were implicitly any).  
 - For example, in `PDFParserAgent.parseBuffer`, if `parseResult` is being assigned from a try/catch, you might need to define its type as `ParseResult | null` and handle that.  
 - Fix function signatures, e.g., if `processUploadedDocument` can return an error string, define its return type as `Promise<ProcessDocumentResult>` and ensure all code paths return { success, ... } objects (which you do).  
3. Add types to the Supabase client responses instead of using generic any. You can import your `Database` type (as you do) and use it for .from() calls for stronger typing on inserts/selects.  
4. Address any places using `any`. For instance, `OCRProcessor.worker` is `any`[56] – you could import the Worker type from tesseract.js if available, or at least make it `unknown` and narrow usage.  
5. After making the code compile under strict, run all tests to ensure nothing broke.  
6. Add comments to critical blocks: e.g., explain the magic numbers or regex in `utils.ts` financial extraction[57][58].

**Visual:** *N/A.* The outcome is improved code quality – which indirectly means fewer runtime errors for users. (For example, strict null checks might catch a case where validationResult.metadata.hasText is undefined and prevent a bug where OCR is wrongly triggered or not triggered.)

**Effort Estimate:** Multiple days if done fully – could be phased. Probably 2–3 days to fix the most important parts and get basic strict compliance, with incremental improvements thereafter.

## OM-to-OM Comparison Tool (Feature Suggestion)

**Priority:** P3  
**Category:** High-Impact Feature (Product Planning)  
**Current Issue:** Currently, users can analyze one document at a time (or with difficulty, multiple as discussed). There is no dedicated tool or interface to directly compare two offering memorandums. CRE professionals often want to compare metrics between deals (cap rates, NOIs, prices, etc.). Right now, they’d have to manually ask for differences via chat or export data and compare externally.

**Impact:** **Opportunity for User Value Boost.** This is not a bug but a feature gap. Implementing an OM comparison feature could greatly enhance the platform’s usefulness. It’s likely in the roadmap (Phase 3 “advanced CRE-specific features” might include it), but not yet started.

**Suggested Fix:** Design and build an OM comparison experience: - **UI:** A screen where a user selects two (or more) documents from their library to compare. Possibly accessible via a “Compare Documents” button. Show key attributes side by side – e.g., property address, purchase price, cap rate, NOI, lease terms, etc. This could be presented in a table or a generated report. - **Data extraction:** Leverage the parsing results: many key metrics might be in document\_analysis or can be derived from document\_chunks. You could create a function that given two doc IDs, pulls known fields (maybe from the metadata JSON where you already extract some CRE\_PATTERNS like capRate, noi[[59]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/services/pdf/enhanced-parser.ts#L42-L50)). If those patterns successfully found data (the EnhancedParser stores them perhaps in parseResult structuredData), you can present them. For missing fields, you could query GPT with a prompt specifically asking to compare (though that would incur cost). - **AI-generated summary:** As a fast MVP, you might simply send both docs’ content (or summaries) to GPT-4 with a system prompt “Compare these two offerings in terms of key financial metrics and highlights.” Then display the response. But this could be expensive and slow for big docs. A better approach is to pre-extract common fields and only use AI for qualitative comparison. - **User flow:** After selecting docs and hitting compare, show a loading indicator while analysis occurs (maybe call a new /api/compare endpoint). Then display either a nicely formatted text summary or a visual comparison (if you have structured data, you can do charts or tables).

This is a substantial feature, so consider it for a future release. In the meantime, document how a user might manually compare via chat (they could attach two docs as above and ask GPT to compare – once multi-select is possible).

**Code Example:** Outline for a compare endpoint:

\*\*Claude Code Prompt:\*\* Plan an OM comparison feature:  
1. Create a new Next.js page at `src/pages/compare.tsx` with a UI to select documents (a multi-select similar to earlier suggestion). Perhaps use a pair of dropdowns: "Select Document A" and "Select Document B".  
2. When two documents are selected, enable a "Compare" button. On click, call a new API endpoint (POST `/api/compare`) with the two document IDs.  
3. For the backend (`src/pages/api/compare.ts`), implement logic:  
 - Fetch each document’s metadata (from `documents` table) and perhaps `document\_analysis` if you have one.  
 - You might use existing extraction patterns (for cap rate, NOI, etc.). Possibly reuse `extractCREData` from EnhancedPDFParser[60][61] by running it on each doc’s full text (which you have in `fullText` or in DB).  
 - Compile a small JSON of key metrics for each doc.  
 - If using AI, feed those metrics and maybe a short summary of each doc into GPT with a prompt: "Compare Document A vs B: A has NOI X, B has NOI Y, ..." etc. But try to limit tokens.  
 - Return a structured response (e.g., { comparisonText: "...", metrics: {...} }).  
4. In the compare page UI, display the `comparisonText` or generate a table of metrics differences.  
5. Style it clearly: e.g., a two-column layout with Document A and B labels and values.

**Visual:** *Concept Wireframe:*

| Metric | Document A | Document B |  
|-----------------|---------------------|---------------------|  
| Cap Rate | 5.2% | 5.8% |  
| NOI | $1,200,000 | $1,100,000 |  
| Occupancy | 95% | 88% |  
| ... | ... | ... |

And below, an AI-generated paragraph: “Document A (123 Main St) offers a slightly higher NOI and occupancy, whereas Document B (456 Oak Ave) has a higher cap rate but more vacancy. Document A is multifamily vs Document B retail, which explains differences in ...” etc.

**Effort Estimate:** 1 week for a basic implementation; more if deep integration and polished UI are desired.

## Other Notable Suggestions (P3 Backlog)

* **Export Conversation or Data:** Provide options to export chat Q&A or extracted tables. E.g., a “Download Q&A as PDF” or “Export tables to CSV” button. This is a nice-to-have that can enhance user workflow (so they can include insights in reports). Effort: ~1 day per export type.
* **Real-time OCR Progress UI:** If processing large scanned PDFs, show a progress bar (since Tesseract provides progress callbacks[[17]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/agents/pdf-parser/utils.ts#L28-L36)). This would improve UX by not leaving users guessing during long OCR tasks.
* **Accessibility Audits:** Ensure color contrast (the app uses Tailwind default, mostly OK but check light gray text on white, e.g., placeholder text might be low contrast). Add aria-label to icon-only buttons like the paperclip “Attach” (currently it has a title but better to add aria-label="Attach file" for screen readers[[62]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L421-L429)). Minor focus state tweaks: e.g., ensure the modal close X is keyboard-focusable (it’s a button, so likely yes).
* **Monitoring & Logging:** Integrate Sentry properly (it’s mentioned as ready in config). Ensure any caught exceptions (especially in parsing or OpenAI errors) call Sentry.captureException. Also consider logging key usage stats (like number of pages processed, time taken) to a monitoring dashboard to watch performance trends.
* **Performance Optimizations:** lazy-load heavy libs on the client if any (though most heavy work is server-side). Check bundle analyzer – if pdfjs or tesseract is not ending up in client bundle (shouldn’t, since they’re used in Node context). Tailwind and shadcn ensure a fairly optimized UI bundle already.

Each recommendation above is aimed at improving OM-AI’s robustness, user satisfaction, and scalability. Addressing the P0/P1 issues (security and critical UX blockers) should come first. Quick wins like showing document names and enabling deletion will immediately improve user trust. Longer-term enhancements (background processing, multi-doc comparison) will set the product apart in the market.

Please let me know if you need further clarification on any of these points. Good luck with the implementation!

[[1]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L24-L32) [[2]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L36-L44) [[5]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L2-L10) [[6]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L34-L42) [[7]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L74-L77) [[33]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L38-L47) [[34]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L40-L48) [[35]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L38-L44) [[38]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql#L42-L50) 20250807\_fix\_document\_search.sql

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250807_fix_document_search.sql>

[[3]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L42-L50) [[4]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L54-L62) [[10]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L56-L65) [[11]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L64-L71) [[29]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L24-L31) [[31]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts#L94-L102) supabase-upload.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/supabase-upload.ts>

[[8]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171900_rls_policies.sql#L22-L30) [[9]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171900_rls_policies.sql#L56-L64) [[46]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171900_rls_policies.sql#L10-L18) 20250722171900\_rls\_policies.sql

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171900_rls_policies.sql>

[[12]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/document-processor.ts#L82-L90) document-processor.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/document-processor.ts>

[[13]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/upload.ts#L84-L93) [[24]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/upload.ts#L196-L204) [[28]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/upload.ts#L34-L41) upload.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/upload.ts>

[[14]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/process-document.ts#L113-L122) process-document.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/process-document.ts>

[[15]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L54-L62) [[16]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L67-L75) [[18]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L146-L155) [[19]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L148-L155) [[20]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L150-L155) [[42]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx#L101-L108) DocumentList.tsx

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentList.tsx>

[[17]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/agents/pdf-parser/utils.ts#L28-L36) [[56]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/agents/pdf-parser/utils.ts#L5-L13) [[57]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/agents/pdf-parser/utils.ts#L112-L121) [[58]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/agents/pdf-parser/utils.ts#L136-L144) utils.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/agents/pdf-parser/utils.ts>

[[21]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L365-L373) [[22]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L375-L379) [[23]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L472-L480) [[41]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L45-L48) [[62]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx#L421-L429) app.tsx

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/app.tsx>

[[25]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/AppSidebar.tsx#L98-L106) [[26]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/AppSidebar.tsx#L96-L104) AppSidebar.tsx

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/AppSidebar.tsx>

[[27]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentUpload.tsx#L124-L132) [[30]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentUpload.tsx#L224-L232) [[32]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentUpload.tsx#L150-L158) DocumentUpload.tsx

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/components/app/DocumentUpload.tsx>

[[36]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L184-L193) [[37]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L202-L211) [[39]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L206-L214) [[40]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L30-L39) [[44]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L113-L121) [[52]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L222-L231) [[53]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts#L277-L285) chat.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/chat.ts>

[[43]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/hooks/useChatPersistent.ts#L108-L116) useChatPersistent.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/hooks/useChatPersistent.ts>

[[45]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171800_initial_schema.sql#L12-L20) 20250722171800\_initial\_schema.sql

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/supabase/migrations/20250722171800_initial_schema.sql>

[[47]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/auth-middleware.ts#L87-L96) [[48]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/auth-middleware.ts#L98-L107) auth-middleware.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/auth-middleware.ts>

[[49]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/__tests__/chat.test.ts#L90-L98) [[50]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/__tests__/chat.test.ts#L104-L112) chat.test.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/pages/api/__tests__/chat.test.ts>

[[51]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/PROJECT_STATUS.md#L144-L152) [[54]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/PROJECT_STATUS.md#L34-L42) [[55]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/PROJECT_STATUS.md#L44-L51) PROJECT\_STATUS.md

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/PROJECT_STATUS.md>

[[59]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/services/pdf/enhanced-parser.ts#L42-L50) [[60]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/services/pdf/enhanced-parser.ts#L180-L188) [[61]](https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/services/pdf/enhanced-parser.ts#L181-L189) enhanced-parser.ts

<https://github.com/mgisrael393823/OM-AI/blob/f6797596ff1545a5e98e75fddb5f509d703a2103/src/lib/services/pdf/enhanced-parser.ts>