Zewail City of Science, Technology and Innovation
University of Science and Technology
School of Computational Sciences and Artificial Intelligence

CSAI 203 - Fall 2025

Introduction to Software Engineering

Unify

Phase 3: Design

Team Number: #27

Team Members:

Ahmed Moatasem 202300917
Jana Mahmoud 202301597
Mohamed Hatem 202301610
Karim Wael 202202212
Ali Mohab 202300786

Representative Contact info: s-ahmed.momtaz@zewailcity.edu.eg

1. Introduction

1.1 Purpose of the Document

The purpose of this Design Document is to provide a detailed technical blueprint for the Unify system, and translates the Software Requirements Specification (SRS) into a concrete design that can be implemented and tested. It defines the system's architecture, the application of the Model–View–Controller (MVC) pattern, module responsibilities, database schema, user interface layout, and key interaction flows, and acts as the foundational architecture, defines the technology stack, specifies internal and external interfaces, and details the data models necessary for implementation. It serves as the primary technical reference for the development and quality assurance teams, ensuring that the final system meets all functional and non-functional requirements (FRs and NFRs) specified in the SRS, and ensures that all team members and stakeholders share a common understanding of how the system will be structured and how its components will interact before coding begins.

1.2 Scope of the Design Phase

Architectural Design:
Defining the system's structure, main components, and how they communicate using the MVC architecture.

Component Specification:
Creating all system architecture diagrams and sequence diagrams to explain how each feature works.

DatDesign:
Building the logical and physical data models for the database, including all required entities and relationships.

UI Design:
Creating wireframes for the main screens to visualize how the system will look and function before implementation.

1.3 Intended Audience

Software Architects and Technical Leads : to validate the system design and ensure alignment with best practices.

Backend and Frontend Developers : to follow the defined architecture, MVC structure, and UI flow during implementation.

Database Designers : to use the data models for building the system’s database structure.

Test Engineers and Maintainers : to derive test cases and ensure long-term stability.

Project Managers and Stakeholders : to understand the technical approach and design decisions.

1.4 Overview of Contents

Section 2: System Overview Provides a high-level description of the Unify system, its main objectives, and the primary design goals and constraints that shaped the overall architecture.

Section 3: Architectural Design Defines the system’s architecture, explaining how the Model–View–Controller (MVC) structure is applied. This section includes the System Architecture Diagram, a description of the architectural style and major components, and the technologies and tools selected for Unify.

Section 4: Detailed Design Presents the technical design in depth, including detailed UML Class Diagrams, Sequence Diagrams, UI Wireframes/Mockups for the main screens, and the complete Data Design (database schema, file structure, and data dictionary).

Section 5: Conclusion Summarizes the key design decisions, artifacts, and outputs produced during this design phase.

2. System Overview

2.1 Brief System Description

Unify is an AI-enhanced, web-based student portal designed to centralize academic tools, automate scheduling, and improve student productivity across university life. It functions as an integrated academic assistant, combining features similar to university portals, Google Calendar, Notion, and AI-based personal organizers in one unified platform.

The main purpose of Unify is to simplify and enhance a student’s academic journey by providing an intelligent system that manages schedules, deadlines, tasks, course information, and academic reminders. Unify uses AI/NLP to analyze student commitments, generate optimized weekly timetables, summarize lecture notes, and offer smart recommendations.

Key functionalities include:

- AI Schedule Optimizer:
  Automatically generates conflict-free weekly schedules using course times, deadlines, and personal constraints.
- Natural Language Query Assistant:
  Students can ask questions such as:
  “When is my next lecture?” or “Show my assignments for Machine Learning.”
- Task & Deadline Manager:
  Add tasks, view deadlines, receive reminders, and categorize academic vs. personal tasks.
- Calendar Integration:
  Syncs university deadlines and schedules with Google Calendar for real-time updates.
- AI Note Summarizer:
  Summarizes uploaded lecture notes and extracts key concepts, saving study time.
- Student Dashboard:
  Displays courses, upcoming deadlines, tasks, reminders, schedule, and recent updates in a unified interface.

Unify acts as a personal academic assistant designed to automate planning, improve organization, and enhance student success through intelligent features.

2.2 Key Design Goals and Constraints

Design Goals (Objectives):

The main objectives of Unify focus on improving academic efficiency, reducing friction in planning, and giving students an AI-supported learning experience:

- Academic Productivity:
  Reduce manual planning effort by automating scheduling, reminders, and task management.
- User Convenience:
  Provide a clean, intuitive platform where all academic tools are available in one place.
- Personalization:
  Deliver tailored schedules, reminders, and summaries based on a student’s courses, workload, and preferences.

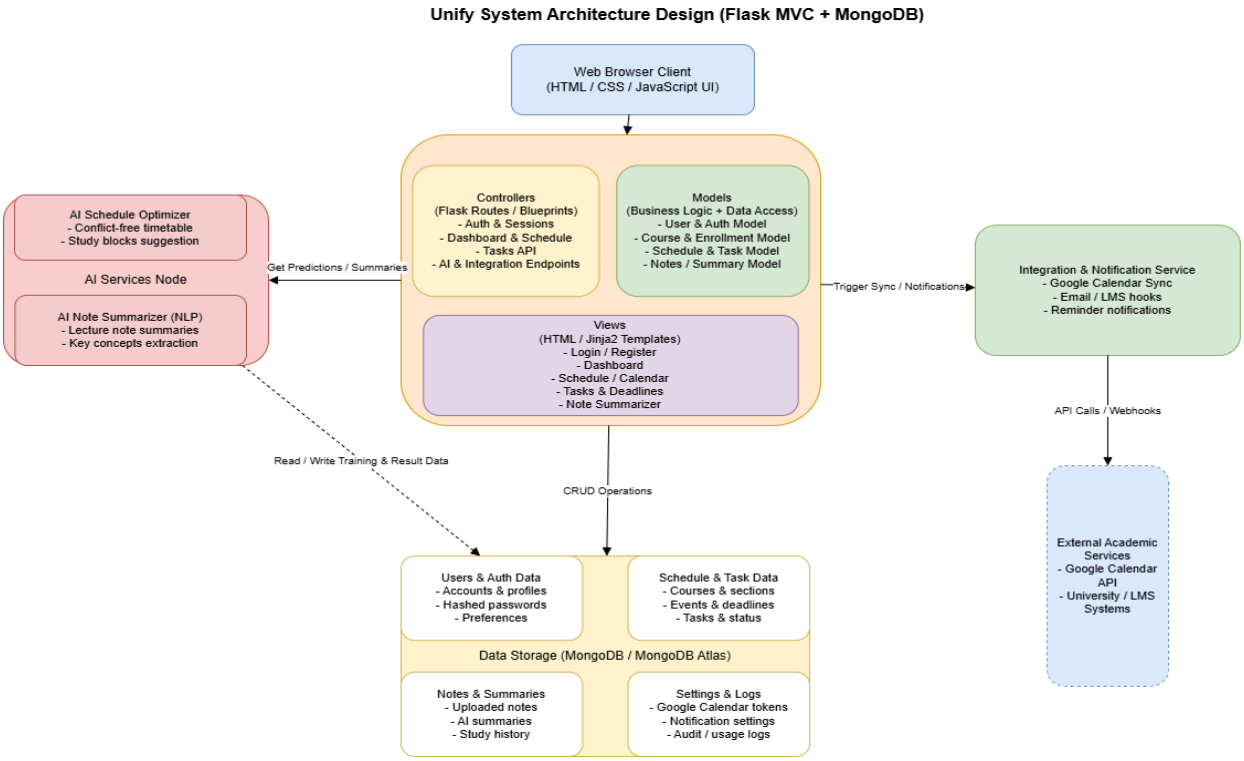
- Accuracy & Automation:
  Use AI/NLP to generate summaries, analyze deadlines, answer academic questions, and optimize schedules based on real data.
- Integration:
  Seamlessly connect with Google Calendar and LMS-style systems for deadlines and course updates.

Unify must be developed under the following constraints:

- Technology Stack:
  Must be built using Flask (Python) with HTML, CSS, JavaScript, and follow the MVC design pattern.
  No use of Java-based backend frameworks.
- Database:
  The system must use MongoDB as the primary database for storing users, schedules, tasks, summaries, and settings.
- Scalability:
  Should efficiently support 5,000+ users with non-relational workload distribution handled by MongoDB.
- Data Privacy:
  Must ensure secure handling of student data and comply with general academic privacy standards.
- API Limitations:
  Required support for Google Calendar API and optional support for LMS-style academic data sync.
- AI Model Constraints:
  AI components (schedule optimizer & note summarizer) must be lightweight, explainable, and executable within Flask.
- Operating Environment:
  Web-based, deployed on a cloud or university server, using MongoDB Atlas or on-prem MongoDB for database hosting.

3. Architectural Design

3.1 System Architecture Diagram



3.2 Discussion of Architectural Style and Components

The Unify system adopts a monolithic web architecture that is logically divided into several layers: the Client Application, the Core Platform (Flask MVC), the AI Services Node, the Integration Layer, and the Data Storage Layer.
This structure simplifies deployment while still separating concerns into clear modules that run inside a single web application stack.

The architecture strictly applies the Model–View–Controller (MVC) pattern:

- The Client Application acts as the primary View, rendering screens and handling user interaction in the browser.

- The Core Platform (Flask) acts as the central Controller, receiving HTTP requests, validating them, orchestrating business logic, and coordinating with models, AI services, and integrations.
- The Data Storage layer (MongoDB) represents the Model/Persistence layer, holding all persistent data used by the application.

Domain/Layer	Component Name	MVC Role	Primary Responsibility
Client Application	Web Browser UI (HTML/CSS/JavaScript)	View	Manages all user interface rendering, user interaction, and data presentation (login, dashboard, schedule, tasks, note summaries). Sends HTTP/HTTPS requests to the Flask backend and updates the UI based on responses.
Core Platform	Routing & API Layer (Flask Routes / Blueprints)	Controller (Interface)	Serves as the secure entry point for all client requests. Handles URL routing, request validation, session management, and dispatches requests to the appropriate business logic functions and models.
Core Platform	Business & Controller Logic	Controller (Core)	Executes core application rules such as authentication, schedule generation requests, task management, and note processing. Coordinates calls to Models, AI Services, and Integration components, then prepares responses for the View.

Core Platform	Domain Models & Data-Access Layer	Model (Logic + Persistence Access)	Encapsulates domain entities (users, courses, enrollments, schedules, tasks, notes, summaries) and provides a clean API for reading/writing data in MongoDB. Ensures validation, consistency, and reuse of data-related logic.
AI Services Node	AI Schedule Optimizer Service	Logic/Service	Provides real-time or on-demand optimized timetables using course data, deadlines, and user preferences. Returns conflict-free schedules and study suggestions to the Core Platform for display in the UI.
AI Services Node	AI Note Summarizer Service (NLP)	Background / Logic Service	Processes uploaded lecture notes or text, generates concise summaries and key points, and returns them to the Core Platform to be stored and displayed to students.
External Integration	Integration & Notification Service	Service/Proxy	Acts as a proxy between Unify and external systems such as Google Calendar or future LMS APIs. Handles API calls, token management,

			data mapping, and triggers notifications or calendar sync operations.
Data Storage	Main MongoDB Collections (Users, Courses, Schedules, Tasks, Notes)	Model (Transactional)	Stores all operational, transactional data: user accounts, course lists, enrollments, schedules, tasks, and AI-generated summaries. Accessed by the Domain Models for standard CRUD operations.
Data Storage	Logs & Settings Collections (Audit, Integrations, Preferences)	Model (Historical / Config)	Stores configuration data (integration tokens, user settings) and audit logs used for monitoring, debugging, and improving system reliability and future analytics.

3.3 Technology Stack and Tools

Frontend : HTML, CSS and JS
Backend : Python (Flask)
Database : MongoDB
Tools : Git,GitHub ,Draw.io, PowerPoint
Testing : Postman , Browser DevTools

,4. Detailed Design

4.1 Model–View–Controller (MVC) Design Pattern

4.1.1 Description of MVC Pattern

The Model–View–Controller (MVC) is an architectural pattern selected to separate the application logic from the user interface. This structure is essential for managing the complexity of Unify, which combines academic scheduling, task management, natural language processing (NLP), AI-driven optimization, and multiple external integrations such as Google Calendar.

Model: The business logic, data management, and core domain objects (e.g., User, Course, Schedule, Task, Note/Summary) reside here. It operates independently of the user interface and communicates directly with the MongoDB database.

View: Responsible for displaying data received from the Model and for capturing user interactions (e.g., clicks, form submissions, task creation). This includes all HTML/Jinja2 templates, CSS styling, and JavaScript-based dynamic behaviors.

Controller: Acts as an intermediary between the View and the Model, receiving user requests, executing the required logic in the Model, invoking AI services when necessary, and selecting the appropriate View to render the response.

Reason for Choosing MVC:
MVC was chosen to satisfy the system’s non-functional requirements for maintainability, extensibility, and scalability. The separation of concerns ensures that the Flask (Python) backend can evolve independently of the HTML/CSS/JavaScript frontend. This structure also supports Unify’s AI components and external service integrations (such as Google Calendar), which may require frequent updates. By isolating logic, data, and presentation, MVC guarantees cleaner code organization and easier future enhancements.

4.1.2 Mapping of Project Components to MVC

MVC Element	Unify Project Component(s)	Rationale
View	Client Application (Web Browser UI) • HTML5 / CSS3 / JavaScript • Jinja2 Templates (Flask)	Responsible for presenting data to the user and capturing user input. It is the end-user interface layer.
Controller	Unify Core Platform (Flask MVC) • Flask Routes & Blueprints • Authentication Controller • Dashboard & Schedule Controller • Task Controller • AI & Integration Controller	Handles all incoming requests from the client, interprets them, performs necessary application logic, and selects the appropriate View.
Model	MongoDB Database (Users, Courses, Enrollments, Schedules, Tasks, Notes, Summaries, Settings) Domain Models (User, Course, Schedule, Task, Summary Models) AI Service Storage (Optimizer & NLP Summary Output)	Manages the application's data, state, business rules, and logic for data manipulation. It is unaware of the View or Controller.

4.1.3 Responsibilities of Model, View, and Controller

Each element within the MVC structure has distinct, single-purpose responsibilities:

Model (Data and Business Logic Management)

The Model layer is composed of the persistent data storage (MongoDB) and the specialized AI processing services used by Unify.

- Data Persistence and Retrieval: MongoDB handles the storage, retrieval, and structural organization of all persistent data (Users, Courses, Enrollments, Schedules, Tasks, Notes, AI Summaries, Settings).

- State and Business Rules: It encapsulates the application’s state and enforces core business logic related to academic data integrity—such as schedule consistency, course enrollment rules, task relations, and summary assignments.

- AI Processing: The AI Services (Schedule Optimizer and NLP Summarizer) act as an extension of the Model layer, managing complex computational logic. They are responsible for generating optimized schedules, processing lecture notes, inferring summaries, and abstracting the complexity of AI algorithms away from the Controller.

View (User Interface)

The View is the entire Client Application (Web Browser), providing the visual and interactive element of the system.

- Presentation: It formats and presents the data retrieved from the Model (via the Controller) to the end-user, such as displaying weekly schedules, upcoming deadlines, tasks, summaries, and Google Calendar synchronization status.
- Input Capture: It captures all user interactions (clicks, form submissions, navigation, note uploads) and forwards these actions directly to the Controller for processing.
- Independence: The View is stateless and passive, containing no business logic or direct knowledge of data structures. It only displays what the Controller instructs it to display.

Controller (Request Handling and Orchestration)

The Controller is the Unify Core Platform (Flask), comprising the routing system, request handlers, and internal business logic modules.

- Request Routing: Flask routes receive HTTP requests from the Client and route them to the appropriate logic modules (e.g., Authentication, Schedule Logic, Task Logic, AI Summarization, Google Calendar Sync).
- Action Execution: Logic modules interpret the request, determine what action is required, and interact with the Model layer (MongoDB or AI Services) to fulfill that request. Examples include generating an optimized timetable, adding a task, retrieving course enrollments, or summarizing notes.
- Response Generation: After the Model updates or returns data, the Controller processes this information and sends an appropriate response back to the View (e.g., an updated schedule table, a task list, a summary result, a success message, or an error notification).

4.1.4 Interaction Between Components

The components interact in a specific, cyclical manner to maintain the separation of concerns.
This flow defines how user actions are processed and how the Unify application manages state and data.

The cycle begins:

User Interaction (View → Controller):

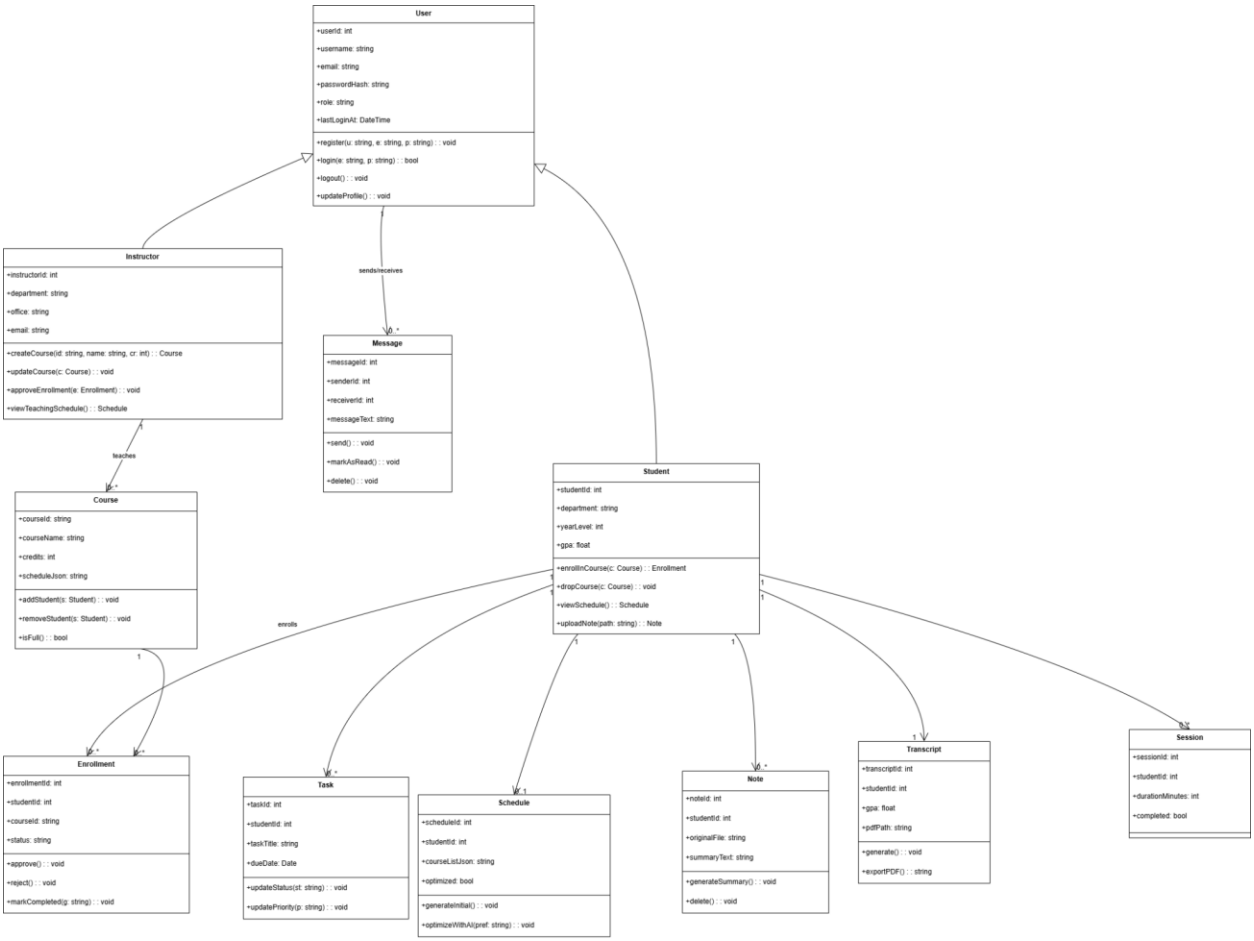
The user interacts with the View (Web Client), initiating an action such as logging in, adding a task, requesting a schedule optimization, uploading notes, or syncing with Google Calendar.
The View captures this input and sends a request (via HTTP/REST) to the appropriate Flask Controller.

Request Handling (Controller):
The Controller receives the request and, after authentication and validation, determines the required business operation.
This may involve accessing user data, generating schedules, retrieving tasks, or invoking AI services.

- Data/Logic Access (Controller → Model):
  The Controller calls upon the Model layer for data access or computational processing.
    - For saving/retrieving data, it interacts with the MongoDB collections (Users, Courses, Tasks, Notes, Schedules, Settings).
    - For AI-driven operations (e.g., schedule optimization, note summarization), it calls the AI Service layer.
- State Update (Model):
  The Model performs the requested operation (e.g., storing a task, generating a summary, retrieving events, computing an optimized schedule) and updates its internal state.
  The Model does not communicate with the View directly; it only returns structured data to the Controller.

View Update (Controller → View):
The Controller receives the result from the Model or AI Services.
It formats this data (e.g., JSON payload, refreshed data block, page render) and sends it back to the View.
The View then updates the user interface accordingly, completing the cycle.

4.2 UML Diagrams
4.2.2 Detailed Class Diagram



1. User

Description:
Represents a student or system user who logs into Unify. The User manages personal information, enrollments, tasks, notes, scheduling preferences, and integrations. Each user interacts with multiple system components and receives notifications.

Attributes:

- userId : string — Unique identifier for the user (MongoDB ObjectId).
- name : string — Full name of the user.
- email : string — Registered email address.

- passwordHash : string — Encrypted password.
- role : string — User role (student by default).
- preferences : dict — Personal settings (theme, language, notification options).

Methods:

- login() — Validates user credentials.
- logout() — Ends current user session.
- updateProfile() — Updates name, settings, or preferences.
- viewDashboard() — Retrieves dashboard items (schedule, tasks, summaries).
- receiveNotification() — Receives system notifications.

Relationships:

- 1..* association with Enrollment (user enrolls in many courses).
- 1..* association with Task (each user owns many tasks).
- 1..* association with Note (each user can store many notes).
- 1..* association with Summary (each user receives many summaries).
- 1..1 association with IntegrationSettings (external sync settings).
- 1..* association with ScheduleEvent (user schedule entries).

2. Course

Description:
Represents an academic course offered to students. Courses are used for enrollment management and schedule generation.

Attributes:

- courseId: string — Unique course identifier.

- code: string — Course code (e.g., MATH101).
- title: string — Course name.
- instructor: string — Instructor’s name.
- credits: int — Number of course credits.

Methods:

- getCourseInfo() — Returns course details.
- listStudents() — Lists all students enrolled.

Relationships:

- 1..* association with Enrollment (a course has many enrollments).
- 1..* association with ScheduleEvent (lectures/labs).
- 1..* association with Note (notes linked to a course).

3. Enrollment

Description:
Represents the relationship between a User and a Course. Each record indicates that a student is enrolled in a specific course for a given semester.

Attributes:

- enrollmentId: string — Unique identifier.
- userId: string — ID of the enrolled student.
- courseId: string — ID of the course enrolled.
- semester: string — Semester code (e.g., Fall 2025).

Methods:

- register() — Enrolls user in a course.
- drop() — Removes enrollment from the course.

Relationships:

- 1 association with User.
- 1 association with Course.

4. ScheduleEvent

Description:
Represents a single calendar entry for a student (lecture, lab, exam, custom event). Appears on the student's weekly schedule.

Attributes:

- eventId: string — Unique identifier.
- userId: string — Owner of the event.
- courseId: string — Associated course (if any).
- type: string — Lecture, Lab, Exam, or Custom.
- startTime : datetime — Starting time.
- endTime: datetime — Ending time.
- location : string — Room or link.

Methods:

- updateEvent() — Edits the schedule entry.
- deleteEvent() — Removes schedule entry.

Relationships:

- Many-to-one with User.
- Optional association with Course.

5. Task

Description:
Represents a user's academic or personal task, such as assignments, deadlines, study items, or reminders.

Attributes:

- taskId : string — Unique ID.
- userId : string — Owner of the task.
- title : string — Task title.
- description : string — Detailed description.
- dueDate : datetime — Deadline.
- status : string — To-Do, In Progress, Done.
- priority : int — Priority level.
- relatedCourseId : string — Optional linked course.

Methods:

- createTask() — Creates a new task.
- updateTask() — Edits task attributes.
- markComplete() — Marks task as done.

Relationships:

- 1..1 association with User.
- Optional association with Course

6. Note

Description:
Represents text notes written or uploaded by the user. Notes can be passed to the AI Summarizer.

Attributes:

- noteId: string
- userId: string
- courseId: string
- rawContent: string
- createdAt: datetime

Methods:

- uploadNote() — Stores new note text.
- deleteNote() — Removes the note.
- summarize() — Sends note to AI summarizer.

Relationships:

- 1..* association with Summary.
- Many-to-one with User.
- Many-to-one with Course.

7. Summary

Description:
Represents an AI-generated summary created from a Note. Summaries provide concise explanations for study purposes.

Attributes:

- summaryId: string
- noteId: string
- userId: string
- summarizedContent: string
- createdAt: datetime

Methods:

- generateSummary() — Produces summary text.
- viewSummary() — Displays summary to the student.

Relationships:

- 1..1 association with Note.
- 1..1 association with User.

8. Notification

Description:
Stores system alerts, reminders, AI suggestions, and important updates sent to the user.

Attributes:

- notificationId: string
- userId: string
- message: string
- type: string
- createdAt: datetime
- isRead: bool

Methods:

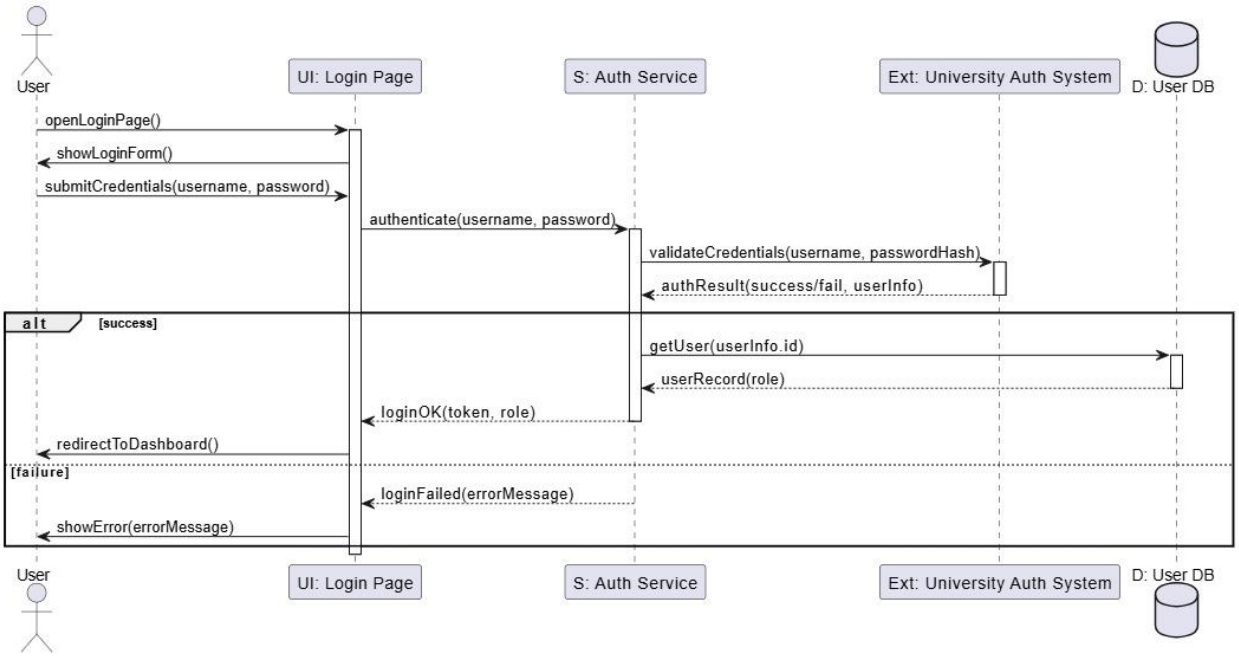
- sendNotification() — Pushes a notification to the user.
- markAsRead() — Marks notification as viewed.

Relationships:

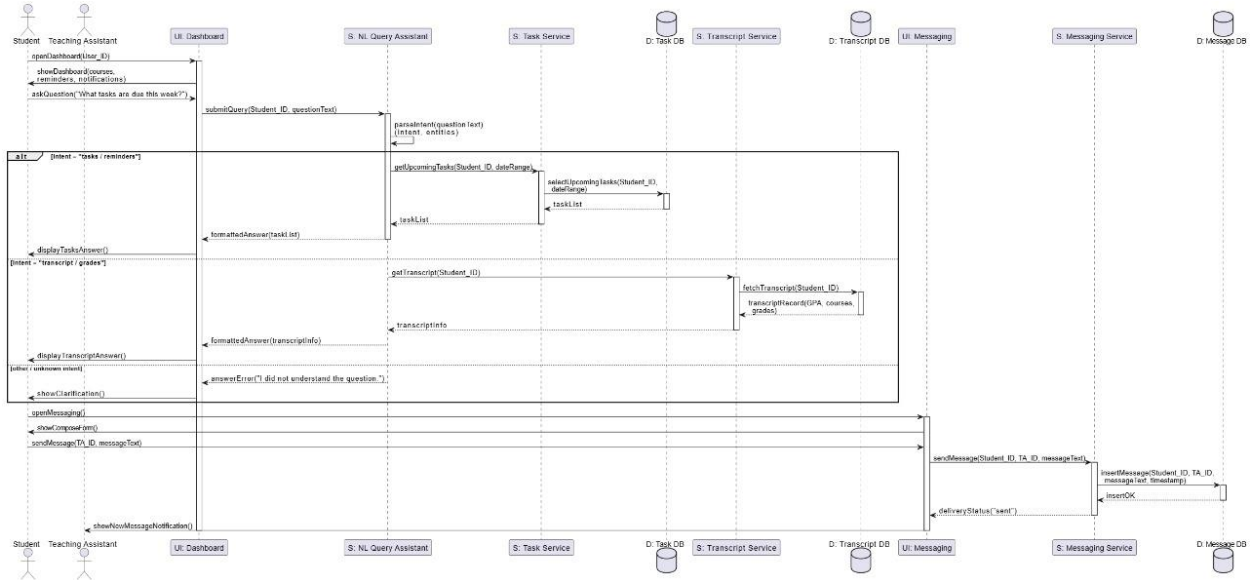
- 1..1 association with User.

4.2.3 Sequence Diagrams

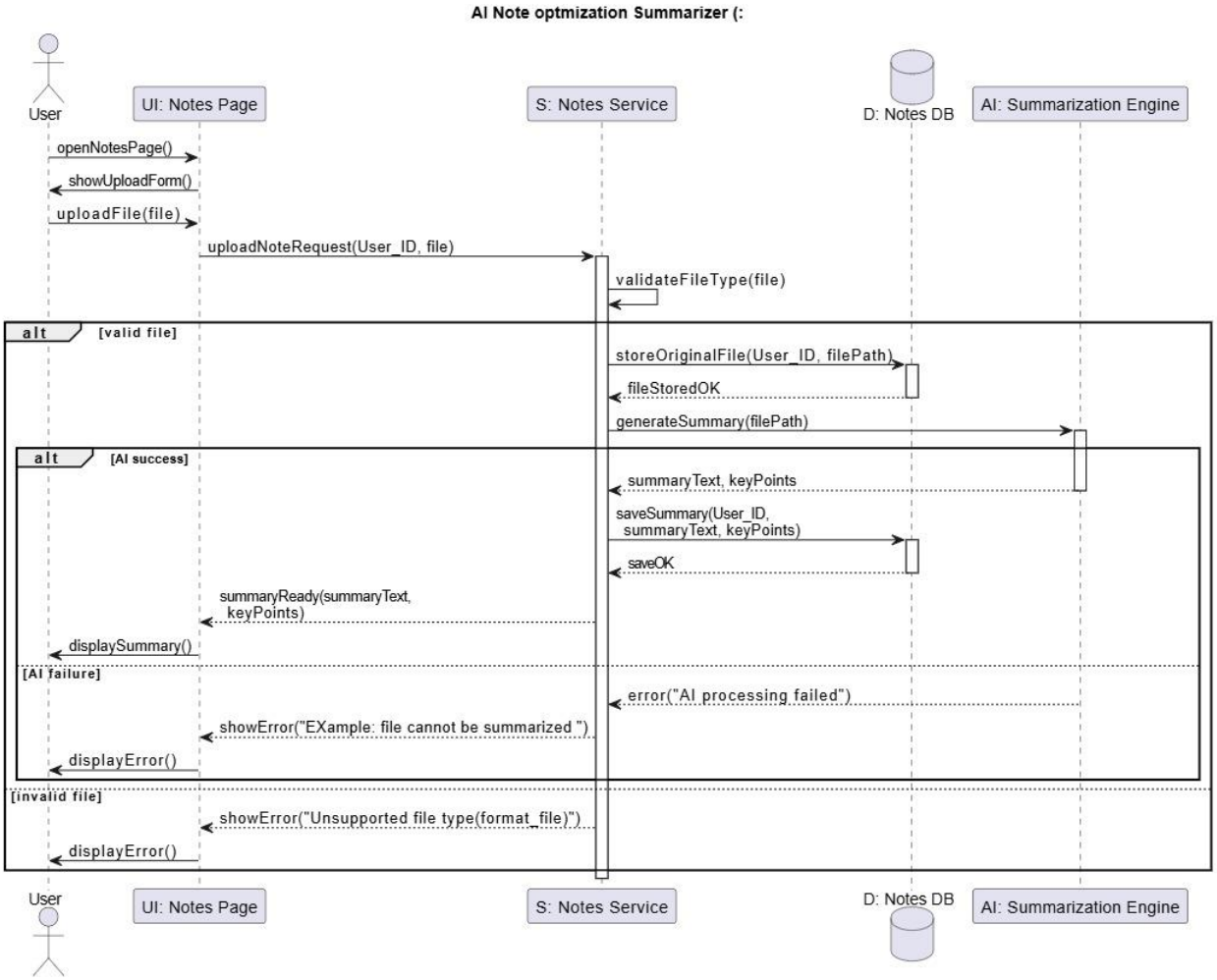
Sequence Diagram 1



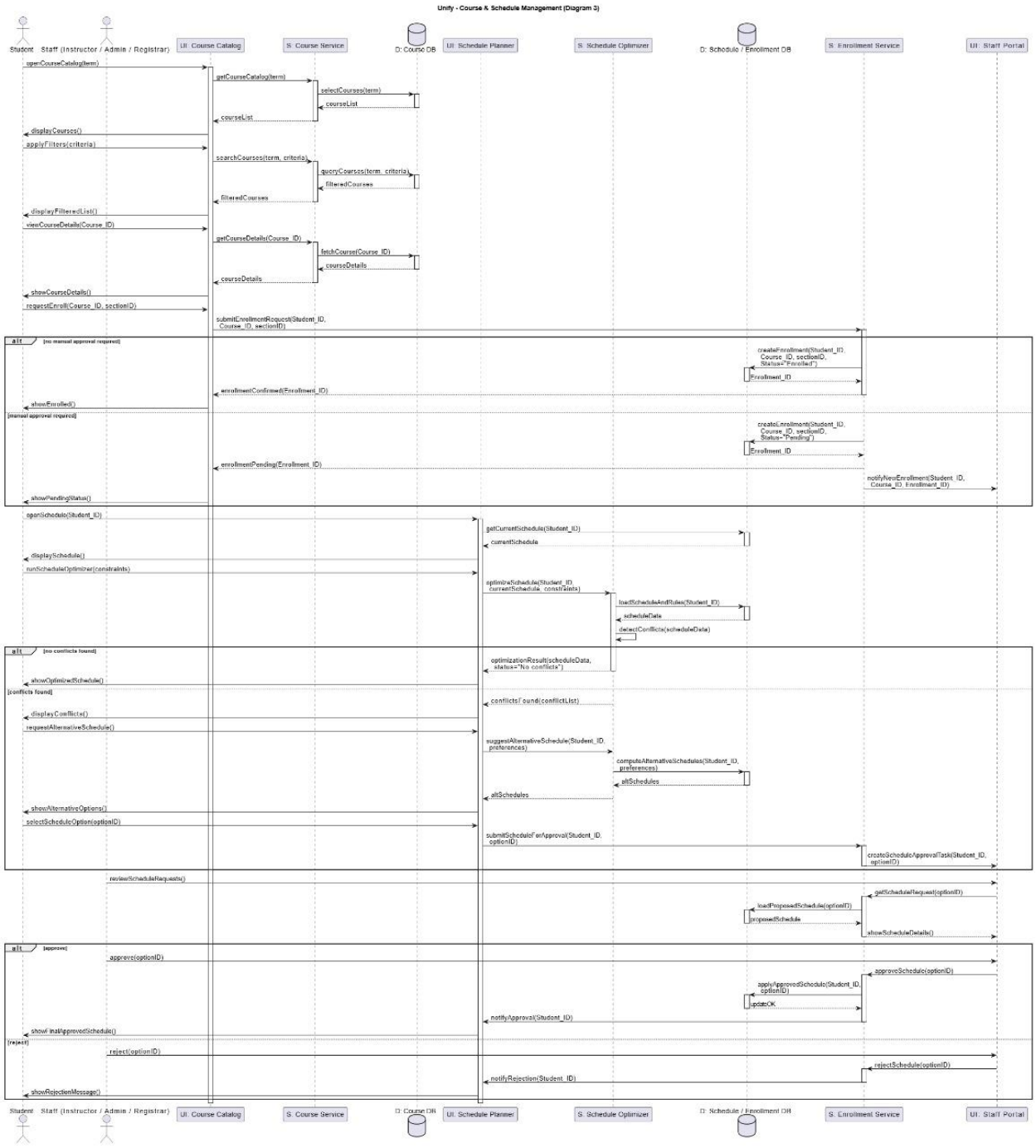
Sequence Diagram 2



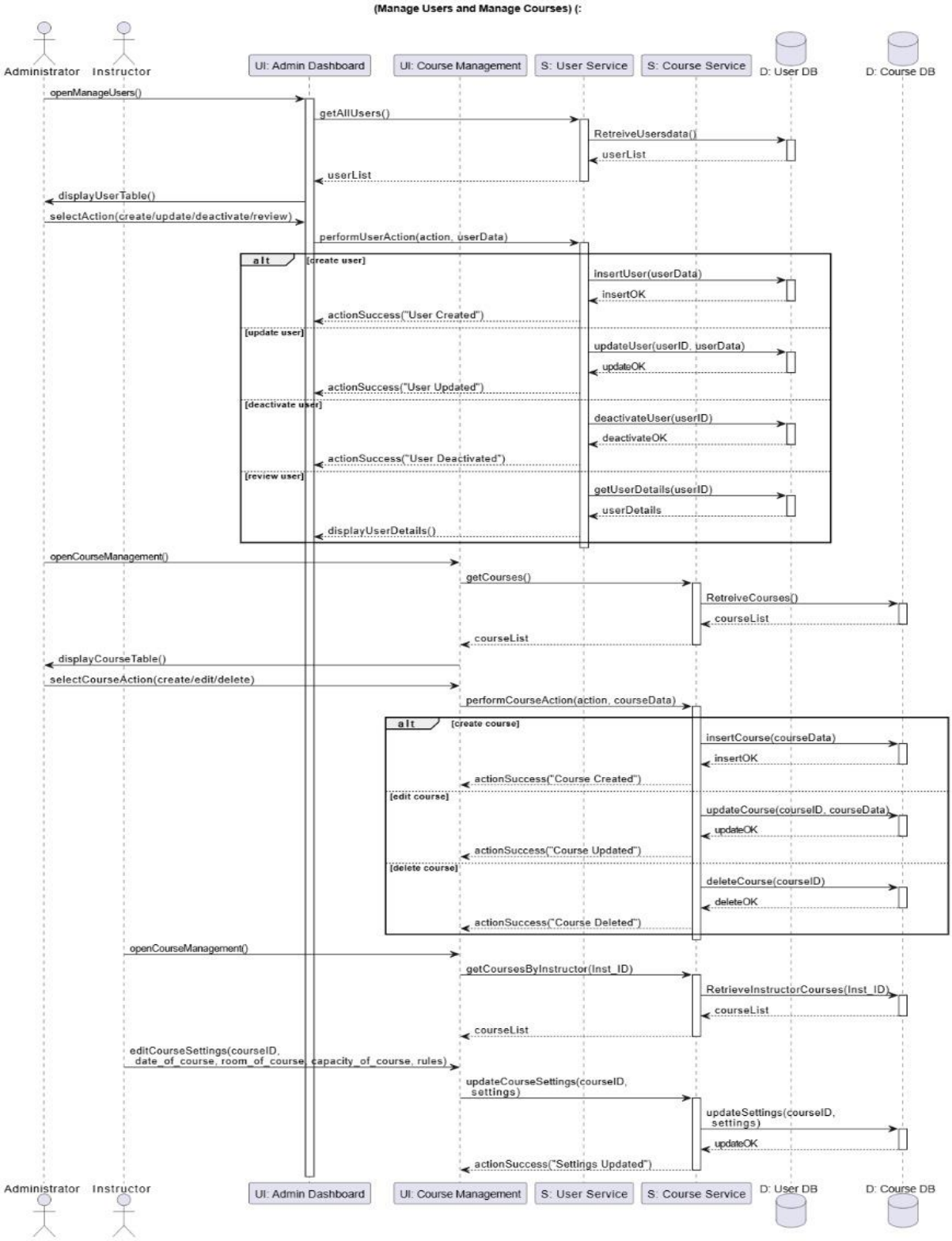
Sequence Diagram 3



Sequence Diagram 4

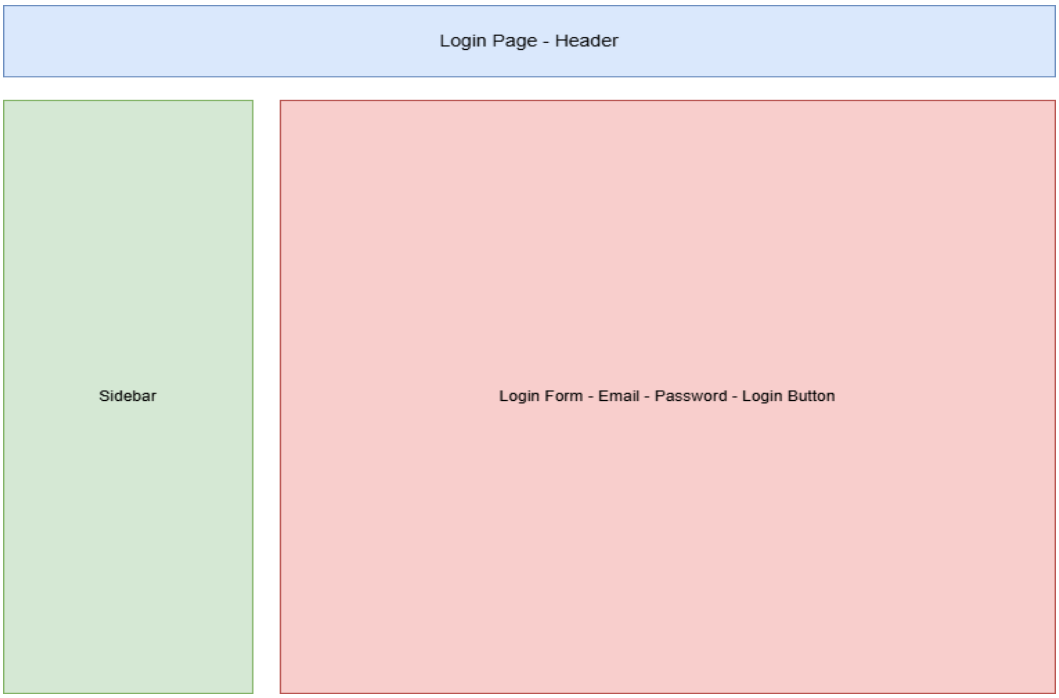


Sequence Diagram 5

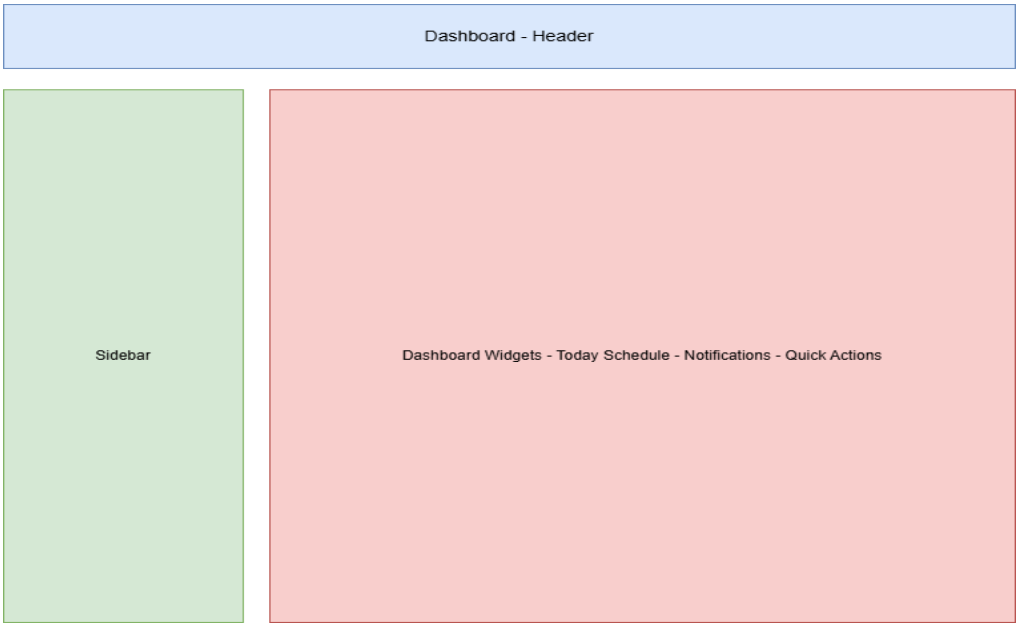


4.3 UI/UX Design

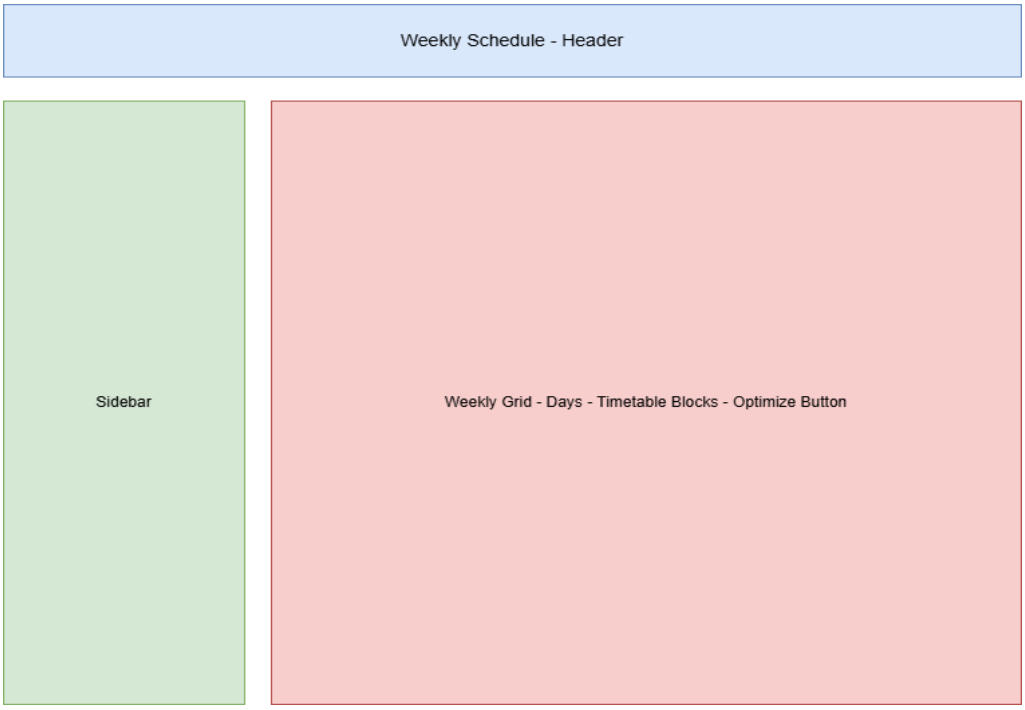
Wireframe 1 – Login Page



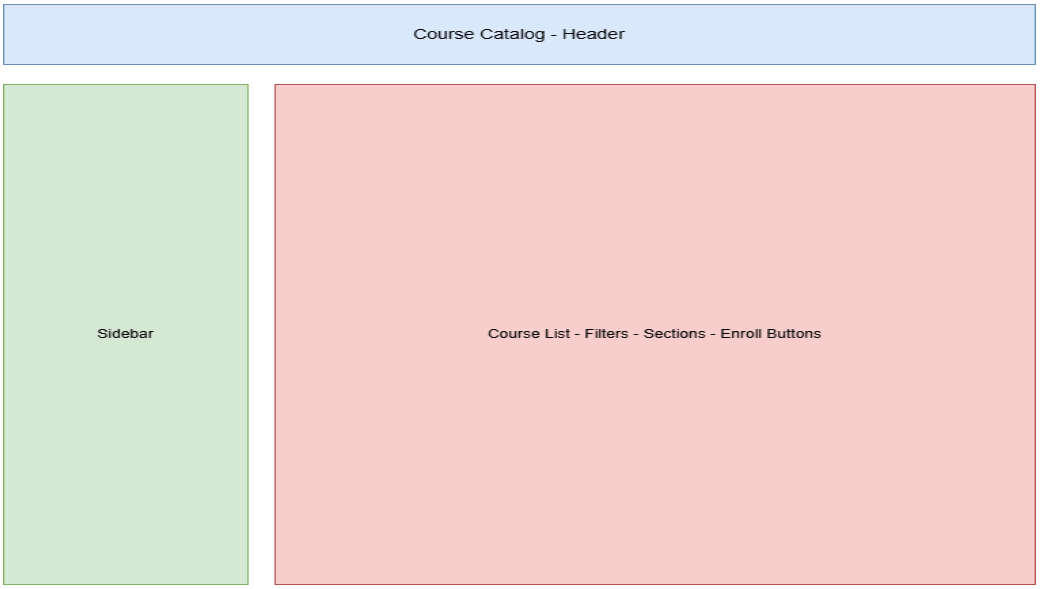
Wireframe 2 – Dashboard Page.



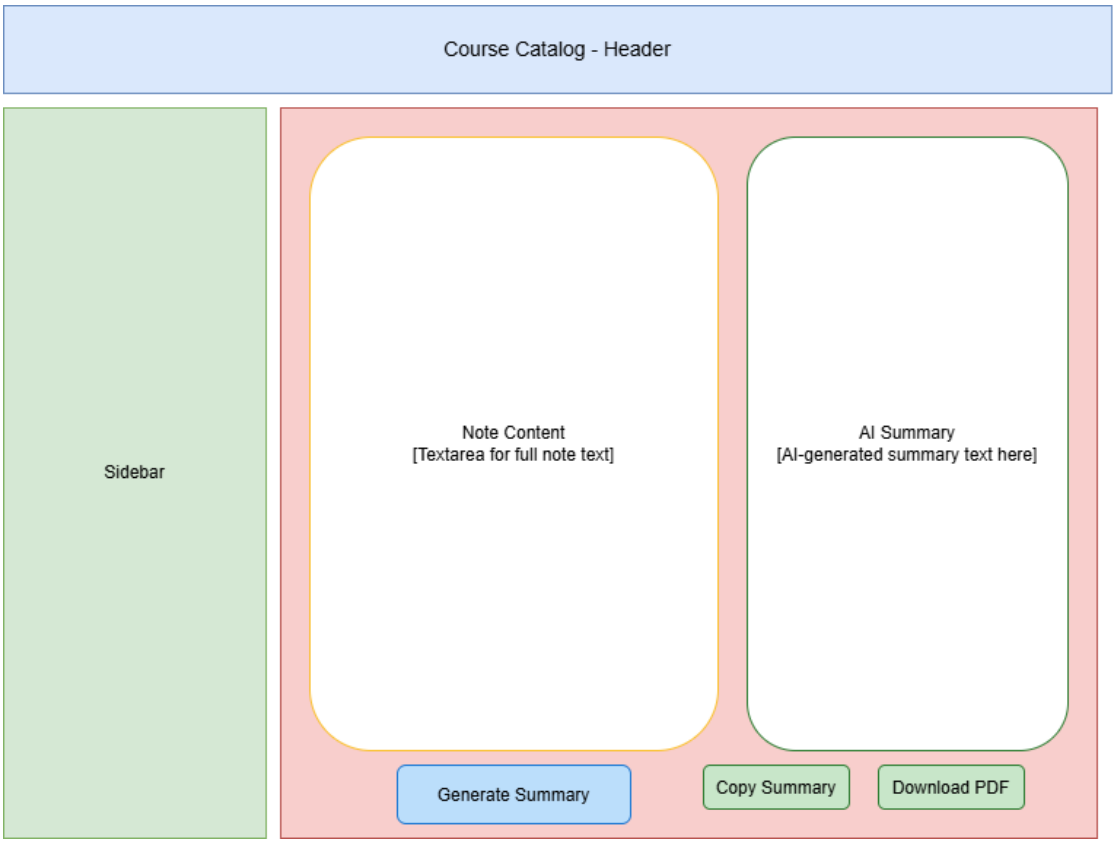
Wireframe 3 – Weekly Schedule Page



Wireframe 4 – Course Catalog Page

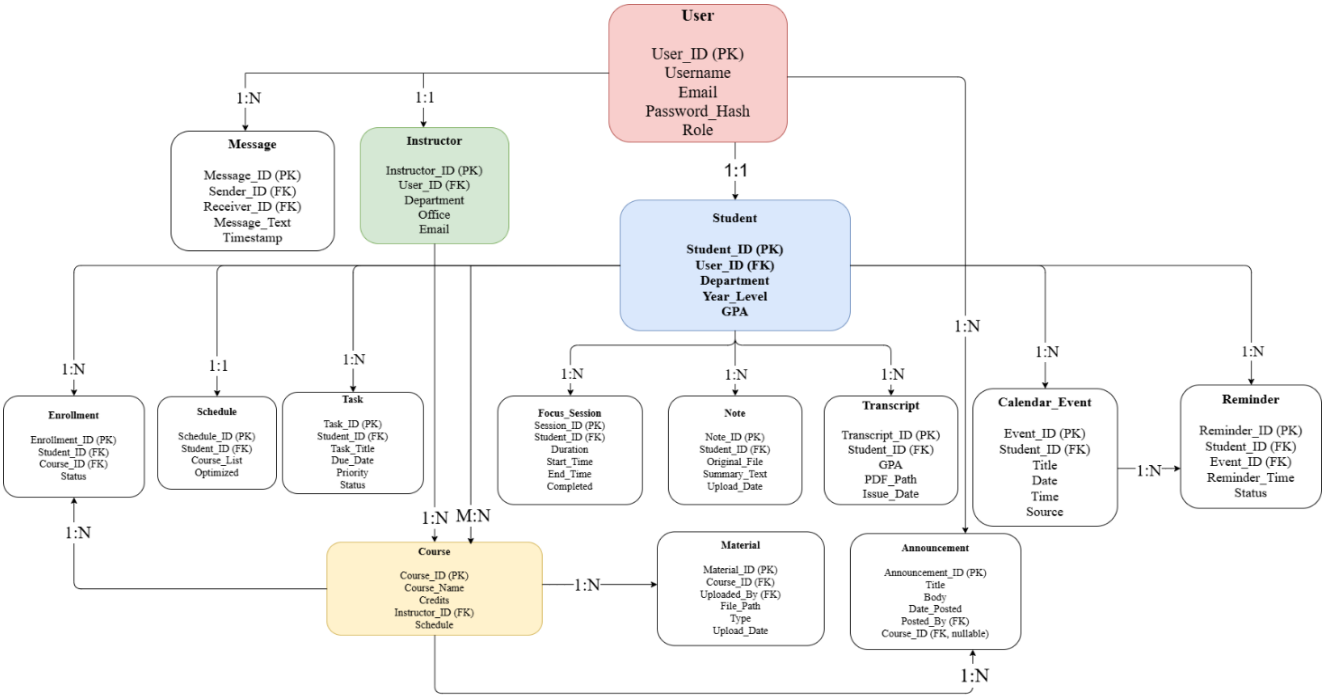


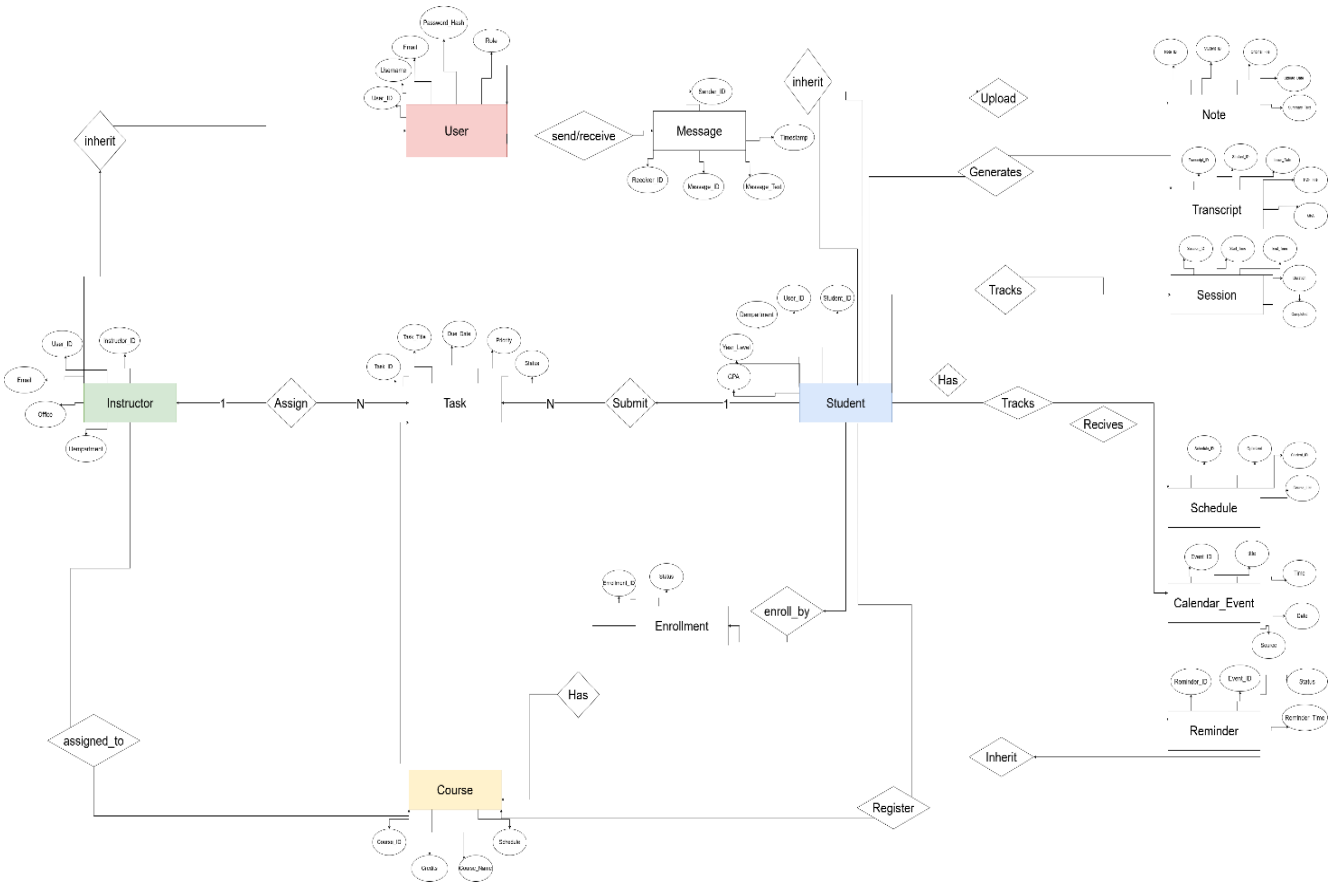
Wireframe 5 – AI Note Summarizer Page



4.4 Data Design

4.4.1 Database Schema / ER Diagram





4.4.3 Data Dictionary

Student	Student_ID	Integer / ObjectId	Unique identifier	PK
	User_ID	Integer / ObjectId	References User	FK
	Department	String	Student department	Optional
	Year_Level	Integer	Year of study	1–5
	GPA	Float	Current GPA	0.0–4.0

Instructor	Instructor_ID	Integer / ObjectId	Unique identifier	PK
	User_ID	Integer / ObjectId	References User	FK
	Department	String	Instructor department	Optional
	Office	String	Office location	Optional
	Email	String	Optional, can differ from User email	Unique

Course	Course_ID	Integer / ObjectId	Unique course ID	PK
	Course_Name	String	Course title	Required
	Credits	Integer	Number of credit hours	Required
	Instructor_ID	Integer / ObjectId	FK to Instructor	Required
	Schedule	JSON	Class schedule (days, times)	Optional

Enrollment	Enrollment_ID	Integer / ObjectId	PK	Unique enrollment record

	Student_ID	Integer / ObjectId	FK to Student	Required
	Course_ID	Integer / ObjectId	FK to Course	Required
	Status	Enum	enrolled/dropped/completed	Required

Task	Task_ID	Integer / ObjectId	PK	Unique task
	Student_ID	Integer / ObjectId	FK to Student	Required
	Task_Title	String	Task description	Required
	Due_Date	DateTime	Deadline	Required
	Priority	Enum	low/medium/high	Optional
	Status	Enum	pending/completed	Required

Schedule	Schedule_ID	Integer / ObjectId	PK	Unique schedule
	Student_ID	Integer / ObjectId	FK to Student	Required
	Course_List	JSON	List of course IDs	Optional
	Optimized	Boolean	Whether schedule is optimized	Default=false

Note	Note_ID	Integer ObjectId	PK	Unique note
	Student_ID	Integer ObjectId	FK to Student	Required
	Original_File	String	Path to uploaded file	Required
	Summary_Text	Text	AI-generated or manual summary	Optional
	Upload_Date	DateTime	Date uploaded	Required

Message	Message_ID	Integer ObjectId	PK	Unique message
	Sender_ID	Integer ObjectId	FK to User	Required
	Receiver_ID	Integer ObjectId	FK to User	Required
	Message_Text	Text	Content of message	Required

	Timestamp	DateTime	Time sent	Required

Transcript	Transcript_ID	Integer ObjectId	PK	Unique transcript
	Student_ID	Integer ObjectId	FK to Student	Required
	GPA	Float	Student GPA	Optional
	PDF_Path	String	Path to transcript PDF	Required
	Issue_Date	DateTime	Date issued	Required

calendar	Event_ID	Integer ObjectId	PK	Unique event
	Student_ID	Integer ObjectId	FK to Student	Required
	Title	String	Event title	Required
	Date	Date	Event date	Required
	Time	Time	Event time	Required
	Source	Enum/String	Source: system/manual	Optional

Reminder	Reminder_ID	Integer ObjectId	PK	Unique reminder
	Student_ID	Integer ObjectId	FK to Student	Required
	Event_ID	Integer ObjectId	FK to Calendar_Event	Required
	Reminder_Time	DateTime	When reminder triggers	Required
	Status	Enum	pending/done	Default=pending

Focus_Session	Session_ID	Integer ObjectId	PK	Unique session
	Student_ID	Integer ObjectId	FK to Student	Required
	Duration	Integer	Duration in minutes	Required
	Start_Time	DateTime	Start of session	Required

	End_Time	DateTime	End of session	Optional
	Completed	Boolean	Whether session completed	Default=false

Announcement	Announcement_ID	Integer / ObjectId	PK	Unique announcement
	Title	String	Announcement title	Required
	Body	Text	Announcement content	Required
	Date_Posted	DateTime	Required	
	Posted_By	Integer / ObjectId	FK to User	Required
	Course_ID	Integer / ObjectId	FK to Course	Optional (nullable)

Material	Material_ID	Integer / ObjectId	PK	Unique material
	Course_ID	Integer / ObjectId	FK to Course	Required
	Uploaded_By	Integer / ObjectId	FK to User	Required
	File_Path	String	File location	Required
	Type	Enum/String	PDF/Doc/Video	Required
	Upload_Date	DateTime	Required	

5. Conclusion

5.1 Summary of Design Phase

The Design Phase Document (DPD) for Unify The AI-Enhanced Student Assistant System successfully transformed the detailed functional and non-functional requirements defined in the Software Requirements Specification (SRS) into a complete, structured, and implementation-ready technical blueprint. This Design Phase establishes a clear architectural foundation that supports Unify's primary mission: providing students with a unified academic dashboard powered by intelligent automation, integrated scheduling, personalized task management, and AI-driven learning support.

Unify adopts a monolithic Flask-based architecture organized according to the Model–View–Controller (MVC) pattern, ensuring clean separation of concerns, scalability, and

maintainability. The Flask Controllers orchestrate all system flows, coordinating interactions between the Web Client (View) developed using HTML, CSS, and JavaScript and the Model layer, which includes MongoDB for persistent data and optional AI/NLP components for schedule optimization and note summarization. This architecture simplifies deployment, accelerates development, and ensures that future enhancements (e.g., additional AI features or analytics modules) can be integrated with minimal disruption.

The confirmed technology stack includes Python (Flask) for backend business logic, MongoDB as the primary storage engine for users, enrollments, schedules, tasks, notes, and AI summary content, and optional integration with external services, such as the Google Calendar API for synchronization and an internal NLP/AI engine for summarization and optimization services. This stack aligns with system constraints, supports Unify's performance requirements, and ensures high accessibility for student use.

To ensure clarity and maintainability, the MVC design was validated through multiple detailed UML artifacts. These include the System Architecture Diagram, which visually represents communication between the View, Controller, Model, and optional AI/Integration layers; a comprehensive UML Class Diagram defining the system's core entities (User, Course, Enrollment, Task, Note, Summary, ScheduleEvent, Notification, IntegrationSettings); and detailed Sequence Diagrams capturing essential user flows such as Login, Task Management, Schedule Generation, Note Summarization, and Google Calendar Synchronization. These diagrams collectively demonstrate how requests flow across system layers and how internal data states update in response to user actions.

Finally, UI/UX Wireframes and Mockups were produced to illustrate the main user-facing screens, including Login, Dashboard, Weekly Schedule, Task Manager, and AI Summary pages. These visual models ensure that the implementation phase begins with a clear understanding of layout, component structure, and user interactions.

This Design Phase Document serves as the complete, approved technical foundation for Unify. All core architectural decisions, entity models, interaction flows, and interface designs have been validated, enabling immediate progression into the Implementation Phase with confidence and alignment across all development stakeholders.