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

Course Registration System

Design Document

Team Number: 23

Team Members:

Hazem Ahmed Refaat – 202401389

Youssof Amgad Abdelrahman – 202400370

Maya Sameh Aboamer – 202401080

Yaseen Mohamed Bilal – 202400463

Representative contact info:

s-hazem.refaat@zewailcity.edu.eg

18/11/2025

Document Outline

1. Introduction
    - 1.1 Purpose of the Document
    - 1.2 Scope of the Design Phase
    - 1.3 Intended Audience
    - 1.4 Overview of the Contents
2. System Overview
    - 2.1 Brief Description of the System
    - 2.2 Key Design Goals and Constraints
3. Architectural Design
    - 3.1 System Architecture Diagram
    - 3.2 Discussion of Architectural Style and Components
    - 3.3 Technology Stack and Tools
4. Detailed Design
    - 4.1 Model–View–Controller (MVC) Design Pattern
        - 4.1.2 Mapping of Project Components to MVC
        - 4.1.3 Responsibilities of Model, View, and Controller
        - 4.1.4 Interaction Between Components
    - 4.2 UML Diagrams
        - 4.2.1 Detailed Class Diagram
        - 4.2.2 Sequence Diagrams
    - 4.3 UI/UX Design

1. Introduction

1.1 This document explains in detail how the requirements stated in the Software Specification Requirement (SRS) will be implemented.

1.2 The design will cover the architecture, User Interface (UI) providing mockups, and data storage.

1.3 This document is aimed at the developers as it will guide them through the implementation of the system.

1.4 The rest of the document describes the system design as follows:

- Section 2: Describes the system at a high level stating its goals and constraints.

- Section 3: Discusses the details of the architecture to be used in the system.

- Section 4: Delves into the specific design features and components of this system

2. System Overview

2.1 The system aims to replace in-person registrar visits and centralize the process of student-course-professor interactions.

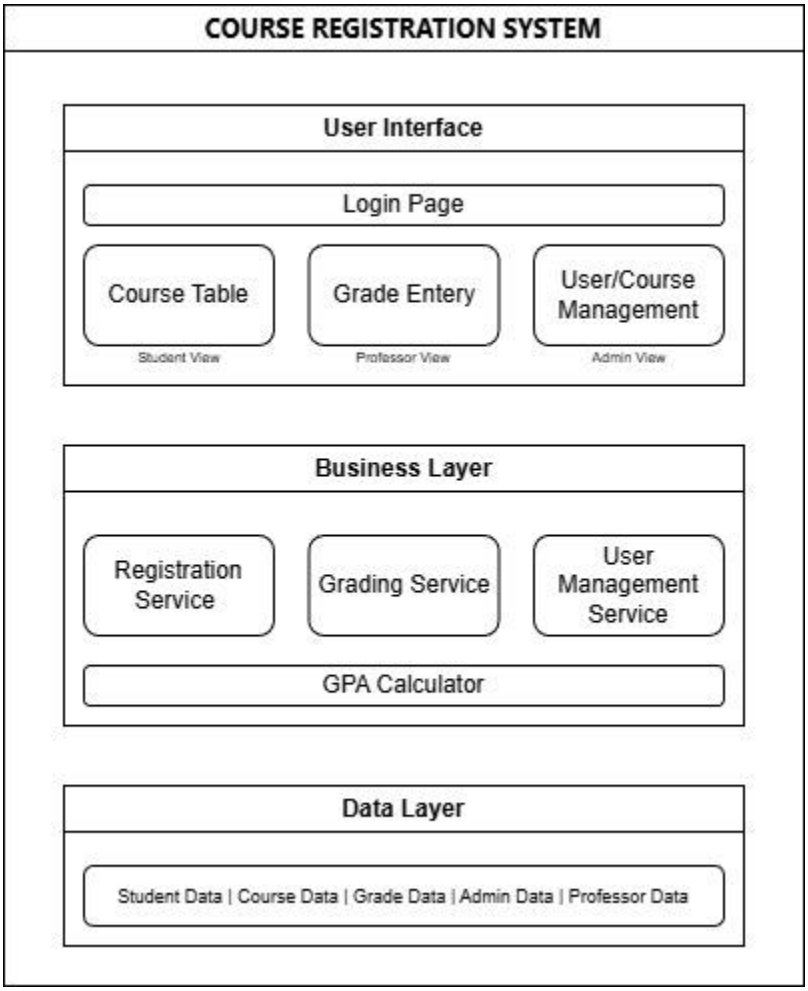
2.2

-Goals: Allow students to add/drop courses and view their grades, professors to manage courses and assign grades and administrators to manage users and courses with an intuitive and simple UI.

-Constraints: Backend must use Flask framework, front end uses HTML/CSS, system must run on a modern browser and user must have a stable internet connection.

3. Architectural Design

3.1



3.2 The system uses a monolithic architecture which includes all the components (Frontend and backend) in one application, this approach is simple to implement and deploy.

Components:

- Frontend: Provides all the different pages for the students, professors and admins representing the UI.

-Backend: Handles the business logic to execute the required functions such as adding/dropping courses, assigning grades, and so on, interacting with the database to retrieve data.

-Database: Stores entities such as the students, courses and professors and their relationships.

3.3

Backend: Python 3, Flask.

Frontend: HTML & CSS.

Database: SQLite.

Development and testing tools: VS Code, Chrome, Flask local server.

4. Detailed Design

4.1

4.1.1 MVC pattern is a way of decoupling system components to increase maintainability, it consists of a Model which handles the business logic and returns requested data, a View which is what the user sees and interacts with to request or view information, and the Controller which takes the input and links the Model and the View, deciding which Model to call and which View to return to the user.

4.1.2

Model: Student, Professor, Admin and Course classes which store each entity’s data and functions.

View: HTML/CSS templates such as the login, course registration and transcript pages.

Controller: student_controller, professor_controller, admin_controller and course_controller controllers to take user input from the View and route to the appropriate Model methods

4.1.3

Component	Responsibility
Model	Store and manage data for Student, Professor, Admin and Course entities Provide the core methods for each class: Student(add/drop, view registered courses, view transcript), Professor(assign student grade, manage courses, add/remove student), Admin(manage accounts, add/remove courses, assign courses to professors). Interact with database.
View	Display information that concerns each entity and take input from the user.
Controller	Receive input from the view and route to the appropriate model method to retrieve the data and select the appropriate view to display it.

4.1.4

1) The user performs an action in the UI for a specific entity.

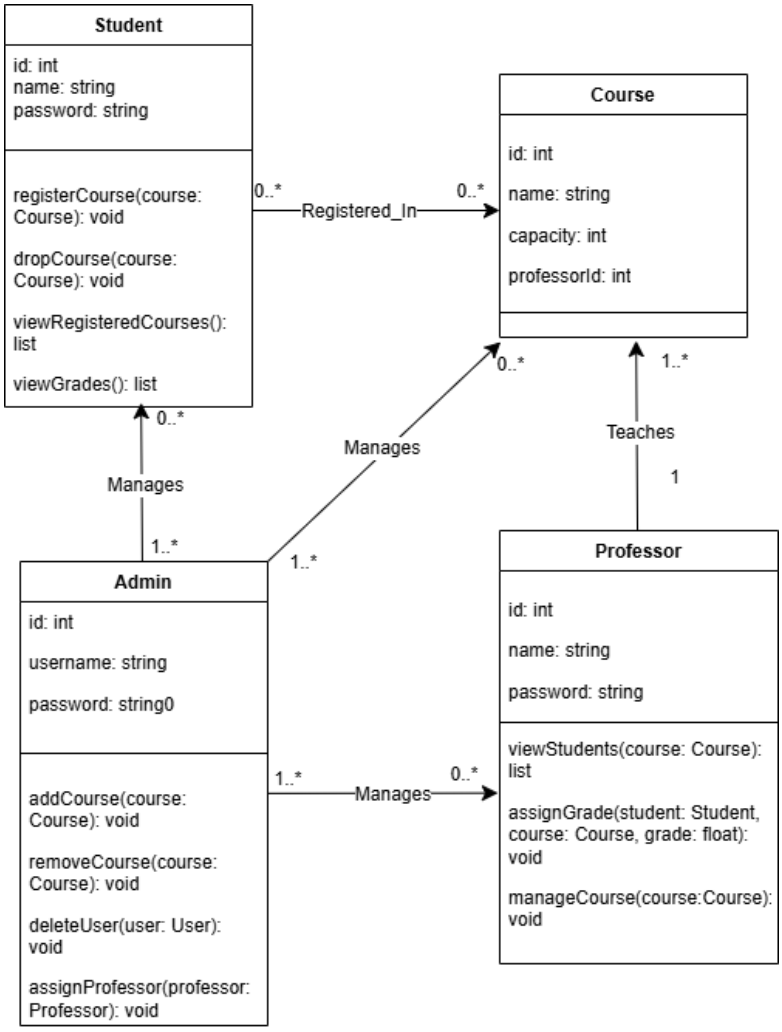
2) The Controller receives the input from the View, validates it, and calls the appropriate Model method.

3) The model performs the requested action that might involve retrieval of or modifying data in the database.

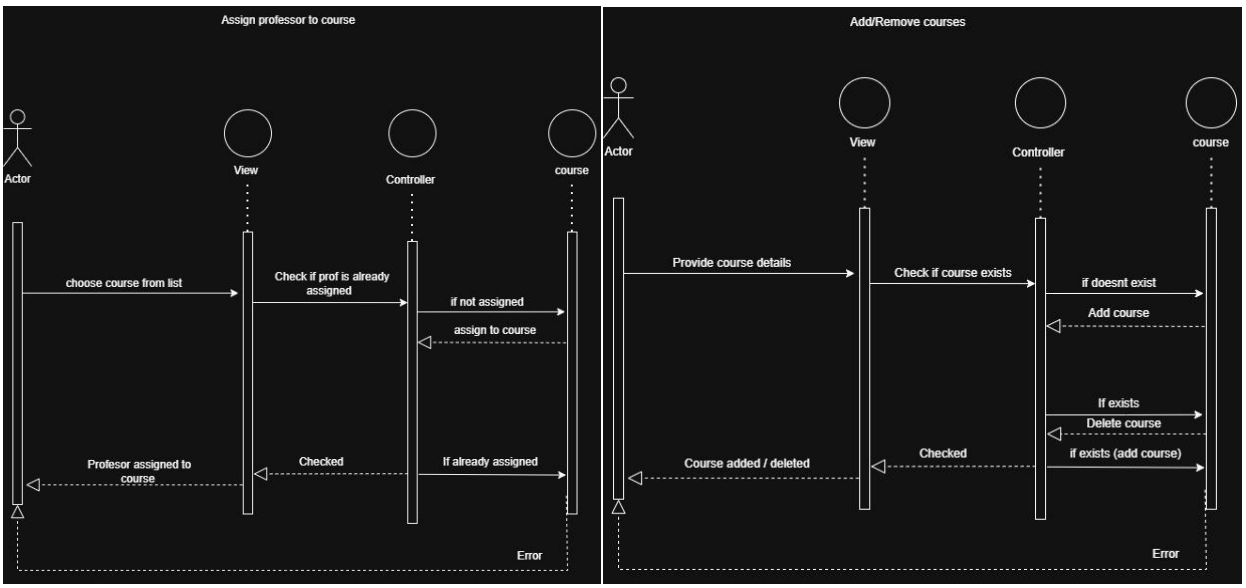
4) The Controller then displays the information returned from the Model using the appropriate template (View).

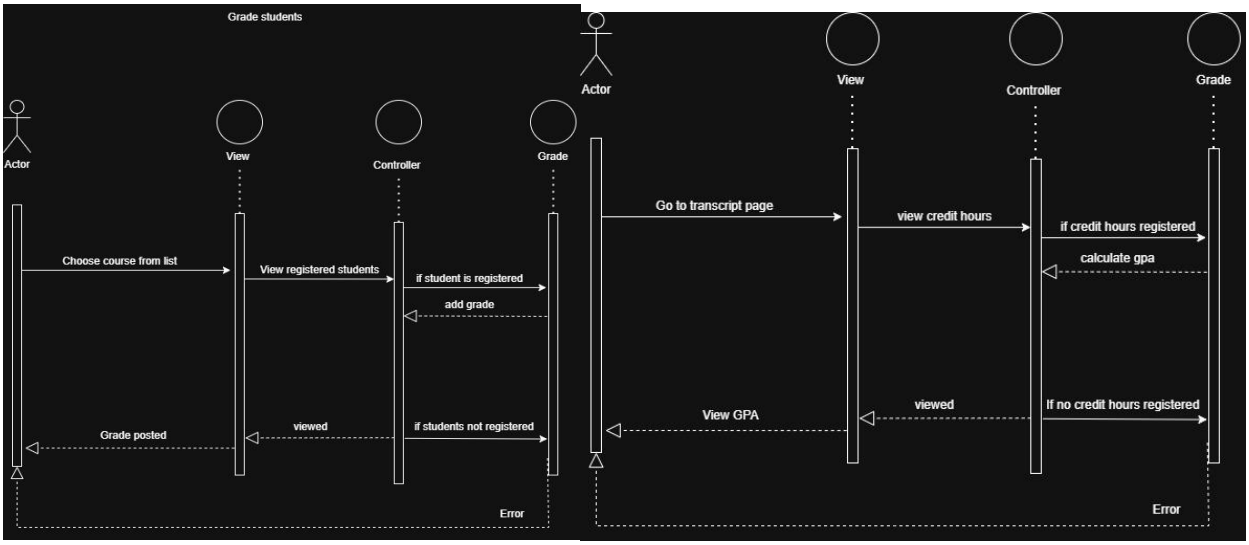
4.2

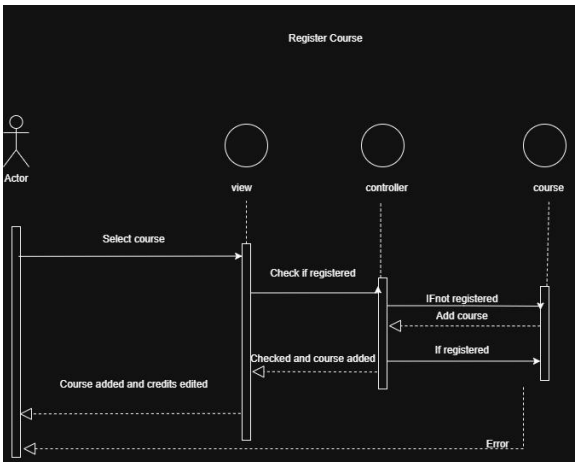
4.2.1


Student
id: int
name: string
password: string
registerCourse(course: Course): void
dropCourse(course: Course): void
viewRegisteredCourses(): list
viewGrades(): list
0..*
Registered_In
0..*
Course
id: int
name: string
capacity: int
professorId: int
0..*
1..*
Manages
0..*
Manages
1..*
1..*
Teaches
1
Admin
id: int
username: string
password: string0
addCourse(course: Course): void
removeCourse(course: Course): void
deleteUser(user: User): void
assignProfessor(professor: Professor): void
1..*
Manages
0..*
Professor
id: int
name: string
password: string
viewStudents(course: Course): list
assignGrade(student: Student, course: Course, grade: float): void
manageCourse(course:Course): void

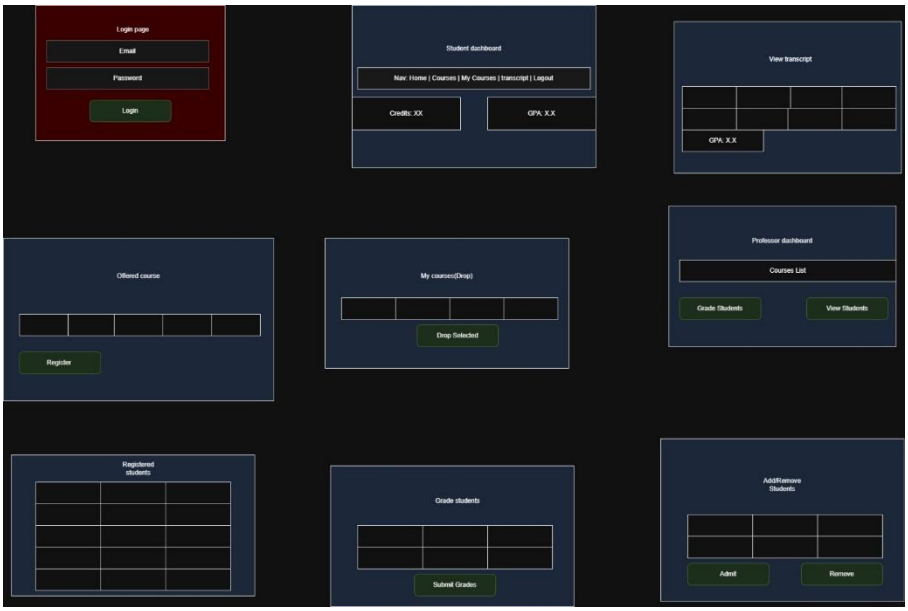
4.2.2


Assign professor to course
Actor
View
Controller
course
choose course from list
Check if prof is already assigned
if not assigned
assign to course
Profesor assigned to course
Checked
If already assigned
Error
Add/Remove courses
Actor
View
Controller
course
Provide course details
Check if course exists
if doesnt exist
Add course
If exists
Delete course
Course added / deleted
Checked
if exists (add course)
Error


Grade students
Actor
View
Controller
Grade
Choose course from list
View registered students
if student is registered
add grade
Grade posted
viewed
if students not registered
Error
Actor
View
Controller
Grade
Go to transcript page
view credit hours
if credit hours registered
calculate gpa
View GPA
viewed
If no credit hours registered
Error

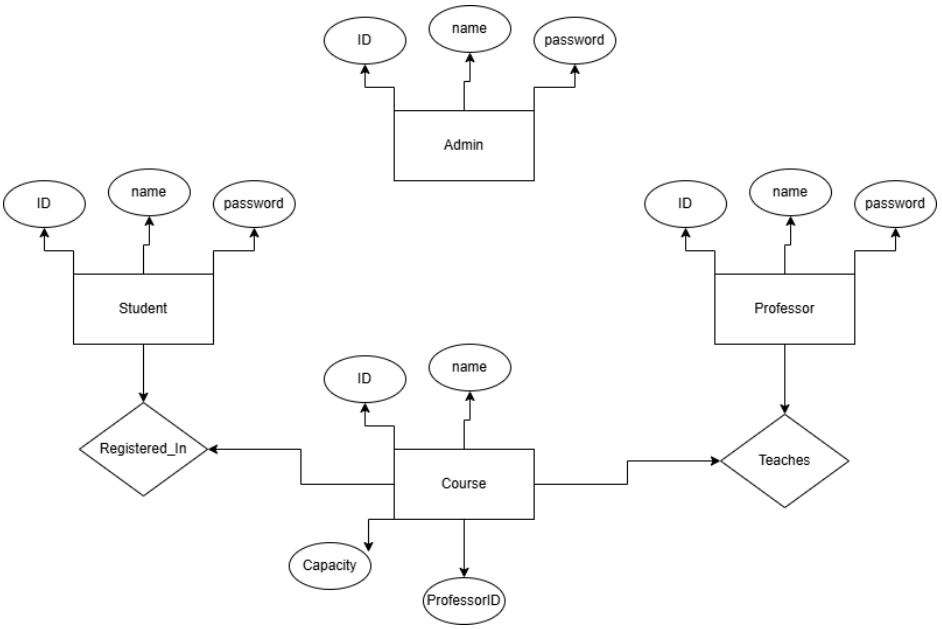

Register Course
Actor
view
controller
course
Select course
Check if registered
IFnot registered
Add course
If registered
Checked and course added
Course added and credits edited
Error

4.3


Login page
Email
Password
Login
Student dashboard
Nav: Home | Courses | My Courses | transcript | Logout
Credits: XX
GPA: X.X
View transcript
GPA: X.X
Offered course
Register
My courses(Drop)
Drop Selected
Professor dashboard
Courses List
Grade Students
View Students
Registered students
Grade students
Submit Grades
Add/Remove Students
Admit
Remove

4.4

4.4.1



4.4.2

Student

Attribute	Type	Description
id	INT (PK)	Unique identifier for each student.
name	VARCHAR	Student's full name.
password	VARCHAR	Password used for authentication.

Professor

Attribute	Type	Description
id	INT (PK)	Unique identifier for each professor.

name	VARCHAR	Professor’s full name.
password	VARCHAR	Password used for authentication.

Admin

Attribute	Type	Description
id	INT (PK)	Unique identifier for each admin.
name	VARCHAR	Admin’s full name.
password	VARCHAR	Password used for authentication.

Course

Attribute	Type	Description
id	INT (PK)	Unique identifier for each course.
name	VARCHAR	Course name.
professorID	INT (FK)	ID for the teaching professor.
Capacity	INT	Maximum number of students the course can hold.

5. Conclusion

5.1 The design document defined the system with a simple monolithic architecture, MVC pattern and a simple SQL database using clear UML diagrams to guide the implementation. These design choices ensure that the implementation of the system will be straightforward, maintainable and easy to deploy.