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

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 [**?**].

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 [**?**].

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 [**?**]. 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 [**?**].

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 [**?**].

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		✓	✓

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			✓
Testing	✓	✓	✓
Documentation (All Sections)	✓	✓	✓

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 / Event Response Table / Storyboarding

CoWriteIA is designed to support writers by providing AI-assisted tools such as project indexing, semantic retrieval, character management, context-aware writing, and dialogue generation. To understand system behavior, user interactions are described using use cases and event-response flows.

A use case illustrates how a user interacts with the system to complete a specific task. It includes the actor, the event that triggers the interaction, the main steps, preconditions, and the expected system responses. These use cases help clarify system behavior and guide the design of system features.

Given the nature of CoWriteIA, storyboarding can also help by visually demonstrating how a writer navigates the interface: uploading documents, searching for context, generating content, managing characters, or using the inline copilot. These visual sequences ensure the design stays consistent with expected functionality.

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, 3, 4]. 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.2).

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

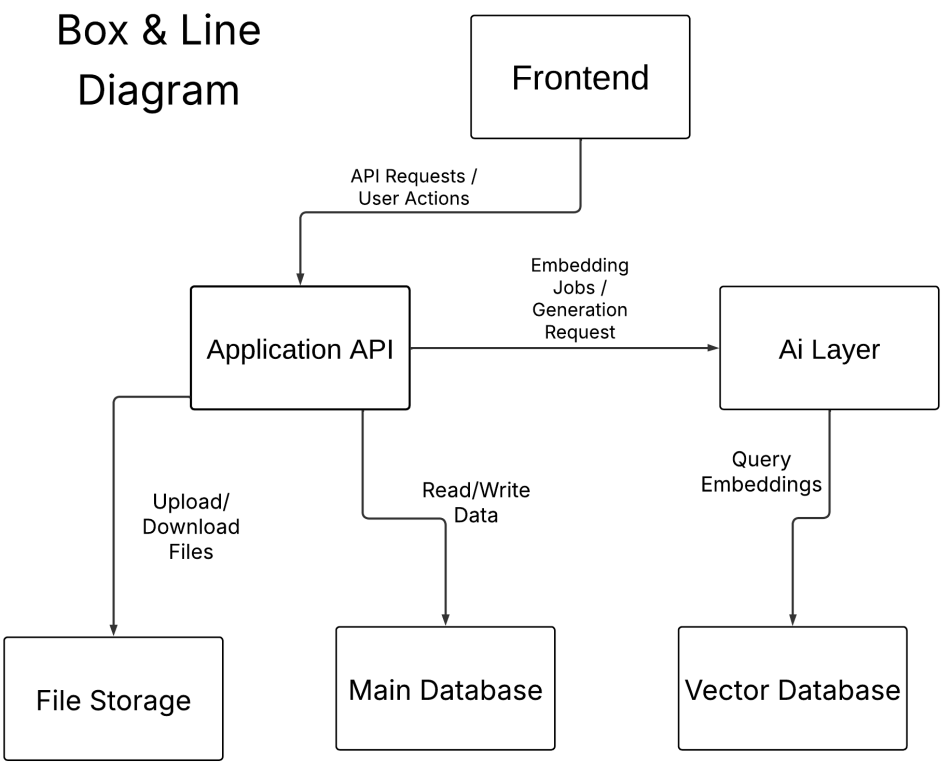

Figure 3.1: Box and Line Diagram showing major subsystems and their connections

- **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 metadata 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.

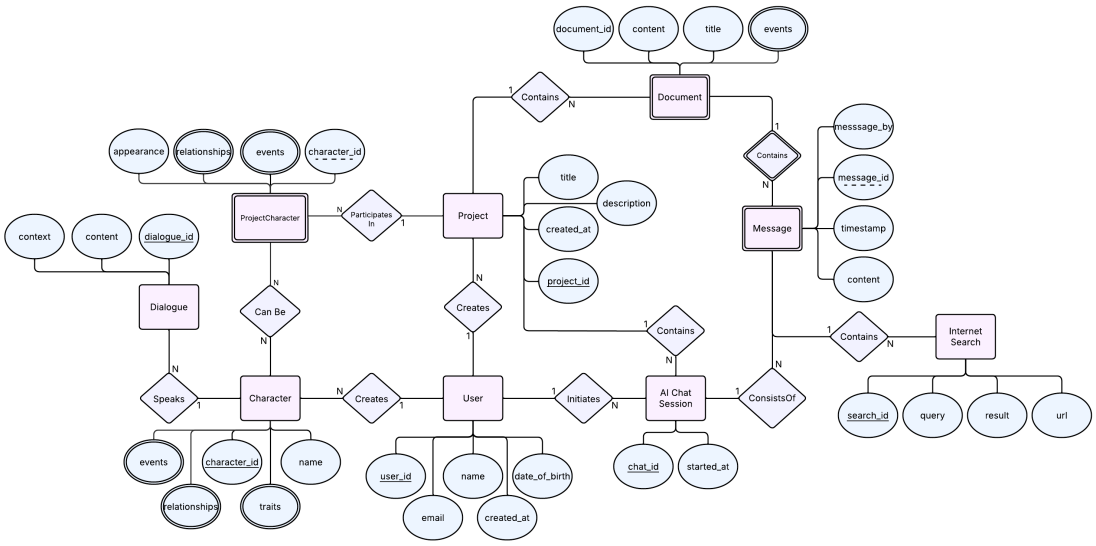

Figure 3.2: Entity–Relationship Diagram for CoWriteIA

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

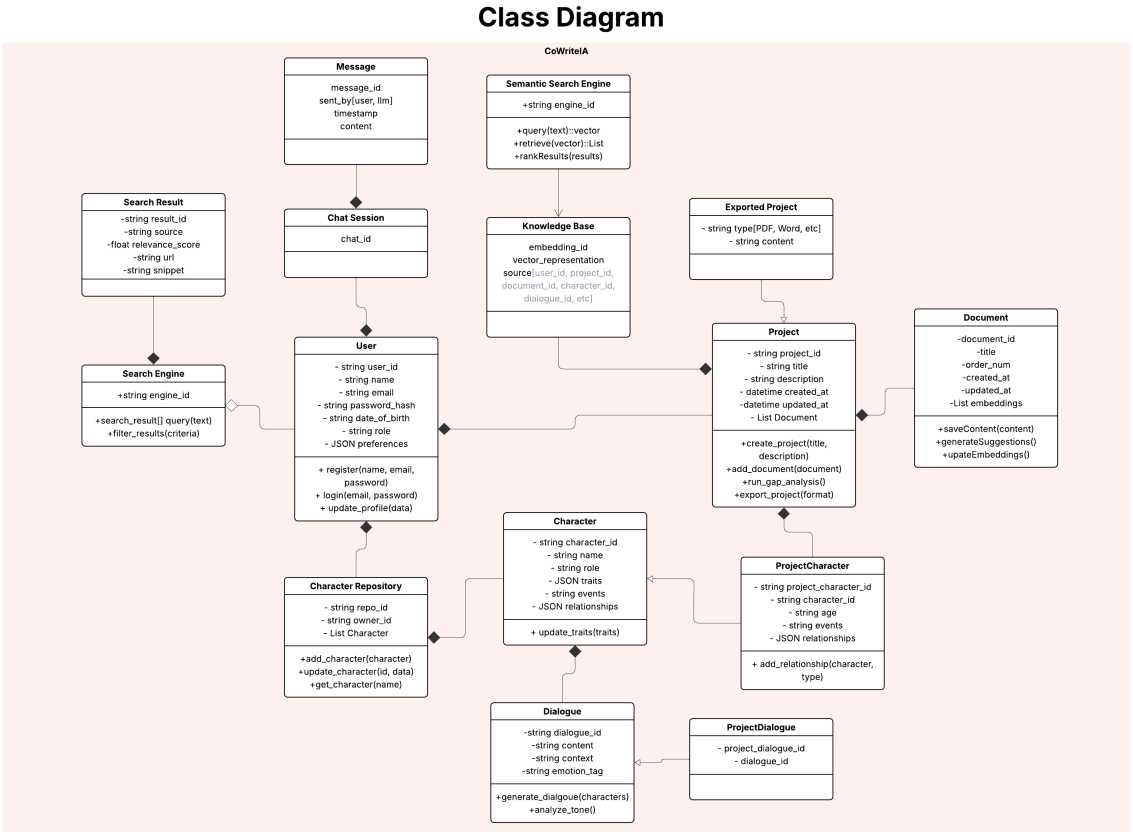

Figure 3.3: Class Diagram for CoWriteIA

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.

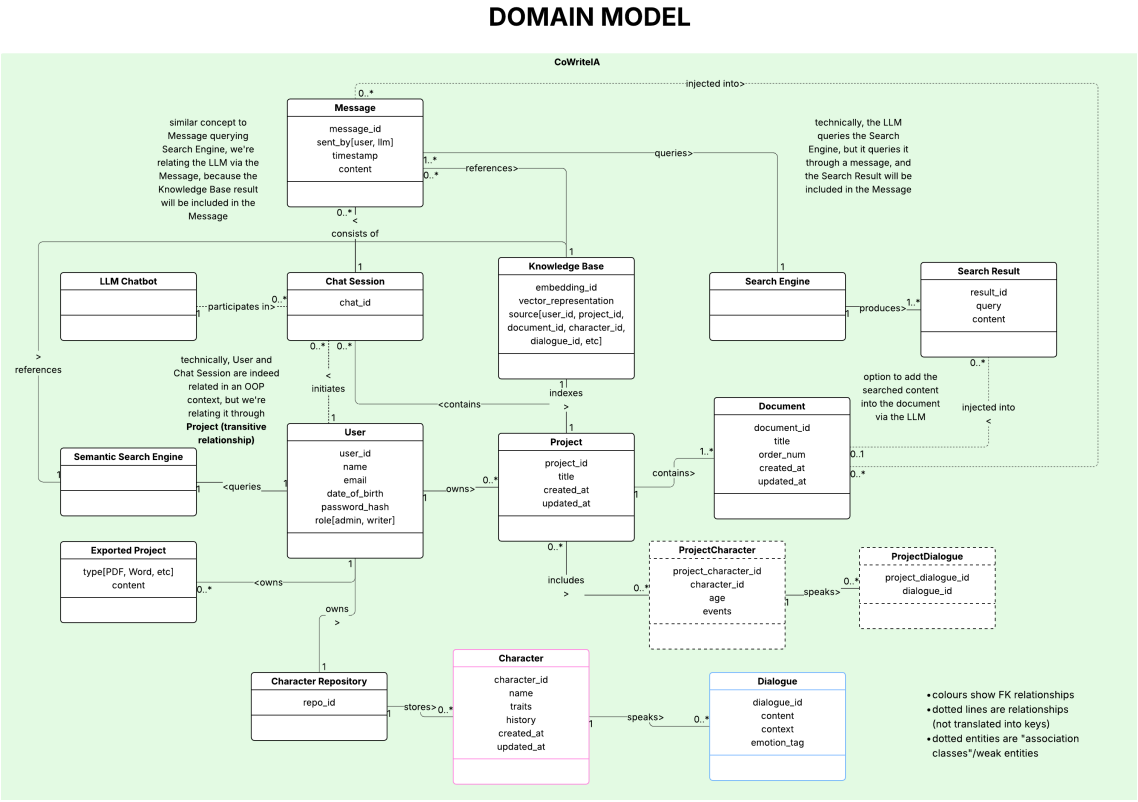

Figure 3.4: Domain Model Diagram for CoWriteIA

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.

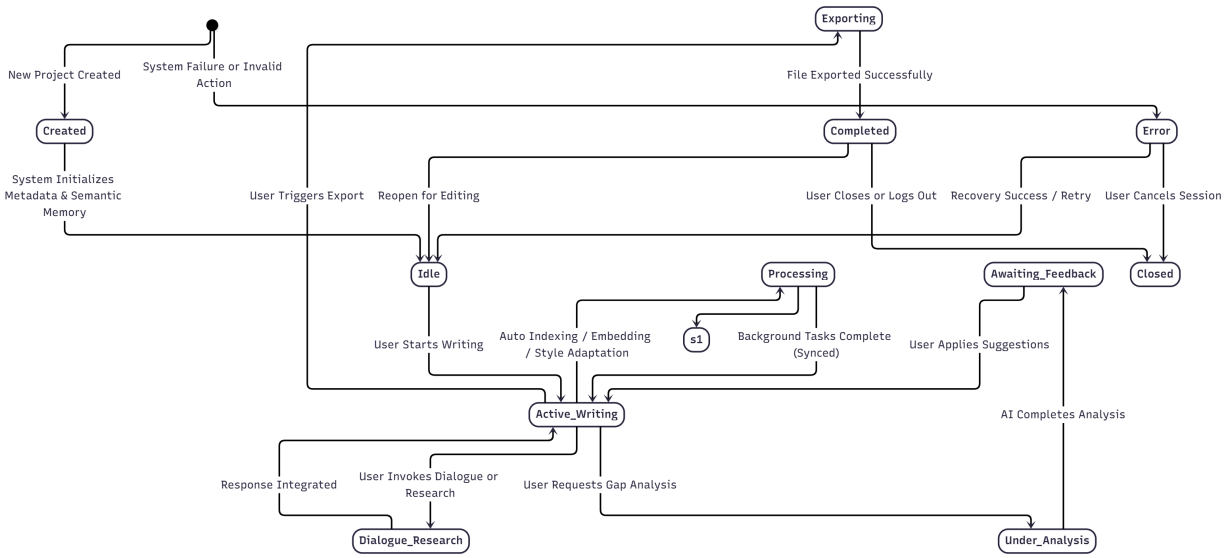

Figure 3.5: State Transition Diagram for CoWriteIA

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.

4.1.1 High-Level Algorithm Flow

The overall system follows this sequence:

1. Document upload by the user
2. Text extraction and chunking
3. Embedding generation for semantic indexing
4. Vector storage
5. Query-based context retrieval
6. AI-based response generation

This structured flow enables efficient and relevant AI-powered responses.

4.1.2 Document Chunking Algorithm

The system divides uploaded documents into smaller logical chunks. This improves semantic accuracy and ensures efficient storage.

Table 4.1: Document Chunking Algorithm

Pseudocode
Algorithm: Document Chunking
Input: Full document text
Output: List of text chunks
Split the document into sentences.
Initialize an empty list of chunks.
Append sentences to a chunk until size limit is reached.
Store the completed chunk and start a new one.
Store remaining text as the final chunk.

4.1.3 Embedding Generation and Storage

Each text chunk is converted into a numerical vector (embedding) that represents semantic meaning. These vectors are stored in a vector database to allow fast similarity search.

Table 4.2: Embedding Generation Algorithm

Pseudocode
Algorithm: Embedding Generation
Input: List of text chunks
Output: Vector embeddings
Read each text chunk.
Convert text chunks into semantic vector embeddings.
Store embeddings in vector database.
Link embeddings with source chunks.

4.1.4 Context Retrieval Algorithm (RAG)

When a user submits a query, the system retrieves the most relevant chunks using semantic similarity search.

Table 4.3: Context Retrieval Algorithm (RAG)

Pseudocode
Algorithm: Context Retrieval (RAG) Input: User query Output: Relevant text chunks Generate embedding for the user query. Search vector database for similar embeddings. Select top-k most relevant chunks. Return these chunks as context.

4.1.5 AI Response Generation Algorithm

The system combines the retrieved context with the user query and sends it to the language model to generate accurate, context-aware responses.

Table 4.4: AI Response Generation Algorithm

Pseudocode
Algorithm: AI Response Generation Input: User query and retrieved context Output: AI generated response Combine query and contextual information. Format the data into a prompt. Send the prompt to the language model. Receive and return the generated response.

4.1.6 Performance Considerations

- Chunking runs in linear time relative to document size.
- Embedding generation dominates processing time.
- Vector search operates in sub-linear time using approximate nearest neighbor indexing.

4.1.7 Reliability and Fault Handling

The system includes fallback mechanisms to handle:

- Missing embeddings
- Vector database unavailability
- Invalid user input

These ensure stable system behavior under failure scenarios.

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 4.5: 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 endpoints.

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.

	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 до	Verify root endpo	Black Box	API server runnir	GET /	200 OK, valid res	High	health
3	TC-02	System Health	Health Check AP	Verify /health enc	Black Box	API server runnir	GET /health	200 OK, status =	High	health
4	TC-03	System Health	API Docs доступн	Verify API docs a	Black Box	API server runnir	GET /docs or /op	200 OK, docs av	Medium	docs
5	TC-04	Authentication	User Registration	Register user wit	Black Box	DB online, user r	POST /auth/regis	201 Created, use	High	REQ-auth-1
6	TC-05	Authentication	User Registration	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 Unauthorize	High	negative
9	TC-08	Authentication	Get Current User	Retrieve user pro	Black Box	Valid access toke	GET /auth/me	200 OK, user dat	Medium	auth
10	TC-09	Authentication	Logout	Invalidate user se	Black Box	Logged in sessio	POST /auth/logo	200 OK, token in	Medium	auth
11	TC-10	Authentication	Refresh Token	Refresh expired	Black Box	Valid refresh toke	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?pa	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	Medium	projects
15	TC-14	Projects	Update Project	Update project m	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 validatic	Medium	validation
19	TC-18	Files	Upload File	Upload file to pro	Black Box	Project exists, au	POST /projects/{	201 Created, file	High	file-upload

Figure 4.1: BlackBox Tests Documentation fig1

	A	B	C	D	E	F	G	H	I	J
30	TC-29	Search	Generate Embec	Generate vector	Black Box	Model available	POST /embeddir	200 OK, vector r	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 ex	POST /search (fi	200 OK, filtered I	Medium	filters
34	TC-33	Search	Search Paginatic	Paginated searcl	Black Box	Large dataset	POST /search (p	Paginated respor	Low	pagination
35	TC-34	Search	Search Without F	Missing project v	Black Box	Invalid/missing p	POST /search	400/422 validatic	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 t	Black Box	Index built	GET/POST /autc	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 to	Black Box	Chat service ava	POST /chat	200 OK, AI respc	Medium	chat
40	TC-39	Chat	Chat With Conte	Contextual conve	Black Box	Chat history exis	POST /chat (with	200 OK, contextu	Medium	context
41	TC-40	Chat	Get Chat History	Retrieve convers	Black Box	Conversation exi	GET /chat/{id}/his	200 OK, history I	Low	history
42	TC-41	E2E	Complete Projec	End-to-end proje	Black Box	All services runn	Full flow executic	Workflow succee	High	e2e
43	TC-42	E2E	File Update Worl	File update and r	Black Box	Existing file	Update file → rei	Updated content	High	e2e
44	TC-43	E2E	Multi-file Search	Cross-file search	Black Box	Multiple files inde	Run cross-file se	Correct cross-file	Medium	similarity
45	TC-44	Integration	Comprehensive I	Full indexing pipe	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 sto	Vectors stored &	Medium	integration
47	TC-46	Integration	Async Indexing	Background asyr	Black Box	Worker queue ac	Trigger async tas	Tasks complete s	Medium	async
48	TC-47	Integration	Relationship Disc	Entity relationshi	Black Box	Extracted entities	Run relationship	Correct relations	Low	NLP

Figure 4.2: BlackBox Tests Documentation fig2

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 focused on verifying the internal logic of isolated system components. Each unit was tested independently to ensure that business logic, validation rules, and service-level operations functioned correctly.

4.3.2.1 Implemented Unit Test Coverage

Unit tests were implemented for the following components:

- Authentication API (register, login, token handling)
- Projects API (CRUD operations)
- Files API (upload, download, deletion)
- Search API (semantic, hybrid, and filter-based queries)
- Chat API (conversation handling and context management)
- Complete workflows (multi-step user journeys)
- Error cases (400, 401, 403, 404, 500)
- Edge cases (pagination and unauthorized access)

4.3.2.2 Testing Approach

Service functions, repositories, and helper methods were tested in isolation using mocks to remove external dependencies such as databases and third-party services.

4.3.2.3 Failure and Boundary Validation

Tests confirmed that invalid inputs, unauthorized access, and unsupported operations were handled safely without system failures.

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

Figure 4.3: 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.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this phase of the project, we built a clear understanding of the problems writers face and designed a complete solution to address them. We explored existing tools, identified their limitations, and defined how CoWriteIA can offer a more helpful and consistent writing experience. All major requirements, system modules, diagrams, and architectural decisions were finalized, giving a strong foundation for development.

We also completed the initial implementations of the core system components such as project indexing, semantic memory, and the basic writing assistant. Overall, FYP-I helped us shape the direction of CoWriteIA and prepare the groundwork needed for building an intelligent, context-aware writing platform.

5.2 Future Work

In the next phase, we will focus on completing the remaining features and integrating all modules into a fully functional system. This includes improving the context-aware writing agent, completing dialogue generation and character management, and enhancing the gap analysis module. We will also work on research integration, better semantic search performance, UI/UX refinement, and overall system testing.

By the end of the next phase, our goal is to deliver a polished and reliable AI writing assistant that supports writers throughout their creative process.

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] Literature and Latte. Scrivener overview. Literature and Latte Website, n.d.

[4] Notion. Everything you can do with notion ai. Notion Website, n.d.

Appendix A

Appendices

A.1 Appendix A

A.1.1 Use Case Diagram

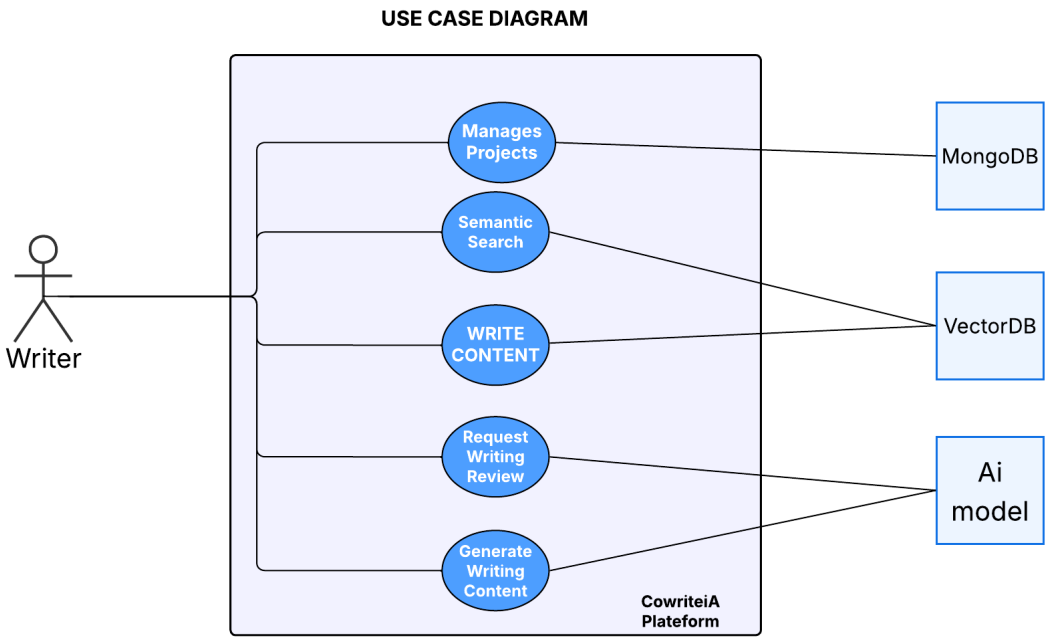

Figure A.1: Use Case Diagram CoWriteIA

A.2 Appendix B

A.3 Appendix C

A.3.1 CoWriteIA Detailed Architecture Diagram

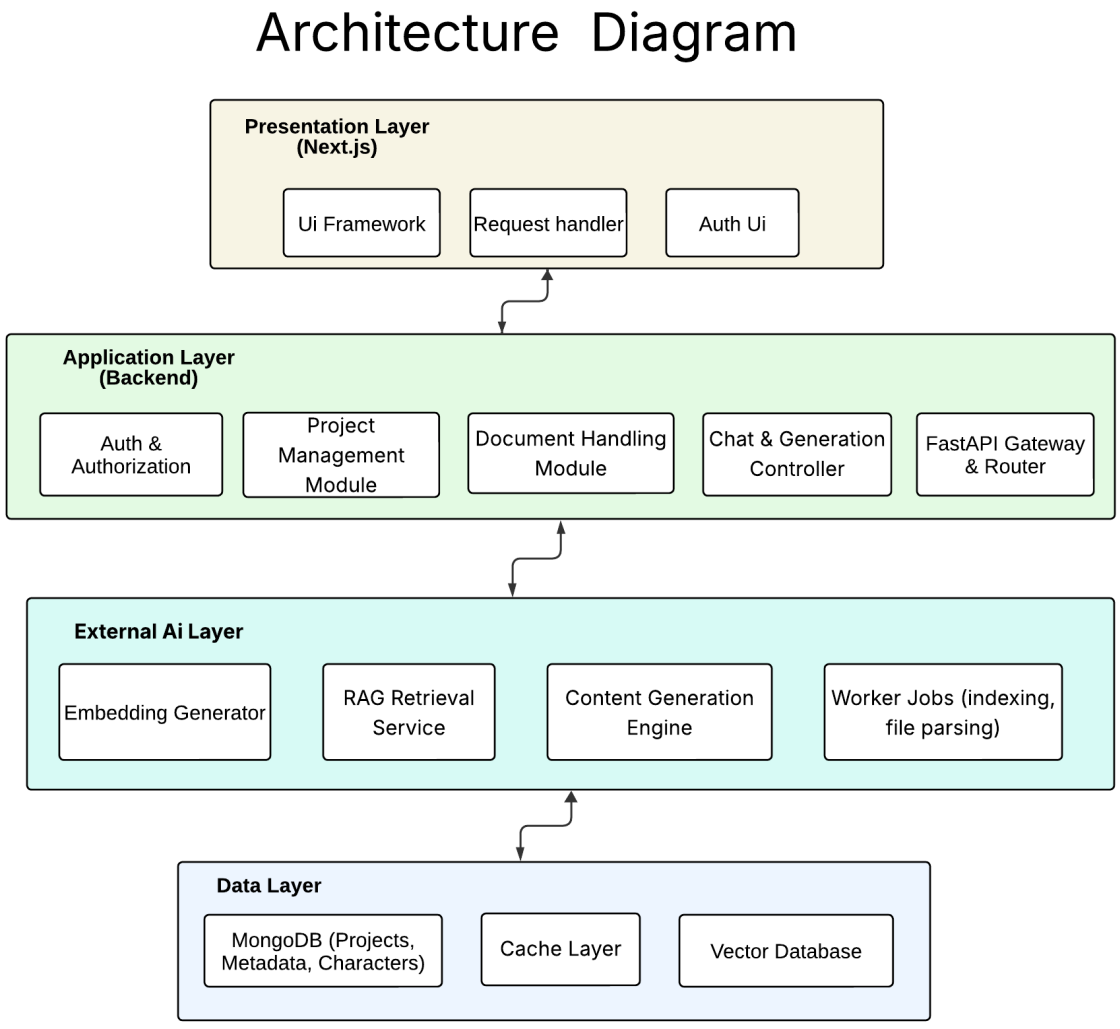

Figure A.2: CoWriteIA Multi-Tier Architecture Diagram

A.4 Appendix D

A.4.1 Activity Diagram

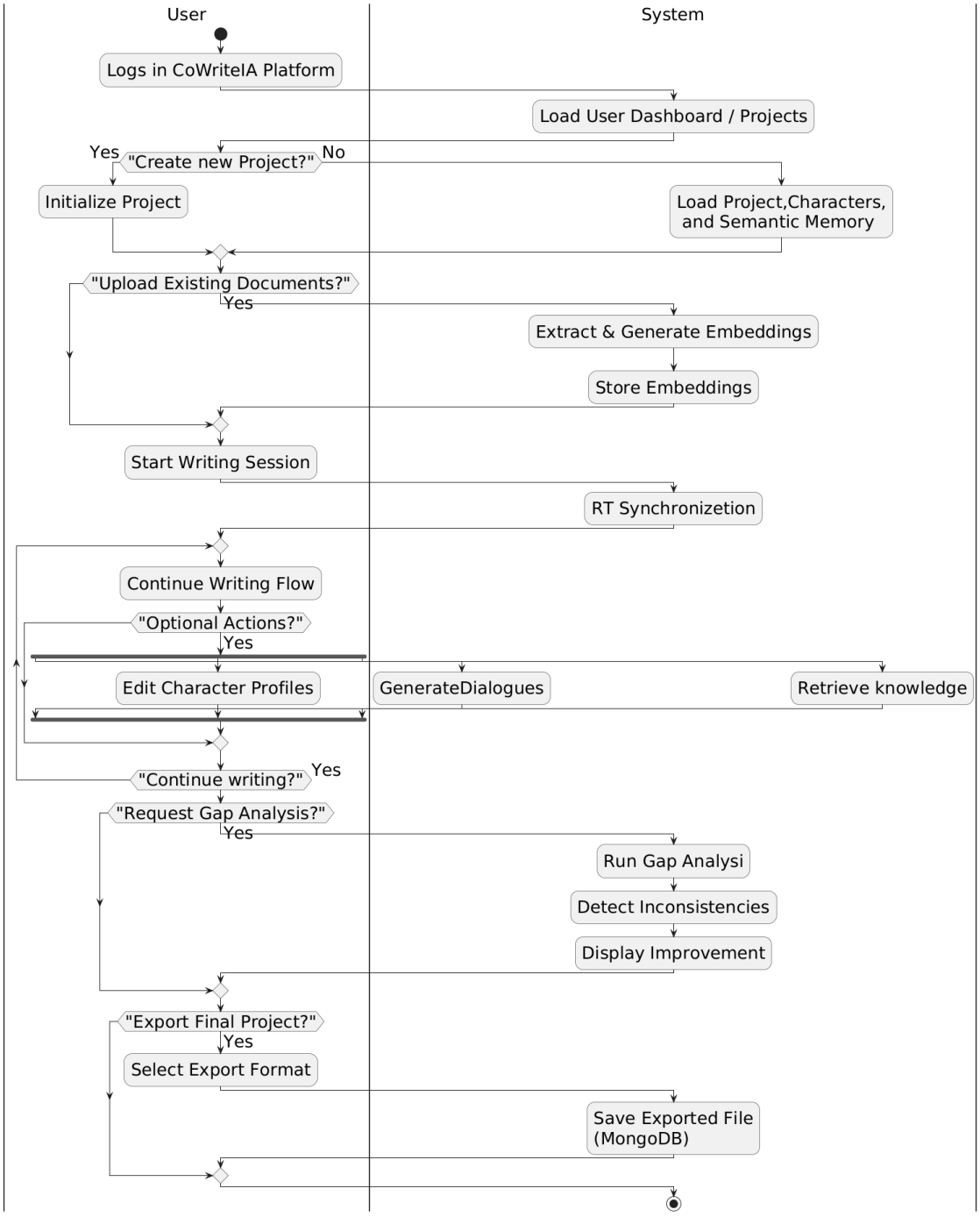

Figure A.3: Activity Diagram