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 [1].

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 [1].

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 [1]. 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].

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].

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			✓
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.

Detailed examples of use cases, storyboards, and event-response tables are placed in Appendix A.

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. 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.8).

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

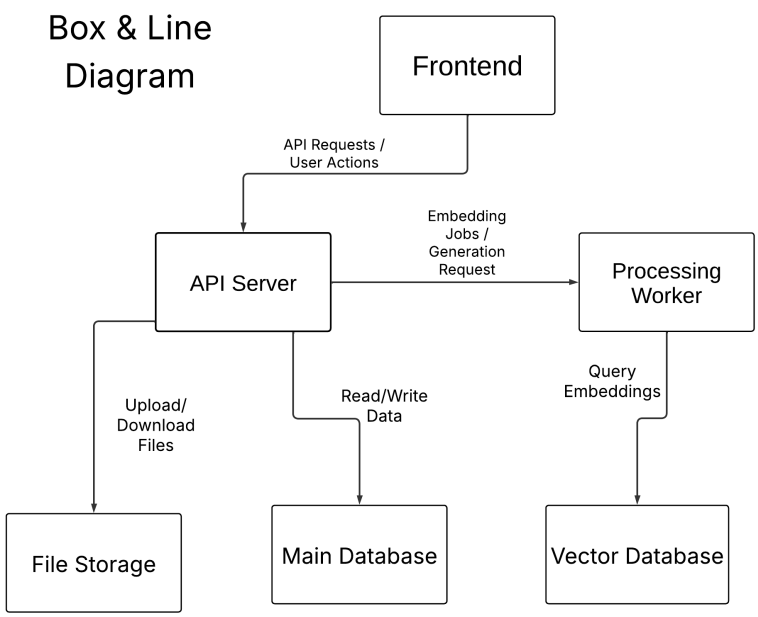

Figure 3.1: Box and Line Diagram showing major subsystems and their connections

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

3.2 Data Design

The data design of CoWriteIA defines how the information domain is structured, processed, and stored across the system. Since the application supports project creation, AI-assisted writing, semantic search, character management, and dialogue modelling, the data layer must accommodate a mix of structured and unstructured information. To achieve this, the system separates its storage into three main components: a primary database for structured data, a vector database for embeddings, and file storage for uploaded and exported documents.

The primary database stores core system entities such as users, projects, documents, characters, chat sessions, and messages. These entities represent the logical workflow of the writing environment and form the backbone of user interaction and content management. The vector database stores high-dimensional embeddings generated from project content, character attributes, and dialogues. These embeddings support semantic search, retrieval-augmented generation, and contextual augmentation during writing sessions. File storage handles uploaded documents and exported project files, ensuring that larger files remain efficiently managed outside the main database.

This separation ensures scalability and performance: structured data is quickly accessed via database queries, semantic operations run on optimized vector indexes, and file operations remain independent of the database workload. Figure 3.3 illustrates how major data entities relate to each other.

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.

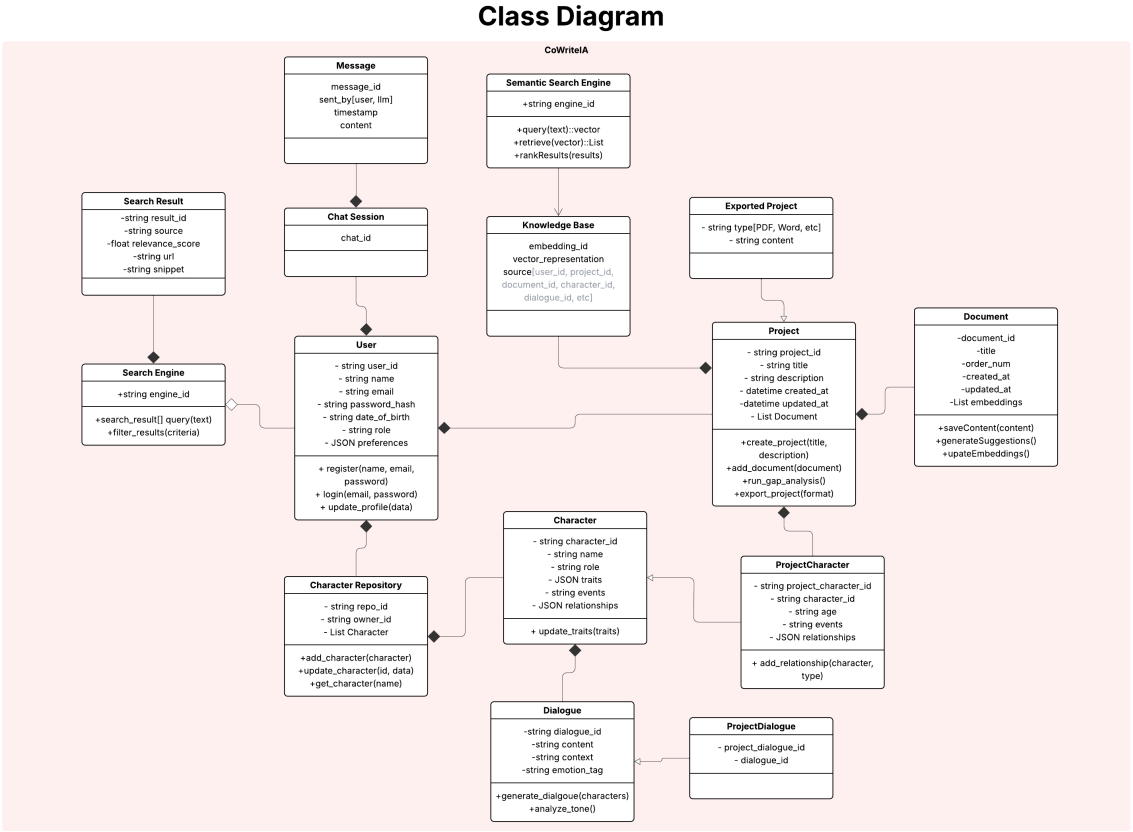

Figure 3.2: Class Diagram for CoWriteIA

3.3 Domain Model

The domain model serves as a conceptual blueprint of the system, describing the key entities, their attributes, and the relationships that define how information flows throughout CoWriteIA. It helps translate functional requirements into a structured and understandable design, guiding both implementation and future system expansion.

3.3.1 Entities and Attributes

The major entities of the system and their attributes are summarized below:

- **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.

3.3.2 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.

3.3.3 Domain Model Diagram

Figure 3.3 presents the complete domain model for CoWriteIA. It includes all major entities, attributes, and associations, including weak entities and conceptual relationships.

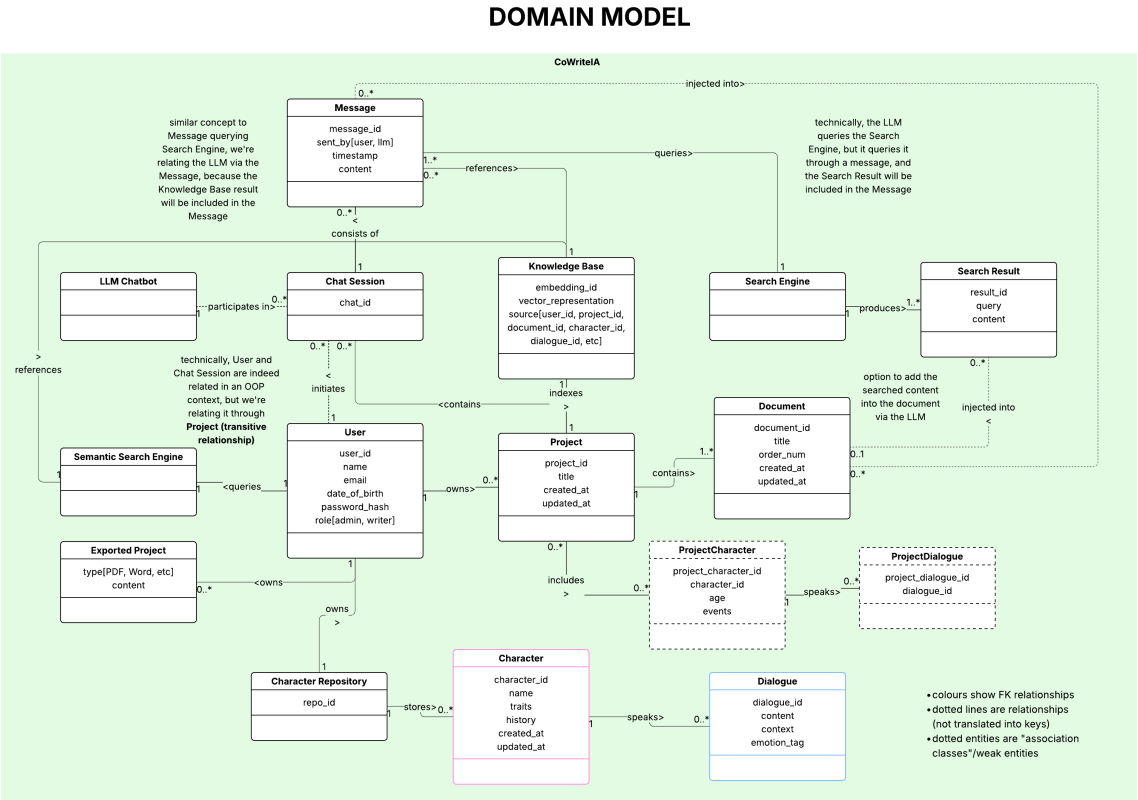

Figure 3.3: Domain Model Diagram for CoWriteIA

3.4 Design Models [UPTO THE CURRENT ITERATION ONLY]

Create design models as are applicable to your system. Provide detailed descriptions with each of the models that you add. Also ensure visibility of all diagrams.

Design Models for Object Oriented Development Approach

The applicable models for the project using object oriented development approach may

include:

• Activity Diagram
• Class Diagram
• Class-level Sequence Diagram
• [OPTIONAL] State Transition Diagram (for the projects which include event handling and backend processes)

Design Models for Procedural Approach

The applicable models for the project using procedural approach may include:

• Activity Diagram
• Data Flow Diagram (data flow diagram should be extended to 2-3 levels. It should clearly list all processes, their sources/sinks and data stores.)
• System-level Sequence Diagram
• [OPTIONAL] State Transition Diagram (for the projects which include event handling and backend processes)

Examples of above diagrams are given in Appendix D.

Chapter 4

Implementation and Testing [UPTO THE CURRENT ITERATION ONLY]

Give a general description of the functionality, context, and design of your project. Provide any background information if necessary.

4.1 Algorithm Design

Mention the algorithm(s) used in your project to get the work done with regards to major modules. Provide a pseudocode explanation regarding the functioning of the core features. Following are few examples of algorithms/pseudocode.

Example:

Algorithm 1 MHCF co-authorsBasedClustering

Input: n groups G_n where each group has set of papers (p_r)

Output: Set of system generated clusters/groups G_n

```
merge ← true
Flag ← false
While(merge=='true') do:
    merge ← false
        for i in range (0: len(G)-1):
            for j in range (i+1: len(G)):
                if (similarCoauthors(GiLco-authors, GjLco-authors) == true) then
                    Flag ← true
                Else (checkNameFragments(GiLco-authors, GjLco-authors) == true) then
                    Flag ← true
                if (Flag == true) then
                    Gi ← Gi U Gj
                    G ← G.pop(j)
                    merge ← true
                end if
            end for
        end for
end while
```

Figure 4.1: Example Of Algorithm Design

4.2 External APIs/SDKs

Describe the third-party APIs/SDKs used in the project implementation in the following table. Few examples of APIs are provided in the table.

API and version	Description	Purpose of usage	API endpoint/function/class used
Stripe (version 2020-08-27)	Credit Card payment integration	Sandbox used orders payment	stripe.paymentMethods.create
Cloudinary	Image and Video management	Uploading Product Images	https://api.cloudinary.com/v1

4.3 Testing Details

Once the system has been successfully developed, testing has to be performed to ensure that the system working as intended.

4.3.1 Unit Testing

Each unit test is designed to test a specific function or method independently from other components, helping to identify issues directly related to the functionality being tested.

Following is the example of Unit testing:

Test case ID	Test Objective	Precondition	Steps	Test data	Expected result	Post-condition	Actual Result	Pass/fail
TC001	Verify admin login with username and password	Admin should be registered with valid email and password before login.	Click on Login button Enter valid username and password	Email-id: abc@xyz.com Password: Xyz123	System displays Admin homepage	Admin should be kept logged in until logout.	As Expected,	Pass

Figure 4.2: Example for Unit Testing

Bibliography

[1] A Kolyshkin and S Nazarovs. Stability of slowly diverging flows in shallow water. *Mathematical Modeling and Analysis*, 2007.

Appendix A

Appendices

A.1 Appendix A

A.1.1 Use Case Diagram example (Online Shopping System)

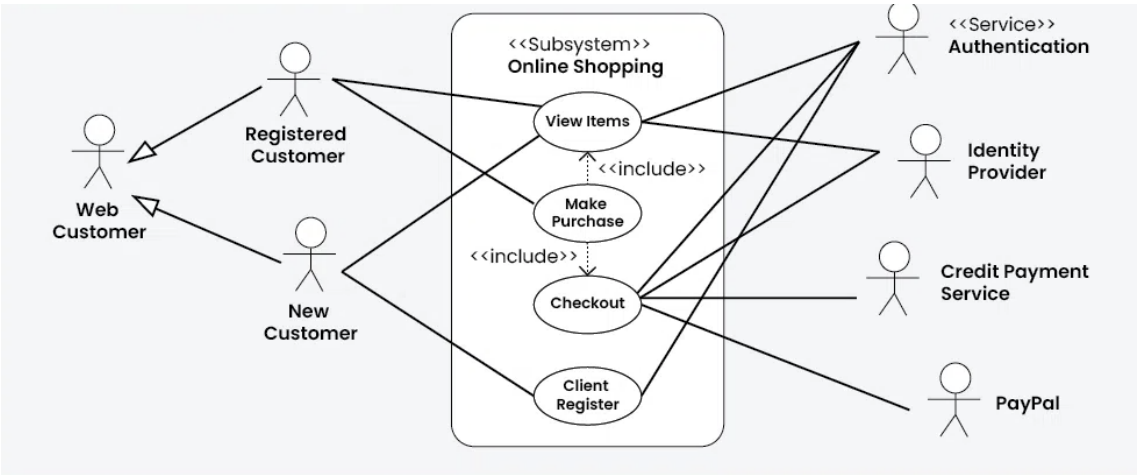

Figure A.1: Use Case Diagram for the Online Shopping System

A.1.2 Detail Use Case Example

ID	#1
Name	Overview elements
Short Description	All elements are shown in a list, where the user can set different filters.
Goal	Displaying elements in a changeable view.
Preconditions	The user got access to the system and is logged in. The user got the right to see schedules.
Success End Condition	The correct elements (filter and sorting) are displayed.
Fall End Condition	Elements not matching the criteria are displayed.
Stakeholder	Customer manager
Trigger	Login in as customer manager (because overview is on the starting screen for this role).
Normal Flow	1. The list of all elements matching default criteria are shown. 2. The user changes the filter criteria. 3. The user presses the Filter button. 4. The list shows all matching elements. Optional: 5. The user presses the Clear button. 6. The list of all elements matching default criteria are shown.
Alternative Flows	With click on the column headers, the list sorting can be changed. Different sorting for the different columns is described in the table below.
Includes	Login
Frequency of Use	About 50 times per day
Constraints and Special Requirements	None
Assumptions	None
Notes and Issues	None

Figure A.2: Detail Use Case Example

A.1.3 Event-Response Table for a Highway Intersection

Event	System State	Response
Road sensor detects vehicle entering left-turn lane.	Left-turn signal is red. Cross-traffic signal is green.	Start green-to-amber countdown timer for cross-traffic signal.
Green-to-amber countdown timer reaches zero.	Cross-traffic signal is green.	1. Turn cross-traffic signal amber. 2. Start amber-to-red countdown timer.
Amber-to-red countdown timer reaches zero.	Cross-traffic signal is amber.	1. Turn cross-traffic signal red. 2. Wait 1 second. 3. Turn left-turn signal green. 4. Start left-turn-signal countdown timer.
Pedestrian presses a specific walk-request button.	Pedestrian sign is solid Don't Walk. Walk-request countdown timer is not activated.	Start walk-request countdown timer.
Pedestrian presses walk-request button.	Pedestrian sign is solid Don't Walk. Walk-request countdown timer is activated.	Do nothing.
Walk-request countdown timer reaches zero plus the amber display time.	Pedestrian sign is solid Don't Walk.	Change all green traffic signals to amber.
Walk-request countdown timer reaches zero.	Pedestrian sign is solid Don't Walk.	1. Change all amber traffic signals to red. 2. Wait 1 second. 3. Set pedestrian sign to Walk. 4. Start don't-walk countdown timer.

Figure A.3: Example of Event Response Table

A.1.4 Story Board Example For Android App

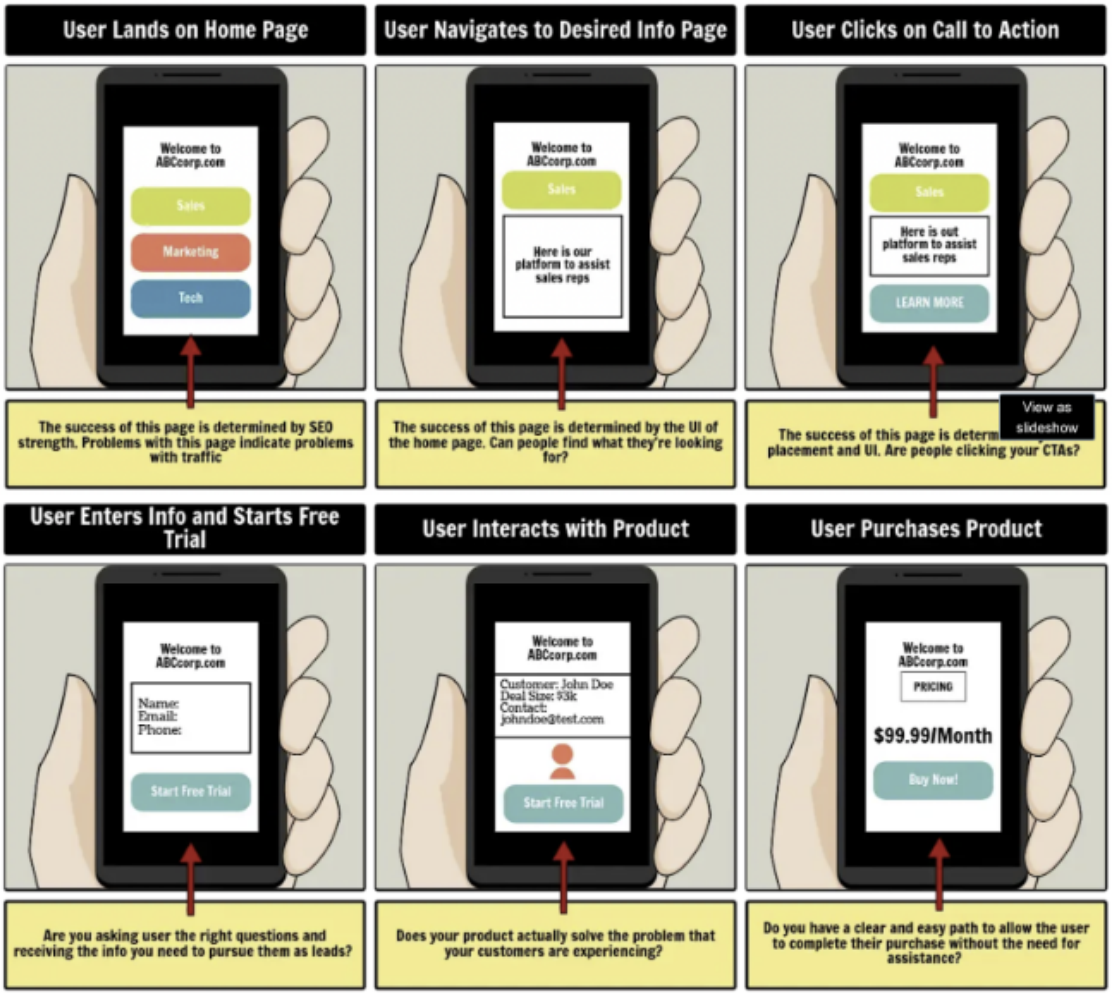

Figure A.4: Example of Story Board

A.2 Appendix B

A.2.1 Domain Model Example For Online Shopping

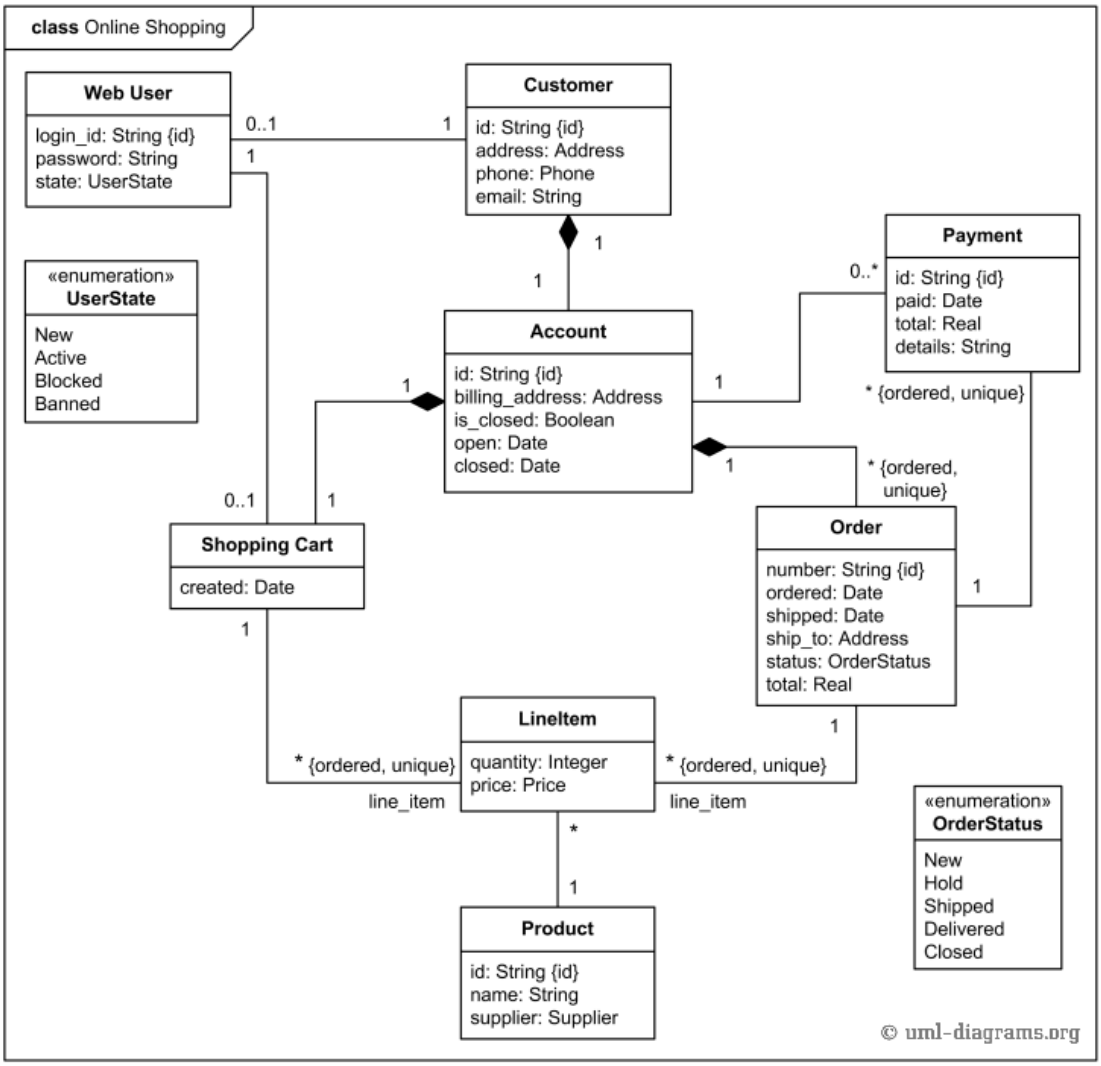

Figure A.5: Domain Model Example For Online Shopping Application

A.3 Appendix C

A.3.1 Box And Line Example For Online Shopping

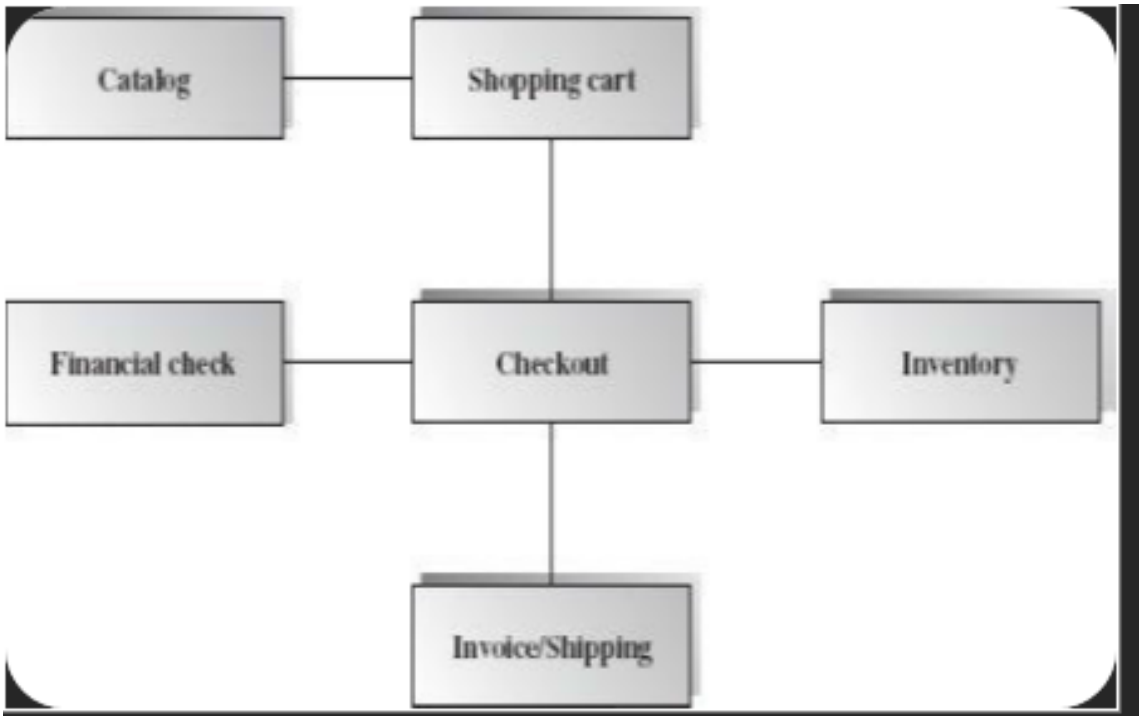

Figure A.6: Box and Line Diagram For Online Shopping Application

A.3.2 Architecture Pattern Example For Online Shopping

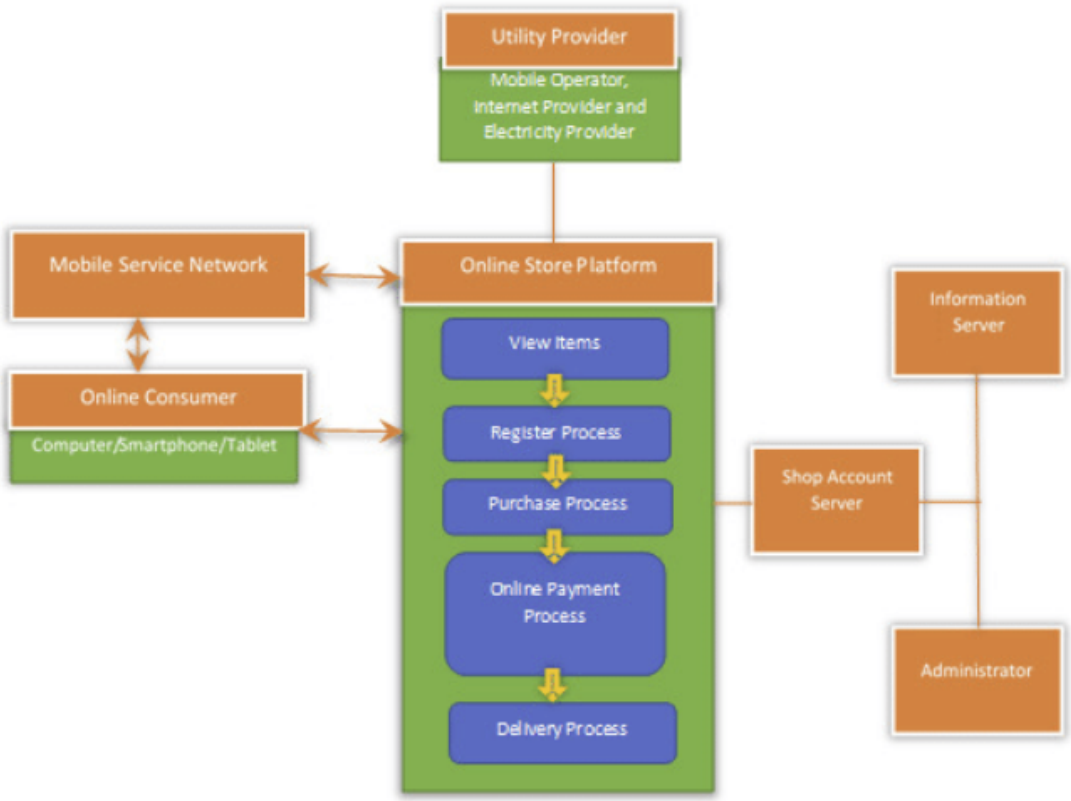

Figure A.7: Architecture Pattern For Online Shopping Application

A.3.3 CoWriteIA Detailed Architecture Diagram

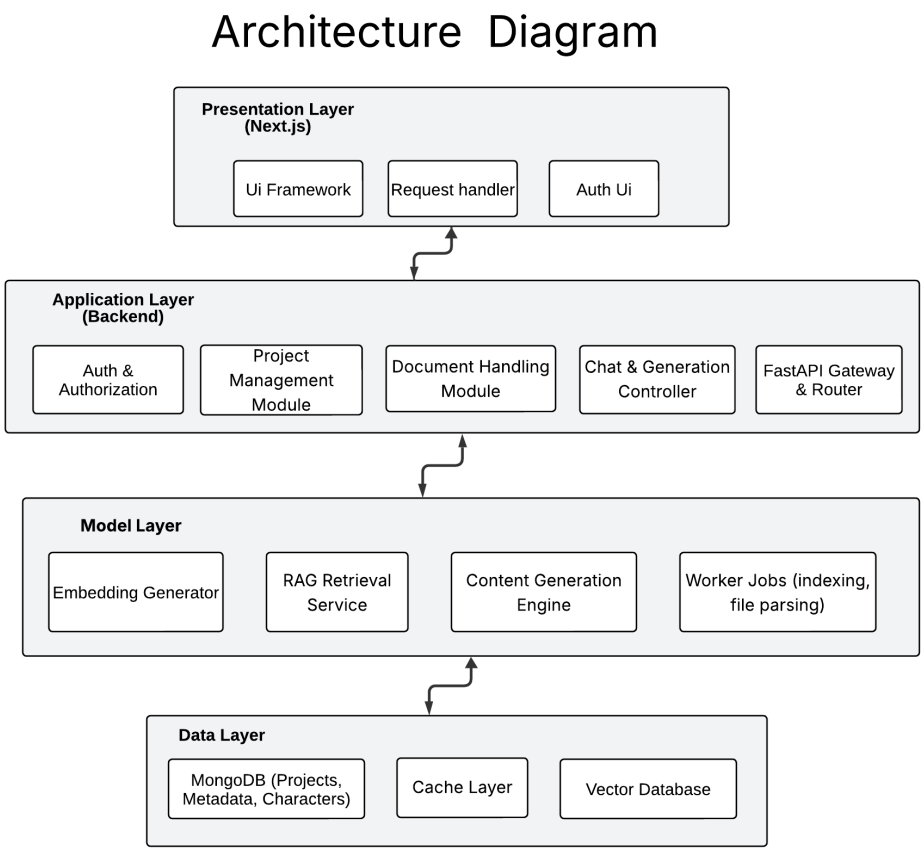

Figure A.8: CoWriteIA Multi-Tier Architecture Diagram

A.4 Appendix D

A.4.1 Activity Diagram

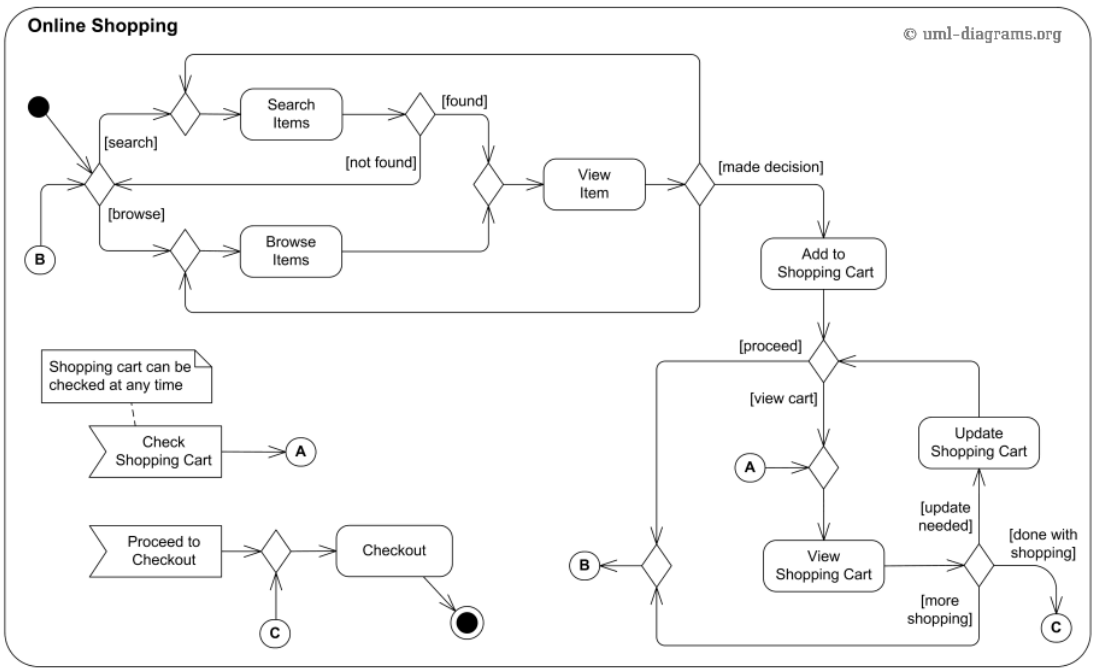

Figure A.9: Activity Diagram For Online Shopping Application

A.4.2 Class Diagram

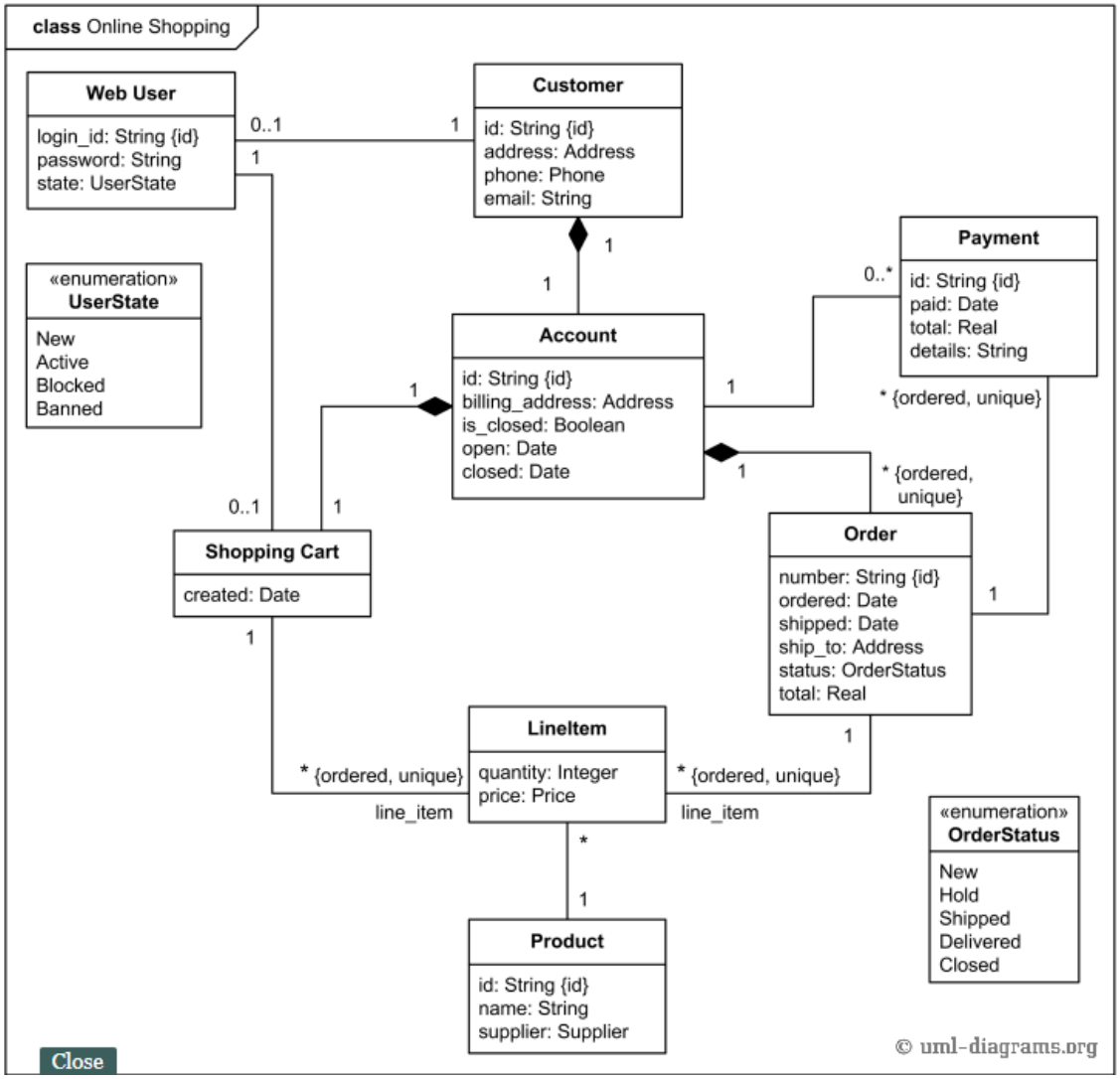

Figure A.10: Class Diagram For Online Shopping Application

A.4.3 Sequence Diagram

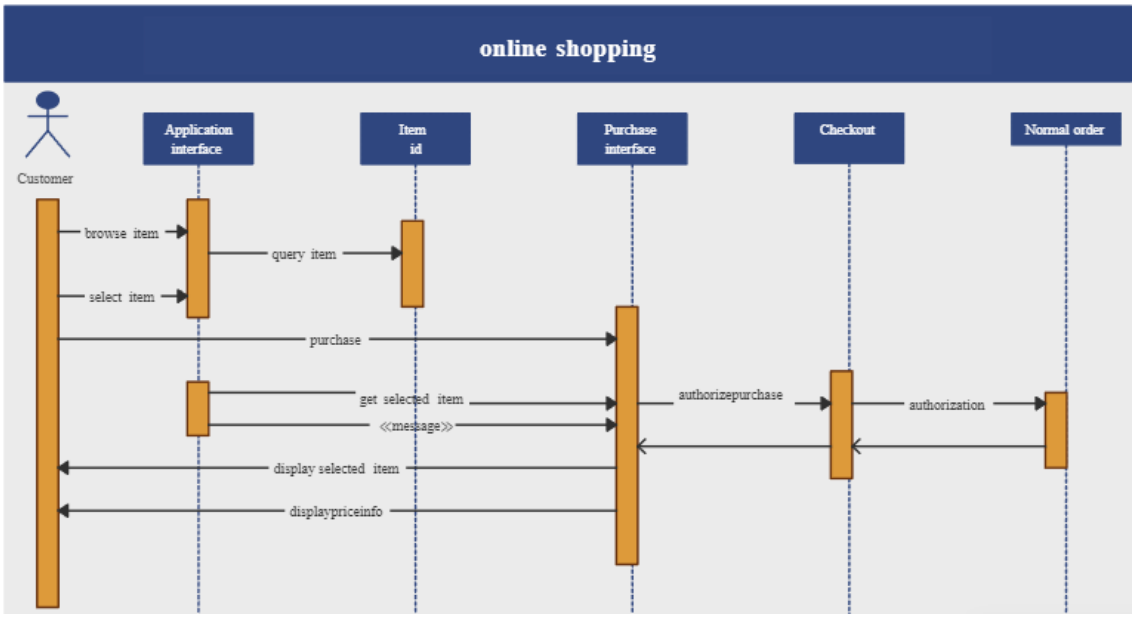

Figure A.11: Sequence Diagram For Online Shopping Application

A.4.4 State Transition Diagram

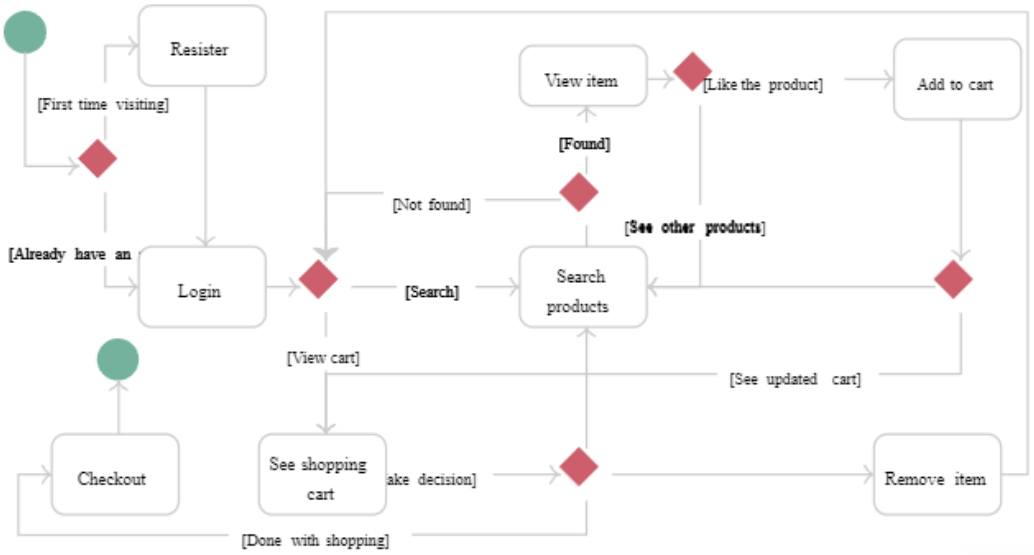

Figure A.12: State Transition Diagram For Online Shopping Application

A.4.5 Data Flow Diagram

Data Flow Diagram

Data flow diagram symbols, symbol names, and examples

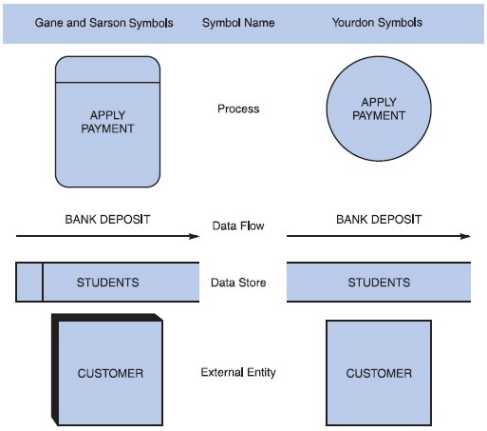

Figure B-5 Data flow diagram symbols, symbol names, and examples of the Gane and Sarson and Yourdon symbol sets.

Guidelines for Drawing DFDs

Step 1: Draw a Context Diagram: The first step in constructing a set of DFDs is to draw a context diagram. A **context diagram** is a top-level view of an information system that shows the system's boundaries and scope. Data stores are not shown in the context diagram because they are contained within the system and remain hidden until more detailed diagrams are created.

Example

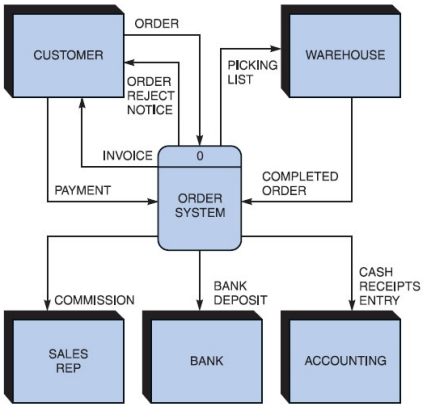

Figure B-6 Context diagram DFD for an order system.

Step 2: Draw a Diagram 0 DFD: To show the detail inside the black box, you create DFD diagram 0. **Diagram 0** zooms in on the system and shows major internal processes, data flows, and data stores. Diagram 0 also repeats the entities and data flows that appear in the context diagram. When you expand the context diagram into DFD diagram 0, you must retain all the connections that flow into and out of process 0.

Example

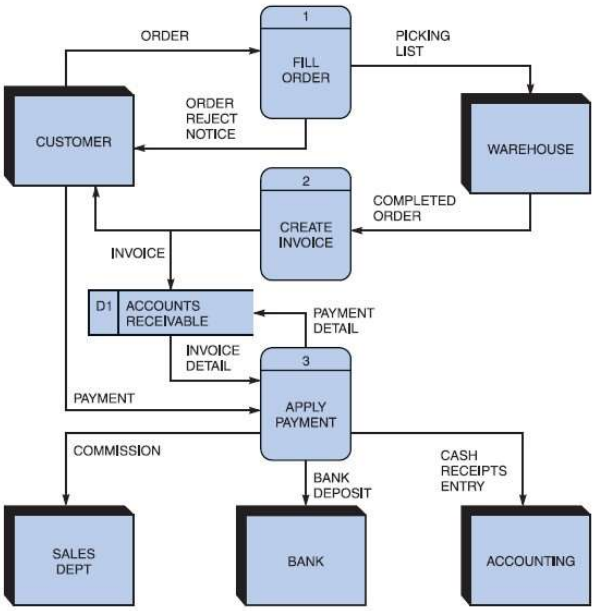

Figure B-7 Diagram 0 DFD for the order system.

Step 3: Draw the Lower-Level Diagrams:

To create lower-level diagrams, you must use leveling and balancing techniques. **Leveling** is the process of drawing a series of increasingly detailed diagrams, until all functional primitives are identified.

Leveling Example

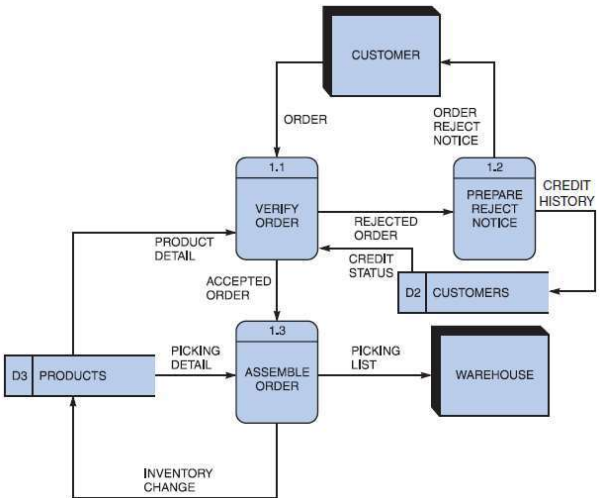

Figure B-8 Diagram 1 DFD shows details of the FILL ORDER process in the order system.