ChatGPT

Intelligent eCTD Co-Author System Design Overview

The **intelligent eCTD Co-Author system** is a comprehensive platform for regulatory document authoring, collaboration, and submission assembly. It combines a Google Docs-based collaborative editor, DocuShare DMS integration for version control, GPT-4 AI assistance, and eCTD-specific workflow tools to streamline the creation of compliant submissions. Key capabilities include real-time co-authoring, automated formatting and compliance checks, multi-region template management, lifecycle controls (Draft → Review → Final → Locked), and one-click export to a valid eCTD package. The system is built on a modern web stack (Node/Express + Supabase backend, React + Tailwind frontend) and is designed to run on Replit with proper OAuth and secret configuration [1] [2]. The goal is to accelerate regulatory writing while ensuring **ICH/FDA eCTD compliance** at every step [3].

Architecture and Platform Components

The platform follows a **client–server architecture** with seamless integration to external services (Google Docs, DocuShare, OpenAI). The user interacts through a React single-page application, while the Node/Express backend orchestrates document storage, AI calls, and eCTD packaging. High-level components include:

- **React Frontend** – Provides a rich UI for document editing, template browsing, search, and export. It embeds Google Docs in an iframe for the editor and includes side panels for AI assistance and metadata.
- **Node/Express API Backend** – Handles business logic and integrates with external APIs. It uses Supabase (PostgreSQL) for persistent data (document metadata, user info, audit trails) [4]. The backend exposes RESTful endpoints for document management, state transitions, AI suggestions, and export.
- **Supabase Database** – Stores application data such as user profiles, document records (with links to Google Docs IDs and DocuShare IDs), version history, templates, and comments. For example, a **Documents** table stores each doc's unique ID, title, associated module, region, Google Doc ID, DocuShare handle, status (Draft/Review/etc), and current version [5].
- **Google Docs & Drive** – Acts as the collaborative editing engine. Documents are created/edited in Google Docs (via embed) and can be exported via the Google Drive API (e.g. to PDF or DOCX) [6]. Google OAuth2 is used for secure access.
- **Xerox DocuShare** – Serves as the regulated Document Management System (DMS) "vault". All official versions are saved to DocuShare via its API, enabling audit logs, access control, and long-term archiving [7]. DocuShare manages versioning (e.g. Version-12345) and retains a single source of truth for each document's history.
- **OpenAI GPT-4** – Provides AI co-authoring features. The backend calls OpenAI's API for content suggestions, drafting assistance, and compliance checks. Prompt engineering aligns the AI output with regulatory writing norms (details in AI section) [8].

- **Replit Hosting** – The application is containerized to run on Replit. All secrets (Google API keys, DocuShare credentials, OpenAI key, etc.) are stored securely via Replit's Secret Manager, and persistent files (e.g. temp PDFs or the compiled submission zip) are stored in `/mnt/data` within the Replit environment [9] [10]. The Node server serves the React app and APIs on a single Replit instance for one-click deployment.

This architecture ensures each piece is modular yet integrated. The **React frontend** provides a smooth UX, while the backend ensures regulatory requirements (version control, audit trail, XML generation) are enforced behind the scenes. External integrations are abstracted behind the API – e.g. the app uses a `/documents` API call, and the server handles whether that means fetching from Google or DocuShare.

Frontend UI Design and Components

The UI is organized into distinct components and pages, each mapped to a core feature of the system. We use Tailwind CSS for a clean, responsive design consistent with enterprise applications (similar to Veeva Vault's UI style). Below is a component-by-component breakdown of the interface:

- **Dashboard Page** – The main landing page (home screen) providing access to all modules of the system. It shows:
- *Recent Documents*: a list of documents the user has recently worked on.
- *Module Progress Overview*: a summary of completion status for each eCTD module in the current project (e.g. which sections are in Draft, which are Final).
- Navigation cards or links to other sections: **Document Editor**, **Template Library**, **Regulatory Search**, and **Export**.

- A user menu (profile info, logout, etc.).

- **Document Editor Page** – A full-featured, live editing environment for eCTD documents (Modules 1–5). Instead of building a custom editor, we embed Google Docs for rich text editing and real-time collaboration [11]. Key UI elements on this page:

- *Google Docs Iframe*: The central area displays the embedded Google Doc editor (with the Google toolbar for formatting, comments, suggesting changes, etc.). This allows multiple users to edit simultaneously, with Google's concurrency handling.
- *AI Assistant Sidebar*: A collapsible side panel powered by GPT-4 suggestions. Users can open this panel to see context-specific writing suggestions, ask the AI to draft a section, or run a compliance check. For example, as the user types, the AI can suggest the next sentence or a clearer phrasing [12]. The sidebar also displays any **real-time compliance warnings** (e.g. missing required sections) and can be toggled on/off.
- *Document Info Panel*: Another sidebar or footer section showing metadata – e.g. the document's module, region, current status (Draft/Review/etc), version number, and links to any related documents (for cross-references).

- *Toolbar Extensions*: In addition to Google's native toolbar, we provide custom buttons: **Insert Template**, **Submit for Review**, **Approve/Finalize**, etc., to integrate our workflow. For instance, an "Insert Template" menu lets users pull in a predefined template snippet for the section they're writing (see Template Library below) [13] [14]. A "Draft with AI" button can trigger the AI to auto-generate content for an empty section based on prompt templates.

- **Template Library Page/Panel** – A library of pre-approved templates for common eCTD documents, organized by region and module [13]. Users can browse or search templates (e.g. an FDA Form 1571 template, a Module 2 clinical overview template, etc.) and insert them into a document:

- The Template Library is accessible from the dashboard ("Browse Templates") and also from within the editor (via the Insert Template action) [14].
- The UI presents a categorized list or tree: first select region (Global/FDA/EMA/PMDA/etc), then module (1 to 5), then the specific document type. For example, **Module 1** templates are grouped by region since Module 1 is region-specific (FDA forms, EMA cover letter, PMDA application forms, etc.), whereas **Modules 2–5** templates are ICH common format (with minor regional tweaks as needed).
- Each template entry shows a name and description (e.g. *"FDA 1571 – Investigational New Drug Application Form"* or *"Module 2.5 Clinical Overview template"*). Clicking it may show a preview or details, and allows insertion into the active document.
- Templates ensure correct heading structure, required sections, and even include "boilerplate" text or prompts for authors. (For instance, a Module 2 summary template might include all the expected headings and a short description of what to write in each section.)

- *Template import:* When a template is inserted, the system can replace placeholders with project-specific metadata (like drug name, applicant name, etc.) if known, and apply consistent formatting.

- **Regulatory Search Panel** – An intelligent search interface to query past submissions and reference materials. This can be a modal or dedicated page accessible via "Explore Knowledge Base" on the dashboard [15]. Features:

- A search bar where users enter keywords or questions (natural language queries).
- Results are pulled from an indexed repository of **prior submission documents** (e.g. all final approved documents in DocuShare) and possibly internal guidance docs. For example, a user writing a toxicology summary can search "previous Phase I tox study summary" to find how similar content was written before [16] [17].
- Results list shows snippets of matching text with highlights, document title, and context. The user can click to view the full document (read-only) or copy relevant text.
- Optionally, an **AI Q&A** feature: the user can ask a question ("What are the common requirements for an EU Module 1?"), and the system uses GPT-4 with retrieval to answer based on the indexed data [18] [19]. This helps users leverage institutional knowledge quickly.

- The search respects access controls – users only see results from submissions they have permission to view [20].

- **Module Progress & Validation Dashboard** – Part of the dashboard or a separate view that tracks the completion status of each section of the eCTD and any validation issues:

- It might display a checklist of all required documents per region (for the product's submission) with status indicators (e.g. green check for completed Final documents, orange for in-progress, red for not started or flagged issues).
- Built-in validation rules are surfaced here. For example, it will show warnings like "Missing Module 2.7.4 Summary of Clinical Safety" if that document hasn't been drafted, or "Module 1 US FDA Form

3674 not yet uploaded" if required. It will also check formatting compliance (like "Some PDFs are not PDF/A" or "Hyperlinks missing in Module 2.5 PDF") as part of pre-publish checks [21] [22].

- Users can click on a section from this dashboard to jump to creating or editing that document (prompting use of the correct template).

- **Export Page** – The interface to generate the eCTD package. Users choose export options via a form or wizard:

- Select target **region(s)** for export (e.g. FDA, EMA, etc., since Module 1 differs by region). They can export one region at a time or generate multiple regional packages if needed.
- Select **sequence number** or submission type if applicable (e.g. initial submission vs a sequence update – for MVP we assume initial).
- The system then displays which documents will be included (with checkboxes, all required by default). Users confirm everything is Final.
- Clicking **"Compile eCTD"** triggers the backend process. A progress indicator is shown, as this may take some time (converting files, assembling XML) [23]. Once done, a download link for the eCTD ZIP appears.
- The page also provides options like "Export to Vault" (to save the package in DocuShare) or "Download as PDF Package" (maybe just a combined PDF for review).
- Basic export settings (like "include bookmarks in PDFs" or "OCR scan images") could be toggled, but defaults are set to compliance standards (PDF text searchable, etc.).

Throughout the UI, consistency and ease-of-use are paramount. Short, guided interactions (with tooltips, help text) ensure that even complex tasks like compiling an eCTD are broken down into clear steps. The design takes inspiration from existing tools like Veeva Vault and Certara's CoAuthor to provide **transparency, consistency, and collaboration** for regulatory teams [24] [25].

Backend API Structure and Database Schema

The backend is a Node.js Express application that exposes a set of RESTful APIs to the frontend. It also connects to external APIs (Google, DocuShare, OpenAI) and the Supabase Postgres database. Key aspects of the backend include:

- **Authentication & OAuth2**: Secure OAuth flows are implemented for Google and DocuShare:
- `GET /auth/google` – Redirects the user to Google OAuth2 consent. A callback endpoint handles Google's response and obtains an access token (and refresh token) for the user [26]. The tokens are stored (encrypted) in the database, associated with the user's account.
- `POST /auth/docushare` – Accepts DocuShare credentials or tokens (depending on DocuShare's auth method, possibly an OAuth or API key). This is used to log in to DocuShare and store a token/session for API calls [27]. (If DocuShare supports SSO/OAuth, a similar redirect flow would be used.)

- We store these tokens in the **Users** table along with user profile info (name, email) [5]. The user login to our app itself can be via Google Sign-In (linking the accounts). All API calls require an active session (JWT or Supabase Auth) to ensure only authenticated users use the system [28].

- **Core API Endpoints**:

- `GET /documents` – List documents the user has access to (with metadata: title, module, status, last modified, etc.) [29] . Supports query params to filter by project, module, or status.
- `POST /documents` – Create a new document. The user provides required metadata (at least title, module, region, maybe template to use). The server then:
    - Creates a new Google Doc via Google Drive API (possibly by copying a template doc if one is specified) [30] .
    - Creates a placeholder in DocuShare via its API (e.g. an empty document record to reserve an ID) [31] .
    - Stores the document record in the database (including Google Doc ID, DocuShare ID, initial state = Draft).
    - Returns the new document's info (including the URL or ID needed to embed the editor).
- `GET /documents/:id` – Fetch detailed info on a specific document, including its current content link. This may construct the **Google Docs embed URL** (which includes the document ID and perhaps an OAuth access token or uses Google's embed capability). The frontend uses this to load the doc in the iframe.
- `POST /documents/:id/save` – Saves a version of the document. (This might not be needed if Google auto-saves, but we may call it when moving to a new state). This would export the latest content from Google Docs (as DOCX or PDF) via the API and upload that version to DocuShare [32] .
- `POST /documents/:id/state` – Change the state of the document (for lifecycle control). E.g., move Draft → In Review, or Review → Final. The server will enforce allowed transitions and trigger side effects (details in the Lifecycle section) [33] .
- `POST /documents/:id/export` – Export an individual document (or a set of docs) to PDF. For instance, if a user wants to download a single file. This uses Google Drive API to convert to PDF and returns the file (or stores it).
- `POST /submissions/export` – Trigger full eCTD compilation (as described above in Export Page). The request includes which region or modules to include. The server will gather all relevant docs in Final state, perform conversions and XML generation, and return the packaged ZIP. (See **eCTD Export Pipeline** below for details.)
- `GET /templates` – List available templates (with filtering by region/module).
- `POST /templates` – (For admin) create or update a template (upload new template content).
- `POST /ai/suggest` – Accepts some input (e.g. document ID or text segment and a prompt type) and returns an AI-generated suggestion from GPT-4 [34] . For example, the frontend might send the last paragraph and ask for a rewrite.
- `POST /ai/validate` – Sends a section or document to GPT-4 asking for compliance check or missing content analysis, and returns the result (a list of issues or recommendations).
- `GET /search?query=...` – Searches across stored documents (could integrate with Supabase full-text search or a vector index) and returns matches (used by the Regulatory Search UI).

All endpoints enforce authorization checks. For example, only users with appropriate roles can change state or export. Express middleware will verify the user's JWT from the frontend on each call (we might use Supabase's built-in auth middleware). Additionally, certain actions (like state transitions or admin template changes) will check the user's role (author vs reviewer vs admin).

- **Database Schema** (Supabase/Postgres):
- **Users**: (id, name, email, role, google_oauth_token, docushare_token, etc.) – Stores user identity and tokens. Role could be an enum {Author, Reviewer, Admin} for basic access control.

- **Documents**: (id, title, module, region, status, google_doc_id, docushare_id, current_version, created_by, created_at, etc.) – Each document's metadata. `module` could be like "M2.5" or "2.5" to indicate location, and region might be "US" "EU" etc. `status` holds Draft/Review/Final/Locked. `current_version` links to the latest entry in DocVersions.
- **DocVersions**: (id, document_id, version_number, docushare_version_id, state_at_save, timestamp, exported_pdf_path, exported_xml_path) – Tracks each saved version. When a document reaches Final, for instance, a new version entry is created referencing the file stored in DocuShare (DocuShare will have its own versioning, but we track for convenience as well) [35] . `state_at_save` notes if that version was Draft vs Final, etc.
- **Templates**: (id, name, module, region, description, file_path, created_by) – Stores the master templates available. `file_path` might point to a Google Doc ID or an HTML fragment or Markdown that is inserted. (We might also store templates as Google Docs that are copied on use, for fidelity of formatting.)
- **Comments** (optional): We may not need a separate comments table if we rely on Google Docs comments. But if we want to consolidate or log them, we could store (id, document_id, author, text, resolved_flag, timestamp). Google's API allows fetching comments so we could sync them here if needed for audit [36] [37] .
- **AuditLogs**: (id, user_id, action, document_id, details, timestamp) – Logs key events (document created, state changed, version saved, exported) for traceability.
- **SearchIndex** (optional): For advanced search, we might maintain a table of indexed text or vectors for documents. However, we can utilize Postgres full-text search on the DocVersions content if we store text, or use an external service.

The backend ensures **transactional integrity** between Google, DocuShare, and our DB. For example, when creating a document, if any step fails, it rolls back (deletes any partial Google Doc or DocuShare entry created). It also carefully handles refresh of OAuth tokens (Google tokens expire, so we refresh using the refresh_token when needed).

Roles & Access Control: Basic role-based permissions will be enforced: - Authors can create/edit documents but may not finalize them. - Reviewers can only view/comment on documents in Review, and approve/reject changes. - Approvers (could be a subset of reviewers or a separate role) can mark documents Final. - Admins can manage templates and user access.

These roles and workflow tie-ins are configurable in code or via a config table (storing which role can transition which state, etc.). We leverage DocuShare's access control as well: for instance, when a doc is in Draft, only its author (and certain others) have edit access; when in Review, it might grant read/comment to a reviewers group. DocuShare's permissions and our app's logic together ensure only the right people can see or edit content at each stage [28] .

Google Docs Integration for Collaborative Authoring

At the heart of the authoring experience is **Google Docs embedded** into our application, providing a familiar WYSIWYG editor with real-time collaboration. The integration works as follows:

- **OAuth2 Authentication**: Users authenticate with their Google account and grant our app access to Google Drive/Docs scopes [38] . Once authorized, the backend obtains an OAuth token to act on behalf of the user (stored securely). This token is used to embed and manipulate documents.

- **Embedding the Editor**: We leverage the Google Docs *embedded mode*. After creating or locating the Google Doc for a given section, the frontend loads an iframe with the Doc's edit URL (for example, `https://docs.google.com/document/d/[docId]/edit?usp=sharing`). Because the user is authenticated, they can edit directly in our app. This approach effectively gives a **Google Docs style** collaborative environment inside our platform [39]. Multiple users can open the same doc page and Google will sync their edits in real-time (with cursors, comments, etc.), achieving true concurrent co-authoring.
- **Features via Google UI**: Using Google Docs means we inherit a rich set of editing features without reinventing them – rich text formatting, change tracking (Suggestion Mode), comments, images, tables, etc. We can instruct users to use **Suggestion Mode** in Google Docs when they want to do "redlining" (track changes) [36]. This way, all edits made during a review are captured as suggestions that can be accepted or rejected, very much like MS Word track changes. **Comments** can be added in the Google Doc interface for specific feedback. These collaborative features are stored in Google's cloud automatically.
- **Capturing Google Doc events**: Our system can optionally use Google Drive API to listen for changes or fetch document content. For instance, when a user indicates they are done editing (or on a timed interval), we might use the Drive API to save a snapshot (export as DOCX) to DocuShare for backup. We can also fetch comments via the Google Docs API if we want to display them in our app's UI or include them in an audit trail [36]. In practice, because Google autosaves, we might not need to constantly save to DocuShare until a state transition (to avoid excessive versioning).
- **Limitations**: Google Docs in an iframe has some limitations (certain dialogs or add-ons might not work). We assume for this use-case (mostly text editing) it's sufficient. In cases where live embed might not fully work (e.g., SSO restrictions), we could fallback to opening the doc in a new tab. But the ideal is in-app editing.
- **Benefits**: By using Google's editor, we offload real-time OT (Operational Transform) complexity. Users get a familiar interface and powerful collaboration out-of-the-box [40]. This also means features like **offline editing** (if the user loses connection, Google can buffer) and **multi-language support** are available inherently.
- **Export via Google**: When needed, we use Google Drive API to export the document to various formats. This ensures high fidelity conversion to PDF or Word. For example, when compiling the eCTD, the backend calls Drive's export endpoint to get a PDF copy of each Google Doc [6]. Google's conversion typically preserves formatting and creates text-searchable PDFs that meet PDF standards (we will verify they are PDF/A if required) [41].
- **Doc Locking**: During workflow transitions (see Lifecycle below), our app can control Google Doc permissions. For instance, when a document moves to **Locked**, we can use the Google Drive API to switch the doc to view-only for everyone (or simply not allow further editing via our UI) [42]. Conversely, when a new Draft is created, the appropriate editors get edit rights.

Overall, the Google Docs integration is a **cornerstone for collaboration**, giving users a familiar environment with the convenience of our regulatory context. It lets authors focus on content while the system later takes care of conversion and compliance.

DocuShare DMS Integration and Version Control

Xerox DocuShare serves as the backend **document vault** to meet regulatory requirements for document management: controlled access, version history, and audit logs. Integration with DocuShare ensures that

every significant document iteration is captured and archived outside of Google's consumer environment, under enterprise control. Key integration points:

- **DocuShare API Usage**: Upon document creation, the backend connects to DocuShare (using provided credentials or a service account) to create a placeholder record for the document [31]. DocuShare might return a unique ID or handle (e.g., *DOC-000123*). This ID is stored in our database record for that document [5].
- **Storage of Versions**: When a document reaches certain milestones (e.g. a user manually saves, or transitions to Review/Final), the content is pushed to DocuShare:
- For example, when a Draft is finalized, the system will export the Google Doc to PDF and/or DOCX and upload that file to the corresponding DocuShare record as a new **version** [35]. DocuShare will then manage that as Version 1, 2, etc., keeping prior versions intact. We also update our DocVersions table to log this event.
- DocuShare's versioning ensures a tamper-evident history – we can always retrieve an older version if needed, and each save is timestamped with the user.
- **Audit Trail**: DocuShare automatically logs events like document created, modified, viewed, etc., satisfying compliance requirements. Our app supplements this by logging actions (like who triggered a state change) in our own AuditLogs, possibly referencing the DocuShare log ID for cross-reference [35]. This dual logging means we can produce a full audit trail report if needed (who edited what, when it was approved, etc.).
- **Access Control via DocuShare**: DocuShare can serve as the source of truth for permissions on documents. For instance, when we create a doc in DocuShare, we can assign it to a workspace/folder for the project and set roles (viewers, editors). Our application's roles will mirror these – e.g., an "Author" might correspond to edit rights in DocuShare, a "Reviewer" to read/comment rights. Whenever a state changes, we could adjust DocuShare permissions (like when moving to Locked, ensure no one except admin can edit the DocuShare record).
- **DocuShare as Archive**: After submission, all Final documents (and the compiled submission package) can be stored in a DocuShare archive folder for long-term retention. This ensures that even if Google Docs links expire or the project closes, the company has the full official content stored in their repository.
- **DocuShare API Details**: We will use DocuShare's REST API or an SDK like PyDocuShare if available [7]. For uploading files, the API likely requires a binary upload to the specific document's version endpoint. For retrieving files (e.g., if we need to include an already PDF attachment), we can fetch by DocuShare ID.
- **Fallback if DocuShare is down**: We might queue version uploads so that if DocuShare is temporarily unavailable, we don't block the user. But assuming high availability, this shouldn't be a major issue for MVP.
- **Supabase as Mini-Vault**: In early phases, we could also store an extra copy of files in Supabase Storage or Replit's filesystem for quick retrieval, but DocuShare is the master. Supabase metadata (DocVersions table) always knows how to get the file from DocuShare via its ID.

By integrating DocuShare, we **bridge the gap between authoring and regulated storage**. Authors work in Google Docs for convenience, but every finalized output is captured in the Vault for compliance. This dual approach draws on best practices (similar to how Veeva Vault or other DMS are used behind editing tools [43]) to ensure nothing falls through the cracks in version control or auditing. DocuShare's robust DMS capabilities (permission management, audit logs, retention policies) complement our system's functionality.

Document Lifecycle Management (Draft → Review → Final → Locked)

Managing the **document lifecycle** is critical in regulatory writing to ensure quality control and compliance. We implement a Veeva Vault-style state model [44] that governs each document's status and allowed actions. The lifecycle stages and behavior:

- **Draft**: When a document is first created, it's in Draft by default [33]. In Draft, authors can freely edit the content in Google Docs. The document is not yet considered complete. Multiple draft iterations can happen with AI assistance and internal edits. Other team members may or may not see the Draft (often only the author or co-authors have access at this stage).
- **Submit for Review**: When an author has completed a draft, they initiate a review. This triggers a transition Draft → In Review. Our backend `POST /documents/:id/state` will verify the current state is Draft and the user has rights, then:
- It may **lock the Google Doc for editing** by changing permissions to comment-only for others [42], so that no further direct edits occur, only suggestions.
- It could notify assigned reviewers (e.g. via email or an in-app notification) that the document is ready for review.
- We log this transition in the audit trail (who sent it for review and when).
- The document's status in the DB becomes "In Review".
- **In Review**: In this state, the content is being reviewed by independent reviewers or QA:
- Reviewers open the document (embedded Google Doc) but instead of editing directly, they use **Suggestion Mode** or comments to provide feedback [36]. Google Docs allows multiple reviewers to simultaneously add comments or suggested edits. This is effectively the "redlining" stage – all proposed changes are tracked.
- Our system can fetch comments via the Google API to display them on a review dashboard or just rely on Google's interface. We ensure that suggestions are not auto-applied until approval.
- The review cycle may involve discussions in comments. Reviewers might mark some comments "Resolved" when done.
- The system doesn't automatically version every comment; it waits until conclusion of review.
- If needed, reviewers without Google access could be given a PDF to mark up offline, but primary plan is to keep everything in Google for single-source.
- **Approve/Finalize**: After reviewers are satisfied, an authorized person (or consensus of reviewers) will move the document to Final. This triggers Review → Final transition:
- Typically, the author or a lead will go through the Google Doc, accept or reject all suggestions (resolving the redlines). Once all changes are resolved and the document is clean, they hit "Approve Final".
- Our backend receives the state change request. It can double-check that no unresolved suggestions remain (optional step for completeness).
- It then changes status to **Final**. At this point, content is considered approved. A final version is saved to DocuShare: the backend exports the Google Doc to PDF and uploads it as the Final version in the DocuShare record [35] (if this wasn't done at review start).
- The Google Doc could be switched to view-only mode (or left editable but since it's final, we prefer lock it). We might keep it editable in case minor last-minute edits are needed, but generally Final means freeze content.

- This stage likely corresponds to a document being "Approved" in regulatory terms (no more content changes).
- **Locked**: As an extra step, once a document is finalized and perhaps after it's been used in a submission, we mark it Locked. This is akin to an archive or "Approved/Effective" state [45] [46]. In Locked state:
- The document is strictly read-only to all end-users in our app. In Google Docs, we ensure it's view-only (to prevent any sneaky edits) [47].
- It indicates the document is complete and *in use* (e.g. submitted to agency). Only an admin could unlock it if absolutely necessary (perhaps to create an erratum or a new version).
- We log the lock event. No further versioning occurs unless unlocked.
- **Lifecycle Enforcement**: The allowed transitions are configured as a simple state machine: Draft → Review → Final → Locked (linear progression). Reverse transitions are generally not allowed except via admin override (e.g., unlocking a Final back to Draft for a revision would create a **new document version** or sequence). Each transition can have conditions:
- Draft to Review: allowed if document has content and user is author.
- Review to Final: allowed if a reviewer/approver user initiates it.
- Final to Locked: perhaps automatic when included in an eCTD export, or manual by an admin once submission sent.
- We implement these checks in the `/documents/:id/state` endpoint (server-side) to prevent illegal transitions [48]. We also reflect allowed actions in the UI (e.g., a Draft document shows a "Submit for Review" button, but a Final document does not).
- **Versioning on State Changes**: Moving to Final and Locked triggers version captures. For example, at Final, we might capture the "clean" Word version too, and at Locked, ensure a PDF is saved if not already. We record the DocuShare version ID in our DB for traceability [35].
- **Collaboration in Each Stage**: During Draft, collaboration is free-form (Google allows even multiple co-authors to type together). In Review, collaboration happens via comments/suggestions among reviewers. We ensure all stakeholders can access the doc in at least comment mode in Review (perhaps adding reviewer emails as commenters in Google as part of transition).
- **Audit and Notifications**: Each stage change could send notifications. E.g., when moved to Review, an email to the review team: "Document X is ready for review". When Final, notify the team or a regulatory affairs lead that "Document X finalized". Our system can integrate with email (via a service or simply rely on mentions in Google which send emails to commenters).

This lifecycle model mirrors common regulatory workflows [49] [33]. It ensures that content quality is checked at **In Review** and only approved content goes to submission. By leveraging Google's suggestion mode for redlining, we avoid implementing a separate track-changes engine and use a familiar review interface. At the same time, our backend keeps record of these state transitions and ensures the **Vault (DocuShare) has the final approved copy** at the end of the process [35]. This gives confidence that the document submitted to authorities is exactly what was reviewed and approved internally.

AI Co-Authoring with OpenAI GPT-4

One of the defining features of this platform is the integration of **GPT-4 AI assistance** to act as a "co-author" for regulatory writers. This AI layer enhances productivity and compliance by offering real-time

suggestions, content generation, and automated checks. We incorporate AI thoughtfully so that the human author remains in control. Key aspects of the AI co-authoring design:

- **OpenAI API Integration**: The backend includes an AI module that communicates with OpenAI's GPT-4 API [34]. We securely store the API key in environment variables on Replit [50]. The AI module exposes endpoints like `/ai/suggest` and `/ai/validate` as described. We may also use streaming responses for better UX (e.g., using OpenAI's streaming API to show the suggestion typing out), though for simplicity we can start with standard calls.
- **Real-time Suggestions**: As the user writes, the system can proactively or on-demand send the current text context to GPT-4 to get suggestions. For example:
- The user pauses after writing a sentence; the AI can suggest the next sentence or a way to complete the paragraph. This might be triggered by a button ("Continue with AI") or even automatically every few minutes. The suggestion appears in the AI Assistant sidebar, and the user can click to insert it if they like [12].
- If the user highlights a sentence or paragraph and asks for rephrasing, we call the AI to provide alternatives (like a built-in editor's rewrite function, but using GPT-4's language prowess).
- **Section Drafting**: For an empty section, the user can use a "**Draft with AI**" feature [51]. For instance, in Module 2.7 (Clinical Summary) introduction, the user can provide some bullet points or simply the section title, and ask GPT-4 to generate a first draft. The prompt might include instructions plus any relevant context (e.g., if Module 5 study reports are available, we could feed summaries of those to GPT). The AI returns a draft which the user can then edit. This jumpstarts writing, though the author will refine the output.
- **Compliance Checks (AI Validation)**: The AI can act as a smart reviewer. The user can click "Validate with AI" to have GPT-4 review the text against known regulations:
- We craft a prompt that instructs GPT-4 to behave like a regulatory QA checker, and list, for example, *"Check if this document is missing any required sections or if any phrasing might be non-compliant with FDA guidance"*. The AI would then produce a list of potential issues or omissions [52] [53].
- These findings are shown to the user (e.g., *"GPT suggests that you mention the clinical trial identifier in the Methods."*). The user can then address these before moving on.
- This AI check supplements rule-based checks (like our validation checklist), possibly catching more subtle issues (tone, clarity, consistency with previous filings).
- **Prompt Engineering Guidelines**: We maintain a library of carefully designed prompts to steer GPT-4 appropriately [8]. The AI is only as good as the prompt, especially in specialized domains:
- **Role prompt**: We often start prompts with a role definition: *"You are an expert regulatory writer with deep knowledge of ICH eCTD guidelines."* This sets the context for GPT-4 to respond in the right tone and with the right focus [54].
- **Module-specific prompts**: Because each eCTD module has a different purpose, we tailor prompts for each. For example:
    - Module 2 (overview/summaries): *"Draft a concise summary of [topic] that captures key points from Module 3/4/5."* The AI should focus on summarizing in a regulatory appropriate tone.
    - Module 3 (quality): *"Explain the manufacturing process for [product] including [specific details], in a clear, factual manner suitable for Module 3."* – Here the AI would be prompted to be technical and precise.
    - Module 4 (nonclinical) vs Module 5 (clinical) might emphasize different data (preclinical studies vs clinical trials).

- These prompt templates are stored in a configuration (possibly in the database) so they can be updated as we refine them, without code changes [55]. We might even have different prompt variants like "Suggest improvement", "Draft text", "Check compliance" as categories.
- We incorporate known regulatory guidance in the prompts. For instance, *"According to ICH M4 granularity guidelines, ensure the text covers X, Y, Z..."* so that GPT-4 gets some hints of what to check [54]. We can supply it bullet points of requirements for a given document type.
- **AI Model Tuning**: We won't train a custom model initially (using GPT-4 via API), but we will test and refine prompts with real examples (some prompt tuning). If needed, we could use few-shot examples in prompts. For instance, including a short example of a well-written excerpt as part of the prompt to guide GPT's style. The roadmap suggests we adjust prompts to act like a QA reviewer for compliance checks [56].
- **Integration in UI**: The AI Assistant Sidebar is the primary UI. It might have tabs for "Suggest" and "Validate" or different modes:
- **Compose** tab: where the user can ask the AI to write something. e.g., "Write the introduction for this section" or just hit a button to auto-compose next paragraph.
- **Review** tab: where the AI lists issues or improvements. This could be triggered after the user stops editing for a moment – the AI lists "3 suggestions to improve clarity" or such.
- The suggestions are presented in a friendly way, and the user can click to accept (insert text) or ignore. We ensure that AI output does not directly overwrite anything without user action, preserving human oversight.
- **AI for Templates**: The AI can also help when using templates. For example, if a template has a placeholder like [Insert study summary], the user could highlight that and ask AI to fill it in (given some context from Module 5). This accelerates filling out templates with actual content.
- **Generative AI Compliance**: We will put guardrails to prevent misuse of AI. For instance, we might limit the AI to certain tasks to avoid it hallucinating regulatory advice. All AI outputs are treated as suggestions that need human validation. This is in line with industry best practices where AI is a helper, not an autonomous author for final content [25].
- **Inspiration from Certara CoAuthor**: Certara's CoAuthor tool uses a specialized GPT model with an eCTD template library to streamline writing [57] [58]. Our approach is similar: by combining a template library with GPT-driven drafting, we ensure the AI works within a familiar structure to produce context-appropriate content. Certara reports significant reductions in drafting time (e.g. 20%+ timeline reduction) by using such technology [59] [60]. We aim for similar efficiency gains while maintaining accuracy.

In summary, the GPT-4 integration provides a **second pair of eyes and a first-pass drafter** at the same time. Writers can lean on it for tedious parts (formatting references, ensuring all sections are present) and for creative assistance (phrasing, summaries), but always with the ability to review and edit the AI's contributions. This co-authoring approach can significantly accelerate document preparation and improve consistency across documents [61] [62], marking a modern shift in regulatory writing practices.

Pre-Approved Templates for Multi-Region eCTD

A robust **Template Library** is essential for consistency and speed. The system provides a comprehensive set of eCTD document templates, covering common documents for FDA, EMA, PMDA, and other regions. These

templates encapsulate best practices and ensure that from the outset, documents are structured correctly. Key points about template management:

- **Coverage of Templates**: We include templates for:
- **Module 1 (Regional)**: e.g., FDA-specific forms and docs – Form FDA 1571 (IND Application) and 356h (NDA cover sheet), FDA cover letter, FDA financial disclosure form; EMA cover letter, EMA application form; PMDA notification forms, etc. Because Module 1 is *region-specific*, each region gets its own set of templates for administrative documents [63].
- **Module 2 (Overviews & Summaries)**: Clinical Overview (M2.5), Nonclinical Overview (M2.4), Clinical Summaries (M2.7), Quality Overall Summary (M2.3), etc. These templates follow ICH guidance in structure. They often have pre-filled section headings and perhaps recommended tables or summaries to include.
- **Module 3 (Quality)**: Templates for drug substance and product sections, e.g., "3.2.S.2.2 Description of Manufacturing Process and Controls" might have a template prompting for description of each step, controls, etc. Quality templates emphasize tabular formats for specs, and placeholders for data.
- **Module 4 (Nonclinical Study Reports)**: While actual study reports are usually imported, we may provide a template for the study report format (though these are often external PDFs). More useful might be templates for Module 4 summaries or bridging documents.
- **Module 5 (Clinical Study Reports & Lists)**: Similar to Module 4, many are external docs (protocols, CSRs). But we include templates for things like an integrated summary of safety/efficacy (if not in Module 2) or other region-specific addenda.

- Also, templates for common **cover letters**, statements, and **project-level documents** (like a Table of Contents, though eCTD's ToC is auto-generated via XML).

- **Template Format**: Each template could be stored as a **Google Doc** that serves as a master copy. This way, when a user inserts a template, we can simply copy its content into the user's working Google Doc. Using Google Docs for templates ensures that formatting (styles, margins) is preserved exactly, and they can include elements like tables, headers/footers.

- **Ensuring Compliance and Quality**: The templates are designed to be "right-first-time" (RFT) – meaning if an author follows the template, the document will inherently meet many eCTD requirements [64] [65]. For example:
- Templates include correct section numbering and headings per ICH.
- They have pre-set styles for headings, normal text, tables that adhere to a predefined style guide (font sizes, margins, etc.), ensuring consistency.
- **Content controls and placeholders**: Some templates might use placeholder text (e.g., "[Insert Study Title]") or instructions in italics that guide the writer on what to fill in each section [66] [67]. In Word, one might use content control fields; in Google Doc, we simply highlight text that needs replacement.
- **Metadata fields**: We can auto-fill certain fields. For instance, when creating a new document from a template, if the system knows the Product Name, we could insert it wherever the template had a placeholder for product name. Similarly for applicant name, application number, etc. This is similar to Celegence's approach of separating content from context to enable reuse [66].

- **Built-in guidance**: Templates may contain hidden comments or tips (which can be removed later). For example, a template might include a comment like "Ensure to include rationale for dose selection as per FDA guidance X" to remind authors of expected content [68] .
- **Multi-Region Handling**: Because we target global submissions, the library supports **region switching**:
- The user can select a region context when starting a new document. E.g., "Create document for EMA region". The template library UI will then filter or automatically pick the EMA version of a template if Module 1, or the common template if Module 2–5.
- For common Modules (2–5), templates are largely the same across regions (thanks to ICH). However, if there are slight differences (perhaps EMA might prefer a different order in a section), we might have slight regional variants or notes in the template.
- The system could allow a document to be marked as "common" or "US-specific", etc. If a company is preparing a global dossier, they might write one Module 2 that is used for US, EU, and Japan. Or they might have separate ones if needed. Our data model can link one logical document to multiple regions if it's identical, or have separate docs if diverged.
- **Auto-Sync Content**: If a Module 2 summary is intended to be common for US and EU, we can avoid duplication by having one document and tagging it for both regions. At export, it gets included in both the US and EU package. If they must diverge (due to specific content for EMA vs FDA), the user can "branch" the document. We'll support either approach but encourage reusing content when possible to maintain consistency. (The phrase "document-to-module mapping with auto-sync" in the requirements suggests that if one section is reused in multiple places, the system should manage that. We handle it by single-sourcing those sections and referencing them in each region's submission, rather than having two copies that might drift.)
- **Template Insertion and Use**: From the editor, when the user chooses a template (say "Clinical Overview"), the content of that template (with all its headings and boilerplate) is inserted into the Google Doc. The user then replaces and expands on each part. The **AI assistant** can also help fill in parts of a template if asked, making the combination powerful (template gives structure, AI fills content).
- **Template Updates**: An admin can update templates in the library (for example, if guidelines change or to improve them). This doesn't retroactively change content already written, but new docs will use the updated template. We maintain versioning for templates too (maybe simple version numbers or last updated date).
- **Inspiration from Dosscriber (Celegence)**: Celegence's Dosscriber templates emphasize consistency and reusability across regions, with features like auto-generated TOCs, embedded styles for easy PDF publishing, and options for granularity [69] [70] . We mirror these ideas:
- Our templates ensure that if followed, the resulting documents will have proper bookmarks and heading styles so that when converted to PDF, a Table of Contents and bookmarks can be generated automatically [70] . (If using Google Docs, we ensure the heading styles map to PDF bookmarks).
- We allow flexibility in granularity: e.g., whether to write a single document covering multiple sections or separate documents. For instance, some companies split Module 2.7 into several documents by subsection. Our library can offer a combined template or individual ones, and our system will accommodate either by how it populates the eCTD XML (multiple leaves vs one leaf) [71] .
- **Example Templates**:
- *FDA Form 1571*: Provided as a fillable template (likely a PDF or an HTML form, since it's a specific form). We might integrate this by either having the user fill a web form that maps to the FDA 1571 PDF or simply instruct them to attach a filled PDF. (This is a special case since it's a standardized

form; for now, one could fill it outside and attach, but an advanced feature could be an interactive form in the app).
- *Module 2.5 Clinical Overview*: A template with all the ICH M4E subsections (2.5.1 Product Development Rationale, 2.5.2 Clinical Pharmacology, etc.) each as placeholders to be completed. This template ensures the author doesn't forget any section. The AI could help populate a high-level summary of each part.
- *Module 3 Quality Overall Summary*: Could include tables to summarize drug substance and product info in a concise way.
- *Cover Letters*: A template that pulls from a project profile (project name, application type, contact info) and standard text.
- These examples illustrate how the system guides authors to start with compliance in mind. As noted, using templates and metadata is known to create consistent, compliant documents with less effort [72].

By providing a rich template library, we **reduce blank-page syndrome** for writers and ensure alignment to regulatory formats from the get-go. Templates combined with the AI assistant (for content suggestions) dramatically expedite document creation while maintaining high quality and consistency across the entire submission dossier.

Automated eCTD Compilation and Export Pipeline

A core goal of the system is **automating the eCTD publishing process**. Once authors have finalized documents, the system can compile them into an eCTD-compliant structure with minimal user effort. The export pipeline handles conversion to PDF, XML backbone generation, and packaging, yielding a submission-ready .zip file. The steps of this pipeline are:

1. **Select Export Options** – As described in the UI section, the user initiates an export, choosing the region and set of modules/documents to include (often all completed docs for that submission sequence). They also choose output format (typically "eCTD zip"), though we might allow "just PDFs" for an internal review packet [73].
2. **Gather Documents** – The backend queries the database (or DocuShare) for the final versions of all documents in scope [74] [73]. Only documents with status Final (or Locked) are considered to ensure we use approved content. We retrieve the Google Doc IDs or DocuShare IDs for each.
3. **Convert to PDF** – For each document, if it's not already a PDF:
4. We use the Google Drive API to export Google Docs to PDF [6]. This ensures proper formatting. (Google's export will handle fonts, images, etc., producing a standard PDF. We will verify these PDFs meet agency requirements like PDF version 1.7, text searchable, no active JavaScript, etc. If needed, we can post-process or use a library to enforce PDF/A compliance [75]).
5. If a document is an attachment (say a literature reference or already a PDF in DocuShare), we retrieve it as-is. We might run checks (like ensure no encryption).
6. For content like Forms (e.g., FDA Form 1571), if those were filled outside and uploaded to DocuShare, they might already be PDF. We just gather them.
7. We temporarily store all these PDFs in a working directory on the server (e.g., in `/mnt/data/export/[submissionID]/` since Replit allows persistent storage during the session).
8. **Generate eCTD XML** – This is the most crucial step. We create the XML backbone files that constitute the table of contents of the eCTD:

9. **index.xml** – the master XML for the submission. It lists all files (as `<leaf>` elements within the appropriate module sections). We have a pre-defined template or use a small library to construct it. For each document, we know its module and we know the output PDF filename, so we insert an entry accordingly [76] .
    - For example, if we have a document "Clinical Overview" which is Module 2.5, we will place a `<leaf>` under `<m2>` section in index.xml with `title="Clinical Overview"` and `href="m2/25-clin-over.pdf"` (and other attributes like ID, checksum if needed).
    - We maintain the CTD hierarchy: Module 1 (if present) entries go under their regional section (like `<fda:applications>` for FDA's us-regional, etc.), Modules 2 to 5 under common sections.
10. **us-regional.xml / eu-regional.xml** – region-specific XML for Module 1. For FDA, a `us-regional.xml` is required containing Module 1 (administrative) contents. We generate this if the region is FDA [77] . Similarly for EU, an eu-regional.xml might be needed (depending on spec version).
11. We can use an XML generation library in Node (like xmlbuilder or fast-xml) to build these files to ensure well-formedness [78] . Or we maintain template files where we inject file references.
12. The system knows the required modules from a configuration (e.g., an array of expected documents per module). We use this to double-check that every required document has a leaf. If something is missing, we can warn (but still generate what's available).
13. Each `<leaf>` entry includes sequence number and operation if needed. For initial submissions, sequence is 0000, operation is "new". We assume initial for MVP. (Future: support later sequences with replace/delete ops).
14. **Package Assembly** – Once all PDFs and XML files are ready in the temp folder, we organize them into the standard eCTD directory structure:
15. Typically: `\modules\` folder with subfolders `1`, `2`, `3`, `4`, `5` (and within those, further subfolders for each subsection). We place each PDF in the appropriate folder (e.g., `module2/` for summary docs, etc.). The exact structure is dictated by the eCTD spec (and the paths we used in index.xml must match).
16. The XML files (index.xml, us-regional.xml) go in the root or a specific folder (for FDA, index.xml at root of sequence, us-regional.xml in `/module1/us/` for example).
17. We ensure naming conventions are followed (often files use short names like "m2-5-clinical-overview.pdf" or specific codes). We can derive these from titles or use a lookup table.
18. Then we create a ZIP archive of the entire folder structure.
19. **Validation Checks** – Before finalizing, we may run an automated validation (if we have a utility or just a checklist) to ensure the package meets specifications:
20. Check that all required files are present.
21. The XML is well-formed and all referenced files exist.
22. All PDFs are readable and text-searchable [22] .
23. If any issues, we log or report them.
24. (Professional tools like Lorenz Validator or FDA's eCTD validator are outside scope, but we aim to catch obvious issues.)
25. **Delivery** – Provide the output to the user:
26. The user can download the ZIP directly through the browser (our API streams the file). Since the file is on the Replit filesystem, we stream it and then (optionally) delete the temp files.
27. We also have the option to automatically upload the zip back into DocuShare for record-keeping (maybe under a "Submissions" folder, with sequence number and timestamp) [79] .
28. We record in our DB that a submission package was generated, who did it, when, and for which region/sequence.

This entire flow, when initiated, is largely automated. It **mimics what a regulatory publisher would do manually**: collating final docs, converting to PDF, writing an XML table of contents, and zipping [80]. By automating it, we reduce the chance of human error (like misplacing a file or typo in XML) and save a lot of time [81].

Implementation Notes: - We will likely implement this in the backend as a series of functions or a queued job. For example, a function `generateEctdPackage(submissionId, region)` that does steps 2–5. - We might use a Node library for zipping (like JSZip or archiver). - For PDF conversion, as noted, using Google's API is simplest. If that fails or if we had non-Google files, we could integrate a conversion library (LibreOffice headless conversion for Word, etc.) but on Replit this might be heavy. Sticking to Google's export for docs and assuming any other files are already PDF is a fair approach for MVP [82] [83]. - The XML generation will be done according to the ICH eCTD v3.2.2 or applicable specification. We can reference sample XML from guidance to ensure correctness. - The output should be a **valid eCTD sequence** that can be loaded into an eCTD viewer or submitted to the agency without further modification [80]. We will test it on a small sample (maybe by opening with a viewer or using FDA's validator if available).

By integrating the export pipeline, **TrialSage** (our hypothetical system name per the docs) can go from authoring to publishing in one platform [84]. This one-click submission assembly is a huge efficiency gain: regulatory teams don't need to manually compile PDFs or outsource publishing; the system ensures that what was written and approved flows directly into the compliant format ready for agency review.

Collaboration and Review Workflows

Collaboration is ingrained throughout the system, leveraging both the Google Docs capabilities and additional workflow features to manage multiple contributors and reviewers:

- **Real-Time Co-Authoring**: Multiple authors can work on the same document simultaneously via the Google Docs embed. They see each other's changes in real time, eliminating edit conflicts. This is especially useful when a document is large and sections can be divided among writers. It's similar to multiple people in a Google Doc normally – our interface just facilitates it.
- **Commenting and Mentions**: Google Docs supports comments and @mentioning users. Reviewers can tag a specific person in a comment (if they have access) and that person will get notified via email (through Google). This can be used during Draft stage too for internal discussions (not formal review).
- **Suggestion Mode for Redlining**: We instruct users to use Suggestion Mode during peer review, as noted. All suggested edits (insertions/deletions) are clearly shown in the text with track changes, attributable to the reviewer (Google shows the name and timestamp) [36]. This satisfies the "redlining" requirement – functionally identical to Word's Track Changes but via Google. The advantage is suggestions can be accepted in one click and merge seamlessly.
- **Consolidated Feedback**: Because all reviewer comments and suggestions live in the document itself, the feedback is consolidated in one place (as opposed to having multiple separate markups to reconcile). The author/approver can go through and **accept/reject changes one by one or all at once** on the Google Doc. This addresses the "consolidated review" requirement – everyone is literally looking at and commenting on the same single document, not siloed copies.
- **Review Assignments**: The system could have a concept of assigned reviewers for each document. When moving to In Review state, we might present a dialog "Choose reviewers" – the selected users

get permissions to comment on the Google Doc (if they didn't have it) and are notified. The document might appear in a "Review Queue" for those users on their dashboard.

- **Review Reminders and Due Dates**: We could allow setting a due date for reviews. The system can send reminder emails or notifications if the date is approaching and reviews are pending. This ensures timely feedback cycles.
- **Cross-Document Linking**: In regulatory docs, it's common to refer to content in another module (e.g., "see Section 4.2.3 for details"). With our system, since all docs are in a structured repository, we could facilitate linking. Perhaps in a comment, an author can paste a link to another document (Google Doc link). In the future, we could implement smart linking that works in the compiled PDF (this is advanced: e.g., a link that at export time resolves to a hyperlink in the PDF pointing to the correct file in the eCTD). For now, we note cross-references and ensure the user knows to include them. (This might be out of scope for MVP, but cross-module linking was mentioned as a desired feature [85]).
- **Document-to-Module Mapping Auto-Sync**: If one piece of content is duplicated in multiple places (for instance, a summary in Module 2 references data in Module 5, and maybe the user wants any changes in Module 5 to reflect in Module 2), we can assist by:
- Using the search feature or an AI diff to detect inconsistencies: e.g., after finalizing, run a check that important values or statements are consistent between related docs (like a number of patients in a study mentioned in Module 2 vs Module 5).
- Or maintain single-source for common content. Some advanced systems let you reuse text via content blocks. We could allow storing a snippet (like a risk statement) in a central library and then embedding it in multiple docs. If updated centrally, it updates everywhere. This "auto-sync" of content could be a future enhancement. For now, we at least help detect divergence.
- **Roles in Collaboration**: Authors are primary editors, reviewers add suggestions, approvers finalize. We clearly delineate these in the UI and by Google permissions. This prevents, say, a reviewer from accidentally fully editing a draft (they would use suggest mode only).
- **Multi-Region Collaboration**: If working on a global submission, teams from different regions might collaborate. Our region filter ensures, for example, EU colleagues focus on Module 1 EU docs, US colleagues on Module 1 US, but they can all collaborate on Modules 2–5 together. The Module Progress dashboard will give a unified view so everyone knows what's done or pending.
- **Activity Logs and Notifications**: Every major action (edit, comment, state change) can generate an entry in an activity feed. We might surface a "Recent Activity" on the dashboard – e.g., "Jane Doe commented on Clinical Overview 1 hour ago." This keeps team members informed. Also, email notifications (or just rely on Google's for comments).
- **Conflict Resolution**: Thanks to Google's engine, conflicts in editing are minimized. However, if two users did end up editing offline copies (not likely here), merging would be an issue. We avoid that by having one source.
- **Connectivity and Offline**: If a user is offline, they might edit the Google Doc offline (Google has that ability) and it syncs later. Our app doesn't directly control that but benefits from Google's capabilities.
- **Training the Team**: We assume users are trained to use the system, but we keep the UI intuitive. They essentially just need to know how to use Google Docs and our few extra buttons for lifecycle and AI.

By designing the workflow like this, we ensure the platform supports **collaborative authoring and review akin to established processes but in one integrated environment**. Authors, reviewers, and approvers all work on the same live documents, drastically cutting down on email back-and-forth and version confusion. The end result is a more efficient team workflow and higher-quality documents (since issues are caught and resolved in-context and early).

AI-Powered Search and Knowledge Management

Writing regulatory documents benefits greatly from leveraging past knowledge – previous submissions, guidelines, etc. The system includes an AI-powered search module to enable users to quickly find relevant information from past documents and reference materials:

- **Indexed Repository**: All final approved documents saved in the DocuShare vault can be indexed for search. We will extract text from PDFs (if not already stored as text) and store it in a search index. Supabase/Postgres can be used for basic full-text search on documents' text content [16]. We might create a table `DocumentText` (doc_id, text) for this purpose. Alternatively, for more advanced search, we can generate embeddings (with OpenAI embeddings API or similar) for paragraphs and use a vector search to enable semantic queries [86].
- **Search Interface**: Described in the UI section, the search bar accepts free text or structured queries. We support:
- **Keyword search**: e.g., "nonclinical pharmacokinetics". The backend will do a full-text search to find documents or sections that mention those terms.
- **Semantic search/Q&A**: e.g., "What is the standard dose of Drug X used in previous trials?" For this, we can use a combination of vector search to get candidate text and then maybe GPT-4 to formulate an answer [18] [19].
- **Scope of Search**: By default, it searches the organization's prior submissions (all projects). We could allow filters by product or by module. For example, filter to only Module 3 documents if the user only wants quality info. Metadata like tags (module tags, document type) can be used for faceted search [87] [88].
- **Inclusion of External Guidance**: We can preload certain public guidance documents or internal SOPs into the search index as well. For example, ICH M4 guidelines, FDA eCTD guidance, or internal writing style guides. This way, when a user searches, they might also find relevant paragraphs from these references (marked as such). This could be very helpful if a user asks "What does ICH say about Module 2.5 content?" – the search could return the section from ICH guidance.
- **AI Assistance in Search**: We integrate GPT-4 in the search workflow via a *retrieval augmented generation* approach:
- When the user asks a question in search, the system will find the top relevant documents/snippets using the search index [16].
- It then composes a prompt for GPT-4: e.g., "Using the following excerpts from prior documents, answer the user's query: … [insert excerpts] … Question: …". GPT-4 can then provide a synthesized answer or summary, citing which past submission it drew from. This is akin to how some knowledge-base assistants work, and can save time for the user in gleaning insight from multiple docs.
- The answer is shown, with an option to view the source documents if needed (to verify context).
- **Security in Search**: We enforce that users can only search content they are allowed to see [20]. If a company has multiple projects with access controls, a user not on Project B shouldn't see Project B's docs in results. We use document permissions from DocuShare or our DB to filter search results.
- **Continuous Learning**: Over time, as more documents and Q&A are done, we might refine the search relevance. We could allow users to tag certain paragraphs as important or add manual keywords to documents to improve findability.
- **Tagging**: The system can auto-tag documents with key topics using AI (e.g., label a doc as "Clinical Pharmacology" or "Oncology" based on content) [89]. These tags improve search filtering. For example, one could filter search results to only those tagged "immunogenicity" or "pediatric study".

We could generate tags by feeding the doc text to an NLP model that picks out keywords or uses a taxonomy.

- **User Files and Reuse**: When starting a new document, the user might search the repository for similar documents (e.g., previous Clinical Overview for another product) to use as reference. They could even copy parts from it if applicable (though careful with copying across products). Our platform fosters content re-use by making it easy to find and retrieve previous text that was already agency-approved.
- **Future AI Integration**: The search results could be integrated into the AI writing assistant – for example, when the user is drafting and asks the AI, "Summarize the outcomes of previous related studies", the system can search and then GPT-4 can summarize those specific bits and insert into the document. This is forward-looking, but the architecture we set up (with search and AI) allows such interplay.

With these search capabilities, our system becomes not just a writing tool, but a **knowledge hub** for regulatory documentation. It shortens the research phase of writing (writers often need to look up how things were phrased before or find data from prior studies). By having this at their fingertips, writers can maintain consistency with past submissions and ensure they don't miss learned lessons or preferred wording. This also helps new writers get up to speed by learning from the library of prior submissions.

Deployment on Replit and DevOps Considerations

The platform is intended to be fully runnable on Replit, which means we account for environment setup and deployment constraints:

- **Replit Environment**: We will use a single Replit container to host the Node.js backend and the React frontend. In development, we can run React in dev mode and Node concurrently, but for deployment on Replit, we can build the React app and serve it as static files from Express (e.g., using Express to serve the `build/` directory). This simplifies deployment to one process.
- **Secrets Management**: All sensitive keys and credentials (Google OAuth client ID & secret, Google API tokens, DocuShare API credentials, OpenAI API key, Supabase URL and service key) are stored using Replit's Secrets feature [56] [50]. They will not be in the code. The application reads them from environment variables at runtime.
- **Google OAuth Config**: We will set the OAuth callback URL to the Replit app URL (which typically is `https://<username>.<replname>.repl.co/callback`). We also must enable "Embedded" Google sign-in because the app runs in an iframe on Replit's domain – we may need to configure OAuth consent screen accordingly.
- **Persistent Storage**: Replit provides a `/mnt/data` volume that persists across restarts. We will use this for any file storage needs:
- For caching template files or storing the compiled eCTD zip before download [90].
- For any temporary files (like Google Drive PDF exports) during export. We ensure to clean up large files after use to save space.
- We do not rely on writing to the container's root filesystem for persistence (since that resets on restart).
- **Supabase Setup**: Supabase is a cloud service; we will use a free project. Our Replit code will connect to Supabase via its URL and API key (provided as secret). Alternatively, if needed, we could run a local Postgres in Replit, but using Supabase is easier and aligns with the plan [91]. We must ensure the Replit container can reach the Supabase endpoint (it should, via internet).

- **DocuShare Connectivity**: The Replit container must be able to reach the DocuShare server. If DocuShare is on-prem and not accessible publicly, we'd need a workaround (likely DocuShare would have to be cloud-accessible or via VPN which is not trivial on Replit). We assume a cloud-accessible DocuShare API for this design.
- **Memory and Performance**: Replit free containers have limited RAM/CPU. We need to be mindful:
- PDF generation via Google is done over network, not heavy CPU on our side.
- The biggest load might be the OpenAI calls (which are external) and processing results, which is fine.
- Zipping a whole submission (which could be hundreds of MB if many files) might strain memory. We can stream zipping file-by-file to avoid high memory usage.
- We should also consider file size limits on Replit. If needed, we can integrate an external storage (like upload the zip to Supabase Storage or cloud storage and give a download link, instead of holding it in memory).
- **Websockets**: If we implement streaming or real-time notifications, Replit supports WebSocket connections on their hosted sites [92] . We can use that for, say, pushing AI suggestions or progress updates. If not, polling or simple HTTP updates can suffice.
- **Testing on Replit**: We will test the full flow (OAuth logins, editing, export) on the Replit deployed URL to ensure all third-party integrations work in that environment.
- **Deployment Steps**:
- Push code to Replit (or use Replit Git integration).
- Add all env vars in the Replit Secrets panel (for Google, DocuShare, OpenAI, Supabase).
- Start the Express server (which also serves the built React).
- Ensure the Replit "Always On" or similar is enabled if we want it running continuously (for persistent backend).
- Use Replit's web IDE to monitor logs. We might add a simple healthcheck endpoint to verify it's up.
- **Container configuration**: In the Replit `replit.nix` or config, include needed packages. E.g., if we need `libreoffice` (if we ever do local conversion) or other binaries, we'd have to add them. For now, we might not need additional system packages; Node libraries should suffice.
- **Scalability**: Replit is fine for development and demos. For production, one might deploy this on a more robust cloud or multiple instances. But within Replit, we design for the constraints: single container, limited resources, but enough for a moderate submission (which might be dozens of documents).
- **Logging and Debugging**: We will output logs to console (visible in Replit log) for key events (auth, errors, etc.). No sensitive data in logs. We can also use Supabase or an error tracking service if needed.
- **Backup**: Since Supabase holds data, ensure Supabase backups are on. Replit file data (if any critical) we might allow export. However, since DocuShare is the source of truth for docs, and Supabase for metadata, a Replit container reset isn't catastrophic – it can reconnect to those external sources and continue.

By following these deployment considerations, we ensure the application is **Replit-ready and easily portable**. A single developer or team can run this entire system in a Replit workspace, which is great for demonstrating the concept or doing integration testing with all components in one place. In a real-world scenario, the same Node and React code could be deployed to a traditional server or cloud VM with minimal changes once proven out.

Conclusion and Future Directions

The proposed intelligent eCTD Co-Author system brings together **collaborative editing, AI assistance, template-driven authoring, and automated publishing** into one platform. By embedding Google Docs for live editing and leveraging DocuShare for enterprise document management, users get the best of both worlds: a familiar, powerful writing interface with the assurance of compliance and version control in the backend. GPT-4 integration offers a leap in efficiency, acting as a smart assistant that can draft and review content in line with regulatory standards. The multi-region template library and lifecycle management workflows ensure that whether you're preparing an FDA IND or an EMA MAA, the system guides you through the correct process with the correct document structure every time.

This design is inspired by and improves upon existing solutions like Certara's AI-enabled CoAuthor (which demonstrated the value of generative AI in regulatory writing) [93] and Celegence's Dosscriber templates (which underscored the importance of reusability and compliance-by-design in authoring) [66] [94]. By integrating similar capabilities into an all-in-one Replit-deployable application, we aim to **democratize** this functionality – making an "enterprise-grade" regulatory co-authoring tool available for teams of any size [43] [95].

Moving forward, potential enhancements could include: support for eCTD v4 / RPS format as it evolves, integration with agency submission portals for direct sending, more advanced content reuse across documents (single-sourcing data like patient numbers or chemical structures), and training custom AI models on the company's own writing style for even more tailored suggestions. The foundation laid out in this implementation plan positions the platform to evolve with the regulatory landscape and continually ease the burden on medical writers and regulatory operations professionals.

In summary, this step-by-step plan provides a clear path to building the intelligent eCTD Co-Author system, detailing everything from OAuth integration to XML generation. Implementing it on Replit validates that the tech stack and integrations work end-to-end in a contained environment. The result will be a **state-of-the-art web application** that transforms how regulatory documents are authored and published – making the process faster, smarter, and more collaborative than ever before.

Figure: The "CTD Triangle" shows the structure of the Common Technical Document: Module 1 is region-specific, while Modules 2–5 are common for all regions [96]. This hierarchy guides our multi-region template design and ensures we manage content appropriately for each jurisdiction.

---

[1] [2] [4] [5] [6] [7] [8] [9] [10] [11] [12] [16] [17] [18] [19] [20] [21] [22] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [40] [41] [42] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [82] [83] [86] [87] [88] [89] [90] [91] [92] [96]

TrialSage eCTD Co-Author Platform Roadmap.pdf
file://file-Vq3DhWVZ4FsNvjYqgaTeiq

[3] [85]

TrialSage eCTD Co-Author Module_ Enterprise-Grade Upgrade Design.pdf
file://file-Uj921zCFZXC6F4joFBvHHG

13 14 15 **TrialSage eCTD Co-Author Module_ Enterprise-Grade Upgrade Design.pdf**
23 39 43 file://file-6aC1Fy8PFeQN1RZJZ1uWzG
72 73 74
75 76 77
78 79 80
81 84 95

24 25 59 **Certara Launches CoAuthor, a Regulatory and Medical Writing Tool | Certara**
60 61 62 https://www.certara.com/announcement/certara-launches-coauthor-a-regulatory-and-medical-writing-tool/
93

63 64 65 **Dosscriber™ eCTD Templates for Efficient Global Submissions**
66 67 68 https://www.celegence.com/dosscriber-ectd-document-templates/
69 70 71
94