College Enrollment System

Software Design Document

Revision History

Date	Revision	Description	Author
03/12/2025	1.0	Intial Version	Aakkash, Christian, Svein, Felipe
03/14/2025	1.1	Finished all the formatting, added Revision history and table of contents.	Aakkash, Christian, Svein, Felipe
03/19/2025	1.2	Started working on the introduction of the Document	Aakkash, Christian, Svein, Felipe
03/20/2025	2.0	Finished the Introduction section and am moving on to System Architecture	Aakkash, Christian, Svein, Felipe
03/22/2025	2.1	Finished the System Architecture section	Aakkash, Christian, Svein, Felipe
03/25/2025	3.0	Started Adding Possible classes and a class diagram with each one	Aakkash, Christian, Svein, Felipe
03/27/2025	3.1	Added more classes and diagrams	Aakkash, Christian, Svein, Felipe
03/29/2025	3.2	Finished the Classes list and added a full class diagram with all the relationships	Aakkash, Christian, Svein, Felipe
04/02/2025	4.0	Started the Use Cases	Aakkash, Christian, Svein, Felipe
04/05/2025	4.1	Added all the Use Cases and a Use Case Diagram for each one	Aakkash, Christian, Svein, Felipe
04/06/2025	4.2	Started on the Sequence Diagrams	Aakkash, Christian, Svein, Felipe
04/07/2025	4.3	Finished all the Sequence Diagrams	Aakkash, Christian, Svein, Felipe

Table Of Contents

1. Introduction

1.1. Purpose of the Design Document

1.1.1. This document provides a comprehensive design of the College Enrollment System, elaborating on and expanding the specifications laid out in the Software Requirements Specification. It outlines the system architecture, component responsibilities, class designs, communication models, and implementation planning

1.2. Scope of the System

1.2.1. The College Enrollment System is a Java-based desktop application designed to facilitate student enrollment and course management across multiple universities. It enables administrators to create and manage course schedules, while students can enroll, drop, and manage their course selections. The system supports prerequisite enforcement, waitlists, reporting, and handles holds on student accounts due to outstanding balances.

1.3. Intended Audience

1.3.1. **Development Team:** To understand the system architecture and design decisions.

1.3.2. **Quality Assurance Team:** To develop test plans based on the system design.

1.3.3. **Project Stakeholders:** To gain insights into the system's design and ensure it meets business requirements.

1.3.4. **Maintenance Team:** For future reference during system updates or troubleshooting.

1.4. Definitions and Abbreviations

1.4.1. **GUI:** Graphical User Interface

1.4.2. **TCP/IP:** Transmission Control Protocol/Internet Protocol

1.4.3. **Admin:** University personnel responsible for managing course offerings

1.4.4. **Student:** University member enrolling in courses

1.4.5. **SRS:** Software Requirements Specification

1.4.6. **UML:** Unified Modeling Language

2. System Architecture

2.1. Overview Diagram

2.1.1. The system follows a client-server architecture. The client application provides the GUI for user interactions, while the server handles business logic and data processing. Communication between the client and server occurs over TCP/IP.

2.2. Component Breakdown

2.2.1. Client GUI (Student/Admin): Provides interfaces for students and administrators to interact with the system.

2.2.2. Server Logic: Manages core functionalities and processes client requests.

2.2.2.1. Course Manager: Handles the creation, updating, deletion, and retrieval of course information.

2.2.2.2. Enrollment Engine: Manages student enrollments, drops, and waitlists, enforcing prerequisites and enrollment limits.

2.2.2.3. Report Generator: Produces reports on course enrollments, waitlists, and available seats.

2.2.2.4. Prerequisite Checker: Verifies if students meet course prerequisites before enrollment.

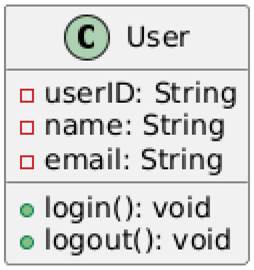
2.3. Communication Flow

2.3.1. The client and server communicate using a TCP/IP-based request/response model. The client sends requests (e.g., enroll in a course), and the server processes these requests and returns appropriate responses.

3. Class Design

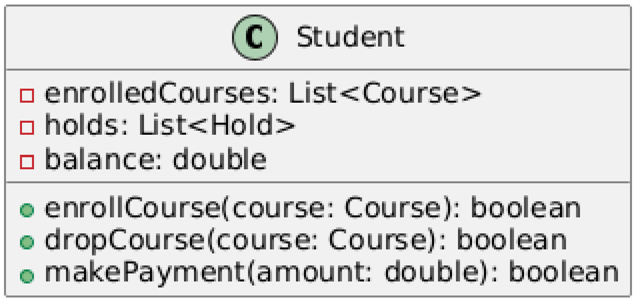
3.1. User (abstract)

3.1.1. base class for all users



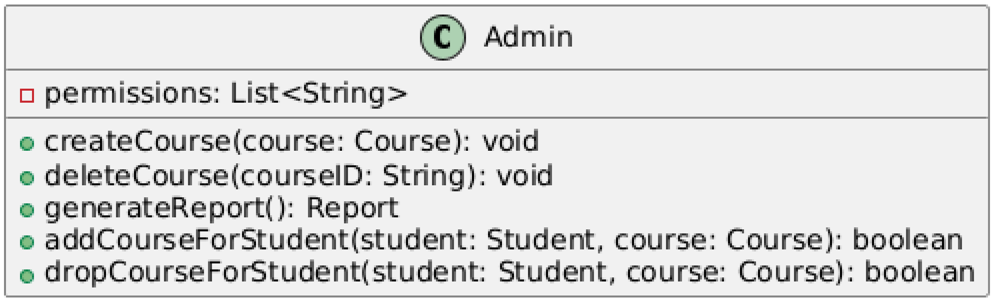
3.2. Student

3.2.1. inherits from User, performs enrollments


C Student
enrolledCourses: List<Course>
holds: List<Hold>
balance: double
enrollCourse(course: Course): boolean
dropCourse(course: Course): boolean
makePayment(amount: double): boolean

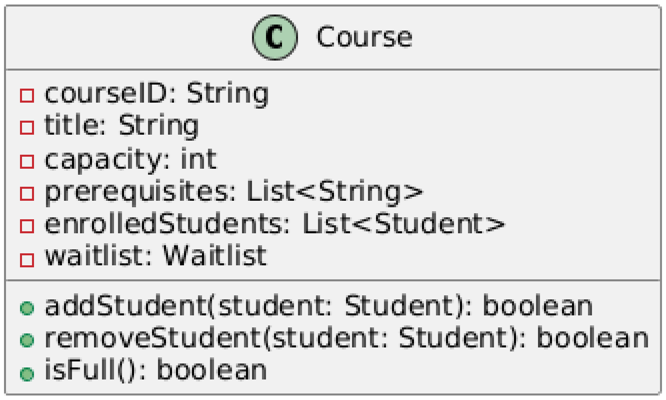
3.3. Admin

3.3.1. inherits from User, manages courses


C Admin
permissions: List<String>
createCourse(course: Course): void
deleteCourse(courseID: String): void
generateReport(): Report
addCourseForStudent(student: Student, course: Course): boolean
dropCourseForStudent(student: Student, course: Course): boolean

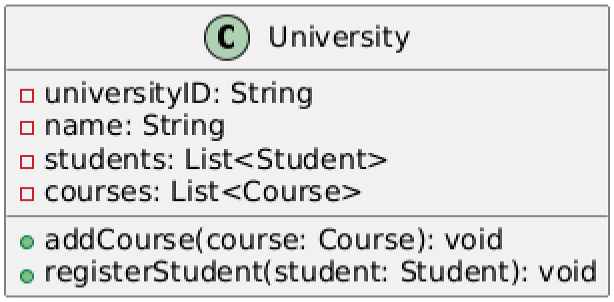
3.4. Course

3.4.1. holds course details


Course
courseID: String
title: String
capacity: int
prerequisites: List<String>
enrolledStudents: List<Student>
waitlist: Waitlist
addStudent(student: Student): boolean
removeStudent(student: Student): boolean
isFull(): boolean

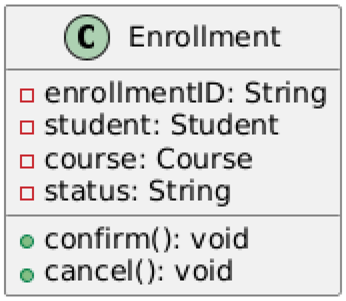
3.5. University

3.5.1. contains students and courses



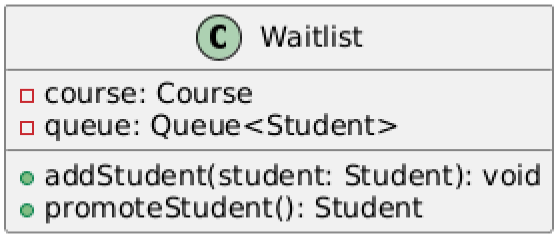
3.6. Enrollment

3.6.1. links student and course



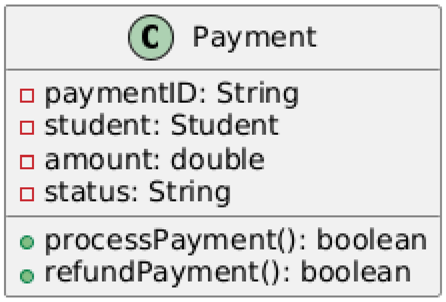
3.7. Waitlist

3.7.1. manages students waiting for full courses



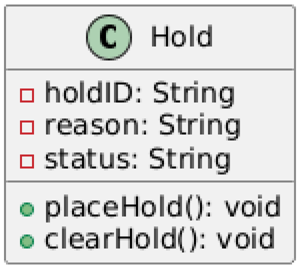
3.8. Payment

3.8.1. handles course payments



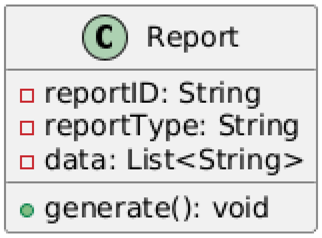
3.9. Hold

3.9.1. represents account holds (e.g., unpaid balance)



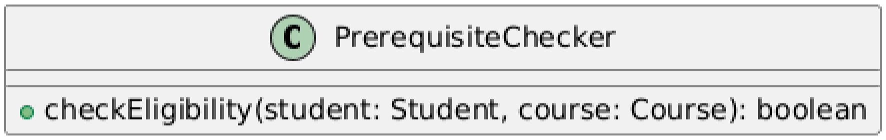
3.10. Report

3.10.1. auto generated for analysis



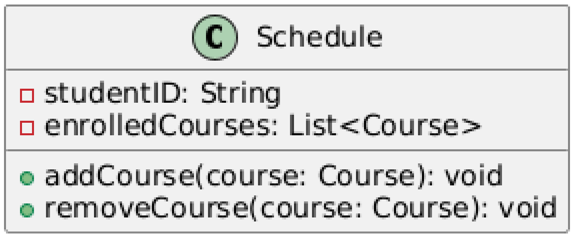
3.11. PrerequisiteChecker

3.11.1. validates enrollment requirements



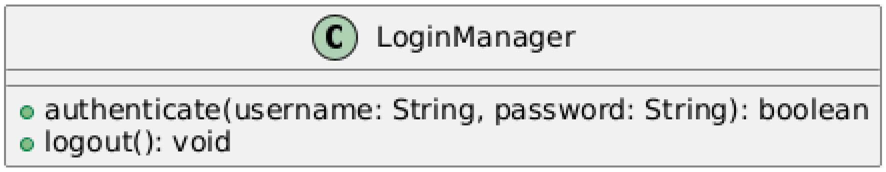
3.12. Schedule

3.12.1. tracks student schedule


C Schedule
studentID: String
enrolledCourses: List<Course>
addCourse(course: Course): void
removeCourse(course: Course): void

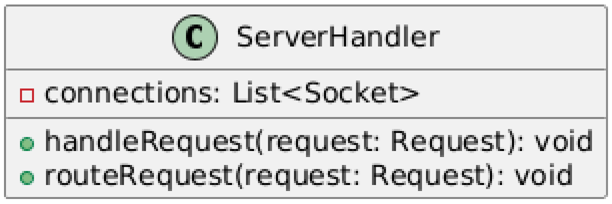
3.13. LoginManager

3.13.1. handles authentication



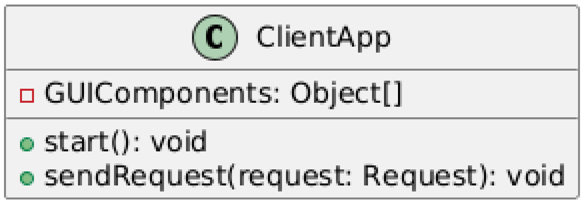
3.14. ServerHandler

3.14.1. manages server logic and routing


C ServerHandler
connections: List<Socket>
handleRequest(request: Request): void
routeRequest(request: Request): void

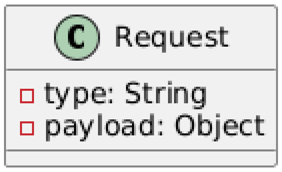
3.15. ClientApp

3.15.1. GUI-based application on the client side



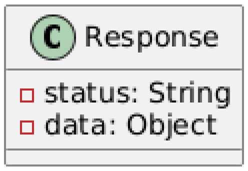
3.16. Request

3.16.1. used for client-server communication



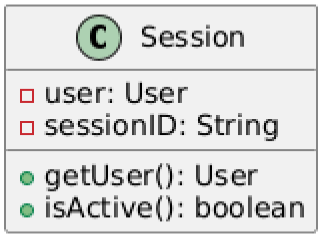
3.17. Response

3.17.1. result returned to client from server



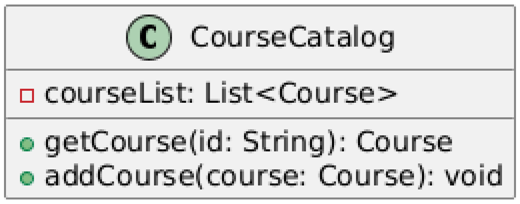
3.18. Session

3.18.1. holds logged-in user session info



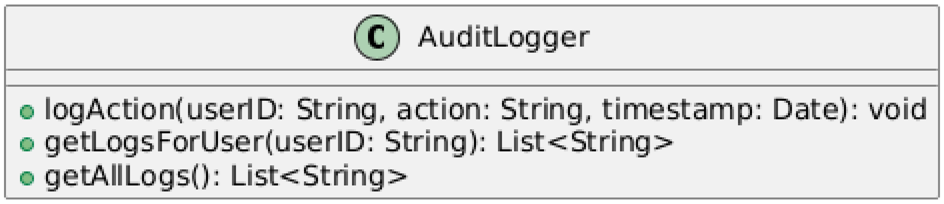
3.19. CourseCatalog

3.19.1. a list of all courses in the system

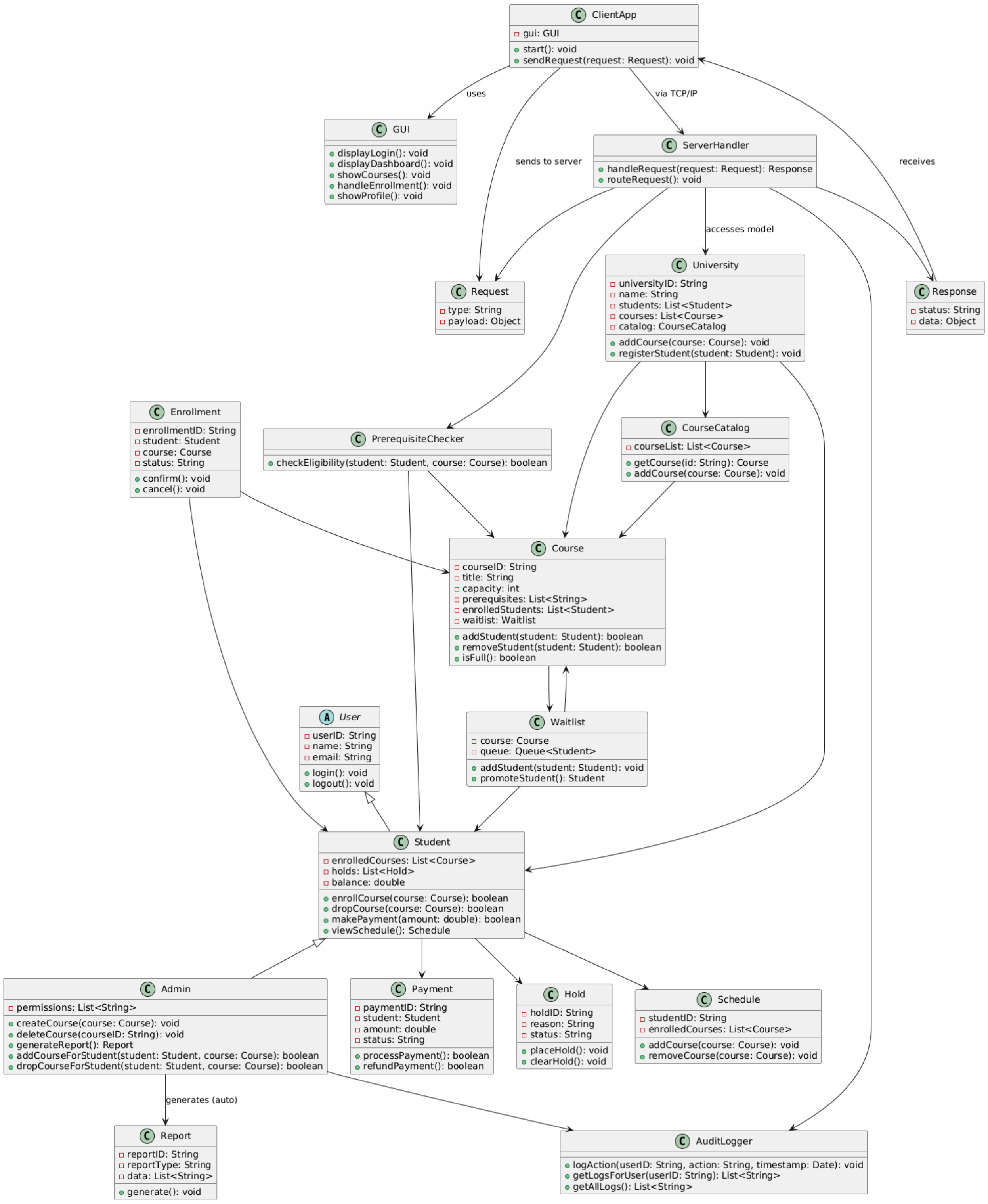

CourseCatalog
courseList: List<Course>
getCourse(id: String): Course
addCourse(course: Course): void

3.20. AuditLogger

3.20.1. tracks and logs important actions


AuditLogger
logAction(userID: String, action: String, timestamp: Date): void
getLogsForUser(userID: String): List<String>
getAllLogs(): List<String>

4. **Class Diagram**

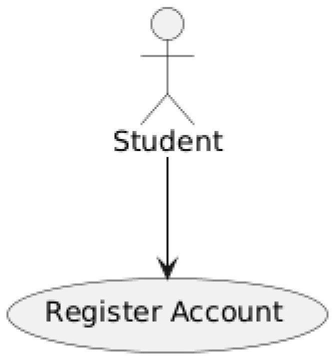

ClientApp
- gui: GUI
+ start(): void
+ sendRequest(request: Request): void
uses
via TCP/IP
sends to server
receives
GUI
+ displayLogin(): void
+ displayDashboard(): void
+ showCourses(): void
+ handleEnrollment(): void
+ showProfile(): void
ServerHandler
+ handleRequest(request: Request): Response
+ routeRequest(): void
accesses model
Request
- type: String
- payload: Object
University
- universityID: String
- name: String
- students: List<Student>
- courses: List<Course>
- catalog: CourseCatalog
+ addCourse(course: Course): void
+ registerStudent(student: Student): void
Response
- status: String
- data: Object
Enrollment
- enrollmentID: String
- student: Student
- course: Course
- status: String
+ confirm(): void
+ cancel(): void
PrerequisiteChecker
+ checkEligibility(student: Student, course: Course): boolean
CourseCatalog
- courseList: List<Course>
+ getCourse(id: String): Course
+ addCourse(course: Course): void
Course
- courseID: String
- title: String
- capacity: int
- prerequisites: List<String>
- enrolledStudents: List<Student>
- waitlist: Waitlist
+ addStudent(student: Student): boolean
+ removeStudent(student: Student): boolean
+ isFull(): boolean
User
- userID: String
- name: String
- email: String
+ login(): void
+ logout(): void
Waitlist
- course: Course
- queue: Queue<Student>
+ addStudent(student: Student): void
+ promoteStudent(): Student
Student
- enrolledCourses: List<Course>
- holds: List<Hold>
- balance: double
+ enrollCourse(course: Course): boolean
+ dropCourse(course: Course): boolean
+ makePayment(amount: double): boolean
+ viewSchedule(): Schedule
Admin
- permissions: List<String>
+ createCourse(course: Course): void
+ deleteCourse(courseID: String): void
+ generateReport(): Report
+ addCourseForStudent(student: Student, course: Course): boolean
+ dropCourseForStudent(student: Student, course: Course): boolean
Payment
- paymentID: String
- student: Student
- amount: double
- status: String
+ processPayment(): boolean
+ refundPayment(): boolean
Hold
- holdID: String
- reason: String
- status: String
+ placeHold(): void
+ clearHold(): void
Schedule
- studentID: String
- enrolledCourses: List<Course>
+ addCourse(course: Course): void
+ removeCourse(course: Course): void
generates (auto)
Report
- reportID: String
- reportType: String
- data: List<String>
+ generate(): void
AuditLogger
+ logAction(userID: String, action: String, timestamp: Date): void
+ getLogsForUser(userID: String): List<String>
+ getAllLogs(): List<String>

5. Use Cases

5.1. UC01: Student Registration

5.1.1. Actor: Student

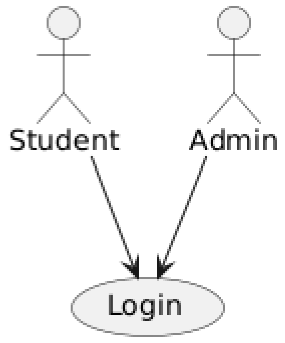
5.1.2. Description: A new user registers by entering personal details. The system creates a new student account.



5.2. UC02: Login

5.2.1. Actor: Student, Admin

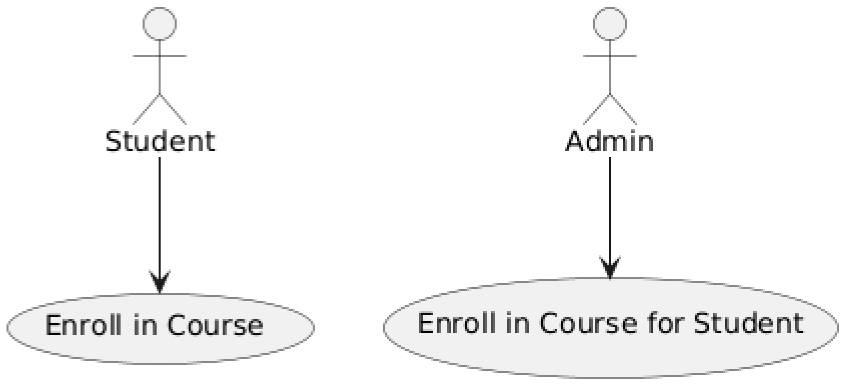
5.2.2. Description: The User enters credentials to gain access to the system with role-based permissions.



5.3. UC03: Course Enrollment

5.3.1. Actor: Student, Admin

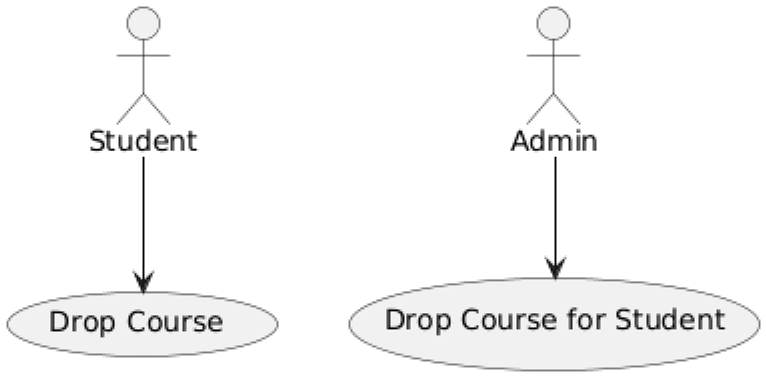
5.3.2. Description: Enroll in available courses if space allows and prerequisites are met. Admin can enroll students, too.



5.4. UC04: Drop Course

5.4.1. Actor: Student, Admin

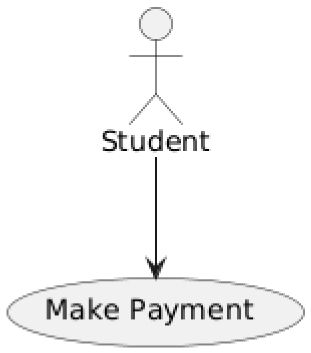
5.4.2. Description: Drop a course before the deadline. The admin can drop a course for any student.



5.5. UC05: Payment Processing

5.5.1. Actor: Student

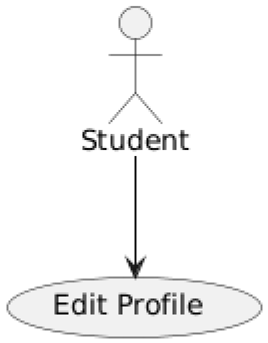
5.5.2. Description: Make tuition payments through the client system. The payment status has been updated accordingly.



5.6. UC06: Manage Student Profile

5.6.1. Actor: Student

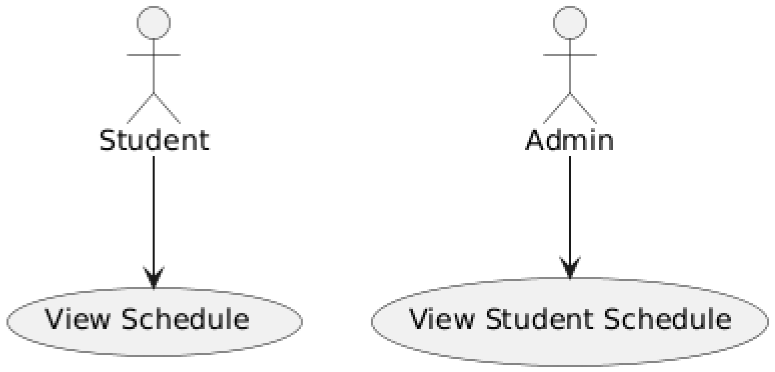
5.6.2. Description: Update name, email, and other personal profile information.



5.7. UC07: View Schedule

5.7.1. Actor: Student, Admin

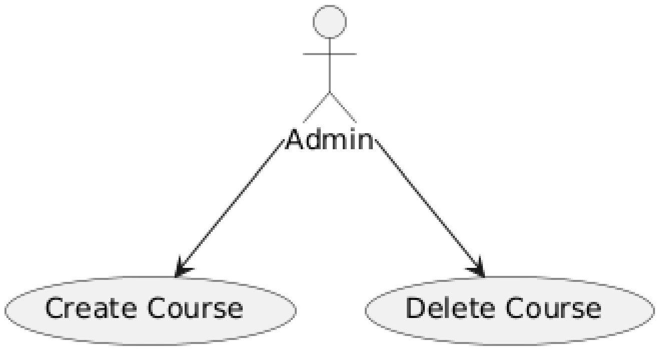
5.7.2. Description: View enrolled course schedule. The admin can view any student's schedule.



5.8. UC08: Admin Creates/Deletes Course

5.8.1. Actor: Admin

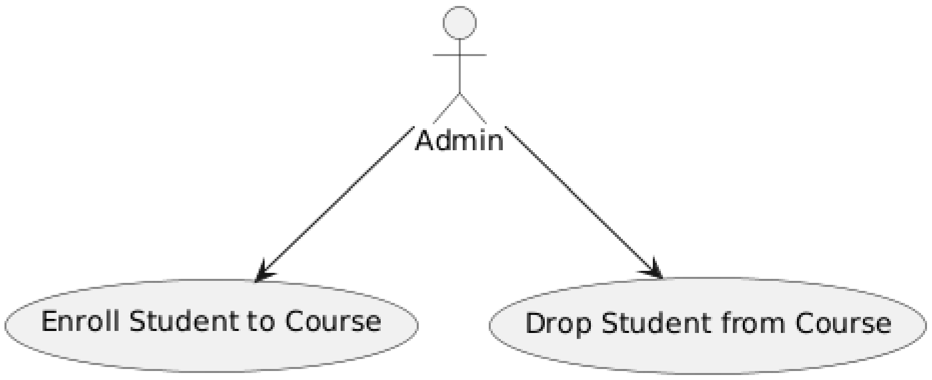
5.8.2. Description: The Admin can create or delete courses from the catalog.



5.9. UC09: Admin Manages Student Enrollment

5.9.1. Actor: Admin

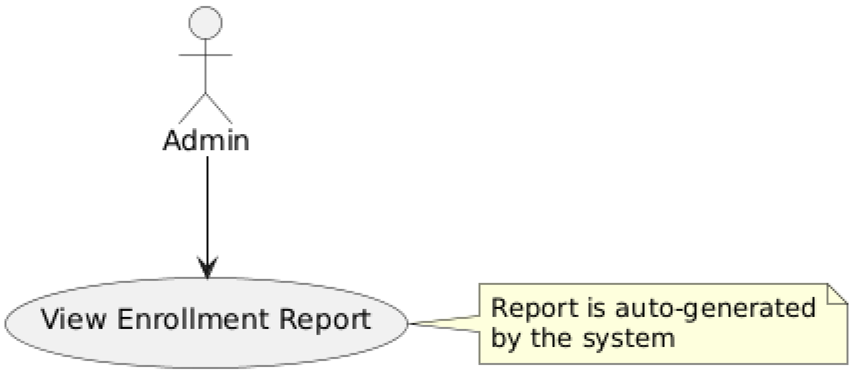
5.9.2. Description: The Admin can manually enroll or drop students from specific courses.



5.10. UC10: Auto-Generate Reports

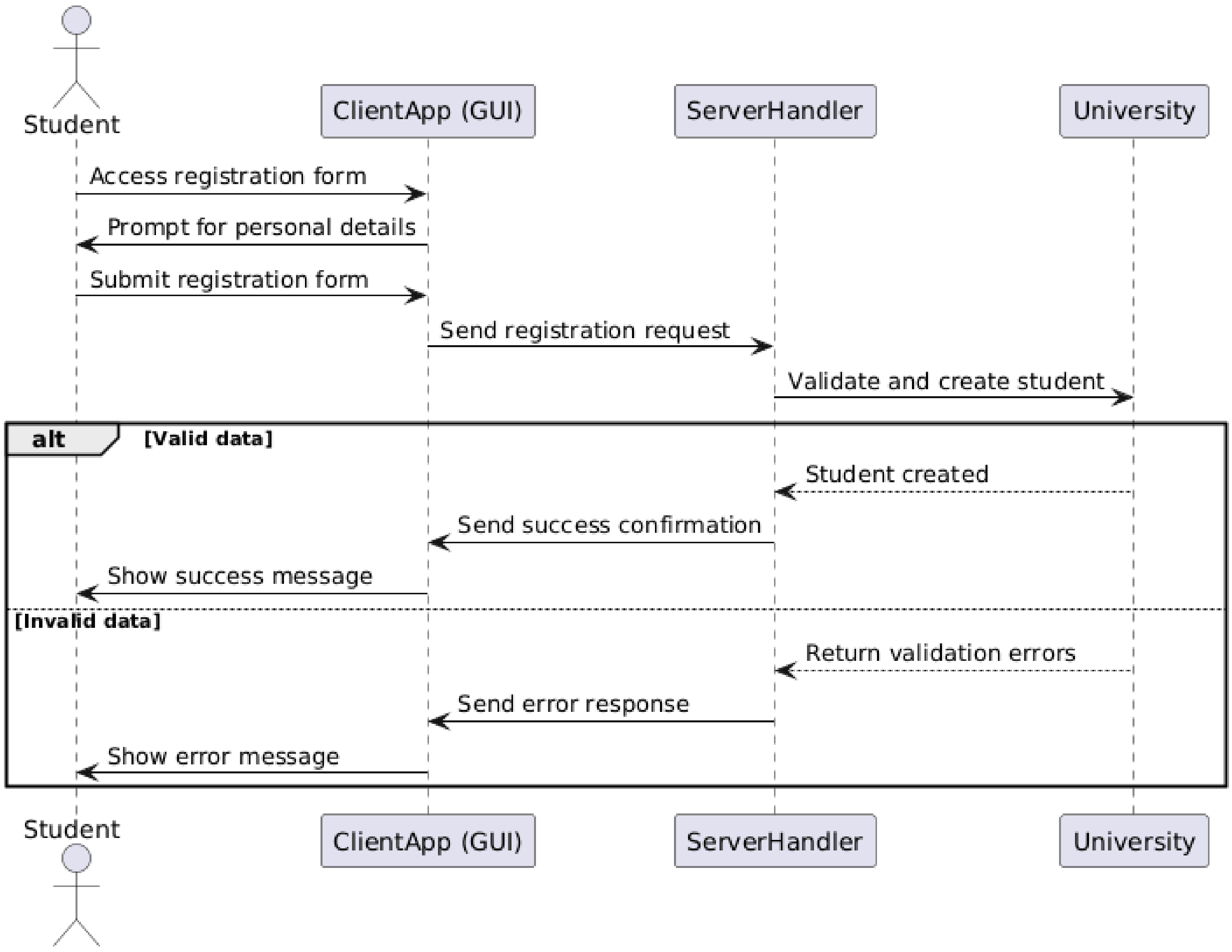
5.10.1. Actor: Admin

5.10.2. Description: Reports on course enrollment, waitlists, and student account statuses are auto-generated and viewable by Admin.

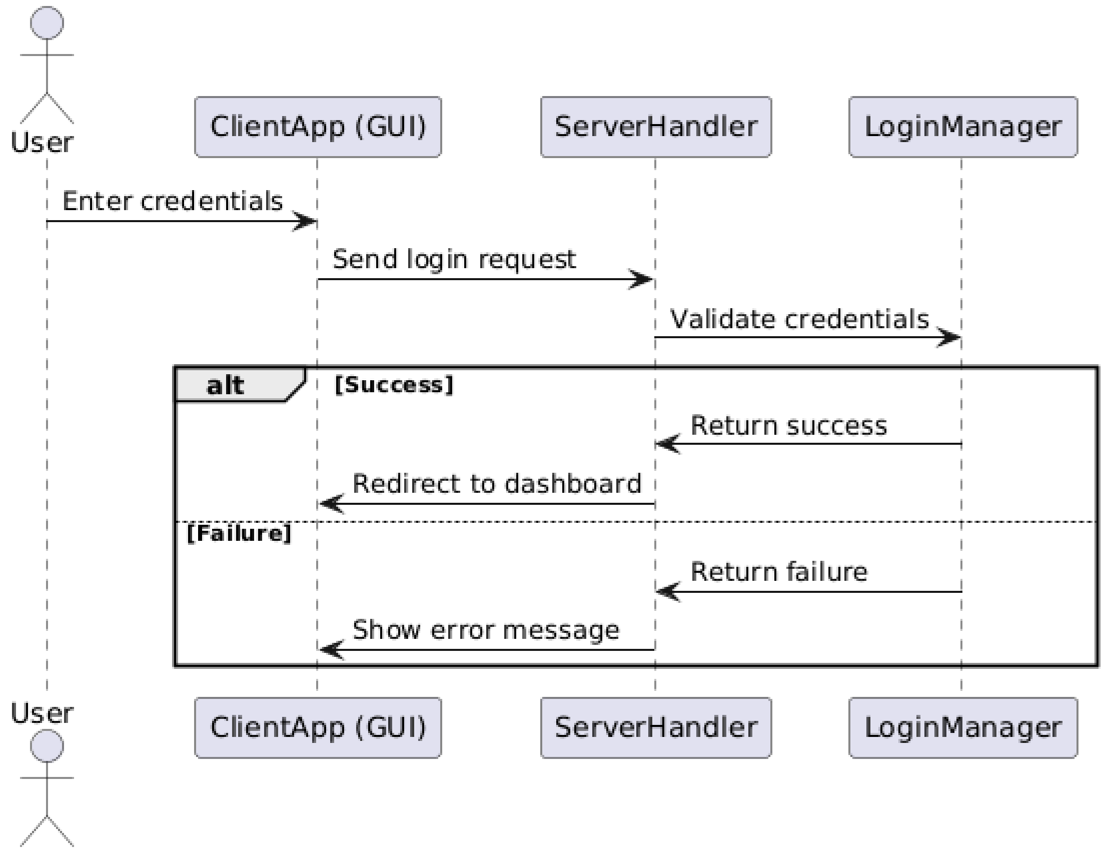


6. Sequence Diagrams

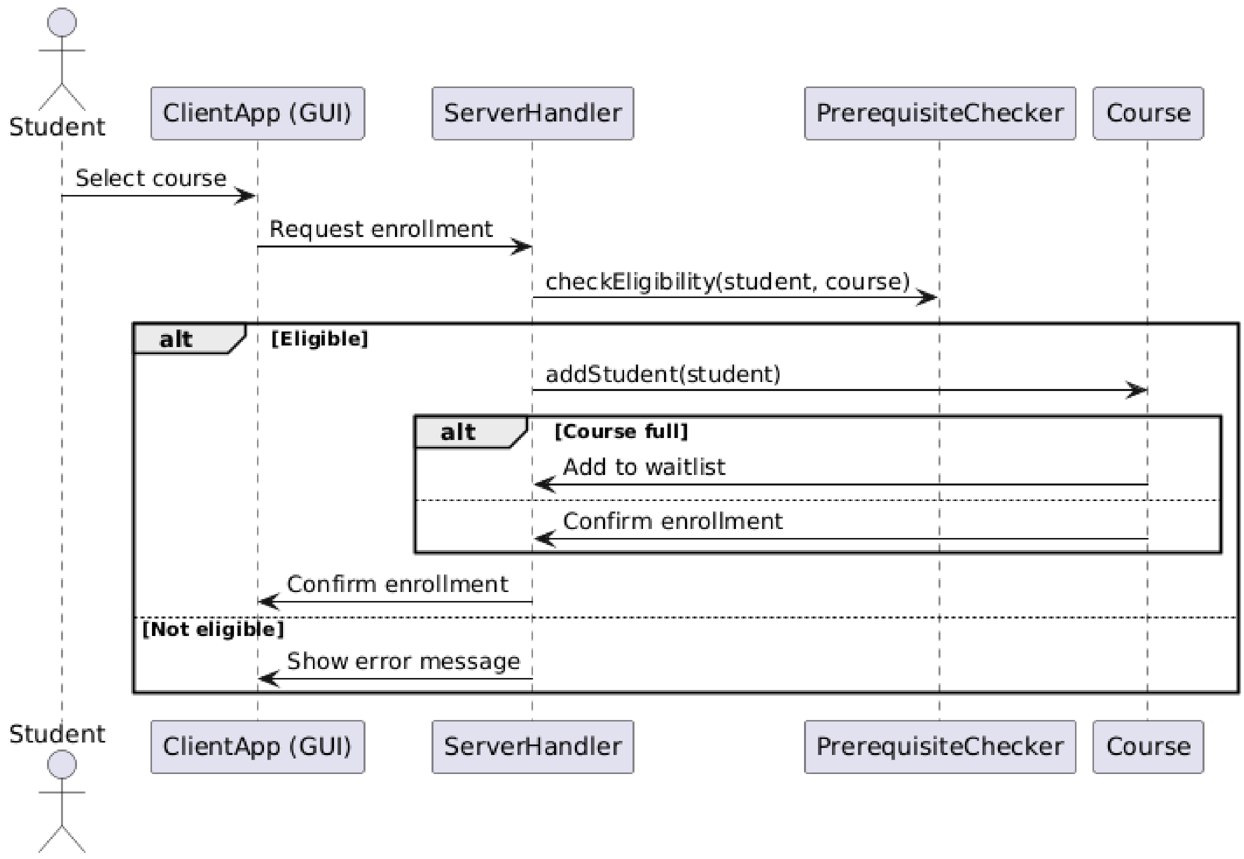
6.1. Student Registration



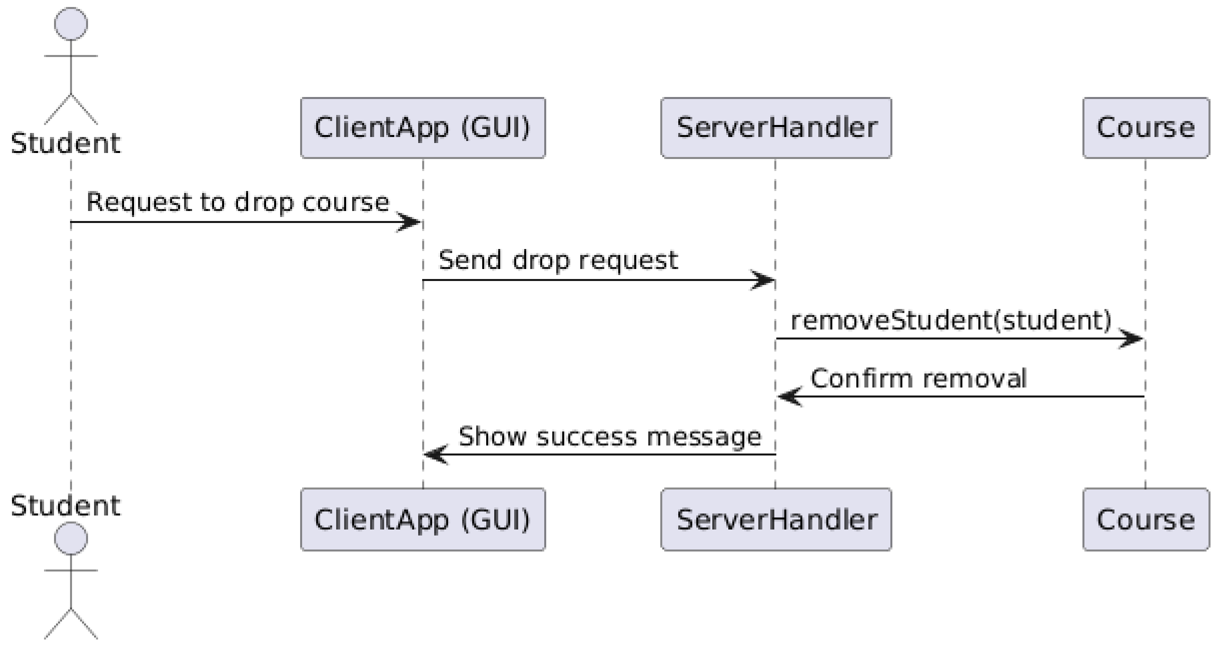
6.2. Login


User
ClientApp (GUI)
ServerHandler
LoginManager
Enter credentials
Send login request
Validate credentials
alt
[Success]
Return success
Redirect to dashboard
[Failure]
Return failure
Show error message

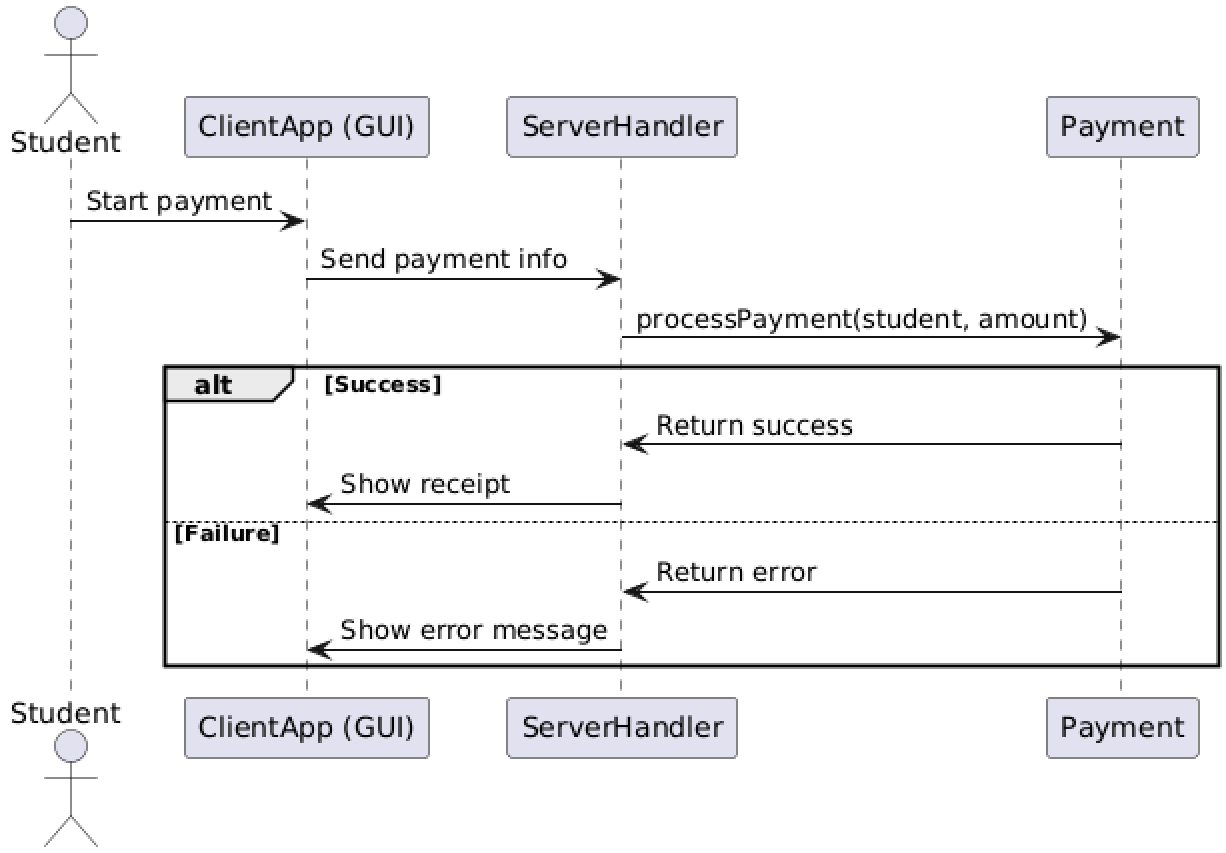
6.3. Course Enrollment



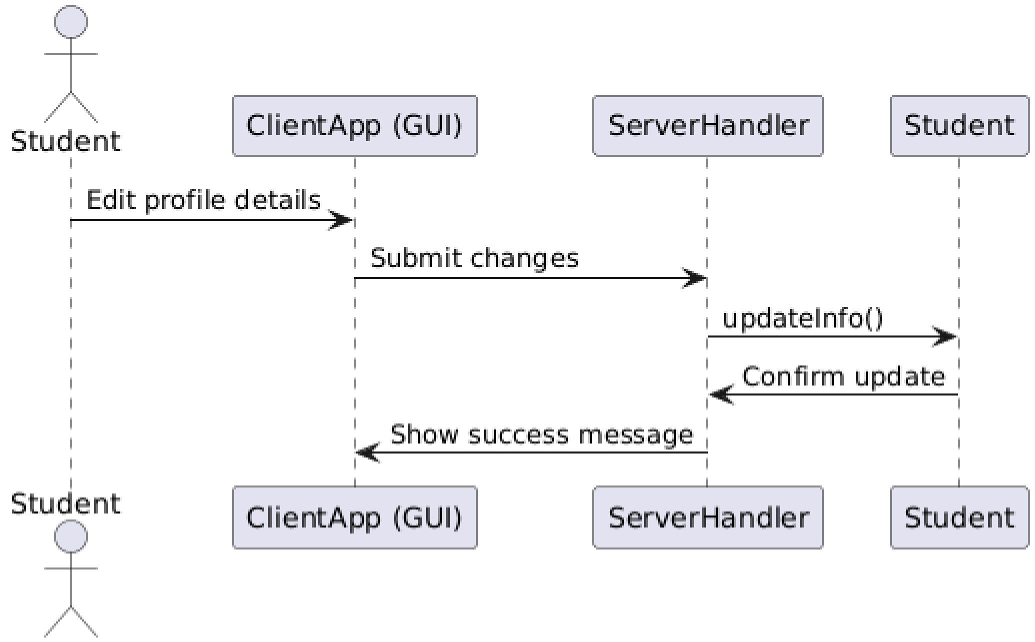
6.4. Drop Course



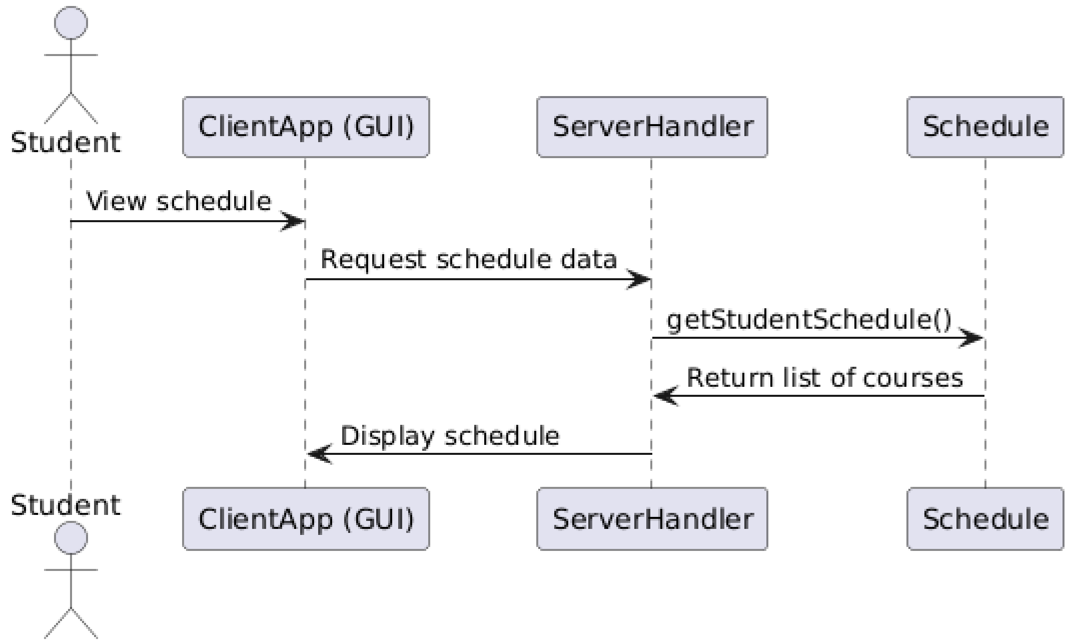
6.5. Payment Processing



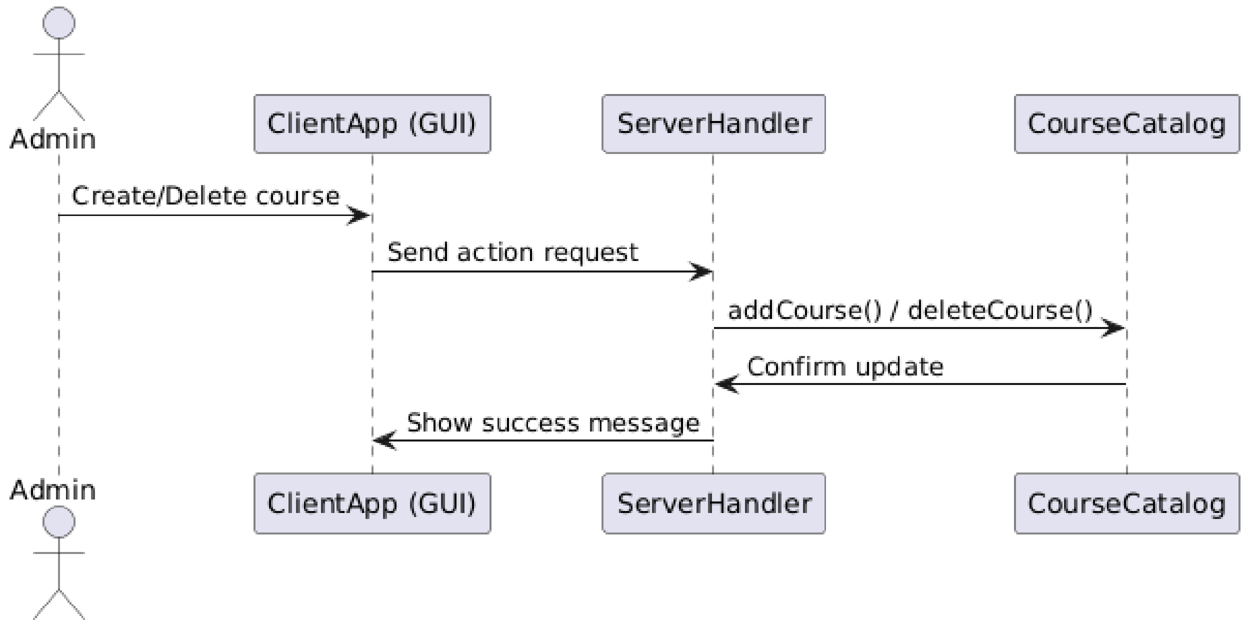
6.6. Manage Student Profile



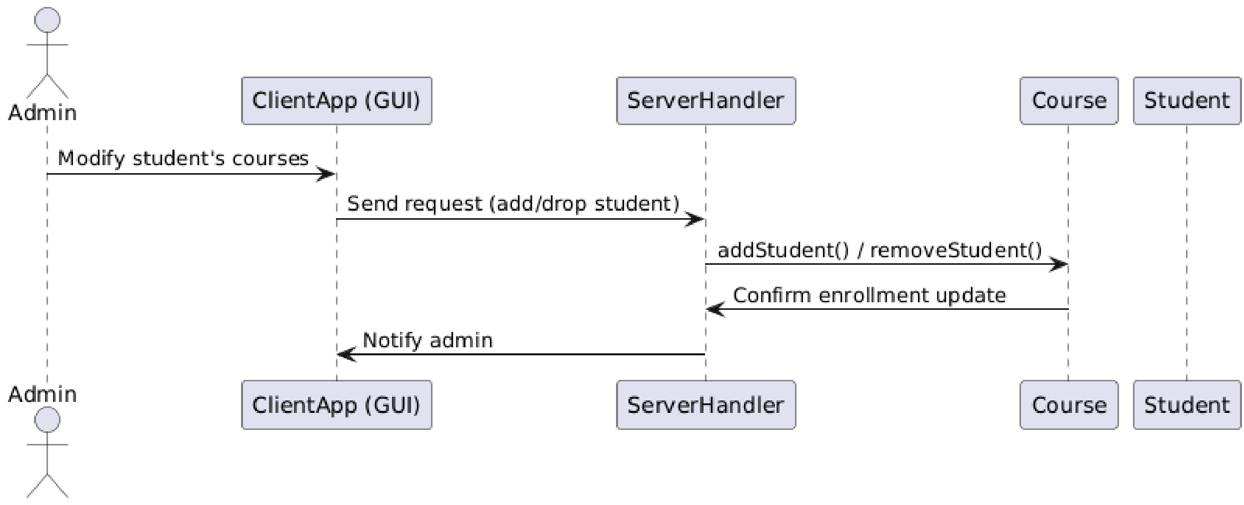
6.7. View Schedule



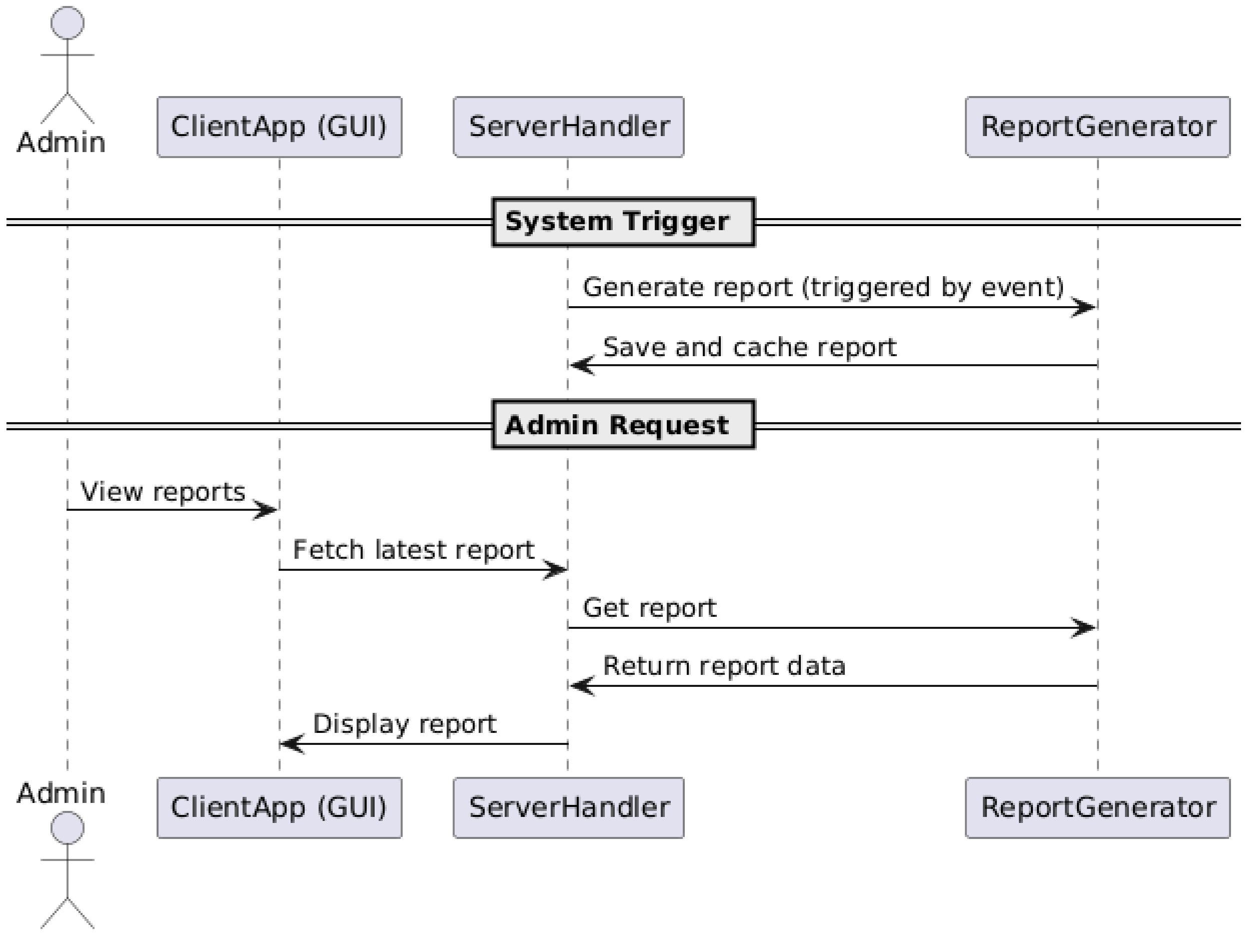
6.8. Admin Creates/Deletes Course



6.9. Admin Manages Student Enrollment


Admin
ClientApp (GUI)
ServerHandler
Course
Student
Modify student's courses
Send request (add/drop student)
addStudent() / removeStudent()
Confirm enrollment update
Notify admin

6.10. Auto-Generate Reports


Admin
ClientApp (GUI)
ServerHandler
ReportGenerator
System Trigger
Generate report (triggered by event)
Save and cache report
Admin Request
View reports
Fetch latest report
Get report
Return report data
Display report