NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY POLITEHNICA BUCHAREST

ADVANCED CONCEPTS IN SOFTWARE ENGINEERING

BrightPath E-Learning App

Software Design Document

Team:
Dumitrof Dan Stefan MSE
Godinel Daria Teodora MSE
Ion Cristina Gabriela MSE
Stavarache Mihaela Cristina MSE
Toma Dana Georgiana MSE

Coordinator:
Prof. dr. ing. Nicolae Goga

Last updated:
Tuesday, January 14, 2025

Table of Contents

System Design

According to the IEEE STD-1016-1998, *IEEE Recommended Practice for Software Design Descriptions*.

1. Introduction

1.1. Purpose

This document aims to present the detailed design of the **E-learning Platform** software application. The application is designed to assist students and professors in managing educational content, facilitating communication, and tracking progress. By providing a centralized platform, it supports learning, collaboration, and efficient course management.
This document presents the modular architecture, key features, data models, and system dependencies required to achieve these objectives.

1.2. Target Public

The target public consists of:

- Students
- Professors
- Educational Institutions
- Technical Stakeholders

1.3. Definitions, Acronyms and Abbreviations

- **CRUD**: Create, Read, Update, Delete – standard operations for interacting with a database.
- **SQLAlchemy**: A Python ORM (Object-Relational Mapper) for managing database operations.
- **Flask**: A lightweight web framework for building Python-based applications.
- **Flask-Babel**: A Flask extension for supporting translations and localization.
- **REST**: Representational State Transfer – an architectural style for designing networked applications.
- **Authentication**: The process of verifying a user's identity.
- **Authorization**: The process of verifying what actions a user is allowed to perform.

- **Role-Based Access Control (RBAC)**: An approach to restricting system access based on user roles (e.g., Student or Professor).
- **Password Hashing**: A security method for securely storing user passwords using hashing algorithms (e.g., bcrypt).
- **Blueprint**: A Flask feature that allows modularization of application components, such as routes.
- **Database**: A structured set of data stored and managed for use by the system (e.g., SQLite).
- **Data Model**: The logical structure of data, defining relationships and properties of entities in the system.
- **Entity**: A representation of a real-world object or concept (e.g., User, Course, Enrollment).
- **Chat System**: A feature enabling real-time communication between students and professors.
- **User Interface (UI)**: The visual elements that users interact with, designed for accessibility and ease of use.
- **Localization**: Adapting the application for different languages and regions.
- **Flash Messages**: Short messages displayed to inform users of the success or failure of actions.
- **Version Control System (VCS)**: A system for tracking and managing code changes (e.g., Git).
- **Responsive Design**: Ensuring the website functions well on various devices (e.g., mobile, tablet, desktop).
- **Object-Relational Mapping (ORM)**: A programming technique for mapping objects to database tables (e.g., SQLAlchemy).

1.4. Document Structure

This document is structured as follows:

Chapter 1: Introduction: Overview of the purpose, target audience, and structure.

Chapter 2: References: Lists the resources and standards referenced in the document.

Chapter 3: Decomposition Description: Describes the modules structure and key components.

Chapter 4: Dependency Description: Details dependencies among modules, processes, and data modules.

Chapter 5: Interface Description: Describes specifications for module and process interfaces.

Chapter 6: Detailed Design: Elaborates on each module and data module, including classes and methods.

Appendices: Includes use case, class, and sequence diagrams, as well as conclusions on the activity.

2. References

List of references used in this paper, possible legislation that governs the application domain of the project/document.

[1] IEEE STD-1016-1998, *IEEE Recommended Practice for Software Design Descriptions*

[2] Spring Framework Documentation: Official documentation for the Spring Boot Framework.

[3] MySQL Reference Manual: Official documentation for the MySQL database.

[4] Thymeleaf Documentation: Official guide for the Thymeleaf template engine.

3. Decomposition Description

3.1. Modules Description

3.1.1. Description of Module 1

Name	User Registration Module
Type	Code Module
Purpose	Manages the registration of new users, allowing them to create accounts by submitting their username, password, email, and role (Student or Professor).
Way of Operating	- The user submits a registration form containing their username, email, password, and role. - auth.py processes the form data. - A new User object is created, and the password is hashed using a secure hashing algorithm. - The user role is assigned using the provided input. - The user data is saved in the database using SQLAlchemy. - Upon successful registration, the system redirects the user to the login page.
Subordination	Subordinate to the Core Module that initializes the application context.
Dependencies	- Flask routes - SQLAlchemy for database operations - Password hashing library (e.g., bcrypt)
Resources	- Flask (Core, Auth) - SQLite Database (User Data Storage) - Password Hashing for secure storage

3.1.2. Description of Module 2

Name	Login Module
Type	Code Module
Purpose	Authenticates users by verifying their credentials and granting access to protected sections of the e-learning platform.
Way of Operating	- The user submits their username and password through the login form. - auth.py fetches the user record from the database based on the username. - The submitted password is compared against the hashed password stored in the database. - If the credentials are valid, the user session is initiated, and the user is redirected to their dashboard. - If invalid, an error message is displayed.
Subordination	Subordinate to the Core Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - Password hashing library (e.g., bcrypt)
Resources	- Flask (Core, Auth) - SQLite Database

3.1.3. Description of Module 3

Name	User Management Module (Admin)
Type	Code Module
Purpose	Allows administrators to manage users by performing actions such as adding, updating, or removing user accounts.
Way of Operating	-Adding: - The admin submits a request through a form containing user details (username, email, password, and role). -The request is processed in auth.py, where the password is hashed, and a new User object is created. -The user is saved in the database using SQLAlchemy. -A success or error message is displayed. -Removal: -The admin selects a user to remove by their unique ID. -The request is processed, and the user is removed from the database. - A success or error message is displayed.
Subordination	Subordinate to the Core Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core, Admin) - SQLite Database

3.1.4. Description of Module 4

Name	Course Management Module
Type	Code Module
Purpose	Manages adding, updating, and removing courses within the platform.
Way of Operating	-Adding: – Professors or admins enter course details (name, description, language, category, and content) through a form. – The course details are validated and saved in the database. – Enrollment: – Students select a course to enroll in from the available courses list. – Enrollment data is stored in the database linking the student to the course. -Viewing: – Enrolled students can access the content of their courses via a dedicated page.
Subordination	Subordinate to the Core Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core, Courses) - SQLite Database

3.1.5. Description of Module 5

Name	Communication Module
Type	Code Module
Purpose	Facilitates communication between students and professors through chats and forums.
Way of Operating	- Users initiate a chat by selecting a recipient (student or professor). - views.py handles chat initiation, checking if a chat session exists, and creating one if needed. - Messages are sent and saved in the database. - Users view chat history retrieved from the database in the chat interface.
Subordination	Subordinate to the Core Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - Chat and Message models
Resources	- Flask (Core, Chat) - SQLite Database

3.1.6. Description of Module 6

Name	Reporting Module
Type	Code Module
Purpose	Generates analytics and reports on user activity, course progress, and overall platform usage.
Way of Operating	- views.py queries the database for relevant data such as course completion, enrollment statistics, and user activity. - Aggregated data is processed and formatted into visual reports. - Reports are displayed on the dashboard or exported as downloadable files (e.g., PDFs).
Subordination	Subordinate to the Core Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core, Reports) - SQLite Database

3.1.7 Description of Module 7

Name	Achievements and Certificates Module
Type	Code Module
Purpose	Tracks student achievements and generates certificates for course completions or milestones.
Way of Operating	- Students view their achievements and certificates on a dedicated "Achievements" page. - Milestones such as completing a course, earning a high quiz score, or completing a specialization are tracked. - Certificates are generated for specific achievements, formatted as downloadable files (e.g., PDF). - The system queries the database to fetch and display earned badges, certificates, and pending achievements.
Subordination	Subordinate to the Reporting Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - Badge and Certificate models
Resources	- Flask (Core, Reports) - SQLite Database

3.1.8 Description of Module 8

Name	Content Delivery Module
Type	Code Module
Purpose	Handles the distribution of course materials such as videos, documents, and other learning resources.
Way of Operating	- Professors upload course content, which is stored and linked to their respective courses in the database. - Enrolled students can access course materials through the course detail page. - Content is served dynamically based on user permissions (students can only view content for their enrolled courses). - The module supports various file types (e.g., PDFs, videos, images) for flexible content delivery.
Subordination	Subordinate to the Course Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - File storage or cloud storage integration (if applicable)
Resources	- Flask (Core, Reports) - SQLite Database

3.1.9 Description of Module 9

Name	Assessment and Evaluation Module
Type	Code Module
Purpose	Manages quizzes, tests, and assignments to evaluate student performance.
Way of Operating	- Professors create quizzes or assignments and upload them to their courses. - Students access assessments through the course content page. - Submissions are stored in the database and evaluated automatically (for quizzes) or manually (for assignments). - Results are displayed to students and logged for professors to review.
Subordination	Subordinate to the Course Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - Assessment models (Quiz, Assignment, Submission)
Resources	- Flask (Core, Reports) - SQLite Database

3.1.10 Description of Module 10

Name	Localization and Language Support Module
Type	Code Module
Purpose	Provides translation and localization support to make the platform accessible in multiple languages.
Way of Operating	- The user's language preference is detected automatically (browser settings) or can be set manually. - Translations for all text displayed on the platform are managed using Flask-Babel. - Language resources are stored in .po and .mo files in a dedicated translations directory. - Users can toggle between supported languages (e.g., English, Romanian, Italian, Chinese, Japanese, Zulu, Hausa).
Subordination	Subordinate to the Core Module.
Dependencies	- Flask-Babel for translation - Language resource files (.po, .mo)
Resources	- Flask (Core, Reports) - SQLite Database

3.2. Description of Concurrent Processes

3.2.1. Description of Process 1

Name	Add User (Admin)
Type	Process
Purpose	Allows administrators to add new users (students or professors) to the platform.
Way of operating	• The Admin submits a form with user details (username, email, password, and role). • The request is processed by the auth.py module. • A User object is created, and the password is hashed using a secure algorithm. • The user is saved in the database. • A success or error message is displayed based on the outcome.
Subordination	Subordinate to the User Management Module.
Dependencies	- Controllers (via Flask route handling) - SQLAlchemy for database operations
Resources	- Flask (Core, Admin) - SQLite Database

3.2.3. Description of Process 2

Name	Remove User (Admin)
Type	Process
Purpose	Allows administrators to remove existing users from the platform.
Way of operating	● The Admin selects a user to remove. ● The request is processed via Flask routes, and the user's ID is used to query the database. ● If the user exists, their entry is deleted from the database. ● A success or error message is displayed.
Subordination	Subordinate to the User Management Module.
Dependencies	- Flask for route handling - SQLAlchemy for database operations
Resources	- Flask (Core, Admin) - SQLite Database

3.2.3. Description of Process 3

Name	User Registration
Type	Process
Purpose	Allows new users to register on the platform.
Way of operating	● The user fills out a registration form and submits it. ● The request is processed in auth.py. ● The data is validated, and the password is hashed. ● A new User entry is created and saved in the database. ● The user is redirected to the login page upon successful registration.
Subordination	Subordinate to the User Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core) - SQLite Database

3.2.4. Description of Process 4

Name	User Login
Type	Process
Purpose	Verifies user credentials and grants access to the platform.
Way of operating	● The user submits login credentials via the login page. ● The request is processed, and the user is retrieved from the database based on the submitted username. ● The password is validated using a hashing algorithm. ● If valid, the user is logged in, and a session is initiated. ● If invalid, an error message is displayed.
Subordination	Subordinate to the User Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core, Login) - SQLite Database

3.2.5. Description of Process 5

Name	Enroll in Course
Type	Process
Purpose	Allows students to enroll in available courses
Way of operating	● The student selects a course from the list of available courses. ● The enrollment request is processed in views.py, where a new Enrollment entry is created. ● The entry links the student's ID with the course ID in the database. ● A success message is displayed, and the student sees the course in their enrolled courses list.
Subordination	Subordinate to the Course Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core, Courses) - SQLite Database

3.2.6. Description of Process 6

Name	Unroll from Course
Type	Process
Purpose	Allows students to unenroll from courses they are no longer interested in.
Way of operating	● The student selects an enrolled course to unroll from. ● The request is processed in views.py, where the Enrollment entry linking the student and the course is deleted. ● A success message is displayed, and the course is removed from their enrolled courses list.
Subordination	Subordinate to the Course Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core, Courses) - SQLite Database

3.2.7. Description of Process 7

Name	View Course Content
Type	Process
Purpose	Allows students to view the content of a course they are enrolled in.
Way of operating	● A student selects an enrolled course from their course list. ● The request is processed in views.py, and the course content is retrieved from the database. ● The course content is displayed on a separate page.
Subordination	Subordinate to the Course Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations
Resources	- Flask (Core, Courses) - SQLite Database

3.2.8. Description of Process 8

Name	Start Chat
Type	Process
Purpose	Enables students to start a chat with professors or professors to initiate a chat with students.
Way of operating	● The user selects a recipient from the list of professors (if a student) or students (if a professor). ● The request is processed in views.py, where the system checks if a chat session already exists between the two users. ● If no chat session exists, a new Chat entry is created in the database. ● The chat interface is displayed, allowing the users to exchange messages.
Subordination	Subordinate to the Communication Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - Chat database model
Resources	- Flask (Core, Enrollment) - SQLite Database

3.2.9. Description of Process 9

Name	Send Message
Type	Process
Purpose	Allows users to send messages to one another within an ongoing chat session.
Way of operating	● The user types a message and submits it via the chat interface. ● The request is processed in views.py, where a new Message entry is created in the database, linking the message to the corresponding chat session. ● The message is displayed in the chat interface for both participants in real time. ● The system retrieves all messages for the chat to ensure the conversation is up to date.
Subordination	Subordinate to the Communication Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - Chat and Message database models
Resources	- **Flask (Core)** - **SQLite Database**

3.2.10. Description of Process 10

Name	Upload Course Content (Professor)
Type	Process
Purpose	Allows professors to upload learning materials (e.g., text, images, videos) to their courses.
Way of operating	● The professor navigates to the "Upload" section of their course management interface. ● The professor fills out the form with course content details (e.g., topic, description, file upload). ● The request is processed in views.py, where the content is validated and saved in the database as part of the Course object. ● The course content is then linked to the respective course and becomes available for enrolled students to access. ● A success message is displayed upon completion.
Subordination	Subordinate to the Course Management Module.
Dependencies	- Flask routes - SQLAlchemy for database operations - File storage (if applicable)
Resources	- Flask (Core) - SQLite Database

3.2.11. Description of Process 11

Name	View Achievements and Certificates (Student)
Type	Process
Purpose	Allows students to view their achievements and earned certificates.
Way of operating	● The student navigates to the "Achievements" page. ● The request is processed in views.py, where the system retrieves data from the database related to the student's completed courses, badges, and certificates. ● The achievements and certificates are displayed on the page, with options to download or print certificates (if applicable).
Subordination	Subordinate to the Reporting Module
Dependencies	- Flask routes - SQLAlchemy for database operations - Badge and certificate database models
Resources	- Flask (Core) - SQLite Database

3.3. Description of Data Modules

3.3.1. Description of Data Module 1

Name	User Data Module
Type	Data module
Purpose	Manages the structure and persistence of user data, including attributes such as username, email, password, and role (Student or Professor).
Way of operating	• The User entity represents user data, which is manipulated through SQLAlchemy models. • CRUD operations (Create, Read, Update, Delete) are performed on the users table in the SQLite database. • Passwords are securely hashed before being stored.
Subordination	Subordinate to the User Management Module, which handles user-related operations.
Dependencies	● SQLAlchemy for database access. ● Flask for route handling. ● Password hashing library (e.g., bcrypt) for secure password storage.
Resources	● SQLite Database ● Flask (Core, Auth)

3.3.2. Description of Data Module 2

Name	Course Data Module
Type	Data module
Purpose	Manages the structure and persistence of course data, such as course name, description, language, category, and content.
Way of operating	• The Course entity represents course data, which is manipulated through SQLAlchemy models. • CRUD operations are performed on the courses table in the SQLite database. • Courses are linked to students and professors through enrollment relationships.
Subordination	Subordinate to the Course Management Module, which handles course-related operations.
Dependencies	● SQLAlchemy for database access. ● Flask for route handling.
Resources	● SQLite Database ● Flask (Core, Courses)

3.3.3. Description of Data Module 3

Name	Enrollment Data Module
Type	Data module
Purpose	Manages the structure and persistence of enrollment data, linking students to the courses they are enrolled in.
Way of operating	• The Enrollment entity represents enrollment data, which is manipulated through SQLAlchemy models. • CRUD operations are performed on the enrollments table in the SQLite database. • Data includes student IDs, course IDs, and enrollment dates.
Subordination	Subordinate to the Course Management Module, which tracks student enrollments.
Dependencies	• SQLAlchemy for database access. • Flask for route handling.
Resources	● SQLite Database ● Flask (Core, Courses)

3.3.4. Description of Data Module 4

Name	Chat Data Module
Type	Data module
Purpose	Manages the structure and persistence of chat session data, including participants and messages.
Way of operating	• The Chat and Message entities represent chat sessions and messages, respectively. • CRUD operations are performed on the chats and messages tables in the SQLite database. • Messages include content, sender IDs, recipient IDs, and timestamps.
Subordination	Subordinate to the Communication Module, which facilitates user interactions.
Dependencies	• SQLAlchemy for database access. • Flask for route handling.
Resources	● SQLite Database ● Flask (Core, Courses)

3.3.5. Description of Data Module 5

Name	Achievement and Certificate Data Module
Type	Data module
Purpose	Manages the structure and persistence of achievement and certificate data, including earned badges, certificates, and associated details.
Way of operating	• The Achievement and Certificate entities represent milestone data, which is manipulated through SQLAlchemy models. • CRUD operations are performed on the achievements and certificates tables in the SQLite database. • Certificates are linked to students and courses, with download options available.
Subordination	Subordinate to the Reporting Module, which tracks student progress.
Dependencies	• SQLAlchemy for database access. • Flask for route handling.
Resources	● SQLite Database ● Flask (Core, Courses)

4. Dependency Description

4.1. Dependencies among Modules

4.1.1. User Management Module

- **Dependencies**:
  - **User Data Module**:
    The User Management Module relies on the User Data Module to store and manage user data, including usernames, passwords, and roles. It accesses the User Data Module during user addition, removal, or updates to validate and persist user records.
- **Interaction**:
  The User Management Module interacts directly with the User Data Module to perform CRUD operations on user records, ensuring proper user and role management.

4.1.2. Course Management Module

- **Dependencies**:
  - **Course Data Module**:
    The Course Management Module uses the Course Data Module to store and retrieve course details, such as course name, description, and content.
  - **Enrollment Data Module**:
    Tracks the relationship between students and courses, allowing the Course Management Module to determine student enrollments.
  - **User Data Module**:
    Validates the users (students or professors) who are associated with courses or enrollments.
- **Interaction**:
  The Course Management Module interacts directly with the Course Data Module and Enrollment Data Module for CRUD operations and indirectly with the User Data Module to validate user roles.

4.1.3. Communication Module

- **Dependencies**:
  - **Chat Data Module**:
    The Communication Module relies on the Chat Data Module to manage chat sessions, including messages and participants.
  - **User Data Module**:
    Validates users initiating chats and ensures proper user association in conversations.
- **Interaction**:
  The Communication Module interacts directly with the Chat Data Module for CRUD operations and indirectly with the User Data Module to validate chat participants.

4.1.4. Reporting Module

- **Dependencies**:
  - **Achievement and Certificate Data Module**:
    Provides data on completed milestones and certificates for generating reports.
  - **User Data Module**:
    Links achievements and certificates to specific students.
  - **Enrollment Data Module**:
    Tracks course completion data for enrolled students.
- **Interaction**:
  The Reporting Module interacts with these modules to generate analytics and display progress for students and professors.

4.2. Dependencies among Processes

4.2.1. User Registration Process

- **Dependencies**:
  - **User Management Module**: Handles the logic for creating new users and assigning roles.
  - **User Data Module**: Stores the newly registered user's data, including username, email, password, and role.
- **Interaction**:
  The User Registration Process interacts with the User Management Module to validate and process registration requests and then stores the user in the User Data Module.

4.2.2. User Login Process

- **Dependencies**:
  - **User Management Module**: Handles credential validation and ensures that only registered users can log in.
  - **User Data Module**: Provides the stored credentials and user information for login validation.
- **Interaction**:
  The User Login Process interacts with the User Management Module for validation and indirectly with the User Data Module to retrieve user records.

4.2.3. Enroll in Course Process

- **Dependencies**:
  - **Course Management Module**: Handles enrollment logic, ensuring students are assigned to valid courses.
  - **Enrollment Data Module**: Stores enrollment details linking students and courses.
- **Interaction**:
  The Enroll in Course Process interacts with the Course Management Module

to validate enrollments and persists the enrollment data using the Enrollment Data Module.

4.2.4. View Course Content Process

- **Dependencies**:
  - **Course Management Module**: Retrieves course content for enrolled students.
  - **Enrollment Data Module**: Validates if the student is enrolled in the course.
- **Interaction**:
  The View Course Content Process interacts with the Course Management Module to fetch course materials and validates enrollments via the Enrollment Data Module.

4.2.5. Start Chat Process

- **Dependencies**:
  - **Communication Module**: Manages chat initiation logic.
  - **Chat Data Module**: Stores chat session details and message history.
  - **User Data Module**: Validates the users participating in the chat.
- **Interaction**:
  The Start Chat Process interacts with the Communication Module for chat initiation logic and the Chat Data Module for persistence. It validates user roles via the User Data Module.

4.2.6. Upload Course Content Process

- **Dependencies**:
  - **Course Management Module**: Handles the logic for uploading and associating content with courses.
  - **Course Data Module**: Stores the uploaded content and links it to courses.
- **Interaction**:
  The Upload Course Content Process interacts with the Course Management Module for validation and persistence and the Course Data Module for storing content details.

4.3. Dependencies among Data Modules

4.3.1. User Data Module

- **Dependencies**: None. The User Data Module is independent and solely manages user data.
- **Interaction**:
  - Provides user-related data to other modules, such as the Enrollment Data Module, Chat Data Module, and Achievement and Certificate Data Module, for validating users or tracking user-specific actions.

4.3.2. Course Data Module

- **Dependencies**:
  - **Enrollment Data Module**: Links courses to enrolled students.
- **Interaction**:
  - Provides course-related data to processes such as course viewing and content uploading.
  - Uses the Enrollment Data Module to track student enrollments.

4.3.3. Chat Data Module

- **Dependencies**:
  - **User Data Module**: Validates chat participants.
- **Interaction**:
  - Provides chat session and message data to processes managing communication.

4.3.4. Achievement and Certificate Data Module

- **Dependencies**:
  - **User Data Module**: Associates achievements and certificates with students.
- **Interaction**:
  - Provides achievement and certificate data to reporting processes for generating progress reports and milestones.

5. Interface Description

5.1. Module Interfaces

5.1.1. Module 1 Interface

Name	User Registration Module
Type	Code Module
Purpose	Manages the registration of new users, allowing them to create accounts by submitting their username, password, email, and role (Student or Professor).
Way of operating	- The user interacts with the registration form to provide their details (username, email, password, and role). - These details are validated and processed to ensure they meet the application's requirements. - Once validated, the user information is securely saved, and a success or error message is displayed.
Interface1	Register User
Input	username: string - The username chosen by the user, which should be unique. password: string - The user's password, which will be encrypted. email: string - The user's email address for verification. role: string - The role assigned to the user (Student or Professor).
Output	result: string - A message indicating whether the registration was successful or not.
Description	This interface accepts user details, validates the inputs, encrypts the password, and saves the user in the database.

5.1.2. Module 2 Interface

Name	User Login Module
Type	Code Module
Purpose	Authenticates users by verifying their credentials and granting access to protected sections of the e-learning platform.
Way of operating	- The user submits their username and password through the login interface. - The system validates these credentials by comparing them against the existing user data. - If the credentials are valid, the user gains access to the system; otherwise, an error message is displayed.
Interface1	Login User
Input	username: string password: string
Output	result: string - A message indicating whether the login was successful or not.
Description	This interface validates the user's credentials by checking the submitted username and password with the existing data. If valid, the user gains access to the system.

5.1.3. Module 3 Interface

Name	User Management Module (Admin)
Type	Code Module
Purpose	Allows administrators to manage users by performing actions such as adding, updating, or removing user accounts.
Way of operating	Adding Users: Admin submits user details (username, email, password, role) via the front-end. The details are processed, validated, and persisted in the database. Removing Users: Admin selects a user by their unique ID and submits a request to delete. - The request is validated, and the user is removed from the database.
Interface1	Add User
Input	username: string password: string email: string role: string
Output	result: string - A message indicating success or failure.
Description	This interface allows the admin to add new users. The module validates the inputs, encrypts the password, and persists the user in the database.
Interface2	Remove User
Input	userId: int - The unique identifier of the user to be removed**.**
Output	result: string - A message indicating success or failure.
Description	This interface allows the admin to remove an existing user. The module checks if the user exists in the database and deletes it if found.

5.1.4. Module 4 Interface

Name	Course Management Module
Type	Code Module
Purpose	Handles adding, enrolling, and viewing courses. It ensures courses are properly managed in the database.
Way of operating	Adding a Course: • Professors or admins submit course details (name, description, language, content) via a form. Enrolling in a Course: • Students select a course from the available list, and their enrollment is saved in the database. Viewing Course Content: • Enrolled students can view course content, which is dynamically retrieved from the database.
Interface1	Add Course
Input	name: string description: string language: string
Output	result: string - A message indicating whether the course was successfully added.
Description	The interface accepts course details, validates them, and stores the course in the database.
Interface2	Enroll in Course
Input	studentId: int courseId: int
Output	result: string - A message indicating enrollment success or failure.
Description	This interface links students to courses and saves the enrollment data in the database.
Interface3	View Course Content
Input	studentId: int courseId: int
Output	content: list - The list of materials for the selected course.
Description	This interface retrieves and displays course content for enrolled students.

5.1.5. Module 5 Interface

Name	Communication Module
Type	Code Module
Purpose	Facilitates real-time messaging between students and professors.
Way of operating	- Users can initiate chats, send messages, and view chat history.
Interface1	Add Shopping Item
Input	senderId: int, recipientId: int
Output	chatId: int - The unique ID of the chat session.
Description	The interface initiates a chat session between two users.
Interface2	Send Message
Input	chatId: int, message: string
Output	result: string - A message indicating success or failure.
Description	This interface sends a message in an existing chat session and saves it in the database.

5.1.6. Module 6 Interface

Name	Reporting Module
Type	Code Module
Purpose	Generates reports and displays analytics related to student progress, course completion, and platform usage.
Way of operating	• The system processes user and course data to generate visual reports. • Achievements and certificates are fetched and displayed for students. • Reports can be exported for external use.
Interface1	Generate Progress Report
Input	studentId: int - The ID of the student for whom the progress report is being generated.
Output	achievements: list - A list of earned achievements and certificates.
Description	This interface retrieves and displays a student's achievements, including badges and certificates, with options for downloading certificates as PDF files.
Interface2	Fetch Achievements and Certificates
Input	studentId: int - The ID of the student.
Output	***result:*** string
Description	The interface retrieves the specified event, applies the updates, and saves the modified event.

6. Detailed Design

6.1. Modules detailed design

6.1.1. Module 1

Name	User Registration Module
Type	Code Module
Purpose	Manages the registration of new users, allowing them to create accounts by submitting their username, email, password, and role (Student or Professor).
Way of operating	Users submit their registration details through a form. The system validates the input, hashes the password, and saves the user data in the SQLite database.
Classes	auth.py, models.py

6.1.1.1. Module 1, class 1

Name	auth.py
Purpose	Manages user registration requests by processing user input, hashing the password, and storing the user in the database.
Methods	register()

6.1.1.1.1. Module 1, class 1, method 1

Name	register
Purpose	Handles the user registration process.
Prototype	def register(username: str, email: str, password: str, role: str)
Input	username: str email: str password: str, role: str
Output	Redirects to the login page or displays an error message.
Caller	Registration form on the user interface.
Calls	User.save()

6.1.1.2. Module 1, class 2

Name	models.py User Model
Purpose	Represents a user entity in the system and provides methods to interact with user data.
Members	id: int username: str email: str password: str (hashed) role: str
Methods	save(): Persists the user entity to the SQLite database. find_by_username(): Retrieves a user by their username.

6.1.1.2. Module 1, class 3

Name	models.py Role Model
Purpose	Represents the roles associated with a user, providing additional metadata about permissions or access levels.
Members	id: int - A unique identifier for each role. role_name: str - The name of the role (e.g., "Student" or "Professor").
Methods	find_by_role_name() - Retrieves a role by its name.

6.1.1.2. Module 1, class 4

Name	forms.py - Registration Form
Purpose	Handles the user input for the registration form, including field validation.
Members	username: str - Field for the username. email: str - Field for the email. password: str - Field for the password. role: str - Dropdown or radio button for selecting the role (Student/Professor).
Methods	validate_username() - Ensures the username is unique in the database. validate_email() - Checks if the email address format is valid and unique. validate_password() - Ensures the password meets security requirements (length, special characters, etc.).

6.1.2. Module 2

Name	User Login Module
Type	Code Module
Purpose	Authenticates users by verifying their credentials and granting access to protected sections of the e-learning platform.
Way of operating	• Users submit their login details (username and password) through a form on the login page. • The form data is processed in auth.py, where the system retrieves the user record by username. • The hashed password stored in the database is compared with the submitted password using a secure hashing algorithm (e.g., bcrypt). • If the credentials are valid, the system initiates a session for the user and redirects them to the dashboard. • If the credentials are invalid, the user is shown an error message on the login page.
Classes	auth.py, models.py

6.1.2.1. Module 2, class 1

Name	auth.py
Purpose	Manages login logic and processes user inputs received from the login form.
Members	None
Methods	login()

6.1.2.1.1. Module 2, class 1, method 1

Name	login
Purpose	Authenticates a user based on their submitted username and password.
Prototype	def login(username: str, password: str)
Input	username: str password: str
Output	Redirects to the dashboard or displays an error message.
Caller	Login form on the user interface.
Calls	User.find_by_username()

6.1.2.1.1. Module 2, class 1, method 2

Name	logout
Purpose	Ends the user's session and logs them out of the platform.
Prototype	def logout()
Input	None
Output	Redirects to the login page or homepage.
Caller	Logout button on the user interface.
Calls	session.clear()

6.1.2.2. Module 2, class 2

Name	models.py - User Model
Purpose	Represents a user entity in the system and provides methods for authentication and user retrieval.
Members	id: int - A unique identifier for each user. username: str - The username of the user. email: str - The user's email address. password: str - The hashed password of the user. role: str - Specifies whether the user is a Student or Professor.
Methods	find_by_username(username: str) verify_password(submitted_password: str)

6.1.2.2.1. Module 2, class 2, method 1

Name	find_by_username
Purpose	Retrieves a user object by their username.
Prototype	def find_by_username(username: str) -> User
Input	username: str - The username of the user.
Output	User object if found, or None if the username does not exist.
Caller	auth.py for login functionality.
Calls	SQLAlchemy query on the User table.

6.1.2.2.2. Module 2, class 2, method 2

Name	verify_password
Purpose	Verifies if the submitted password matches the hashed password stored in the database.
Prototype	def verify_password(submitted_password: str) -> bool
Input	submitted_password: str - The password entered by the user.
Output	True if the password is correct, otherwise False.
Caller	auth.py during the login process.
Calls	bcrypt's check_password_hash method.

6.1.2.2.3. Module 2, class 2, method 3

Name	get_user_from_session
Purpose	Retrieves the logged-in user's details using the session ID.
Prototype	def get_user_from_session(session_id: str) -> User
Input	session_id: str - The ID of the user's current session.
Output	User object if the session is valid, or None if the session has expired or is invalid.
Caller	Any route requiring user authentication.
Calls	SQLAlchemy query on the User table.

6.1.2.2.4. Module 2, class 2, method 4

Name	end_session
Purpose	Terminates the session for a logged-in user
Prototype	def end_session(session_id: str)
Input	session_id: str - The ID of the user’s session.
Output	None
Caller	auth.py for logout functionality.
Calls	Flask’s session.clear()

6.1.3. Module 3

Name	Course Enrollment Management Module
Type	Code Module
Purpose	Allows students to enroll in or unenroll from courses, and enables professors to manage enrolled students.
Way of Operating	• Students interact with the platform's course list to enroll in available courses. • Enrollment actions are processed through Flask routes, which validate the student's eligibility and update the database. • Professors can view the list of enrolled students for their courses and manage course enrollment if required. • The Enrollment model in models.py is used to manage relationships between users and courses.
Classes	views.py, models.py

6.1.3.1. Module 3, class 1

Name	views.py
Purpose	Handles routes for enrolling students in courses and managing enrollments.
Members	None
Methods	enroll_in_course(), unenroll_from_course(), view_enrolled_students()

6.1.3.1.1. Module 3, class 1, method 1

Name	enroll_in_course
Purpose	Allows a student to enroll in a course.
Prototype	def enroll_in_course(course_id: int, student_id: int)
Input	course_id: int - The ID of the course the student wants to enroll in. student_id: int - The ID of the student enrolling in the course.
Output	A success or error message.
Caller	Course enrollment interface via route /enroll/<course_id>.
Calls	None

6.1.3.1.2. Module 3, class 1, method 2

Name	unenroll_from_course
Purpose	Allows a student to unenroll from a course.
Prototype	def unenroll_from_course(course_id: int, student_id: int)
Input	course_id: int - The ID of the course the student wants to unenroll from. student_id: int - The ID of the student unenrolling from the course.
Output	A success or error message.
Caller	Course management interface via route /unenroll/<course_id>.
Calls	Enrollment.delete()

6.1.3.1.2. Module 3, class 1, method 3

Name	view_enrolled_students
Purpose	Allows a professor to view the list of students enrolled in their course.
Prototype	def view_enrolled_students(course_id: int)
Input	course_id: int - The ID of the course.
Output	A list of students enrolled in the course.
Caller	Professor interface via route /course/<course_id>/students.
Calls	Enrollment.find_all_by_course_id()

6.1.3.2. Module 3, class 2

Name	models.py - Enrollment Model
Purpose	Represents the relationship between users (students) and courses.
Members	id: int - A unique identifier for each enrollment record. student_id: int - Links the enrollment to a specific student. course_id: int - Links the enrollment to a specific course. enrollment_date: datetime - The date when the enrollment occurred.
Methods	save() - Saves the enrollment record to the database. delete() - Deletes an enrollment record from the database. find_all_by_course_id(course_id: int) - Retrieves all students enrolled in a specific course.

6.1.3.2.1. Module 3, class 2, method 1

Name	save
Purpose	Saves an enrollment record to the database.
Prototype	def save(self)
Input	None
Output	None
Caller	views.py for enrollments.
Calls	Inserts or updates the enrollment record in the enrollments table.

6.1.3.2.2. Module 3, class 2, method 2

Name	delete
Purpose	Deletes an enrollment record from the database.
Prototype	def delete(self)
Input	None
Output	None
Caller	views.py for unenrollments
Calls	Removes the enrollment record from the enrollments table.epository.deleteById()

6.1.3.2.3. Module 3, class 2, method 3

Name	find_all_by_course_id
Purpose	Retrieves all enrollments for a specific course.
Prototype	def find_all_by_course_id(course_id: int) -> List[Enrollment]
Input	course_id: int - The ID of the course.
Output	A list of Enrollment objects.
Caller	views.py for displaying enrolled students.
Calls	Queries the enrollments table for records linked to the specified course.

6.1.3.3. Module 3, class 3

Name	models.py - Course Model
Purpose	Represents course data in the database and provides methods for interacting with courses.
Members	id: int - A unique identifier for each course. name: str - The name of the course. description: str - A brief description of the course. professor_id: int - Links the course to the professor teaching it.
Methods	find_by_id() find_all()

6.1.3.3.1. Module 3, class 3, method 1

Name	find_by_id
Purpose	Retrieves a course by its unique ID.
Prototype	def find_by_id(course_id: int) -> Course
Input	course_id: int - The ID of the course.
Output	A Course object.
Caller	views.py during enrollment and content viewing.
Calls	Queries the courses table for the record matching the specified ID.

6.1.3.3.2. Module 3, class 3, method 2

Name	find_all
Purpose	Retrieves all available courses.
Prototype	def find_all() -> List[Course]
Input	None
Output	A list of Course objects.
Caller	Course listing routes in views.py.
Calls	Queries the courses table for all available courses.

6.1.4. Module 4

Name	Chat Management Module
Type	Code Module
Purpose	Facilitates real-time communication between students and professors, enabling them to discuss course-related topics or resolve queries.
Way of Operating	Students or professors initiate a chat through the platform interface. The system checks if an existing chat session exists between the users. If not, it creates a new session. Messages are exchanged in real-time using WebSocket or polling mechanisms. Chat histories are stored in the database and retrieved for future reference.
Classes	views.py, models.py

6.1.4.1. Module 4, class 1

Name	views.py - Chat Routes
Purpose	Handles routes for initiating chat sessions and sending messages.
Members	None
Methods	start_chat(), send_message(), get_chat_history()

6.1.4.1.1. Module 4, class 1, method 1

Name	start_chat
Purpose	Initiates a new chat session if none exists between the users.
Prototype	def start_chat(user1_id: int, user2_id: int)
Input	user1_id: int - ID of the first user. user2_id: int - ID of the second user.
Output	The ID of the newly created or existing chat session.
Caller	None
Calls	Chat.find_or_create()

6.1.4.1.2. Module 4, class 1, method 2

Name	send_message
Purpose	Handles sending a message in an existing chat session.
Prototype	def send_message(chat_id: int, sender_id: int, content: str)
Input	chat_id: int - ID of the chat session. sender_id: int - ID of the message sender. content: str - The message content.
Output	A success or error message.
Caller	None
Calls	Message.save()

6.1.4.2. Module 4, class 2

Name	models.py - Chat Model
Purpose	Represents a chat session between users.
Members	id: int - A unique identifier for the chat session. user1_id: int - The first user in the chat. user2_id: int - The second user in the chat.
Methods	find_or_create() - Finds an existing chat session or creates a new one.

6.1.4.3. Module 4, class 3

Name	models.py - Message Model
Purpose	Represents a message within a chat session.
Members	id: int - A unique identifier for the message. chat_id: int - Links the message to a chat session. sender_id: int - The user who sent the message. content: str - The message content. timestamp: datetime - The time the message was sent.
Methods	save() - Saves the message to the database. find_all_by_chat_id(chat_id: int) - Retrieves all messages for a specific chat.

6.1.4.4. Module 4, class 4

Name	Task
Purpose	Represents a task entity with attributes such as name, description, deadline, and status.
Members	id: Long, name: String, description: String, deadline: LocalDate, importance: String, status: boolean
Methods	getName(), setName(), getDescription(), setDescription(), getDeadline(), setDeadline(), getImportance(), setImportance(), getStatus(), setStatus()

6.1.5. Module 5

Name	Quiz and Assessment Module
Type	Code Module
Purpose	Manages quizzes and assignments to evaluate student progress.
Way of Operating	• Professors create quizzes with multiple-choice or open-ended questions. • Students access and complete quizzes through their enrolled courses. • For multiple-choice quizzes, answers are auto-graded; for open-ended assignments, professors review and grade them manually. • Results are stored in the database and displayed to students.
Classes	views.py, models.py

6.1.5.1. Module 5, class 1

Name	views.py - Quiz Routes
Purpose	Handles routes for creating, taking, and grading quizzes.
Members	None
Methods	create_quiz(), take_quiz(), grade_quiz()

6.1.5.1.1. Module 5, class 1, method 1

Name	create_quiz
Purpose	Allows professors to create a new quiz.
Prototype	def create_quiz(course_id: int, questions: List[dict])
Input	course_id: int - ID of the course the quiz belongs to. questions: List[dict] - A list of questions with options and correct answers.
Output	Success or error message.
Caller	None
Calls	Quiz.save()

6.1.5.1.2. Module 5, class 1, method 2

Name	take_quiz
Purpose	Allows students to take a quiz.
Prototype	def take_quiz(quiz_id: int, answers: List[str])
Input	quiz_id: int - ID of the quiz. answers: List[str] - The student's answers.
Output	A success or error message.
Caller	None
Calls	QuizResult.save()

6.1.5.1.3. Module 5, class 1, method 3

Name	grade_quiz
Purpose	Allows professors to manually grade open-ended assignments.
Prototype	def grade_quiz(result_id: int, grade: float)
Input	result_id: int - ID of the quiz result. grade: float - The grade assigned by the professor.
Output	Success message
Caller	None
Calls	QuizResult.update().delete()

6.1.5.2. Module 5, class 2

Name	models.py - Quiz Model
Purpose	Represents a quiz entity in the database.
Members	id: int - A unique identifier for the quiz. course_id: int - Links the quiz to a course. questions: str - JSON-encoded list of questions and options.
Methods	save() - Saves the quiz to the database.

6.1.5.2.1. Module 5, class 2, method 1

Name	save
Purpose	Saves the quiz result to the database.
Prototype	def save(self)
Input	None
Output	None
Caller	views.py when a professor creates a new quiz.
Calls	None

6.1.5.3. Module 5, class 3

Name	models.py - QuizResult Model
Purpose	Represents a student's quiz result.
Members	id: int - A unique identifier for the result. quiz_id: int - Links the result to a quiz. student_id: int - Links the result to a student. score: float - The score achieved by the student.
Methods	save() - Saves the result to the database. update() - Updates the result after manual grading.

6.1.5.3.1. Module 5, class 3, method 1

Name	save
Purpose	Saves the quiz result to the database.
Prototype	def save(self)
Input	None
Output	None
Caller	views.py when a student submits a quiz.
Calls	Database layer

6.1.4.3.2. Module 5, class 3, method 2

Name	update
Purpose	Updates the quiz result after manual grading by a professor.
Prototype	def update(self, grade: float)
Input	grade: float - The updated grade assigned by the professor.
Output	None
Caller	views.py when a professor grades an open-ended question quiz.
Calls	None

6.2. Data Modules Detailed Design

The data modules described in chapter 3.3. are detailed. Detailed diagrams are recommended.

6.2.1. Data module 1

Name	User Data Module
Type	Data Module
Purpose	Manages user-related data, including attributes such as username, email, password, and roles (Student, Professor, or Admin).
Way of Operating	• Stores and retrieves user data from the database using SQLAlchemy. • Provides access to user-related operations such as authentication and authorization.
Classes	User

6.2.1.1. Data module 1, class 1

Name	User
Purpose	Represents a user entity in the system.
Attributes	id: int - A unique identifier for each user. username: str - The username chosen by the user. email: str - The user's email address for verification. password: str - The hashed password for secure storage. role: str - The role assigned to the user (e.g., Student, Professor, or Admin).
Methods	save(): Adds or updates a user in the database. find_by_username(username: str): Retrieves a user by their username from the database.

6.2.2. Data module 2

Name	Course Data Module
Type	Data Module
Purpose	Manages course-related data, including attributes such as course name, description, language, and content.
Way of Operating	• Stores and retrieves course data using SQLAlchemy models. • Handles course-related operations for professors and enrolled students.
Classes	Course

6.2.2.1. Data module 2, class 1

Purpose	Represents a course entity in the system.
Attributes	id: int - A unique identifier for each course. name: str - The course name. description: str - A brief description of the course. language: str - The language of instruction. professor_id: int - Links the course to the professor managing it.
Methods	save(): Adds or updates a course in the database. find_by_id(course_id: int): Retrieves a course by its unique ID.

6.2.3. Data module 3

Name	Enrollment Data Module
Type	Data Module
Purpose	Manages enrollment data, linking students to the courses they are enrolled in.
Way of Operating	Tracks enrollment records using SQLAlchemy models. Supports student enrollment and course management.
Classes	Enrollment

6.2.3.1. Data module 3, class 1

Name	Enrollment
Attributes	id: int - A unique identifier for each enrollment record. student_id: int - The ID of the enrolled student. course_id: int - The ID of the enrolled course. enrollment_date: datetime - The date of enrollment.
Methods	save(): Adds or updates an enrollment record in the database. delete(): Removes an enrollment record from the database. find_all_by_course_id(course_id: int): Retrieves all enrollments for a specific course.
Purpose	Represents the relationship between a student and a course.

6.2.4. Data module 4

Name	Achievement and Certificate Data Module
Type	Data Module
Purpose	Tracks achievements and generates certificates for students upon completing courses or milestones.
Way of Operating	• Stores achievement and certificate records using SQLAlchemy models. • Links achievements and certificates to students and courses.
Classes	Achievement, Certificate

6.2.4.1. Data module 4, class 1

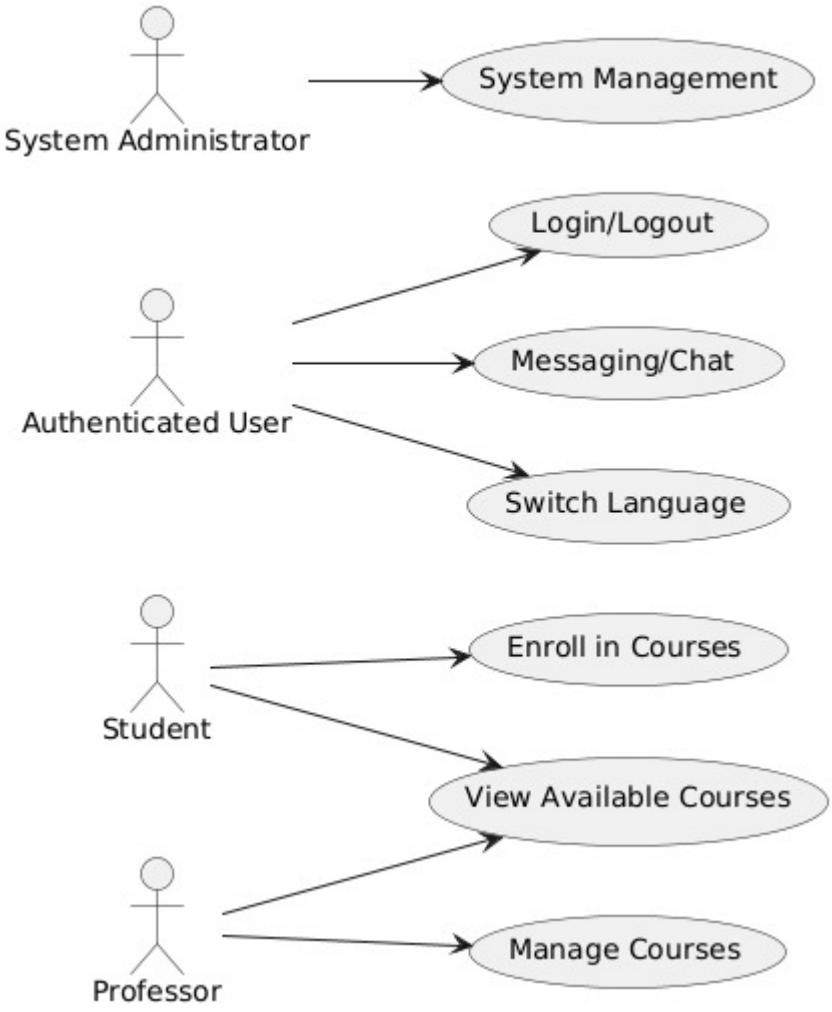
Name	Achievement
Purpose	Represents a user’s achievements in the platform.
Attributes	id: int - A unique identifier for each achievement. student_id: int - The ID of the student. badge: str - The badge earned by the student. course_id: int - Links the achievement to a specific course.
Methods	save(): Adds or updates an achievement in the database.

6.2.4.2. Data module 4, class 2

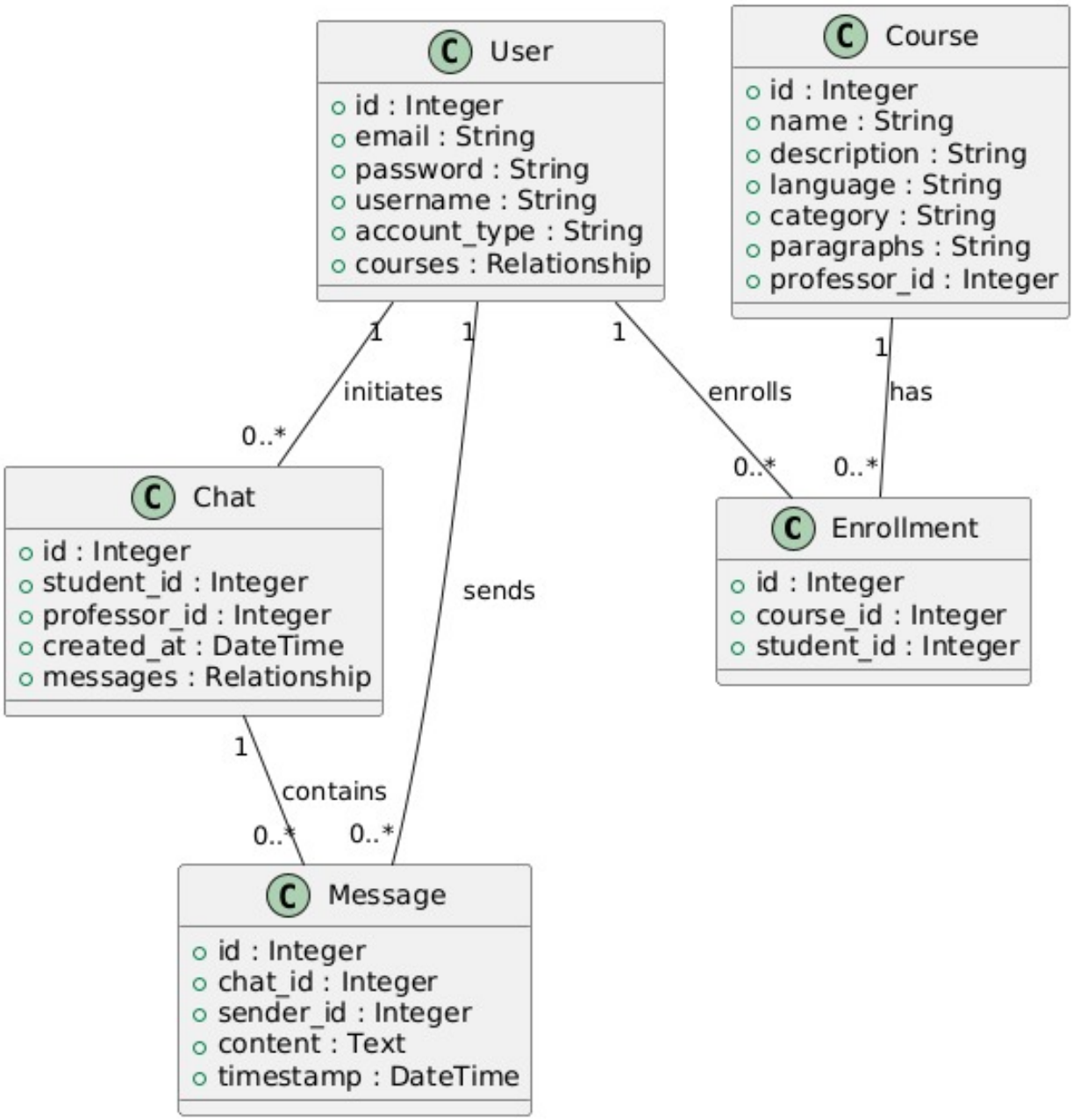
Name	Certificate
Purpose	Represents a certificate awarded to students upon course completion.
Attributes	id: int - A unique identifier for the certificate. student_id: int - The ID of the student earning the certificate. course_id: int - Links the certificate to a completed course. issue_date: datetime - The date the certificate was issued.
Methods	save(): Adds or updates a certificate in the database. find_all_by_student_id(student_id: int): Retrieves all certificates for a specific student.

Appendices

A1. Use cases diagrams

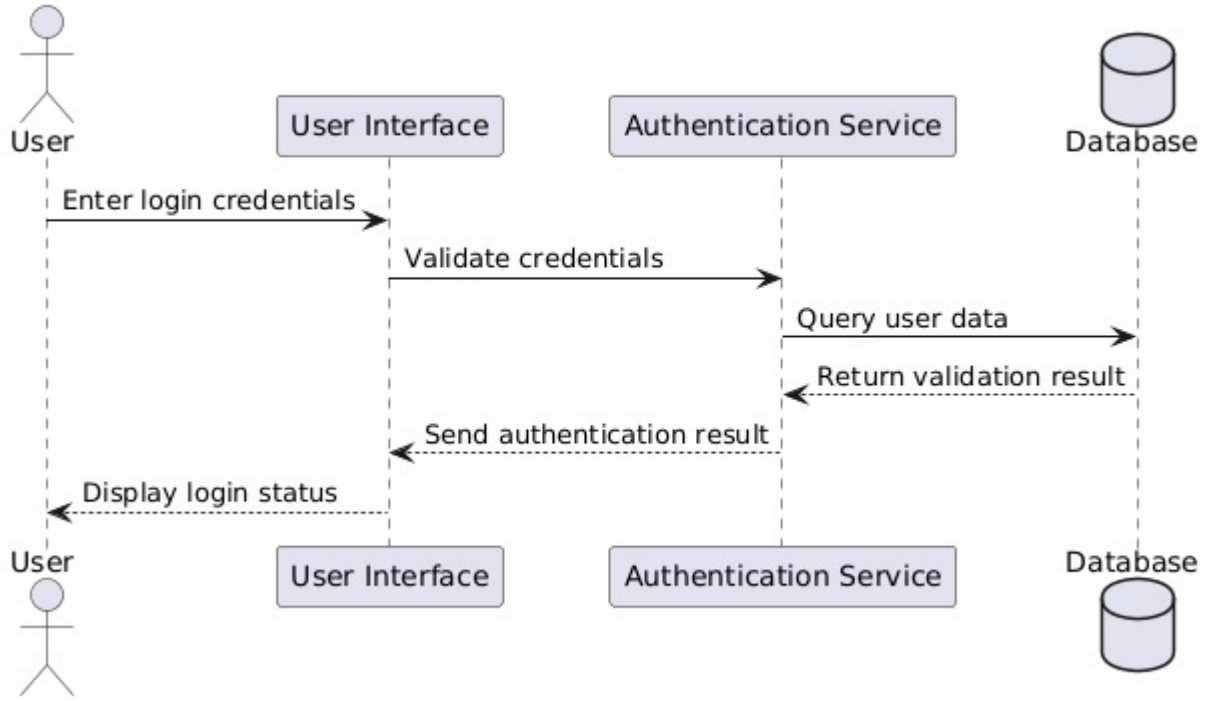


A2. Class diagrams

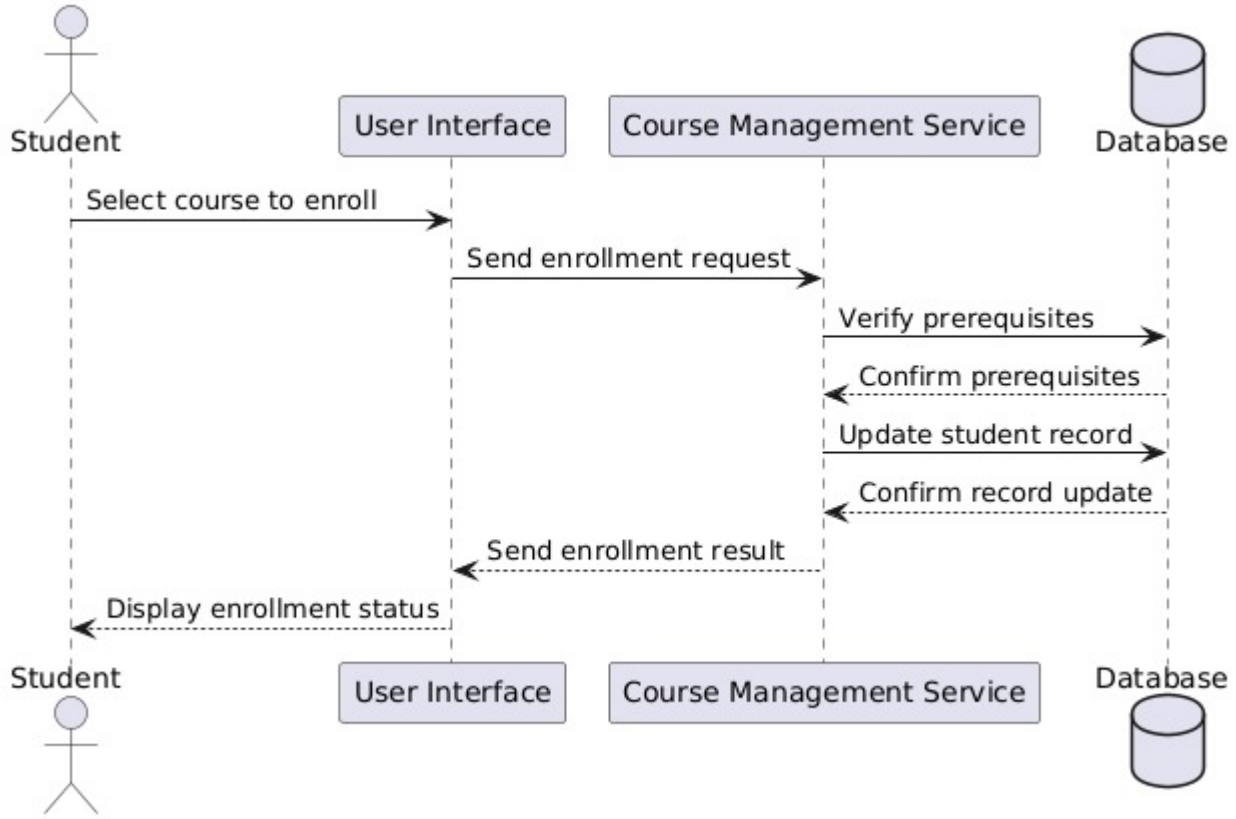


A3. Sequence diagrams

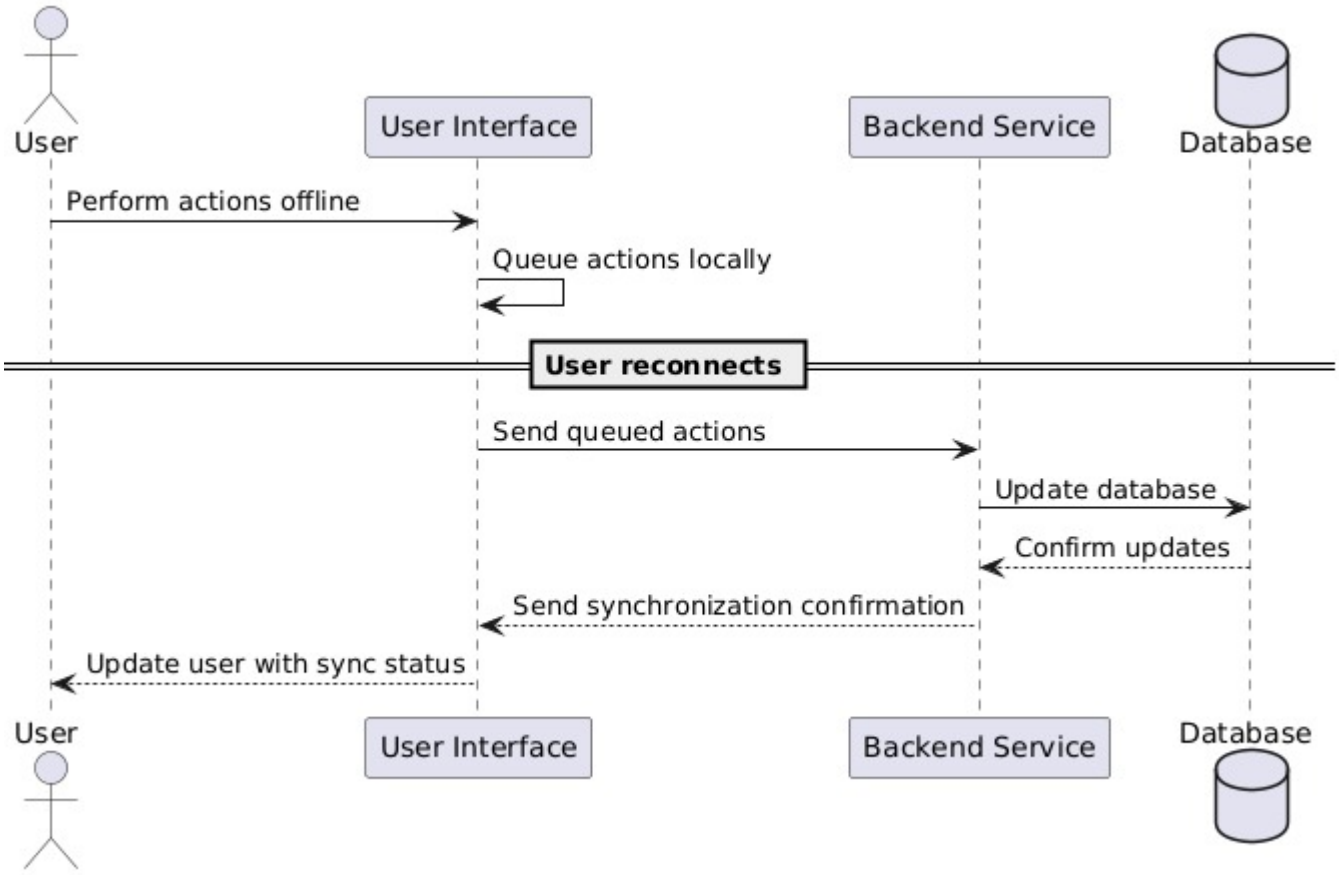
Login



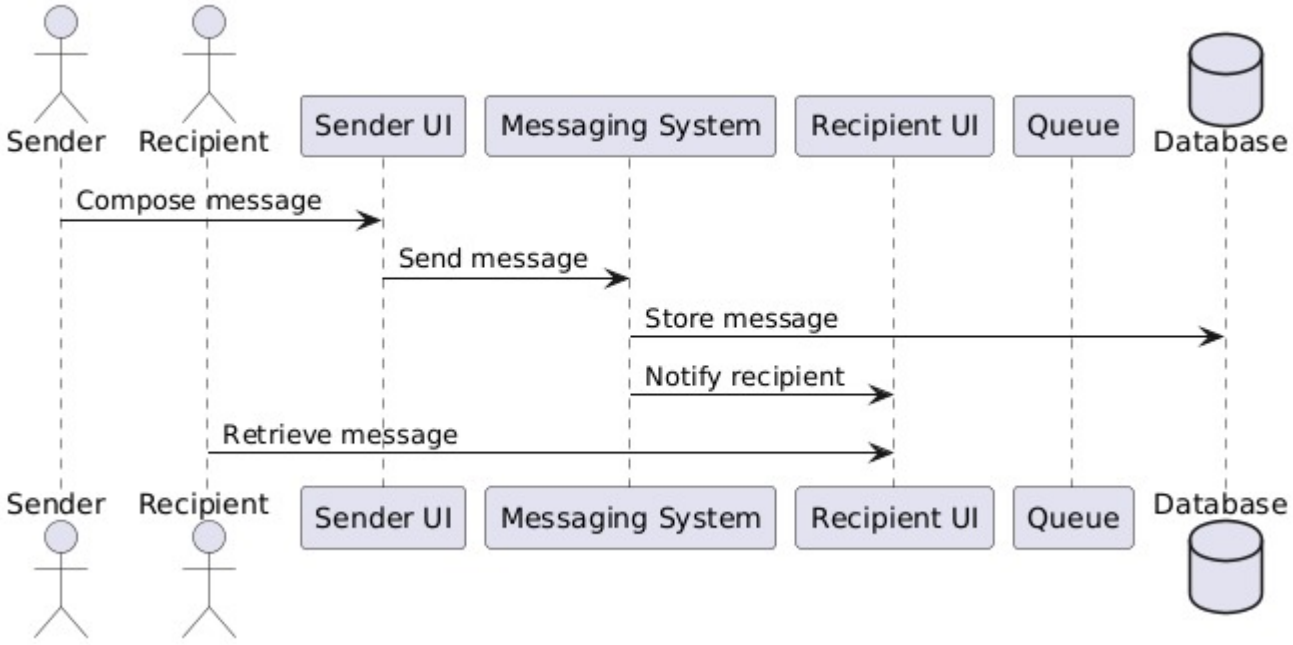
Course enrollment



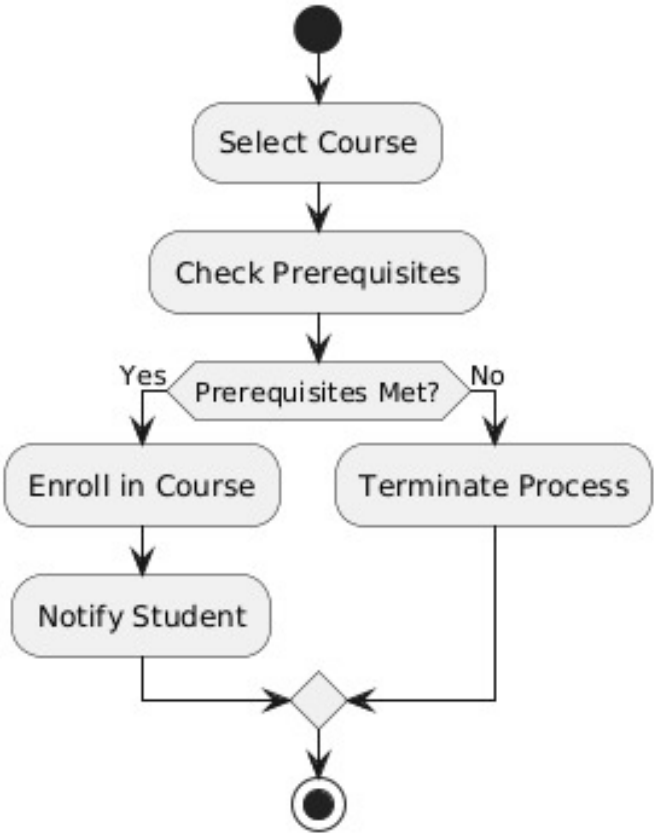
Offline Synchronization



Messaging

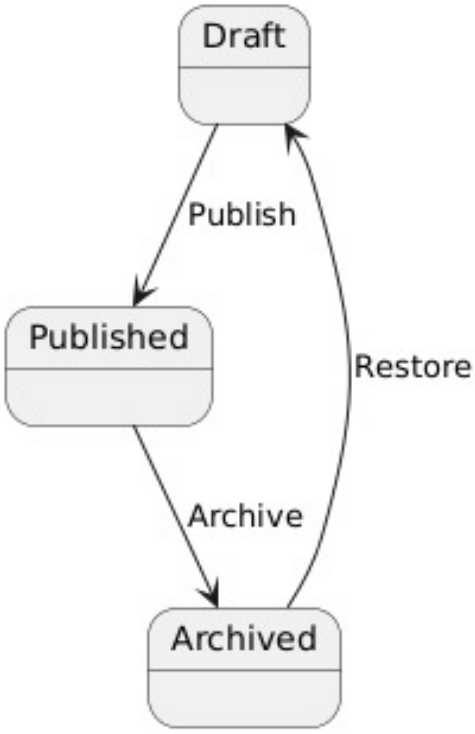


A4. Activity diagrams

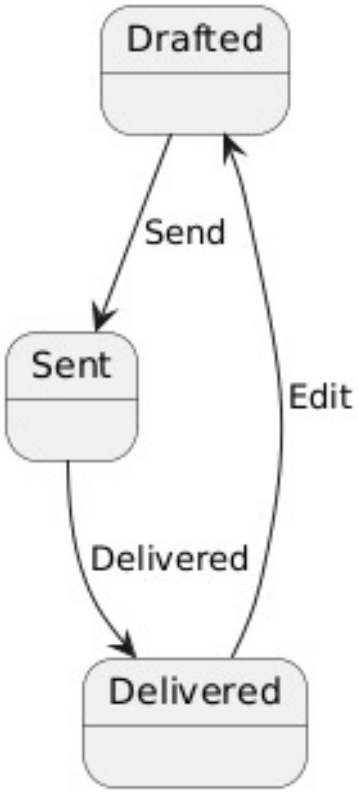


A5. Sequence diagrams

Course



Message



A6. Conclusions regarding the activity

Working on this e-learning platform project was an enriching and valuable learning experience. As a team, we faced challenges in collaboration and communication, especially while coordinating tasks and aligning on technical details. However, these challenges allowed us to develop our teamwork skills and improve our ability to work effectively in a group setting.

Throughout the development process, we successfully applied our knowledge of software development to create a functional e-learning platform that addresses key requirements such as user registration, course enrollment, and real-time communication. This project also gave us hands-on experience with technologies like Flask, SQLite, and SQLAlchemy, as well as implementing features like role-based access, data management, and responsive user interfaces.

Beyond technical skills, we gained practical experience in project management, from initial planning and requirement gathering to detailed design and implementation. Despite initial hurdles, the completion of this project has given us confidence in our ability to work collaboratively and deliver a software solution with real-world applicability.