Course Registration System

https://github.com/Ivan-Wang-tech/course-registration-system

Group#8:
Ivan Wang (yw6505)
Zhengxuan Tian (zt2254)
Hanlin Yan (hy2484)

Date of Submission: Dec 12, 2025

Table of Work

(Please write x in the boxes to mention what each student achieved in this project)

	Ivan Wang	ZhengXuan Tian	Hanlin Yan
Project Description	X	X	X
Use Case Diagram(s) and description			X
Sequence Diagrams			X
Class diagram(s)	X		
Implementation	X	X	
Conclusion		X	

Table of Contents

1. System Analysis

1.1 Project description

1. Project Overview

We propose to develop a Course Registration System that allows students to browse available courses, view course details, register for courses, drop courses, and view their current schedule.

---

2. System Features

The system will support the following core functionalities:

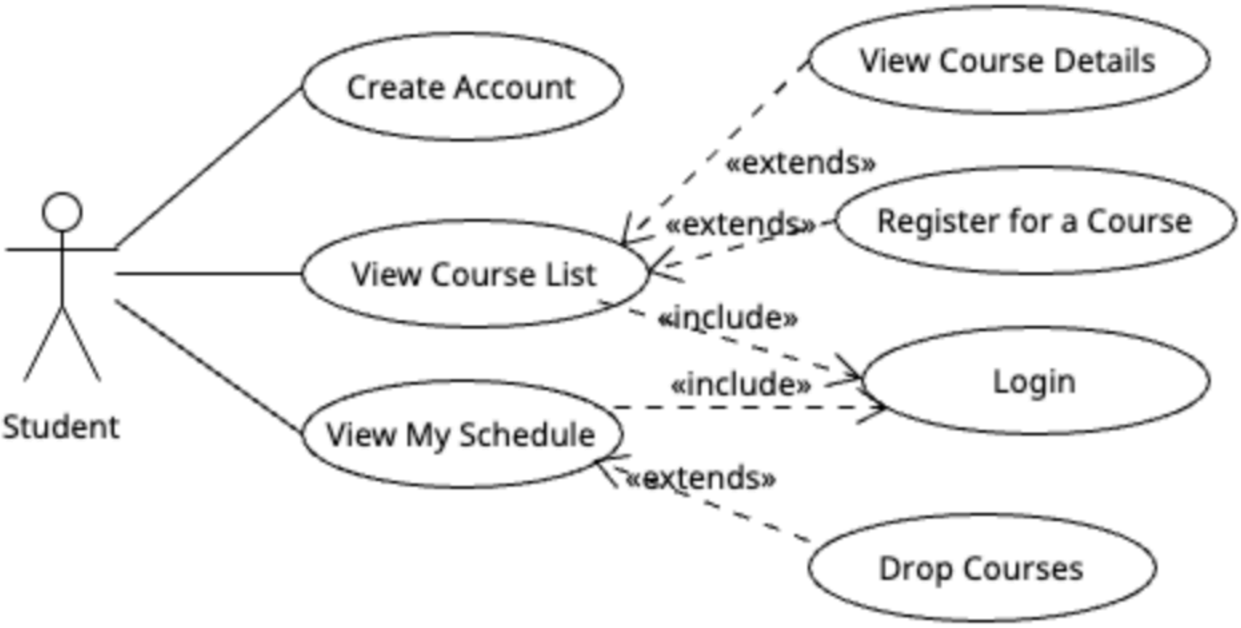
- Student Registration & Login
  Students can create an account and log in before accessing system features.
- View Course List
  Students can browse all available courses.
- View Course Details
  Students can view detailed information of a selected course (course title, instructor, credits, capacity, and number of enrolled students).
- Register for a Course
  Students can register for a course if:
  - the course is not full, and
  - the student is not already registered for it.
- Drop a Course
  Students can remove a previously registered course.
- View My Courses
  Students can view all courses they are currently enrolled in.

---

3. System Design Notes

- ❖ The system will use in-memory data structures (lists in Python) to store course and registration information.
  A database backend will not be used to maintain simplicity.
- ❖ The design will follow object-oriented principles with classes:
  - ➢ UI class: MainGUI
  - ➢ Entity class: Student, Course, Registration
  - ➢ Controller classes: RegistrationSystem(Coordinate the following three managers), CourseCatalog(controller for course), RegistrationManager(controller for registration), AuthenticationService(controller for student)

1.2 Use Cases Diagram(s) and Use Cases Description



UC Reference Name/Number:	Create Account /UC1
Overview	A student can create an account on the course registration system
Related use cases:	N/A
Actors	Student

UC Reference Name/Number:	Login /UC2
Overview	A student must log in before showing other features
Related use cases:	N/A
Actors	Student

UC Reference Name/Number:	View Course List /UC3
Overview	Display all available courses in the system
Related use cases:	Include Login
Actors	Student

UC Reference Name/Number:	View Course Details/UC4
Overview	Show selected course details, including: course title, instructor, credits, capacity, current number of enrolled students
Related use cases:	Extends View Course List
Actors	Student

UC Reference Name/Number:	Register for a Course /UC5
Overview	A student can register for a course if the course is not full, and the student is not already registered for this course
Related use cases:	Extends View Course List
Actors	Student

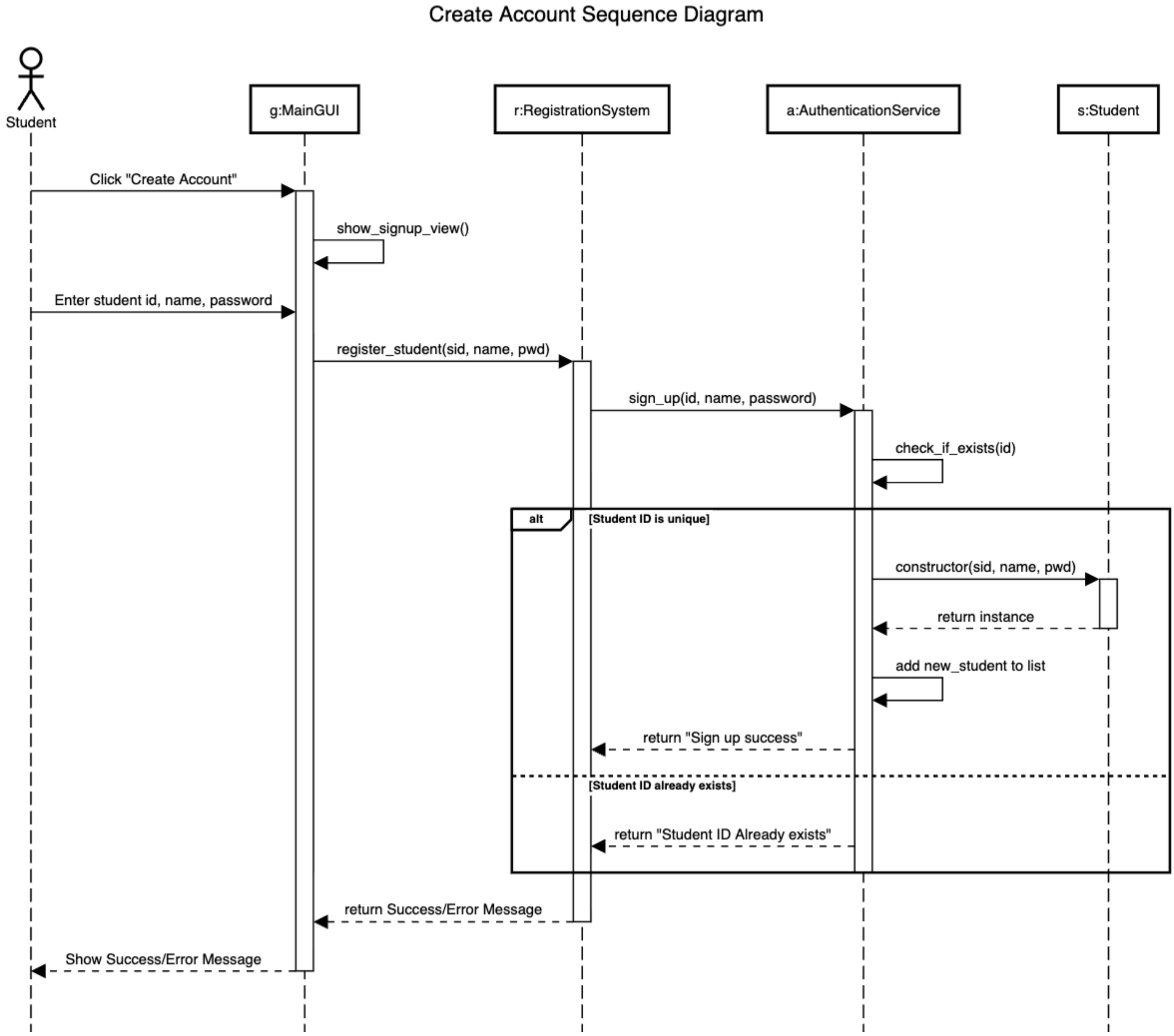
UC Reference Name/Number:	Drop a Course /UC6
Overview	A student can drop a course that they previously registered for.
Related use cases:	Extends View My Schedule
Actors	Student

UC Reference Name/Number:	View My Schedules /UC7
Overview	A student can view all courses they are currently enrolled in.
Related use cases:	Include Login
Actors	Student

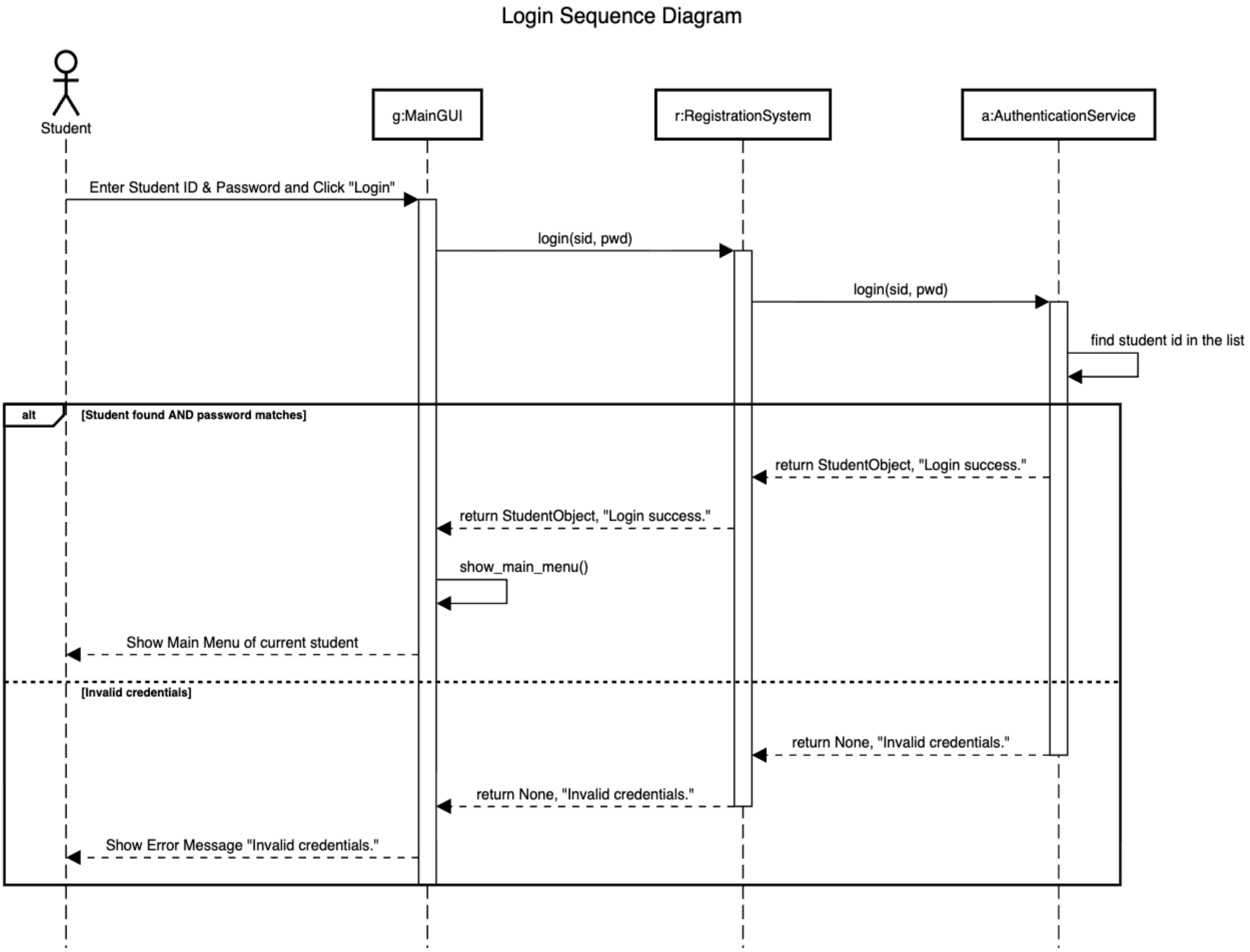
2. System Design

2.1 Sequence Diagrams

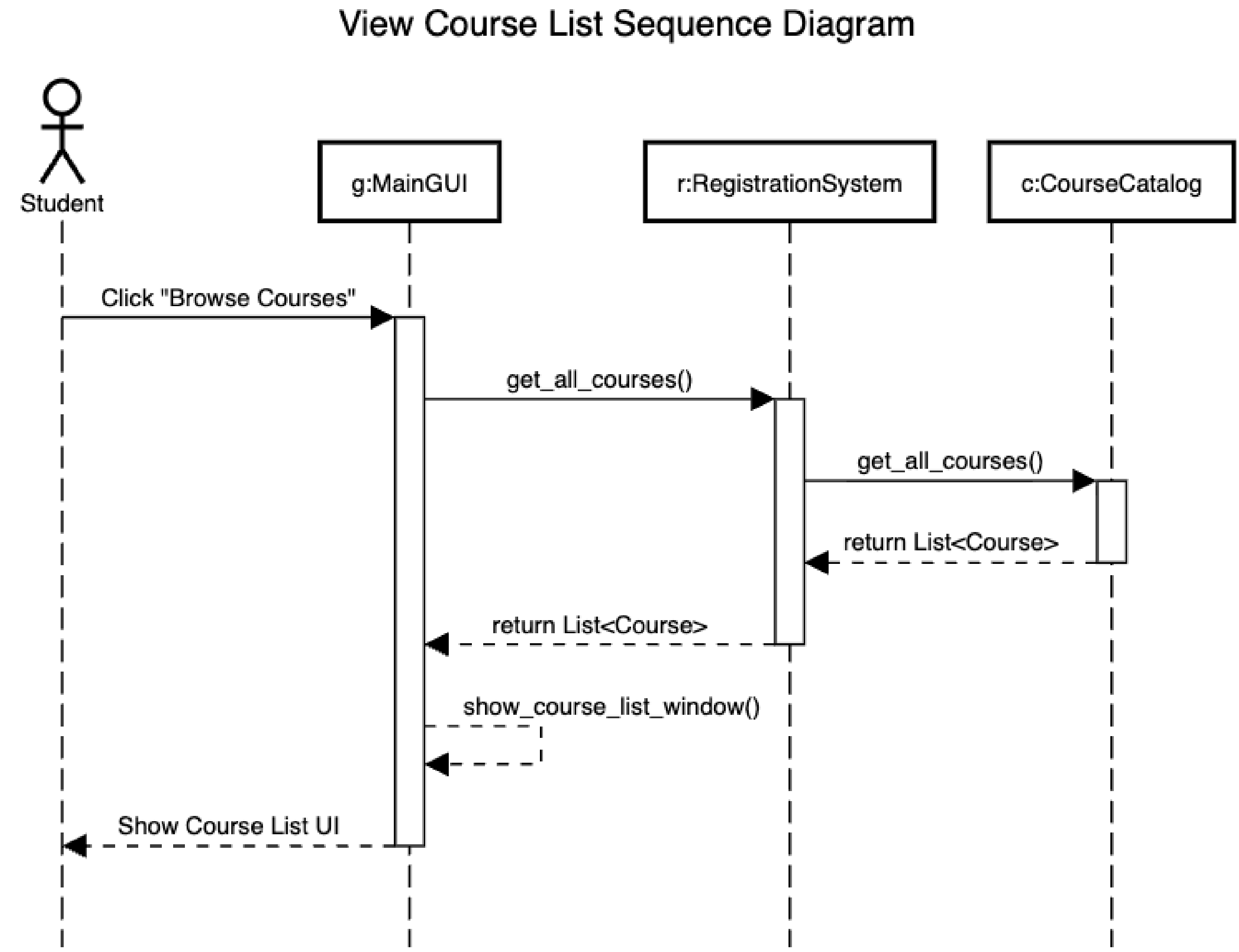
Create Account:


Create Account Sequence Diagram
Student
g:MainGUI
r:RegistrationSystem
a:AuthenticationService
s:Student
Click "Create Account"
show_signup_view()
Enter student id, name, password
register_student(sid, name, pwd)
sign_up(id, name, password)
check_if_exists(id)
alt
[Student ID is unique]
constructor(sid, name, pwd)
return instance
add new_student to list
return "Sign up success"
[Student ID already exists]
return "Student ID Already exists"
return Success/Error Message
Show Success/Error Message

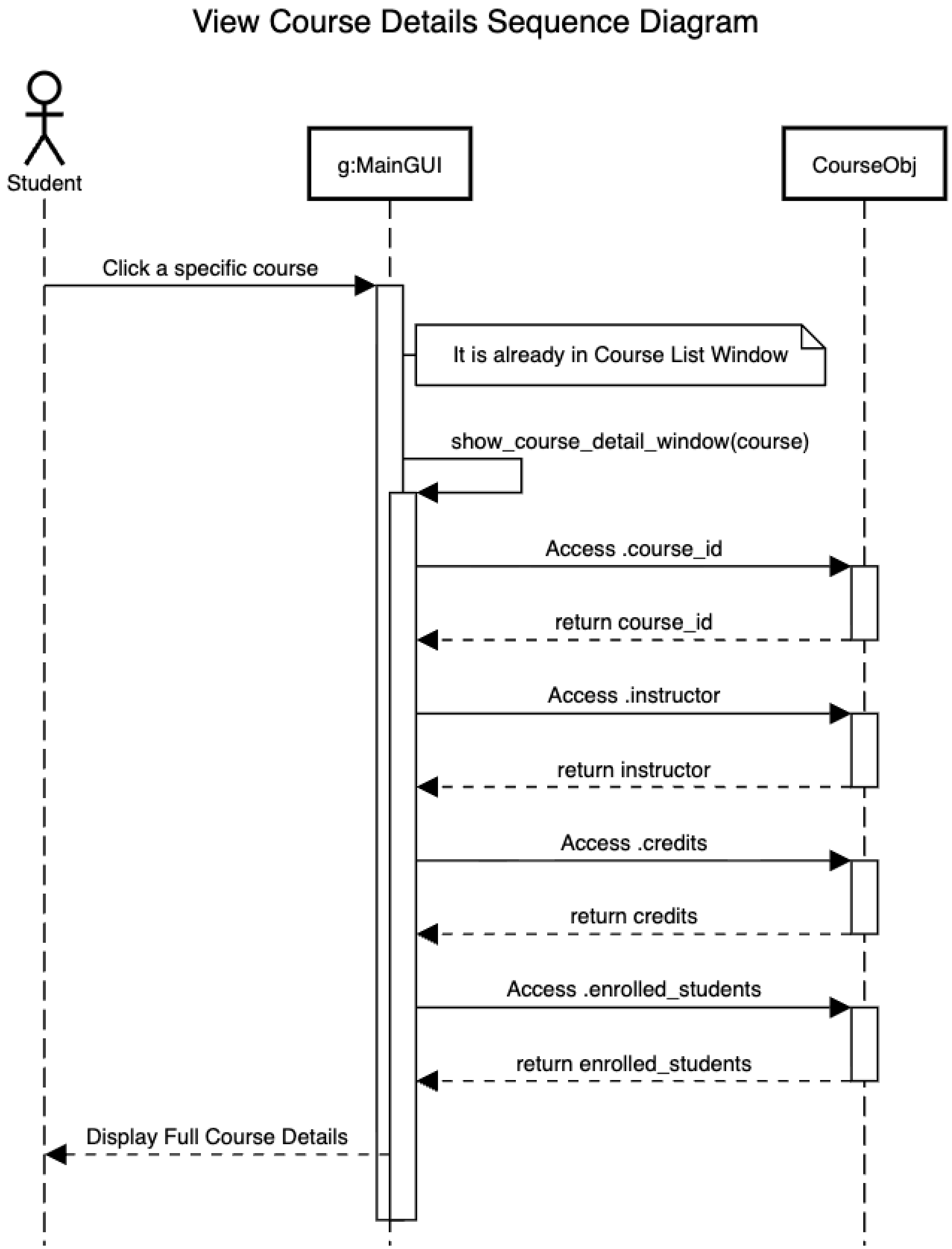
Login:


Login Sequence Diagram
Student
g:MainGUI
r:RegistrationSystem
a:AuthenticationService
Enter Student ID & Password and Click "Login"
login(sid, pwd)
login(sid, pwd)
find student id in the list
alt
[Student found AND password matches]
return StudentObject, "Login success."
return StudentObject, "Login success."
show_main_menu()
Show Main Menu of current student
[Invalid credentials]
return None, "Invalid credentials."
return None, "Invalid credentials."
Show Error Message "Invalid credentials."

View Course List:

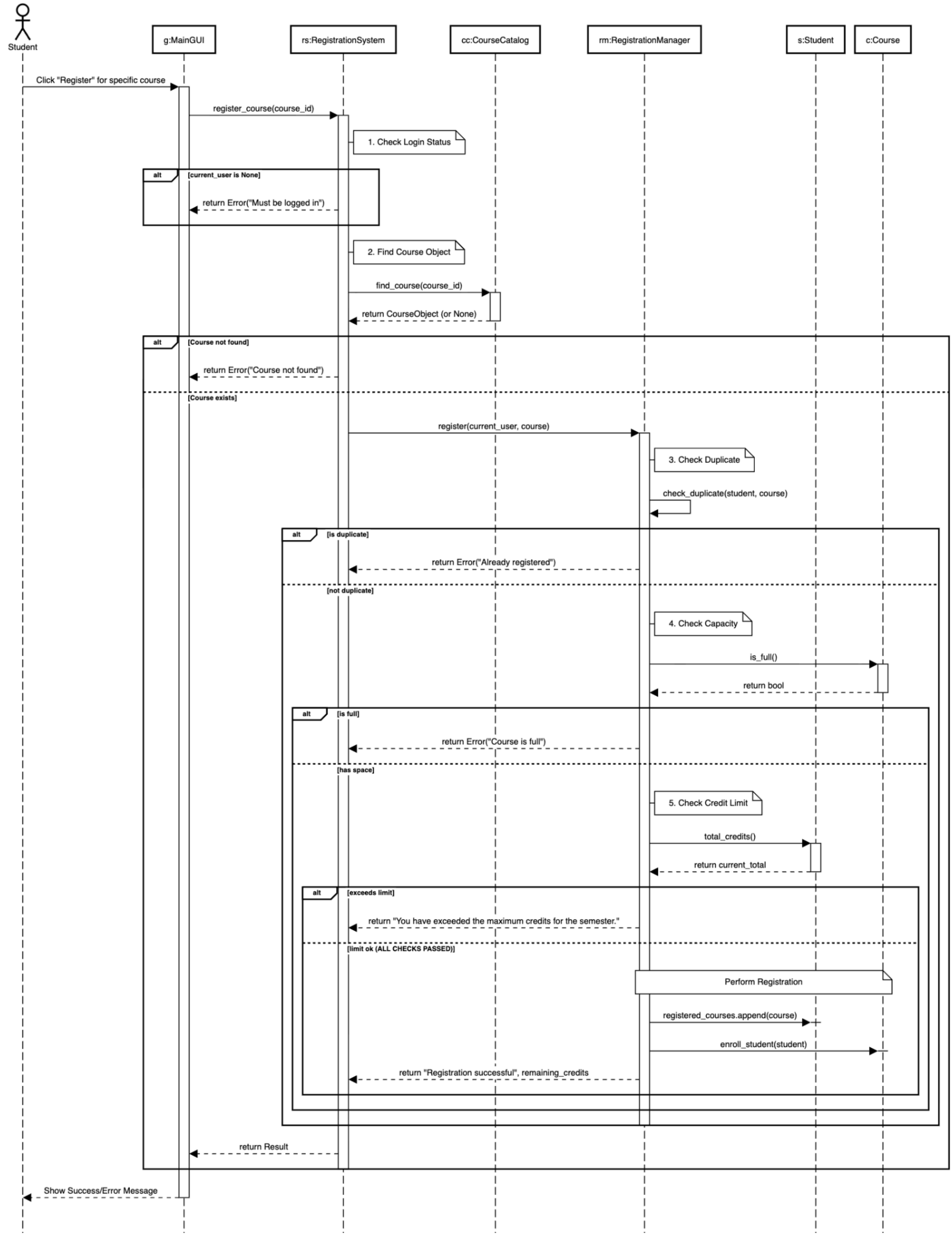


View Course Details:



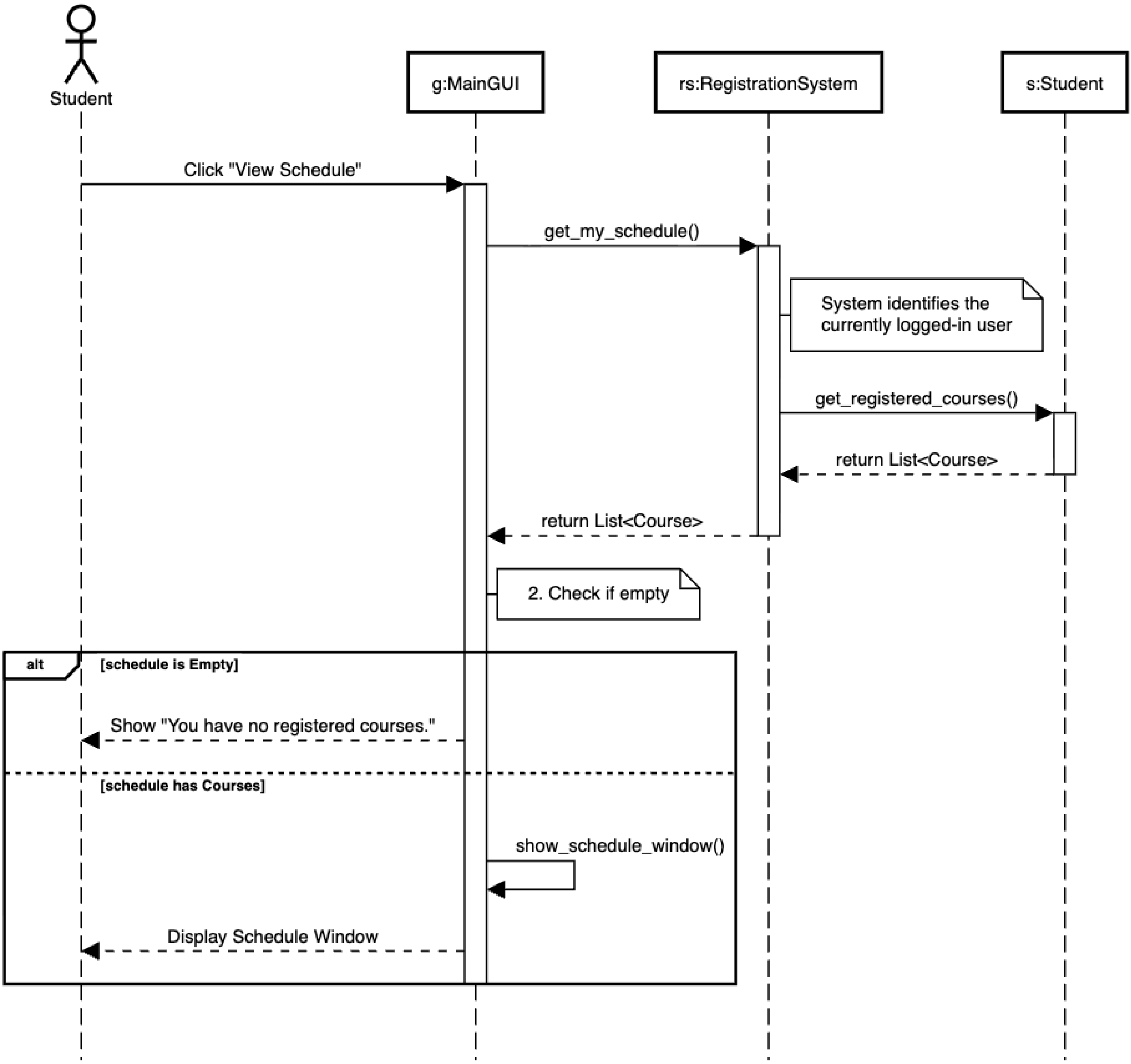
Register for a Course:

Register for a Course Sequence Diagram

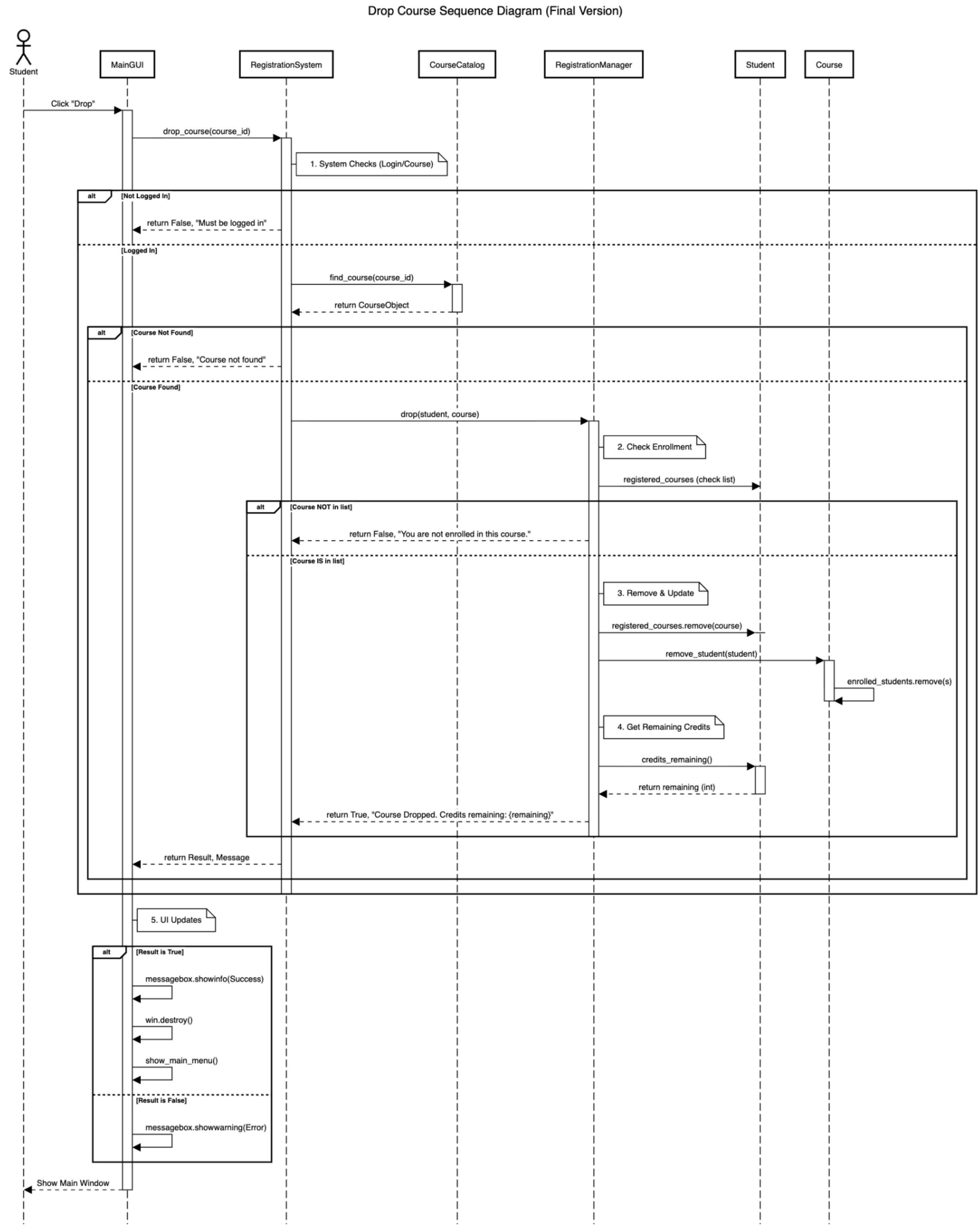

Student
g:MainGUI
rs:RegistrationSystem
cc:CourseCatalog
rm:RegistrationManager
s:Student
c:Course
Click "Register" for specific course
register_course(course_id)
1. Check Login Status
alt
[current_user is None]
return Error("Must be logged in")
2. Find Course Object
find_course(course_id)
return CourseObject (or None)
alt
[Course not found]
return Error("Course not found")
[Course exists]
register(current_user, course)
3. Check Duplicate
check_duplicate(student, course)
alt
[is duplicate]
return Error("Already registered")
[not duplicate]
4. Check Capacity
is_full()
return bool
alt
[is full]
return Error("Course is full")
[has space]
5. Check Credit Limit
total_credits()
return current_total
alt
[exceeds limit]
return "You have exceeded the maximum credits for the semester."
[limit ok (ALL CHECKS PASSED)]
Perform Registration
registered_courses.append(course)
enroll_student(student)
return "Registration successful", remaining_credits
return Result
Show Success/Error Message

View My Schedule:

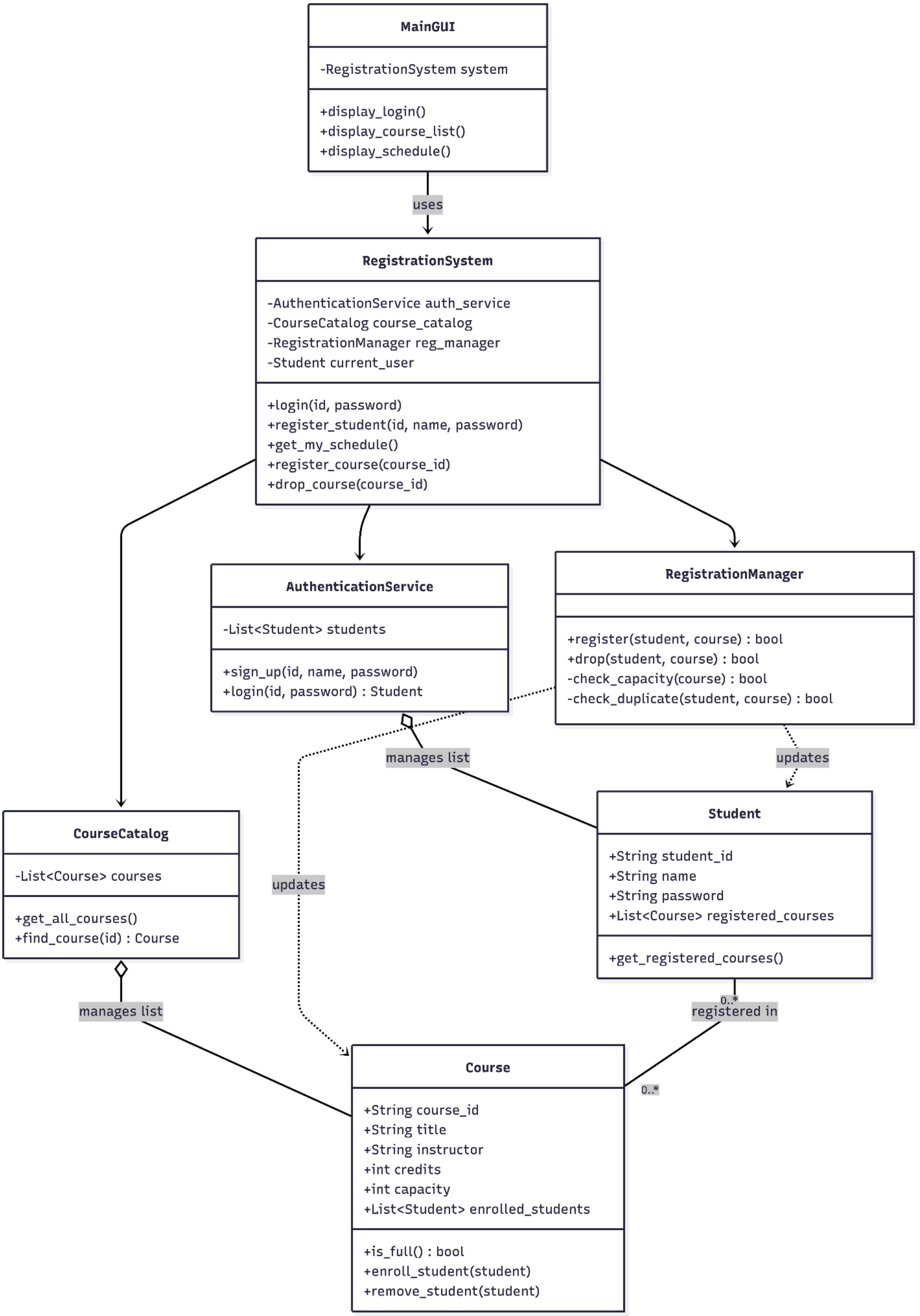
View My Schedule Sequence Diagram



Drop Course:


Drop Course Sequence Diagram (Final Version)
Student
MainGUI
RegistrationSystem
CourseCatalog
RegistrationManager
Student
Course
Click "Drop"
drop_course(course_id)
1. System Checks (Login/Course)
alt
[Not Logged In]
return False, "Must be logged in"
[Logged In]
find_course(course_id)
return CourseObject
alt
[Course Not Found]
return False, "Course not found"
[Course Found]
drop(student, course)
2. Check Enrollment
registered_courses (check list)
alt
[Course NOT in list]
return False, "You are not enrolled in this course."
[Course IS in list]
3. Remove & Update
registered_courses.remove(course)
remove_student(student)
enrolled_students.remove(s)
4. Get Remaining Credits
credits_remaining()
return remaining (int)
return True, "Course Dropped. Credits remaining: {remaining}"
return Result, Message
5. UI Updates
alt
[Result is True]
messagebox.showinfo(Success)
win.destroy()
show_main_menu()
[Result is False]
messagebox.showwarning(Error)
Show Main Window

2.2 Class Diagram(s)



3. Conclusion

Work Summary

The Course Registration System is a Python-based GUI application developed by **Ivan Wang (yw6505), Zhengxuan Tian (zt2254), and Hanlin Yan (hy2484)**. The project implements a simplified yet functional course enrollment platform that allows students to browse available courses, view course details, register or drop courses, and review their academic schedule. The primary goal of the system is to simulate a realistic course registration experience while maintaining accessibility and minimal technical overhead.

Our team adopted an **object-oriented design (OOP)** and structured the system around clean separation of concerns. Core data entities—including **Student**, **Course**, and **Registration**—are modeled as independent classes to encapsulate essential attributes and behaviors. System-level operations are organized into controller components:

- **AuthenticationService** manages account creation and student login.
- **CourseCatalog** stores and provides access to the system's available courses.
- **RegistrationManager** handles registration logic, including enforcing enrollment constraints.
- **RegistrationSystem** coordinates the interaction among all managers and serves as the system's central controller.

The graphical user interface (GUI) is implemented using **CustomTkinter**, providing a user-friendly front-end that allows students to navigate features intuitively. The GUI includes separate views for course browsing, course detail display, registration actions, and schedule review. All application data is maintained through **in-memory Python data structures**, intentionally avoiding a database backend for simplicity and clarity.

Throughout development, the team collaborated on system architecture, interface design, feature implementation, debugging, and testing. Responsibilities were shared across:

- **Backend logic development (course catalogs, registration rules, student accounts)**
- **GUI layout and user interaction design**
- **Integration of controllers and entity classes**
- **Documentation of system behavior and design decisions**

This project demonstrates the team's ability to apply software engineering principles—including modularity, abstraction, and MVC-like separation—to build a complete, functioning application. The resulting system provides a strong foundation for future enhancements such as persistent storage, admin features, or expanded scheduling capabilities.

Difficulties Encountered

During the development of the Course Registration System, several challenges arose from designing and integrating multiple object-oriented components. The system uses a layered structure consisting of entity classes (Student, Course, Registration), controller/manager classes (AuthenticationService, CourseCatalog, RegistrationManager, RegistrationSystem), and a **GUI interface** (MainGUI). Ensuring that these components communicate coherently introduced several technical and design difficulties.

1. Defining Clear Responsibilities Between Entities and Managers

One of the primary challenges was determining **which logic should reside in entity classes** and which should be handled by **manager/controller classes**.

For example:

- The **Course** class stores capacity and enrollment count, but **should NOT** decide whether a student can register.
- The **Student** class stores enrolled courses, but **should NOT** perform registration logic itself.

- All registration rules were centralized in **RegistrationManager**, creating a clear separation of concerns.

This required careful design to avoid:

- Tight coupling between Student ↔ Course
- Duplicated logic across managers
- Unclear responsibility boundaries

2. Managing Relationships and Data Synchronization

Because multiple classes depend on each other, ensuring **consistent updates** was challenging.

Key relationships:

- **Student ↔ RegistrationManager**
  RegistrationManager updates both the registration list and the student's enrolled courses.
- **Course ↔ RegistrationManager**
  RegistrationManager updates enrollment count and checks capacity constraints.
- **CourseCatalog → Course**
  CourseCatalog stores, retrieves, and lists all Course objects, acting as the gateway for GUI display.

Maintaining synchronization across these objects (Student schedule, Course capacity, Registration records) required careful method design and validation.

3. Coordinating System-Wide Operations Through RegistrationSystem

The **RegistrationSystem** acts as the central coordinator, connecting:

- AuthenticationService (student login & creation)
- CourseCatalog (course storage and search)
- RegistrationManager (enrollment logic)
- MainGUI (front-end interface)

This introduces complexity:

- RegistrationSystem must pass object references correctly (e.g., active student, course list)
- It must prevent circular dependencies between managers
- It must allow GUI to call backend methods without exposing internal data structures too directly

Designing this "hub" component required thoughtful architectural planning.

4. Integrating the GUI with Backend OOP Logic

The GUI (MainGUI) needed to interact cleanly with controller classes, which led to challenges such as:

- Passing the logged-in Student object from AuthenticationService → GUI
- Updating GUI elements after backend changes (e.g., after registering a course)
- Ensuring GUI callbacks do not contain business logic (should call manager classes instead)
- Avoiding situations where GUI directly manipulates Course or Student objects incorrectly

Achieving a clean MVC-like structure required refactoring and careful event handling.

5. Handling Error Conditions and System Constraints

Because different layers of the system interact, validating actions became more complex:

- RegistrationManager must check whether a course is full
- Student must not register for a course twice
- A student cannot exceed the credit limit

- GUI must display meaningful error or success messages
- All these conditions must propagate correctly across classes

Coordinating these checks across entities + managers + GUI was non-trivial.

- ### Recommendations

Implement Persistent Storage:

Add a database or file-based storage system so that student accounts, course lists, and registration records are saved between sessions instead of relying on in-memory data.

Improve GUI Usability and Navigation:

Enhance the interface with clearer layouts, search bars, filtering options, and improved error messages to make course browsing and registration more intuitive.

Add Administrative Tools:

Introduce an admin interface that allows instructors or staff to create courses, update course capacities, and manage student enrollments more efficiently.

4. Optional Appendix

1. Website to Check NYU Course Offerings

NYU provides an official public course search page where anyone can view courses by semester, department, instructor, and meeting time.

NYU Public Course Search (Class Search):
https://bulletins.nyu.edu/class-search/

2. OOP Course Registration System Instruction Resource

This is a clear, example-based reference document showing Object-Oriented Analysis & Design for a student/course registration system, including use-case diagrams, class diagrams, and design explanation.

Object-Oriented Analysis and Design for Student Registration System (Example Document):
https://www.scribd.com/document/93784531/Object-Oriented-Analysis-and-Design-for-Student-Registration-System