

F25-314-D-CoWriteIA

Project Team

Aisha Siddiqa	22I-1281
Ayesha Ejaz	22I-0899
Junaid Asrar	22I-0770

Session 2022-2026

Supervised by

Dr. Ali Zeeshan



Department of Computer Science

**National University of Computer and Emerging Sciences
Islamabad, Pakistan**

June, 2026

Contents

1	Introduction	2
1.1	Existing Solutions	2
1.2	Problem Statement	3
1.3	Scope	3
1.4	Modules	4
1.4.1	Module 1: Project Indexing and Semantic Memory	4
1.4.2	Module 2: Context-Aware Writing Assistant	4
1.4.3	Module 3: Character and Scene Management	4
1.4.4	Module 4: Dialogue Generation	4
1.4.5	Module 5: Research Integration	4
1.4.6	Module 6: Project Query Interface	4
1.4.7	Module 7: Gap Analysis Module	4
1.5	Work Division	6
2	Project Requirements	8
2.1	Use-case	8
2.2	Functional Requirements	9
2.2.1	Module 1: Project Indexing and Semantic Memory	9
2.2.2	Module 2: Context-Aware Writing Assistant	9
2.2.3	Module 3: Character and Scene Management	9
2.2.4	Module 4: Dialogue Generation	10
2.2.5	Module 5: Research Integration Module	10
2.2.6	Module 6: Project Query Interface	10
2.2.7	Module 7: Gap Analysis Module	10
2.3	Non-Functional Requirements	10
2.3.1	Usability	10
2.3.2	Performance	11
2.3.3	Security	11
2.3.4	Reliability	11
2.3.5	Maintainability	11
2.3.6	Compatibility	11
2.3.7	Scalability	11

3	System Overview	12
3.1	Architectural Design	12
3.2	Data Design	14
3.2.1	Class Diagram	15
3.3	Domain Model	15
3.3.1	Relationships and Associations	17
3.3.2	Domain Model Diagram	18
3.4	Design Models	18
3.4.1	State Transition Diagram	18
4	Implementation and Testing	20
4.1	Algorithm Design	20
4.1.1	High-level Flow	21
4.1.2	Pseudocode: Document Chunking and Preprocessing	22
4.1.3	Embedding Generation and Storage	24
4.1.4	Pseudocode: Context-aware Retrieval (RAG)	25
4.1.5	Pseudocode: AI Response Generation (RAG + LLM Call)	26
4.2	External APIs and SDKs	28
4.3	Testing Details	28
4.3.1	Black Box Testing	28
4.3.1.1	Purpose of Black Box Testing	29
4.3.1.2	Testing Environment	29
4.3.1.3	Functional API Coverage	29
4.3.1.4	Workflow and Scenario Testing	29
4.3.1.5	Negative and Security Testing	29
4.3.1.6	Error and Edge Case Validation	29
4.3.1.7	Black Box Test Evidence	29
4.3.1.8	Black Box Testing Summary	31
4.3.2	Unit Testing	31
4.3.2.1	Implemented Unit Test Coverage	31
4.3.2.2	Testing Approach	32
4.3.2.3	Failure and Boundary Validation	32
4.3.2.4	Authentication and File Services Unit Tests	32
4.3.2.5	Search, Text Extraction, and Embedding Unit Tests	33
4.3.2.6	LLM, Editing, and Copilot Unit Tests	33
4.3.2.7	RAG Context, Entity Extraction, and Relationship Discovery Unit Tests	33
4.3.2.8	Updated Unit Test Summary	33
4.3.3	Test Evidence	35
4.4	Test Summary	36

4.4.1	Integration Testing	36
4.4.1.1	Summary	36
5	Conclusions and Future Work	37
5.1	Conclusion	37
5.2	Future Work	37
	References	38
A	Appendices	39
A.1	Appendix A: System Diagrams	39
A.1.1	Use Case Diagram	39
A.1.2	Activity Diagram	40
A.2	Appendix B: Architecture Diagram	41
A.2.1	CoWriteIA Detailed Architecture Diagram	41
A.3	Appendix C: User Interface Screenshots	42
A.3.1	Dashboard Interface	42
A.3.2	Writing Environment Interface	42
A.3.3	Writing Assistant Interface	43
A.3.4	Login Interface	43
A.4	Appendix D: Detailed Use Case Specification	45
A.4.1	UC-01: Generate Context-Aware Writing	45
A.5	Appendix E: Pseudo-API Snippets (Call Signatures)	46

List of Figures

1.1	Gantt Chart - Project Timeline	7
3.1	Box and Line Diagram	14
3.2	Entity–Relationship Diagram for CoWriteIA	15
3.3	Class Diagram for CoWriteIA	16
3.4	Domain Model Diagram for CoWriteIA	18
3.5	State Transition Diagram for CoWriteIA	19
4.1	BlackBox Tests fig:1	30
4.2	BlackBox Tests fig:2	30
4.3	Black Box Test 1	31
4.4	Black Box Test 2	31
4.5	Black Box Test 3	31
4.6	Unit Test Evidence 1	33
4.7	Unit Test Evidence 2	34
4.8	Unit Test Evidence 3	35
4.9	Unit Test Evidence 4	35
4.10	Test Execution Results	35
4.11	Integration API Test Evidence	36
A.1	Use Case Diagram CoWriteIA	39
A.2	Activity Diagram	40
A.3	CoWriteIA Multi-Tier Architecture Diagram	41
A.4	CoWriteIA Dashboard Interface	42
A.5	Writing Environment Interface	42
A.6	Writing Assistant Interface	43
A.7	Login Interface	43

List of Tables

1.1	Comparison of Existing Solutions	3
1.2	Work Division for Iteration I	6
1.3	Work Division for Iteration II	6
3.1	Architectural Tiers and Responsibilities	13
4.1	External APIs and SDKs Used	28
4.2	Unit Test Results — Authentication and File Services	32
4.3	Unit Test Results — Search, Text Extraction, and Embedding Services	33
4.4	Unit Test Results — LLM Integration, Edit Proposals, and Copilot	33
4.5	Unit Test Results — RAG Context, Entity Extraction, and Relationship Discovery	34
4.6	Overall Unit Testing Summary	34
A.1	Detailed Use Case: Generate Context-Aware Writing	45

Contents

List of Figures

List of Tables

Chapter 1

Introduction

CoWriteIA is an AI-powered writing assistant designed to support creative writers, novelists, researchers, and content creators. Modern writing workflows require managing notes, drafts, characters, scenes, and references across several disconnected tools, which often breaks focus and reduces productivity. Writers struggle to maintain narrative consistency, track earlier ideas, and keep a coherent writing style when working on long projects. Existing tools offer only isolated features such as grammar correction or basic generation and do not provide project-level understanding or semantic memory [5, 8].

CoWriteIA addresses these issues by creating a unified, intelligent workspace where all project files are indexed, searchable, and semantically connected. By integrating project-level memory, context-aware writing, dialogue generation, character management, and research support, the system helps writers maintain consistency and improve their creative process. The platform is designed to reduce cognitive load, avoid fragmented workflows, and provide meaningful AI assistance throughout the writing journey. This chapter presents the background, existing solutions, problem statement, scope, project modules, and work division that form the foundation of this system.

1.1 Existing Solutions

Several writing and AI-assisted tools exist, but each focuses on limited aspects of the writing process. Grammarly offers grammar correction and style suggestions but lacks deep contextual awareness across long projects. Notion AI and Jasper AI provide generative assistance but do not maintain story-level continuity or user-specific writing style. Tools like Scrivener help with organization but have no semantic understanding or AI memory. As a result, writers must repeatedly switch between applications to manage notes, drafts, characters, and research, leading to inefficiency and inconsistent writing flow.

Table 1.1: Comparison of Existing Solutions

System Name	System Overview	System Limitations
Grammarly	Provides grammar correction, clarity improvements, and tone suggestions.	No project awareness, no memory of earlier chapters, no semantic search.
Notion AI	Offers AI-assisted content generation and note organization.	Cannot maintain narrative consistency; lacks dialogue and character support.
Scrivener	Strong organizational tool for large writing projects with chapter/scene structure.	No AI support, no semantic retrieval, no automated gap or style analysis.

1.2 Problem Statement

Writers working on long-form projects often lose track of earlier ideas, character traits, plot points, and stylistic decisions. This leads to inconsistencies, repeated ideas, and a break in narrative flow. Existing tools either provide isolated writing support or document organization but do not combine both with meaningful context. AI tools can generate text but fail to maintain project-level continuity, making the generated text feel disconnected from the writer’s established style or storyline. Writers spend significant time searching through older drafts to recall information [3, 5].

CoWriteIA aims to solve these problems by offering an intelligent system that continuously indexes all project content, retrieves relevant context, and assists writers in generating text that aligns with their established narrative and writing style. By maintaining a unified knowledge base and supporting character consistency, scene management, dialogue generation, and semantic search, the system reduces cognitive overhead and improves creative flow.

1.3 Scope

The scope of CoWriteIA includes building an AI-driven writing environment that supports project management, semantic search, context-aware writing, dialogue generation, research integration, and style adaptation. The system will allow users to upload documents, create new content, store character information, generate dialogues, and retrieve context from a vector-based semantic memory [4, 8]. It will support long-form writing projects such as novels, research documents, and story-driven content.

The system will not include plagiarism detection, multimedia editing, or full publishing workflows. It focuses strictly on improving the writing process, maintaining consistency, and providing intelligent assistance throughout the creative workflow.

1.4 Modules

The project consists of several modules, each responsible for a unique part of the writing workflow.

1.4.1 Module 1: Project Indexing and Semantic Memory

This module extracts, embeds, and organizes all project content into a semantic database for context-aware retrieval [8].

1. Automatic project indexing and embedding generation.
2. Semantic search based on meaning instead of keywords.

1.4.2 Module 2: Context-Aware Writing Assistant

This module provides intelligent writing suggestions that match the user's tone and project context [5].

1. Generates coherent drafts aligned with past content.
2. Retrieves relevant information to maintain consistency.

1.4.3 Module 3: Character and Scene Management

This module stores and manages characters, scenes, and narrative details.

1. Character profiles with traits and relationships.
2. Scene storage and tracking.

1.4.4 Module 4: Dialogue Generation

Generates natural, character-consistent dialogue suggestions.

1. Dialogue generation based on personality and context.
2. Supports narrative flow within scenes.

1.4.5 Module 5: Research Integration

Fetches factual data from external sources for realistic writing.

1. Web-based research retrieval.
2. Insertable factual references.

1.4.6 Module 6: Project Query Interface

Allows users to ask natural-language questions about their own project.

1. Semantic question-answering.
2. Source-linked responses.

1.4.7 Module 7: Gap Analysis Module

Analyzes draft content to find missing or weak areas.

1. Identifies incomplete sections.
2. Suggests improvements for clarity and consistency.

1.5 Work Division

The work completed during FYP-1 was divided into two major iterations. A summary of responsibilities is presented in the tables below.

Iteration I Tasks

Table 1.2: Work Division for Iteration I

Task	Ayesha	Junaid	Aisha
SRS Document	✓	✓	✓
UML Diagrams	✓	✓	✓
UI Design		✓	✓
Database Connection		✓	✓
Frontend (Main Pages)	✓		
Project Indexing Agent	✓	✓	
Knowledge Retrieval Agent		✓	✓
ChatBot Support		✓	

Iteration II Tasks

Table 1.3: Work Division for Iteration II

Task	Ayesha	Junaid	Aisha
Inline Copilot Support	✓		
Context-Aware Writing Module	✓		
Gap Analysis Module			✓
Style Adaptation		✓	
Testing	✓	✓	✓
Documentation (All Sections)	✓	✓	✓

Project Timeline

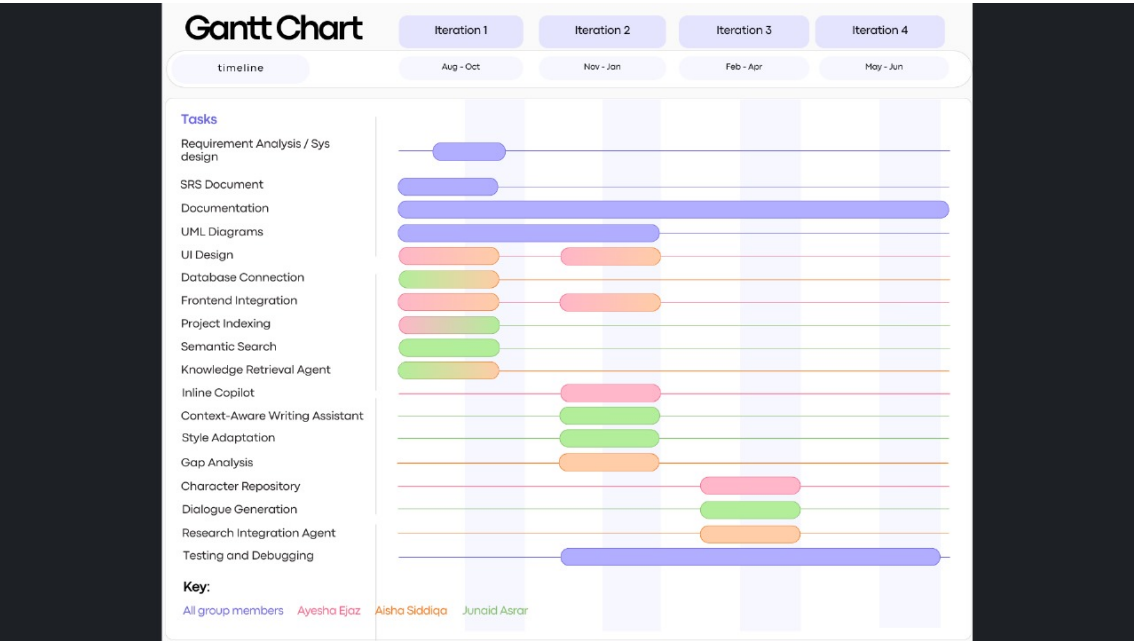


Figure 1.1: Gantt Chart - Project Timeline

Chapter 2

Project Requirements

This chapter defines the requirements essential for developing CoWriteIA. These requirements were derived from the FYP-1 Proposal, Mid Report, and the SRS document. They describe how the system should behave, how users will interact with it, and what constraints and quality standards must be followed. The requirements are divided into functional and non-functional categories, with additional details about expected user interactions.

2.1 Use-case

CoWriteIA provides writers with AI-assisted features such as project indexing, semantic retrieval, character management, context-aware writing, and dialogue generation. To clearly define how users interact with these features, system behavior is modeled through detailed use cases.

A use case describes the interaction between the primary actor (the writer) and the system to accomplish a specific task. Each use case outlines the trigger event, preconditions, main workflow steps, and system responses. These models help ensure that system functionality aligns with user expectations and guide the design of interactive features.

In addition to textual use cases, a visual representation in the form of a Use-Case Diagram provides an overview of the major user interactions with CoWriteIA. This includes actions such as uploading documents, retrieving semantic context, generating content with the writing assistant, managing characters, and initiating dialogue generation. Such diagrams help maintain clarity, consistency, and completeness throughout the system design.

A detailed use case example for the core functionality **Generate Context-Aware Writing (UC-01)** is provided in Appendix D. This detailed use case outlines the complete interaction flow including preconditions, main flow steps, alternate flows, exceptions, and postconditions. The use case demonstrates how the system retrieves semantic context, processes user requests through the AI model, and delivers generated content while han-

dling various edge cases such as missing embeddings or service unavailability.

2.2 Functional Requirements

The functional requirements describe the operations the CoWriteIA system must support. These requirements come directly from the system features identified in the Proposal and SRS.

2.2.1 Module 1: Project Indexing and Semantic Memory

This module handles ingestion, indexing, and semantic storage of project documents.

1. The system shall allow users to upload project files including text documents, chapters, notes, and research material.
2. The system shall extract, segment, and convert uploaded content into embeddings for semantic retrieval.
3. The system shall store embeddings in a vector database.
4. The system shall provide semantic search capabilities based on meaning rather than keyword matching.
5. The system shall return ranked search results with source references.

2.2.2 Module 2: Context-Aware Writing Assistant

This module generates content aligned with user writing style and project context.

1. The system shall analyze previous project content to understand tone, terminology, and style.
2. The system shall allow users to generate context-aware text suggestions based on previous chapters or notes.
3. The system shall retrieve relevant context automatically when generating new content.
4. The system shall maintain style consistency between newly generated and existing content.

2.2.3 Module 3: Character and Scene Management

This module manages characters, their traits, and related narrative structures.

1. The system shall allow users to create, edit, and store character profiles.
2. The system shall store attributes such as personality, relationships, behaviors, and backstory.
3. The system shall allow users to manage scenes and attach characters to scenes.
4. The system shall assist in retrieving character information when generating story text or dialogue.

2.2.4 Module 4: Dialogue Generation

This module generates consistent, character-matching dialogue.

1. The system shall generate dialogue aligned with character personality and tone.
2. The system shall allow users to request dialogue for specific characters or scenes.
3. The system shall ensure continuity between generated dialogue and existing narrative.

2.2.5 Module 5: Research Integration Module

This module retrieves factual information to support realistic writing.

1. The system shall allow users to search for factual references.
2. The system shall fetch external information from trusted research sources.
3. The system shall present research results with citations.

2.2.6 Module 6: Project Query Interface

This module allows users to ask natural-language questions about their project.

1. The system shall process user queries related to characters, scenes, chapters, or events.
2. The system shall fetch relevant information from the semantic memory.
3. The system shall provide answers with linked source passages.

2.2.7 Module 7: Gap Analysis Module

This module identifies missing or weak areas of the writer's draft.

1. The system shall analyze uploaded chapters for missing elements such as incomplete scenes or inconsistent character behavior.
2. The system shall highlight areas needing expansion or clarification.
3. The system shall provide suggestions to improve narrative flow and completeness.

2.3 Non-Functional Requirements

Non-functional requirements ensure that CoWriteIA performs reliably, efficiently, and securely.

2.3.1 Usability

1. The system shall provide a simple and intuitive interface accessible to writers with minimal technical expertise.
2. The system shall allow users to perform core actions such as uploading files, searching, and generating text within no more than three interactions.
3. The UI shall clearly present semantic search results with source references.

2.3.2 Performance

1. The system shall index uploaded documents within 5 seconds for an average chapter-length file.
2. Semantic search results shall appear within 2 seconds of a query.
3. Generated text responses shall be produced within 3–5 seconds depending on context length.

2.3.3 Security

1. User project data shall be stored securely using encrypted connections.
2. Only authenticated users shall access their own documents.
3. The system shall not use project data for training without user permission.

2.3.4 Reliability

1. The system shall maintain uptime of at least 99
2. The system shall recover gracefully from API or model failures by retrying or presenting fallback responses.

2.3.5 Maintainability

1. The codebase shall follow modular architecture to allow updates to individual agents.
2. The system shall support integration of new language models without requiring major structural changes.

2.3.6 Compatibility

1. The system shall run on modern browsers including Chrome, Edge, and Firefox.
2. The frontend shall be built using Next.js and shall be compatible with desktop and tablet interfaces.

2.3.7 Scalability

1. The vector database shall support scaling as the user creates larger projects.
2. The system shall handle multiple concurrent requests without significant performance degradation.

Chapter 3

System Overview

Introduction

This chapter provides a high-level description of the system's overall functionality, context, and architectural design. The aim is to present how the major parts of the system interact and why the system has been decomposed into specific modules and tiers.

System Overview

CoWriteIA is an AI-assisted writing platform designed to support writers through various stages of the creative process [2, 6, 7]. The system enables users to:

- Create and manage writing projects
- Upload and organize documents
- Manage character profiles and story elements
- Generate and refine written content using AI assistance

To efficiently support these features, the system separates concerns into distinct, coordinated components: user interaction, application logic, AI processing, and data management.

3.1 Architectural Design

The architecture of CoWriteIA follows a **multi-tier model**, combining **client-server** and **layered** principles. This structure organizes the system into four logical tiers, each with specific responsibilities to ensure maintainability, scalability, and clarity.

Architectural Tiers

Data Storage Components

- **Main Database:** Stores structured system data (users, projects, characters)
- **Vector Database:** Manages embeddings for semantic retrieval and search

Tier	Component	Responsibilities
Presentation	Frontend Application	User interaction, interface rendering, and user input handling
Application Logic	Backend (API Server)	Authentication, business workflows, and project/document/character management
AI Processing	Processing Worker	Embeddings generation, semantic retrieval, and AI text generation
Data	Storage Systems	Structured data, vector storage, and file management

Table 3.1: Architectural Tiers and Responsibilities

- **File Storage:** Handles uploaded documents and exported files

Architectural Diagram

A detailed architecture diagram illustrating the complete multi-tier structure of CoWriteIA is provided in Appendix C (Figure A.3).

Diagram Development Process

- **Initial Design Stage:** Create a *Box and Line Diagram* for simpler representation of systems
- **Finalization Stage:** After selecting the architecture style/pattern (MVC, Client-Server, Layered, Multi-tiered), create detailed mapping of modules/components to each part of the architecture

Design Principles

This architectural decomposition supports the system’s functional requirements while ensuring:

- **Flexibility:** Components can be modified independently
- **Maintainability:** Clear separation of concerns simplifies updates and debugging
- **Scalability:** Each tier can be scaled independently based on demand
- **Security:** Controlled data flow between tiers with proper authentication

Key Architectural Decisions

- **Separation of AI Processing:** Intensive AI tasks are handled by dedicated workers to maintain API responsiveness
- **Multi-tier Structure:** Enables independent development, testing, and deployment of each tier

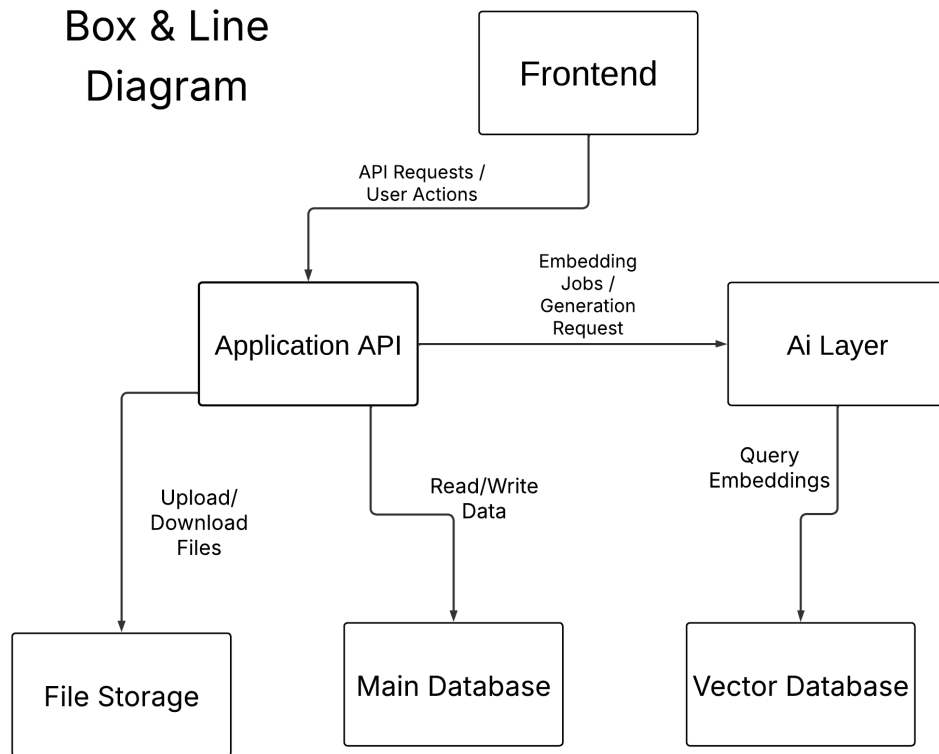


Figure 3.1: Box and Line Diagram

- **Modular Design:** Supports future enhancements and feature additions

3.2 Data Design

The data design of CoWriteIA transforms functional requirements into structured data models that support **project creation**, **AI-assisted writing**, **semantic search**, **character management**, and **dialogue modelling**.

Data Organization

The system organizes data into three primary categories:

- **Relational Data:** User accounts, projects, documents, characters, and chat sessions stored in a structured database with defined schemas and foreign key relationships
- **Vector Data:** High-dimensional embeddings generated from text content, enabling semantic similarity search and context retrieval
- **Binary Data:** Uploaded files and exported documents stored separately with meta-data linkage

Data Flow and Transformation

Data flows through multiple transformation stages:

- **Input Processing:** User actions validate and create/update structured database records
- **Embedding Generation:** Text content is transformed into vector representations for AI operations
- **Retrieval Augmentation:** Semantic queries fetch relevant context from vector and relational stores
- **Output Generation:** AI responses and exports combine retrieved data with generated content

This design ensures **data integrity**, **efficient retrieval**, and **scalable processing**. Recent work on efficient local models such as Phi-3 highlights practical on-device capabilities for embedding and retrieval [1].

Figure 3.2 provides the Entity–Relationship Diagram showing database schema structure, while Figure 3.4 presents the conceptual domain model.

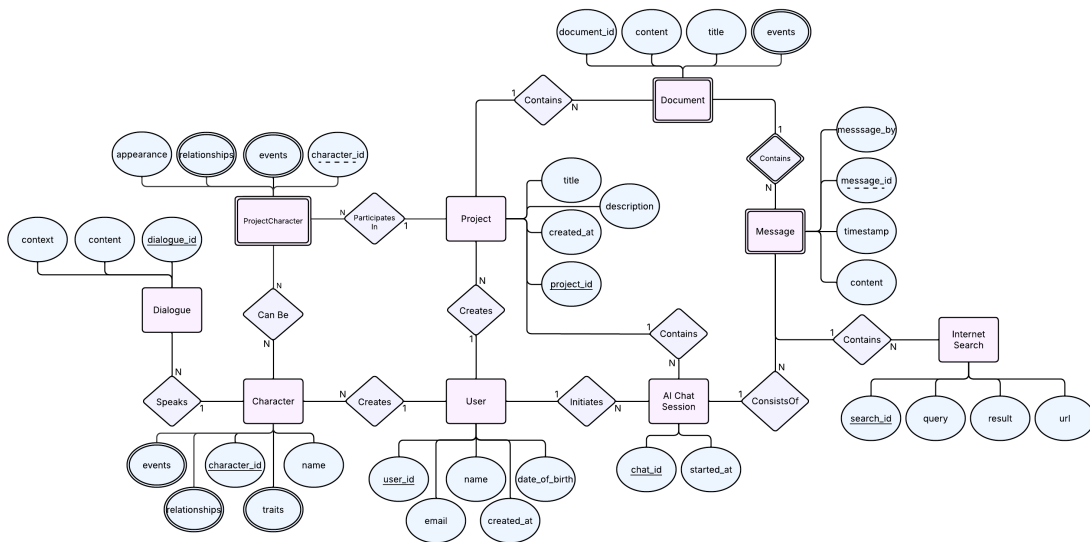


Figure 3.2: Entity–Relationship Diagram for CoWriteIA

3.2.1 Class Diagram

The class diagram represents the object-oriented view of the system and shows classes, attributes, and associations used by the application logic. It provides a structural foundation for backend implementation and helps ensure consistency between the conceptual design and the code-level architecture.

3.3 Domain Model

The domain model provides a conceptual view of the system’s core entities and their relationships, bridging functional requirements and technical implementation. It describes

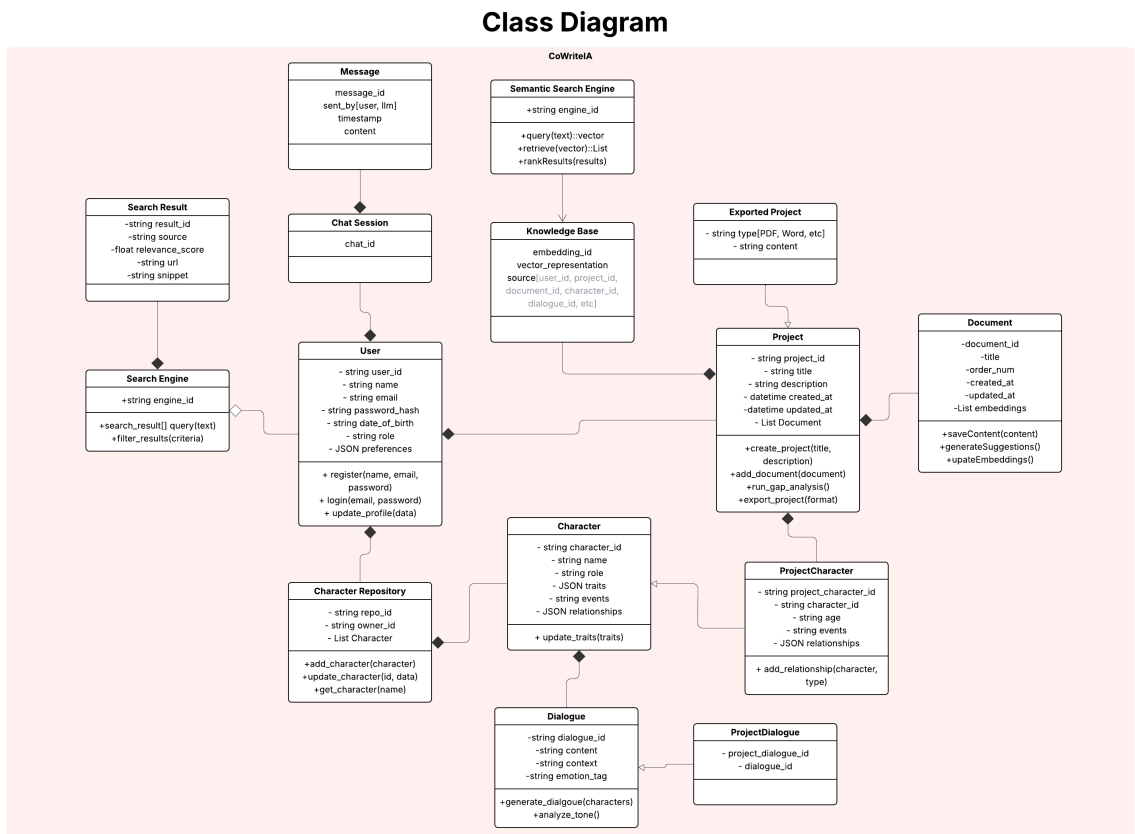


Figure 3.3: Class Diagram for CoWriteIA

the key business objects, their attributes, and how they interact within CoWriteIA.

Entities and Attributes

The major entities of the system include:

- **User:** Represents a system user who owns projects and interacts with the AI. Attributes include `user_id`, `name`, `email`, `date_of_birth`, `password_hash`, `role`, and `created_at`.
- **Project:** A workspace created by the user. Attributes include `project_id`, `title`, `created_at`, and `updated_at`.
- **Document:** Represents a chapter or section of a project. Attributes include `document_id`, `title`, `order_num`, `created_at`, and `updated_at`.
- **Character Repository:** A user-owned collection that groups all character profiles.
- **Character:** Represents a story character. Attributes include `character_id`, `name`, `traits`, `history`, `created_at`, and `updated_at`.
- **ProjectCharacter:** An association class linking characters with specific projects. Stores project-specific values such as age and events.
- **Dialogue:** Represents character dialogues. Attributes include `dialogue_id`, `content`, `context`, and `emotion_tag`.
- **ProjectDialogue:** A linking entity that associates dialogues with a project.
- **Chat Session:** Represents an AI chat interaction initiated by the user. Identified by `chat_id`.
- **Message:** Stores an individual message in a chat session. Attributes include `message_id`, `sent_by`, `timestamp`, and `content`.
- **Search Engine:** Represents the semantic search component.
- **Search Result:** Stores contextual search outputs. Attributes include `result_id`, `query`, and `content`.
- **Knowledge Base:** Stores vector embeddings and source references. Attributes include `embedding_id`, `vector_representation`, and `source_ids`.
- **Exported Project:** Represents generated output files such as PDF or Word exports.

3.3.1 Relationships and Associations

The domain model defines the relationships that structure how information flows across the system:

- A **User** owns multiple **Projects**.
- A **Project** contains multiple **Documents**.
- A **User** owns a **Character Repository**, which stores many **Characters**.
- A **Project** includes multiple **Characters** via **ProjectCharacter**.
- A **Character** speaks one or more **Dialogues**.
- A **Project** associates specific dialogues through **ProjectDialogue**.
- A **User** initiates a **Chat Session**.

- A **Chat Session** stores multiple **Messages**.
- The **LLM Chatbot** interacts with the **Semantic Search Engine** and **Knowledge Base** to generate responses.
- The **Search Engine** produces multiple **Search Results**, which may be injected into messages or documents.

These relationships ensure that writing, character creation, semantic search, and AI interactions remain consistent and interconnected across the system.

3.3.2 Domain Model Diagram

Figure 3.4 presents the complete domain model for CoWriteIA. It includes all major entities, attributes, and associations, including weak entities and conceptual relationships.

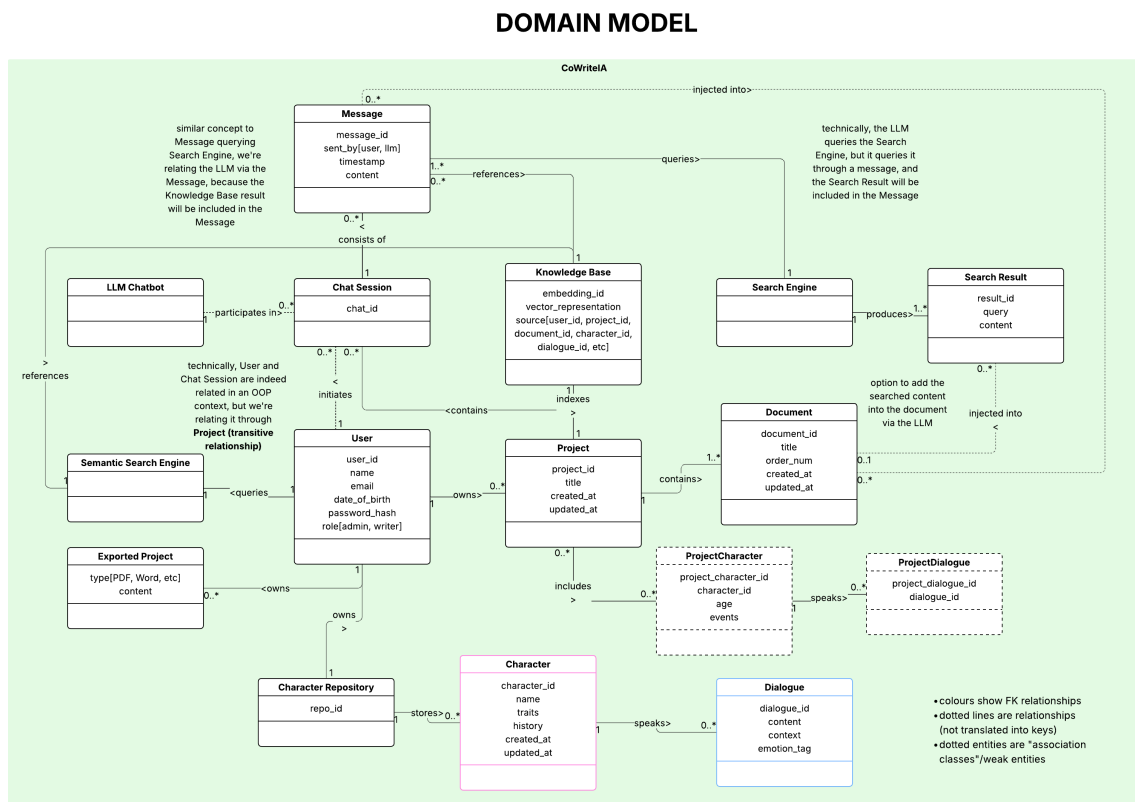


Figure 3.4: Domain Model Diagram for CoWriteIA

3.4 Design Models

3.4.1 State Transition Diagram

Figure 3.5 illustrates state transitions for key entities based on user actions and system events. The diagram shows how entities move between states:

- **Project**: Transitions from Draft → Active → Archived/Deleted based on user ac-

tions

- **Document:** Moves through Creating → Editing → Saved → Publishing → Published states
- **Chat Session:** Cycles between Idle → Processing → Waiting for User → Completed
- **Character:** Changes from Draft → Active → Archived as needed

These transitions ensure proper lifecycle management and data consistency throughout the system.

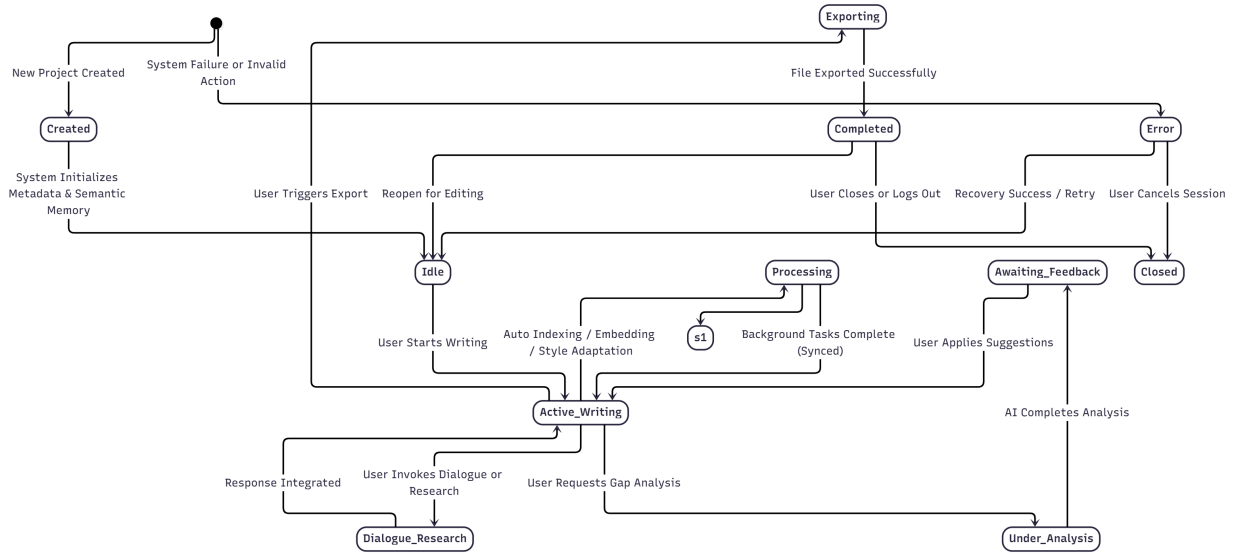


Figure 3.5: State Transition Diagram for CoWriteIA

Chapter 4

Implementation and Testing

This chapter describes the current implementation status of the CoWriteIA system and the testing strategies applied to validate its functionality. The focus of this phase was to implement the core backend services, major APIs, and conduct rigorous black box testing to ensure that all user-facing functionalities behave correctly according to the system requirements.

The implementation described in this chapter reflects the progress achieved up to the current iteration of the project.

4.1 Algorithm Design

The system implements a Retrieval-Augmented Generation (RAG) pipeline to enable intelligent document-based responses. The algorithm focuses only on the core functional requirements: document processing, semantic indexing, context retrieval, and AI response generation.

The CoWriteIA implementation in this repository realizes a semantic indexing + RAG (retrieval-augmented generation) pipeline. The implemented components (and their primary locations in the codebase) are:

- Document chunking and preprocessing
 - `backend/app/services/text_extraction_service.py :: TextExtractionService._create_text_chunk`
- Embedding generation and vector storage
 - `backend/app/services/embedding_service.py :: EmbeddingService.generate_embeddings, store_text_chunks_with_embeddings`
 - `backend/app/services/chroma_service.py :: ChromaService.add_embeddings, search_similar`
 - `backend/app/repositories/text_chunk_repository.py :: update_embedding, get_chunks_without_embeddings, vector_search / vector_search_chroma`

- Context retrieval (semantic similarity)
 - `backend/app/services/rag_context_service.py :: RAGContextService.assemble`
 - `backend/app/services/search_service.py :: SearchService.semantic_search, find_similar_content`
- AI response generation (RAG orchestration and assistant)
 - `backend/app/services/ai_chat_service.py :: AIChatService.chat`
 - `backend/app/services/copilot_service.py :: CopilotService.generate_sugges`

The following subsections contain concise algorithm descriptions derived from the actual implementation and precise pseudocode that maps to the repository functions.

4.1.1 High-level Flow

1. User uploads file via endpoints in `backend/app/api/v1/endpoints/files.py`.
2. File processing triggers text extraction and chunking:
 - `TextExtractionService.extract_text_from_file(...)` and `TextExtractionService`
 - Chunks are created as `TextChunk` models and persisted via `TextChunkRepository`.
3. Embeddings are generated for chunks:
 - `EmbeddingService.generate_embeddings(...)` / `generate_single_embedding(...)`
 - Embeddings saved both in chunk metadata (MongoDB) and in ChromaDB via `ChromaService` when available.
4. Semantic search / RAG:
 - `RAGContextService.assemble_context(...)` calls embedding/search services to retrieve top- k chunks.
 - `SearchService` handles hybrid/fallback logic (ChromaDB preferred, fallback to MongoDB vector search).
5. LLM generation:
 - `AIChatService.chat(...)` or `CopilotService.generate_suggestion(...)` build a prompt from the retrieved context and call the configured LLM (`GroqService` or other).
6. Results returned to the client; metadata and analytics stored as needed.

Example short sequence (mapping code → runtime)

1. User uploads file → API endpoint in `backend/app/api/v1/endpoints/files.py` triggers `process_uploaded_file`.
2. `process_uploaded_file` calls `TextExtractionService.chunk_text` to create chunks.
3. Indexing background task (or immediate synchronous call) calls `EmbeddingService.generate_embeddings` for chunks.
4. `EmbeddingService` stores embeddings via `TextChunkRepository` and `ChromaService`.
5. Later, user sends search/query → `RAGContextService.assemble_context` retrieves top chunks using `ChromaService` or `SearchService`.

6. `AIChatService.chat` builds prompt with context and calls `GroqService (LLM)`; response persisted and returned.

4.1.2 Pseudocode: Document Chunking and Preprocessing

The system splits extracted text into sentence-aware chunks within a target size and optionally applies overlap. Each chunk is materialized using the implementation in `backend/app/services/`

Algorithm 1 ChunkTextAndCreateChunks

Require: *file_id*, *project_id*, *full_text*, *max_chunk_size*, *overlap* (optional), *start_chunk_index* (default 0)

Ensure: List of TextChunk models *chunks*

```

1: sentences  $\leftarrow$  TextExtractionService._split_into_sentences(full_text)
2: chunks  $\leftarrow$  []
3: current_chunk_sentences  $\leftarrow$  []; current_word_count  $\leftarrow$  0; chunk_index  $\leftarrow$ 
   start_chunk_index
4: for each sentence s in sentences do
5:   sentence_words  $\leftarrow$  count_words(s)
6:   if current_word_count + sentence_words > max_chunk_size and current_chunk_sentences not empty then
7:     chunk_text  $\leftarrow$  join(current_chunk_sentences, " ")
8:     start_pos  $\leftarrow$  compute absolute start in full_text
9:     chunk_model  $\leftarrow$  TextExtractionService._create_text_chunk(file_id,
project_id, chunk_text, chunk_index, start_pos, current_word_count)
10:    append chunk_model to chunks
11:    chunk_index  $\leftarrow$  chunk_index + 1
12:    if overlap configured then
13:      current_chunk_sentences  $\leftarrow$  take last overlap words/sentences of current_chunk_sentences
14:      current_word_count  $\leftarrow$  count_words(current_chunk_sentences)
15:    else
16:      current_chunk_sentences  $\leftarrow$  []; current_word_count  $\leftarrow$  0
17:    end if
18:  end if
19:  append s to current_chunk_sentences; current_word_count  $\leftarrow$  current_word_count + sentence_words
20: end for
21: if current_chunk_sentences not empty then
22:   chunk_text  $\leftarrow$  join(current_chunk_sentences, " ")
23:   start_pos  $\leftarrow$  compute absolute start in full_text
24:   chunk_model  $\leftarrow$  TextExtractionService._create_text_chunk(file_id,
project_id, chunk_text, chunk_index, start_pos, current_word_count)
25:   append chunk_model to chunks
26: end if
27: return chunks

```

Notes:

- Uses sentence-aware splitting and target size with optional overlap per TextExtractionService.

- Returns TextChunk Pydantic models; persistence occurs later (e.g., in `store_text_chunks_with_embeddings`)

4.1.3 Embedding Generation and Storage

This stage encodes chunked text into dense vectors and persists both the metadata and vectors. It maps to `EmbeddingService` and `ChromaService` usage in the codebase.

Algorithm 2 `GenerateAndPersistEmbeddings`

Require: *project_id, file_id, list_of_chunks* (content strings)

Ensure: List of created/updated TextChunk records with embeddings

```

1: chunk_texts ← [ chunk.content for chunk in list_of_chunks ]
2: embeddings ← EmbeddingService.generate_embeddings(chunk_texts) ▷ Runs
   threadpool synchronous encoding; model loads lazily
3: created_chunks ← []
4: for each (chunk_info, embedding) in zip(list_of_chunks, embeddings) do
5:   chunk_model ← build TextChunk( file_id = file_id, project_id = project_id,
   content = chunk_info.content, start_position = ..., end_position = ...,
   chunk_index = chunk_info.chunk_index, word_count = ..., embedding_vector =
   embedding )
6:   saved_chunk ← TextChunkRepository.create(chunk_model) ▷ Persist
   metadata in MongoDB
7:   append saved_chunk to created_chunks
8: end for
9: if ChromaService available then
10:   chroma_entries ← map created_chunks to {id, embedding, metadata}
11:   ChromaService.add_embeddings(project_id, chroma_entries)
12: else
13:   Optionally store embeddings only within MongoDB fields
14: end if
15: return created_chunks

```

Implementation highlights:

- **Embedding model:** `sentence-transformers/all-MiniLM-L6-v2` (see embedding service and verification scripts).
- **Dual storage:** Metadata in MongoDB `text_chunk` documents; vectors in ChromaDB via `backend/app/services/chroma_service.py`.
- **Batch background tasks:** Celery jobs generate missing embeddings in `backend/app/tasks/index(batch_generate_embeddings_task)`.

4.1.4 Pseudocode: Context-aware Retrieval (RAG)

This step assembles a rich context from semantically similar chunks and optional entity/scene/relationship metadata.

Algorithm 3 AssembleRAGContext

Require: query Q , $project_id$, optional $file_id$, max_chunks k

Ensure: $context_payload$: list of chunks, entities, scenes, relationships

```

1: query_embedding ← EmbeddingService.generate_single_embedding(Q)
2: if ChromaService available then
3:   vector_hits ← ChromaService.search_similar(project_id,
        query_embedding, n_results = k)
4: else
5:   vector_hits ← TextChunkRepository.vector_search(project_id,
        query_embedding, limit = k)
6: end if
7: top_chunks ← convert vector_hits to chunk payloads (id, content, score, file_id,
        positions)
8: if include_entities then
9:   entities ← EntityRepository.get_by_project(project_id)
10:  entity_matches ← identify entities mentioned in top_chunks
        (EntityExtractionService or string matching)
11: end if
12: if include_scenes then
13:   scenes ← SceneRepository.find_by_project(project_id)
14:   mapped_scenes ← map chunks to scenes when positions overlap or proximity
        matches
15: end if
16: if include_relationships then
17:   relationships ← RelationshipRepository.find_by_project(project_id)
18: end if
19: context_payload ← {"chunks": top_chunks, "entities": entity_matches,
        "scenes": mapped_scenes, "relationships": relationships (filtered)}
20: return context_payload
  
```

Notes:

- `RAGContextService.assemble_context` collects chunk-level results, enriches them with entity and scene metadata, and returns a compact context mapping suitable for prompt building (see `backend/app/services/rag_context_service.py`).
- `SearchService` implements hybrid logic and fallback ordering (Chroma → Mon-

goDB vectorSearch).

4.1.5 Pseudocode: AI Response Generation (RAG + LLM Call)

This step orchestrates context assembly, prompt construction, LLM invocation, and persistence. It corresponds to `AIChatService.chat` and `CopilotService.generate_suggestion`.

Algorithm 4 GenerateAIResponseWithContext

Require: *user_message*, *project_id*, optional *file_id*, *user_id*, *max_context_chunks*

Ensure: *assistant_message*

- 1: **Step A: Assemble RAG context**
 - 2: `context` \leftarrow `RAGContextService.assemble_context(query = user_message, project_id = project_id, file_id = file_id, max_chunks = max_context_chunks)`
 - 3: **Step B: Format context into LLM prompt**
 - 4: `formatted_context` \leftarrow `RAGContextService.format_context_for_llm(context)`
 - 5: `system_prompt` \leftarrow `AIChatService.SYSTEM_PROMPT`
 - 6: `conversation_history` \leftarrow `ConversationRepository.get_latest_messages(conversation_id or new)`
 - 7: `llm_messages` \leftarrow `build_messages_list(system_prompt, conversation_history, user_message, formatted_context)` \triangleright system message, recent assistant/user turns, current user message, appended context
 - 8: **Step C: Call LLM**
 - 9: `assistant_content` \leftarrow `GroqService.generate(messages = llm_messages, max_tokens = ..., model = ...)`
 - 10: **Step D: Postprocess and store**
 - 11: `cleaned` \leftarrow `clean_text(assistant_content)` \triangleright e.g., `CopilotService._clean_suggestion`
 - 12: `MessageRepository.save(role = assistant, content = cleaned, context_used = context)`
 - 13: update conversation metadata (message count, token usage, timestamps)
 - 14: **return** `cleaned`
-

Implementation notes:

- `AIChatService.chat` orchestrates assembling context, formatting prompts, invoking the LLM, and persisting messages.
- `CopilotService.generate_suggestion` targets inline assistance: it infers writing style from RAG results, calls the LLM, then returns a cleaned suggestion and the context used (see `backend/app/services/copilot_service.py`).

Complexity, Reliability, and Practical Concerns

- **Chunking complexity:** $O(n)$ in the number of sentences per document. Overlap increases effective storage but preserves local context for downstream retrieval.
- **Embedding generation:** Dominated by model inference cost per chunk. Batch generation reduces overhead (indexing background tasks `batch_generate_missing_embeddings`).
- **Vector search:** ChromaDB uses efficient ANN (HNSW) with sub-linear query cost; fallback MongoDB vector aggregation may be slower for large collections.
- **Fault tolerance:**
 - Embedding/model calls may fail due to transient network or provider issues; services use `try/catch` with sensible fallbacks (see embedding service and verify scripts).
 - RAG assembly respects available services and degrades gracefully (ChromaDB optional; code falls back to MongoDB vector search when unavailable).
- **Reindexing & incremental updates:** `TextExtractionService.reprocess_file` deletes existing chunks and marks the file ready for reprocessing; batch background tasks handle any missing embeddings.

Algorithm 5 ReprocessFileAndIncrementalUpdate

Require: *file_id, project_id*

Ensure: File is reindexed and embeddings regenerated as needed

```

1: TextChunkRepository.delete_by_file(file_id)
2: FileRepository.mark_ready_for_processing(file_id)
3: full_text ← TextExtractionService.extract_text_from_file(file_id)
4: chunks ← TextExtractionService.chunk_text(project_id, file_id, full_text,
    max_chunk_size, overlap)
5: TextChunkRepository.bulk_insert(chunks)
6: enqueue batch_generate_missing_embeddings(project_id, file_id)

```

Concrete File References (Key Implementation Points)

- **Text chunking and models**
 - `backend/app/services/text_extraction_service.py:: chunk_text(), _create_text_chunk(), _split_into_sentences()`
 - `backend/app/repositories/text_chunk_repository.py:: create(), get_by_file(), get_chunks_without_embeddings(), update_embedding()`
- **Embeddings and vector store**
 - `backend/app/services/embedding_service.py:: generate_single_embedding(), generate_embeddings(), store_text_chunks_with_embeddings(), find_similar_chunks()`
 - `backend/app/services/chroma_service.py:: get_or_create_collection(),`

- ```
add_embeddings(), search_similar()
```
- backend/app/tasks/indexing.py :: index\_document\_task(), batch\_generate\_embeddings()
  - **RAG and search**
    - backend/app/services/rag\_context\_service.py :: assemble\_context(), format\_context\_for\_llm()
    - backend/app/services/search\_service.py :: semantic\_search(), find\_similar\_context(), hybrid logic
  - **LLM orchestration and assistants**
    - backend/app/services/ai\_chat\_service.py :: chat()
    - backend/app/services/copilot\_service.py :: generate\_suggestion(), \_gather\_story\_context()

See Appendix A.5 for pseudo-API call signatures.

## 4.2 External APIs and SDKs

The system integrates multiple third-party APIs and SDKs to support AI capabilities, authentication, and data persistence.

| API / SDK                 | Description                   | Purpose of Usage               | Endpoint/ Function      |
|---------------------------|-------------------------------|--------------------------------|-------------------------|
| OpenAI API (v1)           | Large language model services | Text generation and embeddings | /v1/chat/completions    |
| FastAPI                   | Python web framework          | REST API backend               | @app.get(), @app.post() |
| PostgreSQL                | Relational database system    | User and project data storage  | psycopg2 driver         |
| VectorDB (FAISS/Pinecone) | Vector similarity engine      | Semantic search and retrieval  | similarity_search()     |

Table 4.1: External APIs and SDKs Used

## 4.3 Testing Details

Testing played a critical role in validating the correctness, reliability, and stability of the implemented system. A combination of black box testing and unit testing strategies was adopted to ensure that the system meets its functional requirements.

### 4.3.1 Black Box Testing

Black box testing was used to validate the external behavior of the system without reference to internal implementation details. The focus was on verifying that system features produced correct outputs when interacting through public interfaces such as API end-

points.

#### **4.3.1.1 Purpose of Black Box Testing**

The objective of this testing was to ensure that system functionality aligned with the documented functional requirements by validating response codes, output structures, and observable system behavior.

#### **4.3.1.2 Testing Environment**

Testing was conducted using an isolated staging environment that included a running API server, a connected database, and a REST API testing client. All tests were executed externally without accessing source code.

#### **4.3.1.3 Functional API Coverage**

The following functional modules were tested as black box components:

- Authentication services (registration, login, token validation)
- Project management operations
- File handling operations
- Search and retrieval services
- Chat and conversational APIs

#### **4.3.1.4 Workflow and Scenario Testing**

End-to-end user workflows were validated through multi-step test scenarios, including project creation, file indexing, semantic search, and AI-assisted responses.

#### **4.3.1.5 Negative and Security Testing**

Invalid inputs, unauthorized requests, and forbidden access attempts were tested to verify that the system safely rejected improper usage without exposing internal errors.

#### **4.3.1.6 Error and Edge Case Validation**

Boundary conditions such as missing data, empty result sets, and invalid resource requests were evaluated to ensure stable system responses.

#### **4.3.1.7 Black Box Test Evidence**

All executed test cases were recorded and maintained as structured documentation.

## 4. Implementation and Testing

|    | A     | B              | C                 | D                  | E         | F                  | G                  | H                 | I        | J             |
|----|-------|----------------|-------------------|--------------------|-----------|--------------------|--------------------|-------------------|----------|---------------|
| 1  | TC ID | Module         | Test Case Name    | Description        | Test Type | Preconditions      | Steps / Input      | Expected Result   | Priority | Tags / Req ID |
| 2  | TC-01 | System Health  | Root Endpoint A   | Verify root endp   | Black Box | API server runnir  | GET /              | 200 OK, valid res | High     | health        |
| 3  | TC-02 | System Health  | Health Check AF   | Verify /health end | Black Box | API server runnir  | GET /health        | 200 OK, status =  | High     | health        |
| 4  | TC-03 | System Health  | API Docs doctyp   | Verify API docs a  | Black Box | API server runnir  | GET /docs or /op   | 200 OK, docs av   | Medium   | docs          |
| 5  | TC-04 | Authentication | User Registratio  | Register user wit  | Black Box | DB online, user r  | POST /auth/regis   | 201 Created, use  | High     | REQ-auth-1    |
| 6  | TC-05 | Authentication | User Registratio  | Register with exi  | Black Box | User already exis  | POST /auth/regis   | 400/409 error me  | High     | negative      |
| 7  | TC-06 | Authentication | Login (Valid)     | Login with correc  | Black Box | Registered user    | POST /auth/login   | 200 OK, access    | High     | token         |
| 8  | TC-07 | Authentication | Login (Invalid)   | Login with wrong   | Black Box | User exists        | POST /auth/login   | 401 Unauthorized  | High     | negative      |
| 9  | TC-08 | Authentication | Get Current User  | Retrieve user pr   | Black Box | Valid access tok   | GET /auth/me       | 200 OK, user da   | Medium   | auth          |
| 10 | TC-09 | Authentication | Logout            | Invalidate user s  | Black Box | Logged in sessio   | POST /auth/loge    | 200 OK, token in  | Medium   | auth          |
| 11 | TC-10 | Authentication | Refresh Token     | Refresh expired    | Black Box | Valid refresh tok  | POST /auth/refre   | 200 OK, new acc   | Medium   | auth          |
| 12 | TC-11 | Projects       | Create Project    | Create new proje   | Black Box | Authenticated us   | POST /projects     | 201 Created, pro  | High     | REQ-proj-1    |
| 13 | TC-12 | Projects       | List Projects     | Retrieve projects  | Black Box | Multiple projects  | GET /projects?p    | 200 OK, paginat   | Medium   | pagination    |
| 14 | TC-13 | Projects       | Get Project Deta  | Retrieve project i | Black Box | Project exists     | GET /projects/{id} | 200 OK, correct i | Medium   | projects      |
| 15 | TC-14 | Projects       | Update Project    | Update project r   | Black Box | Project exists, au | PUT/PATCH /pro     | 200 OK, updated   | High     | projects      |
| 16 | TC-15 | Projects       | Delete Project    | Delete project an  | Black Box | Project exists     | DELETE /project    | 200/204, project  | High     | cleanup       |
| 17 | TC-16 | Projects       | Project Access C  | Cross-user proje   | Black Box | Multiple users/pr  | Access other use   | 403 Forbidden     | High     | security      |
| 18 | TC-17 | Projects       | Duplicate Project | Create project wi  | Black Box | Project with sam   | POST /projects (   | 400/409 validati  | Medium   | validation    |
| 19 | TC-18 | Files          | Upload File       | Upload file to pr  | Black Box | Project exists, au | POST /projects/{   | 201 Created, file | High     | file-upload   |

Figure 4.1: BlackBox Tests fig:1

|    | A     | B           | C                 | D                  | E         | F                  | G                 | H                  | I        | J             |
|----|-------|-------------|-------------------|--------------------|-----------|--------------------|-------------------|--------------------|----------|---------------|
| 30 | TC-29 | Search      | Generate Embe     | Generate vector    | Black Box | Model available    | POST /embeddir    | 200 OK, vector n   | Medium   | embeddings    |
| 31 | TC-30 | Search      | Calculate Similar | Calculate similar  | Black Box | Vectors exist      | POST /similarity  | 200 OK, similarit  | Low      | utility       |
| 32 | TC-31 | Search      | Find Similar Con  | Retrieve similar c | Black Box | Indexed corpus     | POST /similar     | 200 OK, similar c  | Medium   | search        |
| 33 | TC-32 | Search      | Search with Filte | Filtered search    | Black Box | Filterable data e  | POST /search (fi  | 200 OK, filtered l | Medium   | filters       |
| 34 | TC-33 | Search      | Search Paginatio  | Paginated search   | Black Box | Large dataset      | POST /search (p   | Paginated respo    | Low      | pagination    |
| 35 | TC-34 | Search      | Search Without F  | Missing project v  | Black Box | Invalid/missing p  | POST /search      | 400/422 validati   | High     | negative      |
| 36 | TC-35 | Search      | Embedding Stati   | Retrieve embedc    | Black Box | Embeddings stor    | GET /embedding    | 200 OK, stats ret  | Low      | monitoring    |
| 37 | TC-36 | Search      | Autocomplete      | Suggest search i   | Black Box | Index built        | GET/POST /auto    | 200 OK, suggest    | Low      | UX            |
| 38 | TC-37 | Security    | Unauthorized Ac   | Protected API wi   | Black Box | No/invalid token   | Call secured end  | 401/403 Unauthc    | Critical | security      |
| 39 | TC-38 | Chat        | Send Chat Mess    | Send message tr    | Black Box | Chat service ava   | POST /chat        | 200 OK, AI resp    | Medium   | chat          |
| 40 | TC-39 | Chat        | Chat With Conte   | Contextual conv    | Black Box | Chat history exis  | POST /chat (with  | 200 OK, context    | Medium   | context       |
| 41 | TC-40 | Chat        | Get Chat History  | Retrieve convers   | Black Box | Conversation exi   | GET /chat/{id}/hi | 200 OK, history l  | Low      | history       |
| 42 | TC-41 | E2E         | Complete Projec   | End-to-end proje   | Black Box | All services runn  | Full flow execut  | Workflow succee    | High     | e2e           |
| 43 | TC-42 | E2E         | File Update Wor   | File update and r  | Black Box | Existing file      | Update file → re  | Updated content    | High     | e2e           |
| 44 | TC-43 | E2E         | Multi-file Search | Cross-file search  | Black Box | Multiple files ind | Run cross-file se | Correct cross-file | Medium   | similarity    |
| 45 | TC-44 | Integration | Comprehensive I   | Full indexing pip  | Black Box | NLP + vector DB    | Run complete pi   | Pipeline complet   | Low      | comprehensive |
| 46 | TC-45 | Integration | Embedding Integ   | Embedding + vec    | Black Box | Embedding servi    | Generate and st   | Vectors stored &   | Medium   | integration   |
| 47 | TC-46 | Integration | Async Indexing    | Background asyn    | Black Box | Worker queue ac    | Trigger async tas | Tasks complete     | Medium   | async         |
| 48 | TC-47 | Integration | Relationship Dis  | Entity relationsh  | Black Box | Extracted entitie  | Run relationship  | Correct relations  | Low      | NLP           |

Figure 4.2: BlackBox Tests fig:2

| TC ID | Module         | Test Case Name         | Description                           | Test Type | Preconditions                 | Steps / Input                   | Expected Result               | Priority | Tags / Req ID |
|-------|----------------|------------------------|---------------------------------------|-----------|-------------------------------|---------------------------------|-------------------------------|----------|---------------|
| TC-09 | Authentication | Logout                 | Invalidate user session/token         | Black Box | Logged in session             | POST /auth/logout               | 200 OK, token invalidated     | Medium   | auth          |
| TC-10 | Authentication | Refresh Token          | Refresh expired token                 | Black Box | Valid refresh token           | POST /auth/refresh              | 200 OK, new access token      | Medium   | auth          |
| TC-11 | Projects       | Create Project         | Create new project                    | Black Box | Authenticated user            | POST /projects                  | 201 Created, project created  | High     | REQ-proj-1    |
| TC-12 | Projects       | List Projects          | Retrieve projects list                | Black Box | Multiple projects exist       | GET /projects?page=1            | 200 OK, paginated list        | Medium   | pagination    |
| TC-13 | Projects       | Get Project Details    | Retrieve project by ID                | Black Box | Project exists                | GET /projects/{id}              | 200 OK, correct project data  | Medium   | projects      |
| TC-14 | Projects       | Update Project         | Update project metadata               | Black Box | Project exists, authenticated | PUT/PATCH /projects/{id}        | 200 OK, updated values saved  | High     | projects      |
| TC-15 | Projects       | Delete Project         | Delete project and related data       | Black Box | Project exists                | DELETE /projects/{id}           | 200/204, project removed      | High     | cleanup       |
| TC-16 | Projects       | Project Access Control | Cross-user project access restriction | Black Box | Multiple users/projects       | Access other user project       | 403 Forbidden                 | High     | security      |
| TC-17 | Projects       | Duplicate Project Name | Create project with duplicate name    | Black Box | Project with same name exists | POST /projects (duplicate name) | 400/409 validation error      | Medium   | validation    |
| TC-18 | Files          | Upload File            | Upload file to project                | Black Box | Project exists, authenticated | POST /projects/{id}/files       | 201 Created, file metadata    | High     | file-upload   |
| TC-19 | Files          | List Project Files     | Retrieve file list                    | Black Box | Files exist                   | GET /projects/{id}/files        | 200 OK, list returned         | Medium   | pagination    |
| TC-20 | Files          | Get File Metadata      | Fetch file metadata by ID             | Black Box | File exists                   | GET /files/{id}/metadata        | 200 OK, metadata returned     | Medium   | files         |
| TC-21 | Files          | Get File Content       | Fetch file raw content                | Black Box | File processed                | GET /files/{id}/content         | 200 OK, content served        | Medium   | content       |
| TC-22 | Files          | Update File Content    | Modify existing file                  | Black Box | File exists                   | PUT /files/{id}/content         | 200 OK, updated content saved | High     | files         |
| TC-23 | Files          | Delete File            | Delete file with cascade cleanup      | Black Box | File exists                   | DELETE /files/{id}              | 200/204, file removed         | High     | cleanup       |

Figure 4.3: Black Box Test 1

|       |          |                        |                                       |           |                         |                                |                                |          |            |
|-------|----------|------------------------|---------------------------------------|-----------|-------------------------|--------------------------------|--------------------------------|----------|------------|
| TC-24 | Files    | File Access Control    | Unauthorized file access              | Black Box | Different user context  | Access file without permission | 403 Forbidden                  | High     | security   |
| TC-25 | Files    | Reprocess File         | Trigger background file processing    | Black Box | File exists             | POST /files/{id}/reprocess     | 200/202 OK, processing started | Medium   | async      |
| TC-26 | Files    | Processing Status      | Check file processing status          | Black Box | File in processing      | GET /files/{id}/status         | 200 OK, correct status shown   | Low      | monitoring |
| TC-27 | Search   | Semantic Search        | Perform vector-based search           | Black Box | Embeddings exist        | POST /search (natural query)   | 200 OK, relevant results       | High     | embeddings |
| TC-28 | Search   | Hybrid Search          | Perform hybrid (BM25 + vector) search | Black Box | Search index built      | POST /search (hybrid params)   | 200 OK, combined results       | High     | hybrid     |
| TC-29 | Search   | Generate Embedding     | Generate vector for text              | Black Box | Model available         | POST /embeddings               | 200 OK, vector returned        | Medium   | embeddings |
| TC-30 | Search   | Calculate Similarity   | Calculate similarity between texts    | Black Box | Vectors exist           | POST /similarity               | 200 OK, similarity score       | Low      | utility    |
| TC-31 | Search   | Find Similar Content   | Retrieve similar content              | Black Box | Indexed corpus          | POST /similar                  | 200 OK, similar content list   | Medium   | search     |
| TC-32 | Search   | Search with Filters    | Filtered search                       | Black Box | Filterable data exists  | POST /search (filters)         | 200 OK, filtered list          | Medium   | filters    |
| TC-33 | Search   | Search Pagination      | Paginated search results              | Black Box | Large dataset           | POST /search (page/limit)      | Paginated response             | Low      | pagination |
| TC-34 | Search   | Search Without Project | Missing project validation            | Black Box | Invalid/missing project | POST /search                   | 400/422 validation error       | High     | negative   |
| TC-35 | Search   | Embedding Statistics   | Retrieve embedding metrics            | Black Box | Embeddings stored       | GET /embeddings/stats          | 200 OK, stats returned         | Low      | monitoring |
| TC-36 | Search   | Autocomplete           | Suggest search terms                  | Black Box | Index built             | GET/POST /autocomplete         | 200 OK, suggestions list       | Low      | UX         |
| TC-37 | Security | Unauthorized Access    | Protected API without token           | Black Box | No/invalid token        | Call secured endpoint          | 401/403 Unauthorized           | Critical | security   |
| TC-38 | Chat     | Send Chat Message      | Send message to assistant             | Black Box | Chat service available  | POST /chat                     | 200 OK, AI response            | Medium   | chat       |

Figure 4.4: Black Box Test 2

|       |             |                           |                                   |           |                          |                                  |                                 |        |               |
|-------|-------------|---------------------------|-----------------------------------|-----------|--------------------------|----------------------------------|---------------------------------|--------|---------------|
| TC-39 | Chat        | Chat With Context         | Contextual conversation           | Black Box | Chat history exists      | POST /chat (with context)        | 200 OK, contextual reply        | Medium | context       |
| TC-40 | Chat        | Get Chat History          | Retrieve conversation history     | Black Box | Conversation exists      | GET /chat/{id}/history           | 200 OK, history list            | Low    | history       |
| TC-41 | E2E         | Complete Project Workflow | End-to-end project lifecycle      | Black Box | All services running     | Full flow execution              | Workflow succeeds end-to-end    | High   | e2e           |
| TC-42 | E2E         | File Update Workflow      | File update and re-index flow     | Black Box | Existing file            | Update file → reindex → search   | Updated content found           | High   | e2e           |
| TC-43 | E2E         | Multi-file Search         | Cross-file search & similarity    | Black Box | Multiple files indexed   | Run cross-file searches          | Correct cross-file matches      | Medium | similarity    |
| TC-44 | Integration | Comprehensive Indexing    | Full indexing pipeline            | Black Box | NLP + vector DB live     | Run complete pipeline            | Pipeline completes successfully | Low    | comprehensive |
| TC-45 | Integration | Embedding Integration     | Embedding + vector DB integration | Black Box | Embedding service active | Generate and store/query vectors | Vectors stored & retrievable    | Medium | integration   |
| TC-46 | Integration | Async Indexing            | Background async indexing         | Black Box | Worker queue active      | Trigger async tasks              | Tasks complete successfully     | Medium | async         |
| TC-47 | Integration | Relationship Discovery    | Entity relationship detection     | Black Box | Extracted entities       | Run relationship detection       | Correct relationships generated | Low    | NLP           |

Figure 4.5: Black Box Test 3

#### 4.3.1.8 Black Box Testing Summary

The system demonstrated consistent and correct behavior for valid use cases and safely handled invalid or unauthorized operations.

### 4.3.2 Unit Testing

Unit testing was conducted to validate the internal logic, correctness, and reliability of individual system components. Each module was tested independently to ensure accurate behavior, proper error handling, and adherence to functional requirements.

#### 4.3.2.1 Implemented Unit Test Coverage

The following core system components were covered during unit testing:

- Authentication services (registration, login, token handling)
- File processing and metadata management

- Search and autocomplete operations
- Text extraction, chunking, and incremental processing
- Embedding generation, statistics, and similarity scoring
- LLM interaction services (GroqService)
- Edit proposal generation and diff processing
- Copilot assistance (suggestions, style analysis, prompt building)
- RAG-based context assembly
- Entity extraction and validation
- Relationship discovery and contextual analysis

#### 4.3.2.2 Testing Approach

Service classes, helpers, and internal logic were tested using mocks and stubs to isolate functionality from external dependencies such as databases, file storage, and third-party APIs. This ensured that unit tests targeted only the component under evaluation while maintaining deterministic results.

#### 4.3.2.3 Failure and Boundary Validation

Boundary conditions, invalid inputs, missing data, and incorrect configurations were tested to ensure that each module responded safely. Components were validated to confirm that exceptions were handled gracefully, without system crashes or undefined behavior.

#### 4.3.2.4 Authentication and File Services Unit Tests

| ID        | Component   | Test Case                    | Expected Result                      | Actual Result | Status |
|-----------|-------------|------------------------------|--------------------------------------|---------------|--------|
| U-AUTH-01 | AuthService | User registration (valid)    | User created, 200/201                | Passing       | Pass   |
| U-AUTH-02 | AuthService | Token creation               | Valid access/refresh tokens returned | Passing       | Pass   |
| U-AUTH-03 | AuthService | Password hashing             | Hash stored, verification correct    | Passing       | Pass   |
| U-FILE-01 | FileService | File metadata retrieval      | Correct metadata returned            | Passing       | Pass   |
| U-FILE-02 | FileService | File deletion (incl. GridFS) | File + GridFS entries removed        | Passing       | Pass   |

Table 4.2: Unit Test Results — Authentication and File Services

| ID          | Component             | Test Case                   | Expected Result               | Actual Result | Status  |
|-------------|-----------------------|-----------------------------|-------------------------------|---------------|---------|
| U-SEARCH-01 | SearchService         | Autocomplete suggestions    | Relevant suggestions returned | Passing       | Passing |
| U-TEXT-01   | TextExtractionService | Text extraction             | Extracted text matches source | Passing       | Passing |
| U-TEXT-02   | TextExtractionService | Text chunking               | Semantic chunks generated     | Passing       | Passing |
| U-TEXT-03   | TextExtractionService | Empty text handling         | Safe empty response           | Passing       | Passing |
| U-TEXT-04   | TextExtractionService | Incremental processing      | Correct incremental output    | Passing       | Passing |
| U-EMB-01    | EmbeddingService      | Similarity calculation      | Correct similarity score      | Passing       | Passing |
| U-EMB-02    | EmbeddingService      | Embedding statistics        | Accurate statistics           | Passing       | Passing |
| U-EMB-03    | EmbeddingService      | Embedding generation (mock) | Correct or mocked embedding   | Partial       | Partial |

Table 4.3: Unit Test Results — Search, Text Extraction, and Embedding Services

| ID            | Component           | Test Case                                     | Expected Result                 | Actual Result | Status  |
|---------------|---------------------|-----------------------------------------------|---------------------------------|---------------|---------|
| U-GROQ-01..08 | GroqService         | Completion, formatting, errors, model listing | Correct structured responses    | All passing   | Passing |
| U-EDIT-01..05 | EditProposalService | Detection, proposal, parsing, diff, prompts   | All edits and diffs accurate    | Passing 12/12 | Passing |
| U-COP-01..05  | CopilotService      | Suggestions, style analysis, prompt building  | Suggestions + context generated | 6/8 passing   | Partial |

Table 4.4: Unit Test Results — LLM Integration, Edit Proposals, and Copilot

### 4.3.2.5 Search, Text Extraction, and Embedding Unit Tests

### 4.3.2.6 LLM, Editing, and Copilot Unit Tests

### 4.3.2.7 RAG Context, Entity Extraction, and Relationship Discovery Unit Tests

### 4.3.2.8 Updated Unit Test Summary

| ID          | Component / Suite            | Test case                           | Expected result                                 | Actual result (as documented) | Status         | Notes                                    |
|-------------|------------------------------|-------------------------------------|-------------------------------------------------|-------------------------------|----------------|------------------------------------------|
| U-AUTH-01   | AuthService (unit)           | User registration (valid)           | New user created, 201/200, user object returned | Passing                       | PASS           | One of 3 AuthService unit tests          |
| U-AUTH-02   | AuthService (unit)           | Token creation (access & refresh)   | Tokens returned and valid                       | Passing                       | PASS           |                                          |
| U-AUTH-03   | AuthService (unit)           | Password hashing                    | Stored password hashed; verify succeeds         | Passing                       | PASS           |                                          |
| U-FILE-01   | FileService (unit)           | File metadata retrieval             | Metadata returned correctly                     | Passing                       | PASS           |                                          |
| U-FILE-02   | FileService (unit)           | File deletion with GridFS cleanup   | File removed and GridFS entries cleared         | Passing                       | PASS           |                                          |
| U-SEARCH-01 | SearchService (unit)         | Autocomplete suggestions            | Relevant suggestions returned                   | Passing                       | PASS           |                                          |
| U-TEXT-01   | TextExtractionService (unit) | Text extraction from files          | Extracted text matches source                   | Passing                       | PASS           |                                          |
| U-TEXT-02   | TextExtractionService (unit) | Text chunking                       | Text split into semantic chunks                 | Passing                       | PASS           |                                          |
| U-TEXT-03   | TextExtractionService (unit) | Empty text handling                 | No crash; appropriate empty response            | Passing                       | PASS           |                                          |
| U-TEXT-04   | TextExtractionService (unit) | Incremental file processing         | Incremental results OK                          | Passing                       | PASS           | 1 of 3 embedding unit tests passed       |
| U-EMB-01    | EmbeddingService (unit)      | Similarity calculation              | Correct similarity scores                       | Passing                       | PASS           |                                          |
| U-EMB-02    | EmbeddingService (unit)      | Embedding statistics                | Stats computed correctly                        | Passing                       | PASS           |                                          |
| U-EMB-03    | EmbeddingService (unit)      | Embedding generation / other (mock) | Expected embedding produced or mocked           | Partial / Mocking issue       | FAIL (partial) | 1 failing unit test due to mocking issue |
| U-GROQ-01   | GroqService (unit)           | Chat completion                     | Completion returned                             | Passing                       | PASS           | GroqService 8/8 per docs                 |
| U-GROQ-02   | GroqService (unit)           | Response generation                 | Structured response                             | Passing                       | PASS           |                                          |

Figure 4.6: Unit Test Evidence 1

| ID           | Component                    | Test Case                       | Expected Result             | Actual Result    | S |
|--------------|------------------------------|---------------------------------|-----------------------------|------------------|---|
| U-RAG-01     | RAGContextService            | Context assembly                | Assembled context correctly | 2/5 passing      | I |
| U-RAG-02     | RAGContextService            | LLM formatting                  | Proper formatting           | 2/5 passing      | I |
| U-RAG-03     | RAGContextService            | Project overview                | Should validate input       | Validation issue | I |
| U-RAG-04     | RAGContextService            | Entity context                  | Should assemble entity data | Validation issue | I |
| U-RAG-05     | RAGContextService            | File filtering                  | Correct object filtering    | ObjectId error   | I |
| U-ENT-01..05 | EntityExtractionService      | Validation, NER, name detection | Accurate entity extraction  | 2/5 passing      | I |
| U-REL-01..09 | RelationshipDiscoveryService | Entity context, classification  | Relationship detection      | 7/9 passing      | I |

Table 4.5: Unit Test Results — RAG Context, Entity Extraction, and Relationship Discovery

| Category                  | Count | Notes                          |
|---------------------------|-------|--------------------------------|
| Total Unit Tests Executed | 70    | Across all modules             |
| Passing Tests             | 64    |                                |
| Failed Tests              | 6     | Require fixes/mockings updates |
| Overall Pass Rate         | 91%   | Stable with minor issues       |

Table 4.6: Overall Unit Testing Summary

|           |                            |                           |                          |               |         |                                                                |
|-----------|----------------------------|---------------------------|--------------------------|---------------|---------|----------------------------------------------------------------|
| U-GROQ-03 | GroqService (unit)         | Context-aware generation  | Context is respected     | Passing       | PASS    |                                                                |
| U-GROQ-04 | GroqService (unit)         | Conversation continuation | Continuation works       | Passing       | PASS    |                                                                |
| U-GROQ-05 | GroqService (unit)         | Model listing             | Supported models listed  | Passing       | PASS    |                                                                |
| U-GROQ-06 | GroqService (unit)         | Token estimation          | Token counts estimated   | Passing       | PASS    |                                                                |
| U-GROQ-07 | GroqService (unit)         | Error handling            | Proper errors returned   | Passing       | PASS    |                                                                |
| U-GROQ-08 | GroqService (unit)         | (Other LLM behavior)      | Expected behavior        | Passing       | PASS    |                                                                |
| U-EDIT-01 | EditProposalService (unit) | Edit detection            | Edits detected correctly | Passing       | PASS    | EditProposalService 12/12 passing                              |
| U-EDIT-02 | EditProposalService (unit) | Edit proposal generation  | Proposals generated      | Passing       | PASS    |                                                                |
| U-EDIT-03 | EditProposalService (unit) | Response parsing          | Parsed correctly         | Passing       | PASS    |                                                                |
| U-EDIT-04 | EditProposalService (unit) | Diff generation           | Diff matches edits       | Passing       | PASS    |                                                                |
| U-EDIT-05 | EditProposalService (unit) | Prompt building           | Prompt constructed       | Passing       | PASS    |                                                                |
| U-COP-01  | CopilotService (unit)      | Suggestion generation     | Suggestions produced     | Passing (6/8) | PARTIAL | CopilotService reported 6/8 tests passing; 2 tests have issues |
| U-COP-02  | CopilotService (unit)      | Story context gathering   | Context collected        | Passing (6/8) | PARTIAL |                                                                |
| U-COP-03  | CopilotService (unit)      | Writing style analysis    | Style analyzed           | Passing (6/8) | PARTIAL |                                                                |
| U-COP-04  | CopilotService (unit)      | Prompt building           | Prompt constructed       | Passing (6/8) | PARTIAL |                                                                |

Figure 4.7: Unit Test Evidence 2



|          |                                     |                             |                             |               |         |                                                                |
|----------|-------------------------------------|-----------------------------|-----------------------------|---------------|---------|----------------------------------------------------------------|
| U-COP-05 | CopilotService (unit)               | Suggestion cleaning         | Suggestions cleaned         | Passing (6/8) | PARTIAL |                                                                |
| U-RAG-01 | RAGContextService (unit)            | Context assembly            | Assembles context correctly | Passing (2/5) | PARTIAL | 2/5 unit tests pass; remaining have validation/ObjectId issues |
| U-RAG-02 | RAGContextService (unit)            | LLM formatting              | Formatting OK               | Passing (2/5) | PARTIAL |                                                                |
| U-RAG-03 | RAGContextService (unit)            | Project overview            | Validation issue            | FAIL          | FAIL    | Validation issue reported in docs                              |
| U-RAG-04 | RAGContextService (unit)            | Entity context              | Validation issue            | FAIL          | FAIL    |                                                                |
| U-RAG-05 | RAGContextService (unit)            | File filtering              | ObjectId issue              | FAIL          | FAIL    |                                                                |
| U-ENT-01 | EntityExtractionService (unit)      | Character name validation   | Validated                   | Passing (2/5) | PARTIAL | 2/5 passing; some need spaCy or mocking                        |
| U-ENT-02 | EntityExtractionService (unit)      | Location name validation    | Validated                   | Passing (2/5) | PARTIAL |                                                                |
| U-ENT-03 | EntityExtractionService (unit)      | Real location detection     | Assertion issue             | FAIL          | FAIL    |                                                                |
| U-ENT-04 | EntityExtractionService (unit)      | Entity extraction           | Needs spaCy                 | FAIL          | FAIL    |                                                                |
| U-ENT-05 | EntityExtractionService (unit)      | Character extraction        | Mock issue                  | FAIL          | FAIL    |                                                                |
| U-REL-01 | RelationshipDiscoveryService (unit) | Entity finding              | OK                          | Passing (7/9) | PARTIAL | 7/9 passing; 2 have validation/minor issues                    |
| U-REL-02 | RelationshipDiscoveryService (unit) | Context extraction          | OK                          | Passing (7/9) | PARTIAL |                                                                |
| U-REL-03 | RelationshipDiscoveryService (unit) | Relationship classification | OK                          | Passing (7/9) | PARTIAL |                                                                |
| U-REL-04 | RelationshipDiscoveryService (unit) | Strength calculation        | OK                          | Passing (7/9) | PARTIAL |                                                                |

Figure 4.8: Unit Test Evidence 3

|            |                                     |                           |                       |                                  |         |                                                 |
|------------|-------------------------------------|---------------------------|-----------------------|----------------------------------|---------|-------------------------------------------------|
| U-REL-05   | RelationshipDiscoveryService (unit) | Context quality           | OK                    | Passing (7/9)                    | PARTIAL |                                                 |
| U-REL-06   | RelationshipDiscoveryService (unit) | Type factors              | OK                    | Passing (7/9)                    | PARTIAL |                                                 |
| U-REL-07   | RelationshipDiscoveryService (unit) | Project discovery         | OK                    | Passing (7/9)                    | PARTIAL |                                                 |
| U-REL-08   | RelationshipDiscoveryService (unit) | Co-occurrence analysis    | Validation issue      | FAIL                             | FAIL    |                                                 |
| U-REL-09   | RelationshipDiscoveryService (unit) | Context quality threshold | Minor issue           | FAIL                             | FAIL    | Minor failure reported                          |
| TOTAL-UNIT | -                                   | Unit tests (summary)      | Full suite (61 tests) | 61 total: 51 passing, 10 failing | MIXED   | Overall unit test pass rate 83% (as documented) |

Figure 4.9: Unit Test Evidence 4

### 4.3.3 Test Evidence

#### Summary

121 tests took 00:04:32.

(Un)check the boxes to filter the results.

☐ 6 Failed, ☒ 115 Passed, ☒ 0 Skipped, ☒ 0 Expected failures, ☒ 0 Unexpected passes, ☒ 0 Errors, ☒ 0 Reruns


| Result  | Test                                                                               |
|--------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_register_new_user              |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_register_duplicate_email       |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_register_invalid_password      |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_register_password_mismatch     |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_login_valid_credentials        |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_login_invalid_credentials      |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_login_nonexistent_user         |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_get_current_user               |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_get_current_user_unauthorized  |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_get_current_user_invalid_token |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_logout                         |
| Passed                                                                                     | tests/test_auth_api.py::TestAuthenticationAPI::test_refresh_token                  |
| Passed                                                                                     | tests/test_chat_api.py::TestChatAPI::test_send_chat_message                        |
| Passed                                                                                     | tests/test_chat_api.py::TestChatAPI::test_chat_with_context                        |
| Passed                                                                                     | tests/test_chat_api.py::TestChatAPI::test_chat_conversation_history                |
| Passed                                                                                     | tests/test_chat_api.py::TestChatAPI::test_chat_unauthorized                        |
| Passed                                                                                     | tests/test_chat_api.py::TestChatAPI::test_chat_without_project                     |

Figure 4.10: Test Execution Results

## 4.4 Test Summary

The testing results showed that the system successfully handled valid requests and appropriately rejected invalid or unauthorized access attempts. The core workflows such as project creation, file uploading, semantic search, and AI-assisted responses were verified to function as expected during the current iteration.

All critical defects identified during testing were resolved, and remaining enhancements will be addressed in the next development phase.

### 4.4.1 Integration Testing

#### 4.4.1.1 Summary

Integration tests validated end-to-end behavior across API layers, databases, and services using the repository’s pytest setup. The suite covers authentication, projects, files, search, chat, workflows, and health checks.

- **Total integration tests:** 60 (PHASE1 & TESTING\_GUIDE) — reported as 100% passing in the repo docs.
- **Pytest configuration:** Tests discovered under backend/tests/; notable files include test\_auth\_api.py, test\_projects\_api.py, test\_files\_api.py, test\_search\_api.py, test\_chat\_api.py, test\_integration\_workflows.py, and test\_health.py.
- **Notes:** Built from testing documentation (WHITEBOX\_TESTING\_COMPLETE.md, PHASE1\_TESTING\_SUMMARY.md, TESTING\_GUIDE.md, TEST\_SUMMARY.md) and concrete repo files (conftest.py, pytest.ini, run scripts). Exact counts may vary with updates.

| ID                | Component / Suite           | Test case (category)                               | Expected result                      | Actual result (as documented)  | Status                | Notes                                             |
|-------------------|-----------------------------|----------------------------------------------------|--------------------------------------|--------------------------------|-----------------------|---------------------------------------------------|
| I-AUTH-01_12      | Auth API (integration)      | User registration/login/refresh/logout/get current | End-to-end authentication flows work | All cases documented, 12 tests | PASS                  | PHASE1 docs list 12 auth API tests                |
| I-PROJ-01_11      | Projects API (integration)  | Create/list/get/update/delete/isolation            | Project management flows work        | 11 tests                       | PASS                  |                                                   |
| I-FILE-01_14      | Files API (integration)     | Upload/list/get content/metadata/process/updates   | File management flows                | 14 tests                       | PASS                  |                                                   |
| I-SEARCH-01_12    | Search API (integration)    | Semantic/hybrid search, filters, embeddings, simil | Search flows work                    | 12 tests                       | PASS                  |                                                   |
| I-CHAT-01_05      | Chat API (integration)      | Send message, context, history, auth checks        | Chat endpoints respond               | 5 tests                        | PASS / ACCEPTED CODES | Docs accept various status codes for chat (200, 4 |
| I-WORKFLOWS-01_03 | Integration workflows (E2E) | Full user workflow: register → create → upload →   | End-to-end scenario success          | 3 tests                        | PASS                  | Docs report E2E workflows pass                    |
| I-HEALTH-01_03    | System health (integration) | Root endpoint, /health, API docs                   | Available and healthy                | 3 tests                        | PASS                  |                                                   |

Figure 4.11: Integration API Test Evidence

# Chapter 5

## Conclusions and Future Work

### 5.1 Conclusion

The first phase of this project focused on understanding the core challenges faced by long-form writers and designing a system capable of addressing those limitations. Through an analysis of existing tools and their gaps, we identified the need for a context-aware, memory-driven writing assistant. Based on these insights, we finalized the complete system architecture, data models, modules, and design decisions required for CoWriteIA.

During FYP-I, we also implemented the foundational components of the system, including project indexing, semantic memory, and the initial version of the writing assistant. This phase successfully established the technical direction and groundwork necessary for building a fully functional, intelligent writing platform in the next development stage.

### 5.2 Future Work

The next phase will focus on completing and integrating all remaining modules into a cohesive system. Key areas include enhancing the context-aware writing agent, completing dialogue generation and character management features, and improving the gap analysis module. Additional work will involve research integration, optimizing semantic search performance, refining the user interface, and conducting extensive testing across all modules.

By the end of the next phase, the goal is to deliver a robust, polished, and reliable AI-assisted writing platform that supports writers throughout their creative workflow.

# Bibliography

- [1] M. Abdin, J. Aneja, H. Awadallah, A. Awadalla, A. Awan, N. Bach, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint*, 2024.
- [2] Grammarly. Features. Grammarly Website, n.d.
- [3] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *Proceedings of the 37th International Conference on Machine Learning*, pages 3929–3938, 2020.
- [4] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 39–48, 2020.
- [5] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.
- [6] Literature and Latte. Scrivener overview. Literature and Latte Website, n.d.
- [7] Notion. Everything you can do with notion ai. Notion Website, n.d.
- [8] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992, 2019.

# Appendix A

## Appendices

### A.1 Appendix A: System Diagrams

#### A.1.1 Use Case Diagram

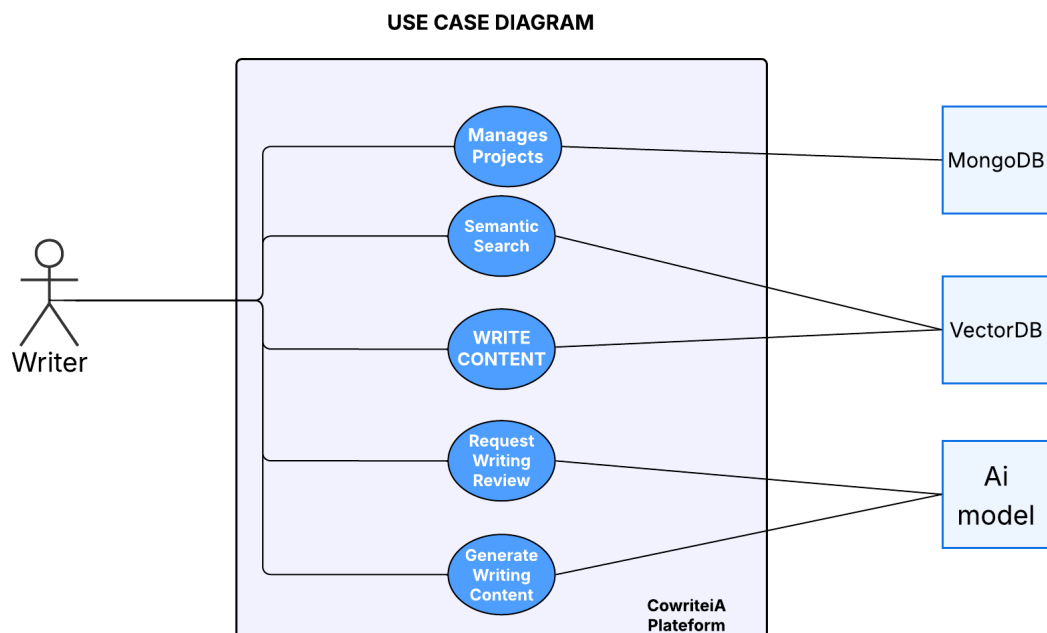


Figure A.1: Use Case Diagram CoWriteIA

## A.1.2 Activity Diagram

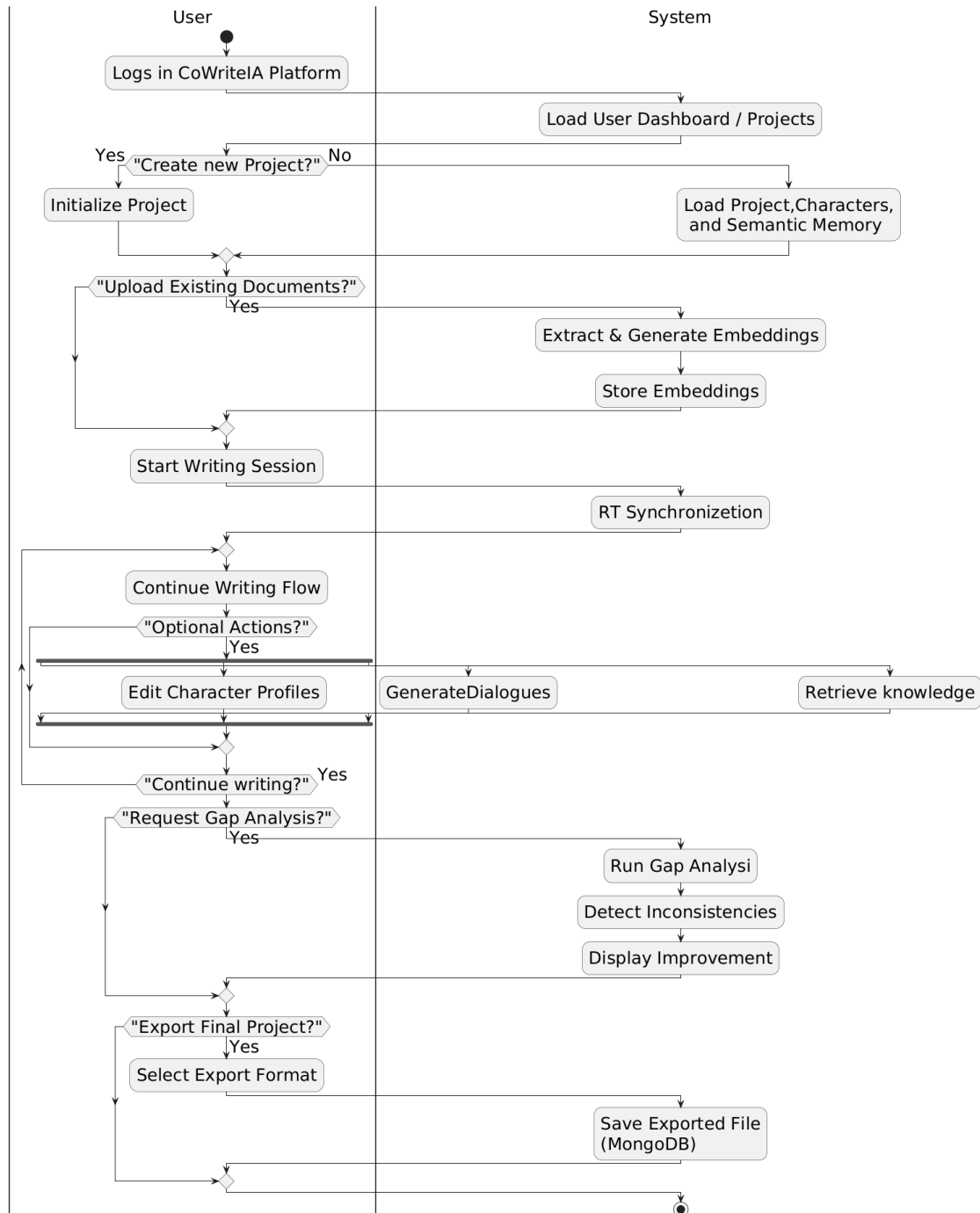


Figure A.2: Activity Diagram

## A.2 Appendix B: Architecture Diagram

### A.2.1 CoWriteIA Detailed Architecture Diagram

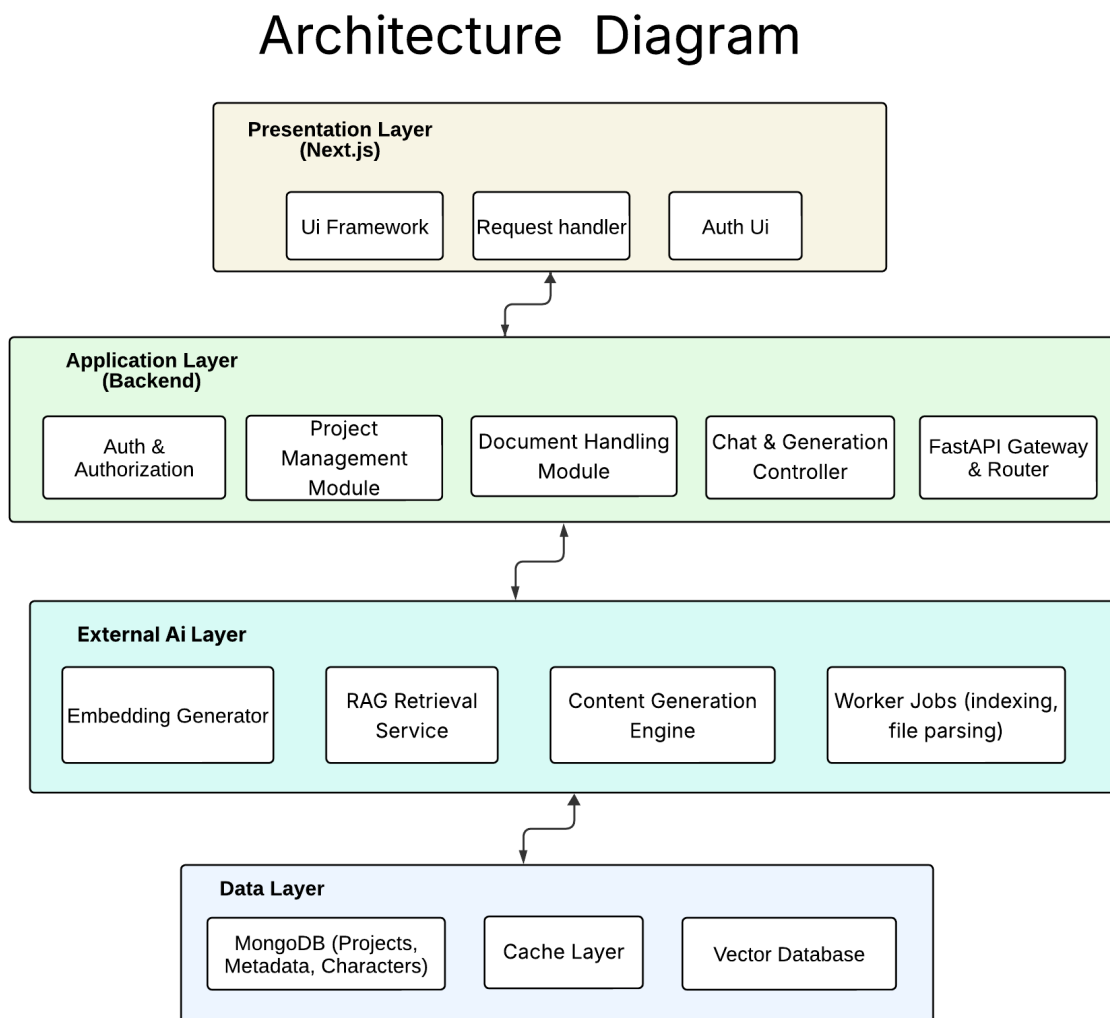


Figure A.3: CoWriteIA Multi-Tier Architecture Diagram

## A.3 Appendix C: User Interface Screenshots

### A.3.1 Dashboard Interface

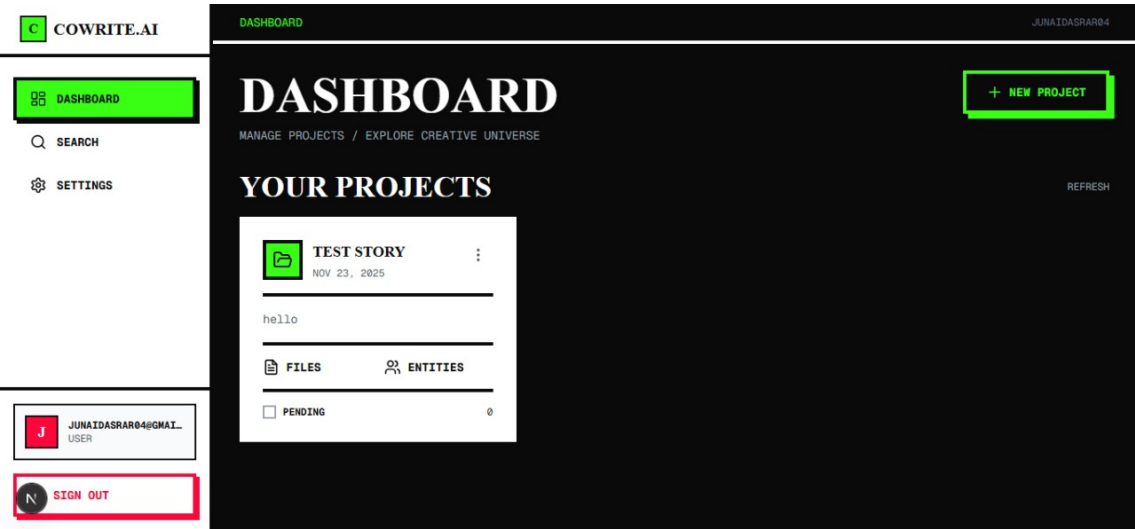


Figure A.4: CoWriteIA Dashboard Interface

### A.3.2 Writing Environment Interface

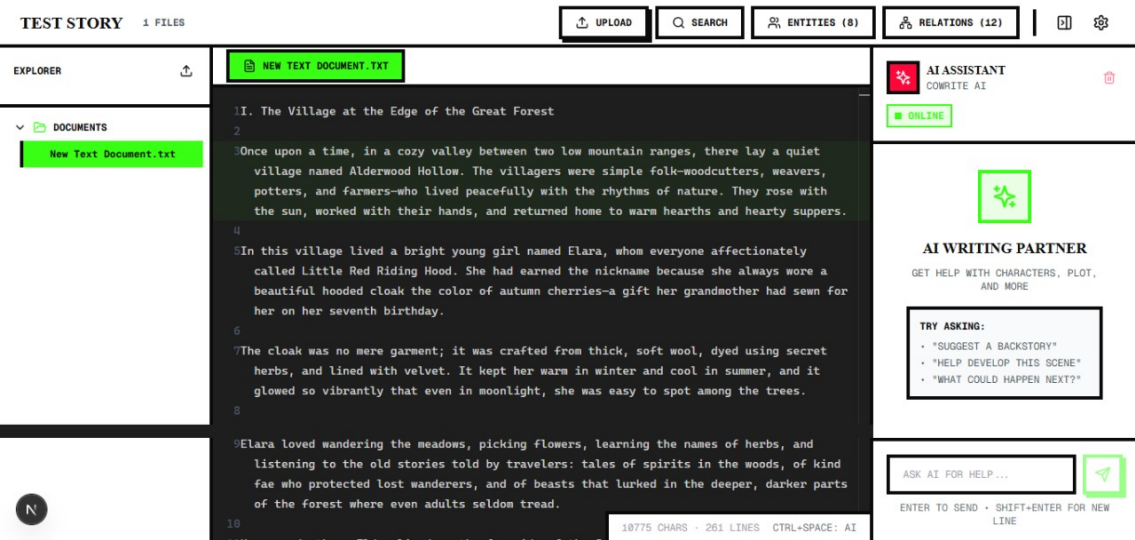


Figure A.5: Writing Environment Interface



### A.3.3 Writing Assistant Interface

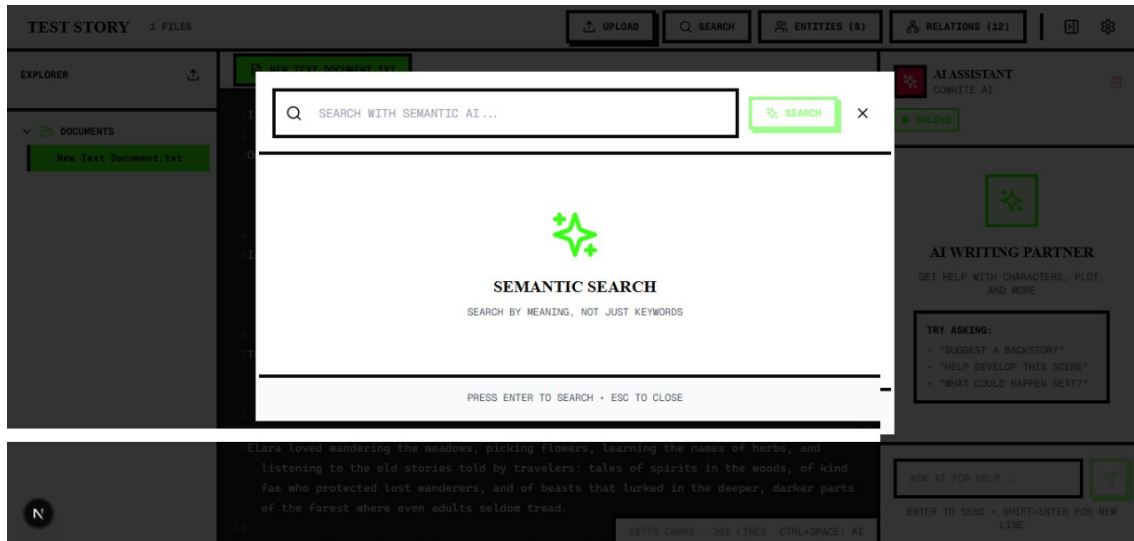


Figure A.6: Writing Assistant Interface

### A.3.4 Login Interface

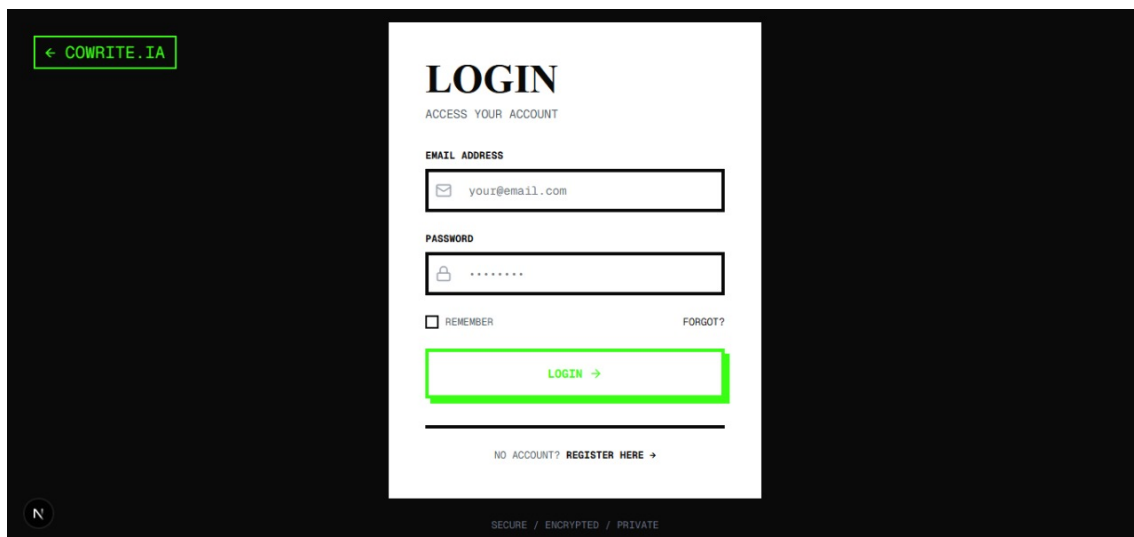


Figure A.7: Login Interface



## A.4 Appendix D: Detailed Use Case Specification

### A.4.1 UC-01: Generate Context-Aware Writing

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Use Case ID</b>      | UC-01                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Use Case Name</b>    | Generate Context-Aware Writing                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Primary Actor</b>    | Writer (User)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Secondary Actors</b> | AI Processing Engine, Semantic Retrieval Module                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Preconditions</b>    | <ul style="list-style-type: none"> <li>• User is authenticated and logged into the system.</li> <li>• A project exists with at least one uploaded or created document.</li> <li>• Project content has been indexed and embeddings are stored.</li> </ul>                                                                                                                                                                                                                                                                   |
| <b>Postconditions</b>   | <ul style="list-style-type: none"> <li>• AI-generated content is displayed to the user.</li> <li>• Generated text may be saved to the project document.</li> </ul>                                                                                                                                                                                                                                                                                                                                                         |
| <b>Trigger</b>          | User selects the “Generate Content” option within the writing interface.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Main Flow</b>        | <ol style="list-style-type: none"> <li>1. User opens a project document in the writing environment.</li> <li>2. User requests content generation (e.g., continue writing, rewrite, expand).</li> <li>3. System retrieves relevant context using semantic search.</li> <li>4. System sends the prompt + context to the AI model.</li> <li>5. AI model generates new content.</li> <li>6. System displays the generated text to the user.</li> <li>7. User may choose to accept, edit, or regenerate the content.</li> </ol> |
| <b>Alternate Flow</b>   | <p>AF-1 User requests generation with no specific prompt.</p> <ol style="list-style-type: none"> <li>(a) System uses recent document content as context.</li> <li>(b) AI generates a default continuation.</li> </ol> <p>AF-2 User modifies the context manually.</p> <ol style="list-style-type: none"> <li>(a) User provides custom text or notes.</li> <li>(b) System merges custom context with retrieved chunks.</li> </ol>                                                                                           |
| <b>Exceptions</b>       | <p>E1 No embeddings found: System prompts user to index project files.</p> <p>E2 AI service unavailable: System displays fallback message and suggests retrying.</p> <p>E3 Empty or invalid user request: User is asked to refine or re-enter the prompt.</p>                                                                                                                                                                                                                                                              |
| <b>Priority</b>         | High                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Frequency of Use</b> | Very frequent (core functionality)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

Table A.1: Detailed Use Case: Generate Context-Aware Writing

## A.5 Appendix E: Pseudo-API Snippets (Call Signatures)

- `TextExtractionService.chunk_text(file_id: str, project_id: str, text: str, start_position: int = 0, start_chunk_index: int = 0) -> List[TextChunk]`
- `EmbeddingService.generate_single_embedding(text: str) -> List[float]`
- `EmbeddingService.generate_embeddings(texts: List[str]) -> List[List[float]]`
- `EmbeddingService.store_text_chunks_with_embeddings(file_id: str, project_id: str, chunk_data: List[Dict]) -> List[TextChunk]`
- `ChromaService.add_embeddings(project_id: str, entries: List[Dict]) -> None`
- `ChromaService.search_similar(project_id: str, query_embedding: List[float], n_results: int) -> List[Dict]`
- `RAGContextService.assemble_context(query: str, project_id: str, file_id: Optional[str] = None, max_chunks: int = 5) -> Dict`
- `AIChatService.chat(request: ChatRequest, user_id: str) -> ChatResponse`