

Design Document

Group 3

CS401-02

<https://github.com/chenAndrew4/CS401ClassEnrollmentSystem>

Revision History

Date	Revision	Description	Author
10/16	1.0	Initial Version	Anthony Kungo
10/18	1.0	Adding initial class designs	Chen Li
10/20	1.0	Adding initial data table designs	Chen Li
10/24	1.0	Adding initial test case designs	Chen Li
10/24	1.0	Worked on test case	Andrew Tang
10/26	1.0	Adding initial GUI designs	Chen Li
10/26	1.0	Worked on Detailed Test Plan	Anthony Kungo
10/26	1.0	Worked on GUI designs	Andrew Tang
10/26	1.0	Worked on Use Cases	Anthony Kungo
10/28	1.0	Adding Class UML diagram	Chen Li
10/30	1.0	Formatted the Document	Andrew Tang
10/30	1.0	Added Use Case Diagrams & Sequence Diagrams	Connor McMillan
10/30	1.0	Formatted the Document	Anthony Kungo

Table of Content

- [1. System Overview](#)
- [2. Design Goals](#)
- [3. Detailed Class Designs](#)
 - [3.1 User Class](#)
 - [3.2 Student Class \(Extends User\)](#)
 - [3.3 Faculty Class \(Extends User\)](#)
 - [3.4 Administrator Class \(Extends User\)](#)
 - [3.5 Course Class](#)
 - [3.6 Schedule Class](#)
 - [3.7 RegistrationManager Class](#)
 - [3.8 FileDataManager Class](#)
 - [3.9 Enums](#)
 - [3.9.1 Location Enum](#)
 - [3.9.2 Campus Enum](#)
 - [3.9.3 Room Enum](#)
- [4. File Storage and Data Management](#)
 - [4.1 Data Persistence](#)
 - [4.1.1 User Class](#)
 - [Table Fields in User.csv](#)
 - [4.1.2 Student Class \(Extends User\)](#)
 - [Table Fields in Student.csv](#)
 - [4.1.3 Faculty Class](#)
 - [Table Fields in Faculty.csv](#)
 - [4.1.4 Administrator Class](#)
 - [Table Fields in Administrator.csv](#)
 - [4.1.5 Course Class](#)
 - [Table Fields in Course.csv](#)
 - [4.1.6 Schedule Class](#)
 - [Table Fields in Schedule.csv](#)
 - [4.1.7 Waitlist Class](#)
 - [Table Fields in Waitlist.csv](#)
 - [Explanation of Methods](#)
 - [4.2 Data Serialization](#)
- [5. System Processes](#)
 - [5.1 Registration Process](#)
 - [5.2 Waitlist Management Process](#)
- [6. Detailed GUI Design](#)
 - [6.1 Login Screen](#)

[Layout: BorderLayout](#)

[Additional Elements](#)

[6.2 Role-Based Dashboards](#)

[6.2.1 Student Dashboard](#)

[Layout: BorderLayout](#)

[Additional Elements](#)

[6.2.2 Faculty Dashboard](#)

[Layout: BorderLayout](#)

[Additional Elements](#)

[6.2.3 Admin Dashboard](#)

[Layout: BorderLayout](#)

[Additional Elements](#)

[6.3 Course Catalog Screen](#)

[Layout: BorderLayout](#)

[Additional Elements](#)

[6.4 Schedule Viewer](#)

[Layout: BorderLayout](#)

[Additional Elements](#)

[6.5 Waitlist Viewer \(Admin and Faculty\)](#)

[Layout: BorderLayout](#)

[Additional Elements](#)

[7. Detailed Test Plan](#)

[7.1.1 Authentication and Access Control](#)

[7.1.2 Student Course Registration](#)

[7.1.3 Drop Course](#)

[7.1.4 Schedule Conflict Detection](#)

[7.1.5 Faculty Course Management](#)

[7.1.6 Admin Course Management](#)

[7.1.7 File Data Persistence](#)

[7.1.8 Waitlist Management](#)

[7.1.9 GUI Validation](#)

[7.2.1 Login Functionality](#)

[7.2.1 Course Registration](#)

[7.2.2 Drop Course](#)

[7.2.3 Schedule Conflicts](#)

[7.2.4 Syllabus Management](#)

[7.2.5 Data Persistence](#)

[7.2.6 Waitlist Management](#)

[7.2.7 GUI Responsiveness](#)

[7.3.1 User Class](#)

[7.3.2 Student Class](#)

[7.3.3 Faculty Class](#)

[7.3.4 Administrator Class](#)

[7.3.5 Course Class](#)

[7.3.6 Schedule Class](#)

[7.3.7 Waitlist Class](#)

[8. Use Cases](#)

[8.1 System Login](#)

[8.2 Student Course Registration](#)

[8.3 Faculty Grade Submission](#)

[8.4 Administrator Course Management](#)

[8.5 Student Course Drop](#)

[8.6 Administrator Creates Schedule](#)

[8.7 User Management](#)

[8.8 View Class Roster](#)

[8.10 Update Course Syllabus](#)

[8.11 View Course Catalog](#)

[8.12 Generate Enrollment Report](#)

[8.13 Search Courses](#)

[8.14 View Waitlist](#)

[8.15 Generate Waitlist Report](#)

[8.16 Modify Course Waitlist](#)

[8.17 View Personal Waitlist Position](#)

[8.2 Use Case Diagram](#)

[9. Sequence Diagrams](#)

[9.1 System Login](#)

[9.2 Student Course Registration](#)

[9.3 Faculty Grade Submission](#)

[9.4 Administrator Course Management](#)

[9.5 Student Course Drop](#)

[9.6 Administrator Creates Schedule](#)

[9.7 User Management](#)

[9.8 View Class Roster](#)

[9.9 Update Course Syllabus](#)

[9.10 View Course Catalog](#)

[9.11 Generate Enrollment Report](#)

[9.12 Search Courses](#)

[9.13 View Waitlist](#)

[9.14 Generate Waitlist Report](#)

[9.15 Modify Course Waitlist](#)

[9.16 View Personal Waitlist Position](#)

[10. Data Integrity and Security](#)

[11. Concurrency and Synchronization](#)

[12. Error Handling](#)

[13. Future Enhancements](#)

[14. Class UML Diagrams](#)

1. System Overview

The College Class Enrollment System is a Java application that enables students to enroll in courses, faculty to manage class rosters, and administrators to control course assignments and generate reports. The system is file-based, using the file system for data persistence, and it's designed as a standalone application accessible via a Java GUI (Java Swing).

2. Design Goals

- **Modularity:** Each functional unit (e.g., user management, course registration) is encapsulated in separate classes.
- **Data Integrity:** Immediate data persistence to ensure accuracy and reliability of student and course data.
- **Role-based Access Control:** Restrict access based on user roles (Student, Faculty, Administrator).
- **Concurrency Management:** Ensuring thread-safe operations in multi-user environments.

3. Detailed Class Designs

3.1 User Class

- **Description:** The base class representing a generic user.

- **Attributes:**

- private String userID: Unique identifier for each user.
- private String firstName: The user's first name.
- private String lastName: The user's last name.
- private String password: Hashed password for secure authentication.
- private boolean isAuthenticated: Tracks if the user is logged in.

- **Methods:**

- public boolean authenticate(String password): Validates the password during login.
- public boolean login(): Sets isAuthenticated to true upon successful authentication.
- public void logout(): Sets isAuthenticated to false and ends the user session.

3.2 Student Class (Extends User)

- **Attributes:**

- private List<Course> enrolledCourses: List of courses the student is currently enrolled in.
- private List<Course> waitlistedCourses: List of courses where the student is on the waitlist.
- private AcademicRecord academicRecord: Tracks the student's grades and academic status.

- **Methods:**

- public boolean registerForCourse(Course course): Enrolls a student in a course if

prerequisites are met.

- `public void dropCourse(Course course)`: Removes the student from an enrolled course.
- `public List<Course> viewSchedule()`: Returns the student's current course schedule.
- `public Map<Course, Integer> getWaitlistPositions()`: Displays the waitlist position for each course where the student is waitlisted.

3.3 Faculty Class (Extends User)

- **Attributes:**

- `private List<Course> assignedCourses`: List of courses taught by the faculty member.

- **Methods:**

- `public List<Student> viewClassRoster(String courseID)`: Returns the roster for a specific course.
- `public boolean updateSyllabus(Course course, String syllabusText)`: Allows the faculty to upload or update a syllabus.

3.4 Administrator Class (Extends User)

- **Methods:**

- `public void assignCourses(Faculty faculty, String courseID)`: Assigns a course to a faculty member.
- `public Report generateReport(ReportType type)`: Generates various reports on

enrollment, schedules, etc.

- public Course createCourse(String courseID): Creates a new course.
- public boolean deleteCourse(String courseID): Deletes an existing course.

3.5 Course Class

- **Attributes:**

- private String courseID: Unique identifier for each course.
- private String name: The course name.
- private int enrollmentLimit: Maximum student enrollment.
- private List<Course> prerequisites: List of prerequisite courses.
- private ClassRoster classRoster: Roster for students enrolled in the course.
- private Waitlist waitlist: Waitlist for students when a course is full.

- **Methods:**

- public boolean addStudent(Student student): Adds a student to the course if space is available.
- public void removeStudent(Student student): Removes a student from the course roster.
- public boolean isFullyEnrolled(): Checks if the course has reached its enrollment limit.
- public boolean addToWaitlist(Student student): Adds a student to the waitlist if the course is full.

3.6 Schedule Class

- **Attributes:**
 - private String classID: Unique identifier for the schedule.
 - private Days[] days: Array of days the class meets.
 - private Time startTime: The start time of the class.
 - private Time endTime: The end time of the class.
- **Methods:**
 - public boolean doesThisScheduleConflict(Schedule otherSchedule): Checks for schedule conflicts.

3.7 RegistrationManager Class

- **Attributes:**
 - private Map<String, Course> courseRegistry: Registry of all courses.
 - private List<RegistrationRule> rules: List of registration rules.
- **Methods:**
 - public boolean processCourseRegistration(Student student, Course course):
Registers a student for a course.
 - public void processDropCourse(Student student, Course course): Drops a course for a student and updates waitlists.
 - public boolean checkPrerequisites(Student student, Course course): Ensures prerequisites are met.
 - public void handleWaitlist(Student student, Course course): Adds or removes students from the waitlist as needed.

3.8 FileDataManager Class

- **Methods:**
 - `public Object readData(String fileKey)`: Reads data from the corresponding file.
 - `public boolean writeData(String fileKey, Object data)`: Writes data to file.

3.9 Enums

3.9.1 Location Enum

- **Description:** Represents various campus locations.
- **Values:**
 - `MEIKLEJOHN_HALL("Meiklejohn Hall")`
 - `SCIENCE_BUILDING("Science Building")`
 - `ART_EDUCATION("Art and Education Building")`
 - `MUSIC_BUILDING("Music Building")`
 - `LIBRARY("University Library")`
 - `STUDENT_UNION("Student Union")`
 - `GYM("Gymnasium")`

3.9.2 Campus Enum

- **Description:** Represents different CSU East Bay campuses.
- **Values:**
 - `HAYWARD("Hayward Campus")`

- CONCORD("Concord Campus")
- ONLINE("Online Campus")

3.9.3 Room Enum

- **Description:** Represents rooms numbered from Room1 to Room50.
- **Values:**
 - ROOM1, ROOM2, ..., ROOM50

4. File Storage and Data Management

4.1 Data Persistence

Data will be stored in separate files for each entity (e.g., users.csv, courses.csv, schedules.csv, waitlists.csv). Each file will be formatted to ensure easy parsing and loading of data:

4.1.1 User Class

Table Fields in **User.csv**

Field	Description
userID	Unique identifier for user
firstName	User's first name
lastName	User's last name
password	Hashed password
role	User role (e.g., Student)
isAuthenticated	Indicates login status

4.1.2 Student Class (Extends User)

Table Fields in **Student.csv**

Field	Description
userID	Reference to User
enrolledCourses	Course IDs (semicolon-separated if multiple)
waitlistedCourses	Waitlisted course IDs (semicolon-separated)
academicRecord	Serialized academic data

4.1.3 Faculty Class

Table Fields in **Faculty.csv**

Field	Description
userID	Reference to User
assignedCourses	List of course IDs (semicolon-separated)

4.1.4 Administrator Class

Table Fields in **Administrator.csv**

Field	Description
userID	Unique identifier for each administrator (inherited from User)
firstName	First name of the administrator
lastName	Last name of the administrator
password	Hashed password for secure access
role	Set to "Admin" for all administrator records
permissions	Comma-separated list of specific permissions

4.1.5 Course Class

Table Fields in **Course.csv**

Field	Description
courseID	Unique identifier for each course
name	Name of the course
description	Course description
enrollmentLimit	Maximum enrollment
prerequisites	Prerequisite course IDs (semicolon-separated)

4.1.6 Schedule Class

Table Fields in **Schedule.csv**

Field	Description
classID	Course reference
days	Days of the week (semicolon-separated)
startTime	Start time of class
endTime	End time of class

Here's a detailed design for a **Waitlist** class in the College Class Enrollment System, including the file structure, fields, save/load functions, and corresponding test cases. The **Waitlist** class is responsible for managing students who are waiting to enroll in a course that is currently full.

4.1.7 Waitlist Class

Table Fields in **Waitlist.csv**

Field	Description
courseID	Unique identifier for the course associated with this waitlist
studentIDs	List of student IDs on the waitlist for the course (semicolon-separated)
maxWaitlist	Maximum number of students that can be on the waitlist

Explanation of Methods

1. **saveWaitlist**: Writes the waitlist information for each course to a row in **Waitlist.csv**.

Each row includes **courseID**, **studentIDs** (semicolon-separated), and **maxWaitlist**.

2. **loadWaitlist**: Reads from `Waitlist.csv` to load the waitlist for a specific course.
3. **addStudent**: Adds a student ID to the waitlist if the `maxWaitlist` limit has not been reached.
4. **removeStudent**: Removes a student ID from the waitlist if they drop their spot or are promoted to enrollment.
5. **getWaitlistPosition**: Returns the position of a specific student in the waitlist.

This setup ensures that the `Waitlist` class efficiently handles students waiting to enroll in full courses, with clear and consistent storage and retrieval operations, and robust test cases to verify functionality.

4.2 Data Serialization

- **Process**: Each entity will be serialized to a JSON-like format or CSV for easy export and import.
- **Storage**: Each entity class (e.g., `User`, `Course`) will implement methods to convert data to and from storage format.

5. System Processes

5.1 Registration Process

1. **Course Selection:** Student selects a course from the catalog.
2. **Prerequisite Check:** RegistrationManager validates prerequisites.
3. **Schedule Conflict Check:** Confirms there are no conflicts with existing schedules.
4. **Enrollment or Waitlisting:** Enrolls student if space is available, or adds them to the waitlist if full.

5.2 Waitlist Management Process

1. **Adding to Waitlist:** If a course is full, RegistrationManager places the student in the course waitlist.
2. **Notification of Available Slot:** WaitList notifies the next student in the queue when a spot opens.

6. Detailed GUI Design

6.1 Login Screen

Layout: BorderLayout

- **North: JLabel** - Displays the system logo or title.
- **Center: JPanel with GridLayout** - Contains form fields for login.
 - **JLabel:** "Username"
 - **TextField:** Username input
 - **JLabel:** "Password"
 - **PasswordField:** Password input

- **JLabel:** "Role"
- **JComboBox:** Role selection (Student, Faculty, Admin)
- **South: JPanel with FlowLayout** - Contains action buttons.
 - **JButton:** "Login"
 - **JButton:** "Reset"

Additional Elements

- **JOptionPane** for error messages (e.g., incorrect credentials or role not selected).

6.2 Role-Based Dashboards

6.2.1 Student Dashboard

Layout: BorderLayout

- **North: JToolBar** - Quick access buttons.
 - **JButton:** "View Course Catalog"
 - **JButton:** "View Schedule"
 - **JButton:** "Logout"
- **West: JTabbedPane** - Tabs for "Enrolled Courses" and "Waitlisted Courses".
 - **JList** (within **JScrollPane**): Displays a list of enrolled courses or waitlisted courses.
- **Center: JPanel with BorderLayout** - Displays course details.
 - **Center (Nested JPanel with GridLayout):**
 - **JLabel:** "Course Name"

- **JTextArea**: Displays selected course information.
 - **JButton**: "Register"
 - **JButton**: "Drop Course"
- **South: JProgressBar** - Indicates processing of registration or drop.

Additional Elements

- **JTable** (within **JScrollPane**) for schedule view with day/time, course name, and location.
- **JOptionPane** for registration confirmation and waitlist notifications.

6.2.2 Faculty Dashboard

Layout: BorderLayout

- **North: JToolBar** - Quick access buttons.
 - **JButton**: "View Class Roster"
 - **JButton**: "Update Syllabus"
 - **JButton**: "Logout"
- **West: JTabbedPane** - Tabs for "Assigned Courses" and "Student Roster".
 - **JList** (within **JScrollPane**): Displays a list of assigned courses.
- **Center: JPanel** with **BorderLayout** - Displays roster or syllabus.
 - **Center (Nested JPanel with GridLayout)**:
 - **JTable** (within **JScrollPane**): Shows student roster for the selected course.
 - **JButton**: "Download Roster"

- **JFileChooser**: For uploading syllabus files.

Additional Elements

- **JOptionPane** for syllabus upload success or error messages.
- **JSpinner** for setting specific enrollment limits in specific courses if applicable.

6.2.3 Admin Dashboard

Layout: BorderLayout

- **North: JToolBar** - Admin actions.
 - **JButton**: "Create Course"
 - **JButton**: "Assign Instructor"
 - **JButton**: "Generate Report"
 - **JButton**: "Logout"
- **Center: JTabbedPane** - Tabs for "Course Management" and "User Management".
 - **Course Management Tab**:
 - **JTable** (within **JScrollPane**): List of courses with edit options.
 - **JPanel** with **GridLayout**: Course creation/edit form.
 - **TextField**: Course ID, Course Name
 - **ComboBox**: Instructor assignment
 - **JSpinner**: Enrollment limit
 - **JButton**: "Save Course"
 - **JButton**: "Delete Course"

- **User Management Tab:**
 - **JTable** (within **JScrollPane**): List of users (students, faculty, admin).

Additional Elements

- **JOptionPane** for course creation and deletion confirmations.
- **JFileChooser** for importing/exporting data.

6.3 Course Catalog Screen

Layout: BorderLayout

- **North: JTextField** for the search bar and **JComboBox** for department filters.
- **Center: JList** (within **JScrollPane**) - Displays filtered list of available courses.
- **South: JPanel** with **FlowLayout** - Action buttons.
 - **JButton**: "View Details"
 - **JButton**: "Register"
 - **JButton**: "Back"

Additional Elements

- **JOptionPane** for registration success and error messages.
- **JTextArea** in **BorderLayout (East)** for course description details.

6.4 Schedule Viewer

Layout: BorderLayout

- **North: JPanel with GridLayout** - Date filters.
 - **JLabel:** "Start Date"
 - **JSpinner:** Start date picker
 - **JLabel:** "End Date"
 - **JSpinner:** End date picker
 - **JButton:** "Filter by Date"
- **Center: JTable (within JScrollPane)** - Displays schedule with day/time, course name, and location.

Additional Elements

- **JOptionPane** for conflict notifications.
- **JProgressBar** to show loading or filtering progress.

6.5 Waitlist Viewer (Admin and Faculty)

Layout: BorderLayout

- **North: JPanel** - Heading and filter options.
 - **JLabel:** "Waitlist for Course"
 - **JComboBox:** Course selection for viewing the waitlist.
- **Center: JTable (within JScrollPane)** - Displays waitlisted students.
- **South: JPanel with FlowLayout** - Action buttons.
 - **JButton:** "Notify Student"
 - **JButton:** "Promote from Waitlist"
 - **JButton:** "Back"

Additional Elements

- **JOptionPane** for notification and promotion confirmations.
- **JProgressBar** for waitlist processing.

7. Detailed Test Plan

7.1.1 Authentication and Access Control

- **Test Case 1:** Verify successful login with correct username/password.
 - **Expected Result:** User is redirected to the dashboard based on role.
- **Test Case 2:** Attempt login with incorrect password.
 - **Expected Result:** Display error message for invalid credentials.
- **Test Case 3:** Attempt login without selecting a role.
 - **Expected Result:** Display warning to select a role.
- **Test Case 4:** Test role-based access (Student can only access student features, etc.)
 - **Expected Result:** Each user role has access only to their specific dashboard.

7.1.2 Student Course Registration

- **Test Case 1:** Register for a course with no prerequisites.
 - **Expected Result:** Student is successfully registered.
- **Test Case 2:** Register for a course with prerequisites met.
 - **Expected Result:** Registration succeeds, and the course is added to the student's schedule.

- **Test Case 3:** Register for a course with unmet prerequisites.
 - **Expected Result:** Display an error message about unmet prerequisites.
- **Test Case 4:** Register for a course that is fully enrolled.
 - **Expected Result:** Student is added to the waitlist, and a message is displayed.

7.1.3 Drop Course

- **Test Case 1:** Drop an enrolled course.
 - **Expected Result:** Course is removed from student's enrolled list and waitlist is updated.
- **Test Case 2:** Attempt to drop a course that the student is not enrolled in.
 - **Expected Result:** Display an error message stating that the student is not enrolled in the course.

7.1.4 Schedule Conflict Detection

- **Test Case 1:** Register for overlapping courses.
 - **Expected Result:** Display conflict message and deny registration.
- **Test Case 2:** Register for non-overlapping courses.
 - **Expected Result:** Registration succeeds without conflict.

7.1.5 Faculty Course Management

- **Test Case 1:** View class roster for an assigned course.
 - **Expected Result:** Roster displays with all enrolled students.
- **Test Case 2:** Update syllabus for an assigned course.

- **Expected Result:** Syllabus is successfully updated.
- **Test Case 3:** Attempt to view a course roster for a course not assigned to the faculty member.
 - **Expected Result:** Display error indicating unauthorized access.

7.1.6 Admin Course Management

- **Test Case 1:** Create a new course.
 - **Expected Result:** Course is added to the course list and available in the catalog.
- **Test Case 2:** Delete an existing course.
 - **Expected Result:** Course is removed from the course list and is no longer available for registration.
- **Test Case 3:** Assign a faculty member to a course.
 - **Expected Result:** Faculty is successfully assigned, and the course appears on their dashboard.

7.1.7 File Data Persistence

- **Test Case 1:** Register for a course and verify data is saved.
 - **Expected Result:** Registered course data is saved to the file system.
- **Test Case 2:** Drop a course and verify data is saved.
 - **Expected Result:** Updated course data reflects in the file.
- **Test Case 3:** Restart application and verify data consistency.
 - **Expected Result:** All data is loaded correctly from files.

7.1.8 Waitlist Management

- **Test Case 1:** Add a student to a waitlisted course.
 - **Expected Result:** Student appears in the waitlist for the course.
- **Test Case 2:** Remove a student from a waitlist.
 - **Expected Result:** Student is removed, and the next student in the queue is promoted.
- **Test Case 3:** Promote a waitlisted student to enroll when a slot opens.
 - **Expected Result:** Student is moved from waitlist to enrolled.

7.1.9 GUI Validation

7.2.1 Login Functionality

- **Test:** Valid credentials, invalid credentials, empty fields, role selection.
 - **Expected Results:** Appropriate dashboard loads, error dialogs display as needed.

7.2.1 Course Registration

- **Test:** Register with prerequisites met, unmet, and course full.
 - **Expected Results:** Course adds to schedule, prerequisite or full course messages display.

7.2.2 Drop Course

- **Test:** Drop course, check waitlist updates.
 - **Expected Results:** Course removed from schedule, waitlist adjusts if applicable.

7.2.3 Schedule Conflicts

- **Test:** Register for overlapping/non-overlapping courses.
 - **Expected Results:** Conflict message or success registration.

7.2.4 Syllabus Management

- **Test:** Upload syllabus file for course, download roster.
 - **Expected Results:** Syllabus uploads and replaces correctly, roster downloads as expected.

7.2.5 Data Persistence

- **Test:** Register/drop course, verify file saves, reload data.
 - **Expected Results:** All data persists and loads accurately.

7.2.6 Waitlist Management

- **Test:** Add/remove students from waitlist, promote students.
 - **Expected Results:** Waitlist updates correctly, promotions succeed.

7.2.7 GUI Responsiveness

- **Test:** All buttons, dropdowns, and input fields.
 - **Expected Results:** All components respond as expected, providing appropriate feedback.

7.3.1 User Class

- **Test Case 1:** Verify that `User.csv` includes a new row with the correct information after saving.
 - **Expected Results:** The correct information is shown after saving.
- **Test Case 2:** Confirm that `loadUser` returns the correct `User` object for an existing `userID`.
 - **Expected Results:** The correct `User` is returned.
- **Test Case 3:** Ensure `isAuthenticated` reflects login/logout status accurately.
 - **Expected Results:** The correct information is shown after authentication.

7.3.2 Student Class

- **Test Case 1:** Ensure a new entry in `Student.csv` with the correct course IDs.
 - **Expected Results:** The correct information is shown after saving.
- **Test Case 2:** Validate that `loadStudent` correctly restores a `Student` object.
 - **Expected Results:** The correct information is loaded.
- **Test Case 3:** Confirm `enrolledCourses` and `waitlistedCourses` lists match data in `Student.csv`.
 - **Expected Results:** The correct information matches.

7.3.3 Faculty Class

- **Test Case 1:** Confirm data in `Faculty.csv` after saving a faculty member.
 - **Expected Results:** The correct information is shown after saving.

- **Test Case 2:** Verify that `loadFaculty` retrieves the correct courses for a faculty member.
 - **Expected Results:** The correct information is loaded.
- **Test Case 3:** Check `assignedCourses` reflects correctly after updating in `Faculty.csv`.
 - **Expected Results:** The correct information matches.

7.3.4 Administrator Class

- **Test Case 1:** Save a new administrator entry to `Administrator.csv`.
 - **Expected Results:** A new row is added in `Administrator.csv` with the correct `userID`, `firstName`, `lastName`, `password`, `role`, and `permissions`.
- **Test Case 2:** Load an existing administrator from `Administrator.csv`.
 - **Expected Results:** The correct `Administrator` object is returned based on the provided `userID`.
- **Test Case 3:** Update the list of permissions for an administrator.
 - **Expected Results:** The `permissions` list in the `Administrator` object is updated correctly, and when saved, `Administrator.csv` reflects the updated permissions.
- **Test Case 4:** Remove an administrator's entry from `Administrator.csv`.
 - **Expected Results:** The `Administrator.csv` file no longer contains the deleted administrator's `userID`.
- **Test Case 5:** Convert an `Administrator` object to JSON format.
 - **Expected Results:** The JSON object correctly represents the administrator's details, including permissions.

7.3.5 Course Class

- **Test Case 1:** Verify `Course.csv` correctly saves course attributes.
 - **Expected Results:** The correct information is shown after saving.
- **Test Case 2:** Ensure `loadCourse` retrieves the course data accurately.
 - **Expected Results:** The correct data is loaded.
- **Test Case 3:** Confirm `enrollmentLimit` matches value stored in `Course.csv`.
 - **Expected Results:** The correct information is shown after matching.

7.3.6 Schedule Class

- **Test Case 1:** Check that `Schedule.csv` saves each field correctly.
 - **Expected Results:** The correct information is shown after saving.
- **Test Case 2:** Ensure `loadSchedule` accurately returns schedule data.
 - **Expected Results:** The correct data is loaded.
- **Test Case 3:** Verify start and end times conform to expected format.
 - **Expected Results:** The correct information is shown after matching.

7.3.7 Waitlist Class

- **Test Case 1:** Save a new waitlist entry to `Waitlist.csv`.
 - **Expected Results:** A new row is added in `Waitlist.csv` with the correct `courseID`, `studentIDs`, and `maxWaitlist` values.
- **Test Case 2:** Load an existing waitlist from `Waitlist.csv`.
 - **Expected Results:** The correct `Waitlist` object is returned for a given `courseID`.

- **Test Case 3:** Add a student to the waitlist if it has not reached its maximum capacity.
 - **Expected Results:** The student ID is added to `studentIDs`, and the method returns `true`.
- **Test Case 4:** Attempt to add a student when the waitlist is full.
 - **Expected Results:** The method returns `false`.
- **Test Case 5:** Remove a student from the waitlist if they decide not to wait or are enrolled.
 - **Expected Results:** The student ID is removed from `studentIDs`, and the method returns `true`.
- **Test Case 6:** Attempt to remove a student who is not on the waitlist.
 - **Expected Results:** The method returns `false`.
- **Test Case 7:** Remove a student from the waitlist if they decide not to wait or are enrolled.
 - **Expected Results:** The student ID is removed from `studentIDs`, and the method returns `true`.
- **Test Case 8:** Attempt to remove a student who is not on the waitlist.
 - **Expected Results:** The method returns `false`.
- **Test Case 9:** Retrieve the position of a specific student on the waitlist.
 - **Expected Results:** The correct 1-based position of the student is returned.
- **Test Case 10:** Attempt to get the position of a student who is not on the waitlist.
 - **Expected Results:** The method returns `-1` or an indication of non-existence.

8. Use Cases

8.1 System Login

Use Case ID: UC-SL-001

Use Case Name: Logging in to the system
Relevant Requirements: 3.1.3.1, 3.1.3.2
Primary Actor: Student, Administrator, Faculty

Pre-conditions:

- User has login information

Post-conditions:

- User is logged in

Basic Flow or Main Scenario:

1. User enters username, password, and school code
 - Students will then have access to their class schedule, and will be able to enroll/drop classes.
 - Administrators will be able to access courses, classes, student information, and student enrollment.
 - Faculty will be able to update their course and class information, and submit grades.

Extensions or Alternate Flows:

- None

Exceptions:

- Username doesn't exist
- Password doesn't match given username
- School code is wrong

Related Use Cases:

- Student Course Registration
- Faculty Grade Submission
- Administrator Course Management
- Student Course Drop
- Administrator Creates Schedule
- User Management

8.2 Student Course Registration

Use Case ID: UC-SCR-001
Use Case Name: Register for a Course
Relevant Requirements: 3.1.3.1, 3.1.3.2, 3.1.3.4
Primary Actor: Student

Pre-conditions:

- Student is logged into the system
- Student has met all prerequisites for the desired course
- The course has available slots

Post-conditions:

- Student is enrolled in the course
- Course enrollment count is updated
- Student's schedule is updated

Basic Flow or Main Scenario:

1. Student navigates to the course registration page
2. Student searches for the desired course
3. System displays course details and availability
4. Student selects the course and clicks "Register"
5. System checks for schedule conflicts and prerequisites
6. System enrolls the student in the course
7. System displays confirmation message
8. System updates student's schedule and course enrollment count

Extensions or Alternate Flows:

- 4a. Course is full:
 1. System adds student to the course waitlist
 2. System displays waitlist confirmation
- 5a. Schedule conflict detected:
 1. System displays conflict warning
 2. Student can choose to cancel registration or proceed with conflict

Exceptions:

- Student does not meet course prerequisites
- System experiences a file I/O error during enrollment update

Related Use Cases:

- View Course Catalog
- Student Course Drop
- System Login

8.3 Faculty Grade Submission

Use Case ID: UC-FGS-001

Use Case Name: Submit Student Grades

Relevant Requirements: 3.1.4.2, 3.1.4.3

Primary Actor: Faculty

Pre-conditions:

- Faculty is logged into the system
- Faculty is assigned to the course
- The grading period is open

Post-conditions:

- Student grades are recorded in the system
- Grade submission is marked as complete for the course

Basic Flow or Main Scenario:

1. Faculty navigates to the grade submission page
2. System displays list of assigned courses
3. Faculty selects a course
4. System displays the class roster with grade input fields
5. Faculty enters grades for each student
6. Faculty submits the completed grade sheet
7. System validates the entered grades
8. System saves the grades and marks submission as complete
9. System displays confirmation message

Extensions or Alternate Flows:

- 6a. Faculty saves grades as draft:
 1. System saves current progress without finalizing
 2. Faculty can return later to complete submission
- 7a. Invalid grade detected:
 1. System highlights invalid entries
 2. Faculty corrects errors and resubmits

Exceptions:

- System experiences a file I/O error during grade saving
- Grading period closes before submission is complete

Related Use Cases:

- View Class Roster
- Update Course Syllabus
- System Login

8.4 Administrator Course Management

Use Case ID: UC-ACM-001

Use Case Name: Create New Course

Relevant Requirements: 3.1.2.1, 3.1.2.2, 3.1.2.4

Primary Actor: Administrator

Pre-conditions:

- Administrator is logged into the system
- The course does not already exist in the catalog

Post-conditions:

- New course is added to the course catalog
- Course is available for student registration (if set as active)

Basic Flow or Main Scenario:

1. Administrator navigates to the course management page
2. Administrator selects "Create New Course" option
3. System displays course creation form
4. Administrator enters course details (code, name, description, credits, prerequisites)
5. Administrator sets maximum enrollment capacity
6. Administrator sets course status (active/inactive)
7. Administrator submits the new course information
8. System validates the entered information
9. System adds the new course to the catalog
10. System displays confirmation message

Extensions or Alternate Flows:

- 8a. Duplicate course code detected:
 1. System displays warning message
 2. Administrator can modify code or cancel creation
- 6a. Administrator sets course as inactive:
 1. Course is added to catalog but not available for registration

Exceptions:

- System experiences a file I/O error during course creation
- Invalid data format detected during validation

Related Use Cases:

- Edit Existing Course
- Set Course Prerequisites

- System Login

8.5 Student Course Drop

Use Case ID: UC-SCD-001

Use Case Name: Student Drops Course

Relevant Requirements: 3.1.1.5, 3.1.3.3, 3.1.4.2

Primary Actor: Student

Pre Conditions:

- Student is logged into system
- Student is in the course

Post Condition:

- Student is dropped from course
- Related Faculty is notified of student dropped
- System updates the number of students in course

Basic Flow or Main Scenario:

1. Student navigates to drop course page
2. Student chooses the course to drops and submits
3. Student is dropped from said course
4. Related faculty is notified of the student dropping the course
5. System updates to account for the dropped student
6. Display confirmation message

Extension or Alternate Flows:

- Detected that the student will go below full-time
 1. Warning message will appear on screen
 2. Will ask for another confirmation to drop course

Exceptions:

- File I/O error when updating file for course and student

Related Use Case:

- Student Course Registration
- Student Waitlist
- System Login

8.6 Administrator Creates Schedule

Use Case ID: UC-ACS-001

Use Case Name: Administrator creates a schedule

Relevant Requirements: 3.1.3.4, 3.1.3.5

Primary Actor: Administrator

Pre Conditions:

- Administrator is logged into the system

Post Condition:

- A schedule is created for one particular course

Basic Flow or Main Scenario:

1. Administrator logs into the system
2. Administrator clicks on 'Create a schedule' button
3. Administrator selects a course to apply the schedule to
4. Administrator selects the days of the week and time slots
5. Administrator clicks 'Finish'
6. System adds the schedule for the course into the database

Extension or Alternate Flows:

- Detected that there exists a schedule for that particular course
 1. Warning message will appear on screen
 2. Will suggest to modify the existing schedule

Exceptions:

- File I/O error when updating file for schedule

Related Use Case:

- Student Course Registration
- Student Waitlist
- System Login

8.7 User Management

Use Case ID: UC-UM-001

Use Case Name: Administrator creates, deletes, or modifies a user

Relevant Requirements: 3.1.1.1

Primary Actor: Administrator

Pre Conditions:

- Administrator is logged into the system

Post Condition:

- A user is created, deleted, or modified

Basic Flow or Main Scenario:

1. Administrator logs into the system
2. To create a user:
 - a. Administrator clicks on 'Add user' button
 - b. Administrator fills out a basic form containing the following fields
 - i. Username
 - ii. Password
 - iii. Institution Code
 - iv. Drop down: Student or Administrator
 - c. Administrator clicks 'Finish'
 - d. System adds the user into the database
3. To edit a user:
 - a. Administrator clicks on 'Edit user' button
 - b. Administrator can modify the following fields
 - i. Username
 - ii. Password
 - iii. Institution Code
 - iv. Drop down: Student or Administrator
 - c. Administrator clicks 'Finish'
 - d. System reinserts the user into the database
4. To delete a user:
 - a. Administrator clicks on 'Remove user' button
 - b. Administrator selects the username(s) to delete
 - c. System removes the user(s) from the database

Extension or Alternate Flows:

- Detected that there are empty fields for adding or modifying users
 - Warning message will appear on screen
- Detected that there are no users selected to delete
 - Warning message will appear on screen
- Detected that the only administrator account is being deleted
 - Error message will appear on the screen. You cannot delete the only registered administrator account.

Exceptions:

- File I/O error when updating a user

Related Use Case:

- System Login

8.8 View Class Roster

Use Case ID: UC-VCR-001 Use Case Name: View Class Roster Relevant Requirements: 3.1.4.3 Primary Actor: Faculty

Pre-conditions:

- Faculty is logged into the system
- Faculty is assigned to at least one course

Post-conditions:

- Class roster is displayed for the selected course

Basic Flow or Main Scenario:

1. Faculty navigates to the "My Courses" section
2. System displays a list of courses assigned to the faculty
3. Faculty selects a specific course
4. System retrieves the class roster from the course data file
5. System displays the roster with student names, IDs, and enrollment status

Extensions or Alternate Flows:

- 2a. No courses assigned:
 1. System displays message indicating no courses are currently assigned
- 5a. Faculty requests to export roster:
 1. System provides option to export as CSV or plain text file
 2. Faculty selects format and initiates export
 3. System generates and saves the exported file

Exceptions:

- System cannot access course data file
- Course assignment data is corrupted or incomplete

Related Use Cases:

- Submit Student Grades
- Generate Course Report

8.10 Update Course Syllabus

Use Case ID: UC-UCS-001 Use Case Name: Update Course Syllabus Relevant Requirements: 3.1.4.4 Primary Actor: Faculty

Pre-conditions:

- Faculty is logged into the system
- Faculty is assigned to the course
- Course exists in the system

Post-conditions:

- Course syllabus is updated in the system
- Update timestamp is recorded

Basic Flow or Main Scenario:

1. Faculty navigates to course management section
2. System displays list of assigned courses
3. Faculty selects a course to update
4. System loads current syllabus content
5. Faculty modifies syllabus content
6. Faculty submits updated syllabus
7. System validates the content
8. System saves the updated syllabus
9. System confirms successful update

Extensions or Alternate Flows:

- 5a. Faculty saves draft:
 1. System saves current progress without publishing
 2. Faculty can return later to complete updates
- 7a. Content exceeds size limit:
 1. System displays warning
 2. Faculty must reduce content before submitting

Exceptions:

- File system error during save operation
- Concurrent update attempt by another user

Related Use Cases:

- View Class Roster
- Create New Course

8.11 View Course Catalog

Use Case ID: UC-VCC-001 Use Case Name: View Course Catalog Relevant Requirements: 3.1.3.1 Primary Actor: Student

Pre-conditions:

- User is logged into the system
- Course catalog data is available

Post-conditions:

- Course catalog is displayed to the user

Basic Flow or Main Scenario:

1. User navigates to the course catalog section
2. System retrieves course catalog data
3. System displays list of all available courses
4. User can scroll through the catalog
5. System provides filtering and sorting options

Extensions or Alternate Flows:

- 3a. User applies filters:
 1. User selects filtering criteria (department, course level, etc.)
 2. System updates display with filtered results
- 3b. User uses search function:
 1. User enters search terms
 2. System displays matching courses

Exceptions:

- Course catalog data is unavailable
- System encounters error while filtering/sorting

Related Use Cases:

- Register Course
- Search Courses

8.12 Generate Enrollment Report

Use Case ID: UC-GER-001 Use Case Name: Generate Enrollment Report Relevant Requirements: 3.1.6.1, 3.1.6.2, 3.1.6.3, 3.1.6.5 Primary Actor: Administrator

Pre-conditions:

- Administrator is logged into the system
- Enrollment data is available in the system

Post-conditions:

- Requested report is generated and available for viewing/export

Basic Flow or Main Scenario:

1. Administrator navigates to the reporting section
2. System displays available report types
3. Administrator selects "Enrollment Report"
4. Administrator configures report parameters (term, department, etc.)
5. Administrator selects report format (CSV, plain text)
6. System generates the report
7. System displays report preview
8. Administrator can export or print the report

Extensions or Alternate Flows:

- 4a. Administrator selects historical data:
 1. System retrieves archived enrollment data
 2. Report includes historical trends
- 6a. Report generation takes longer than expected:
 1. System displays progress indicator
 2. Administrator can cancel report generation

Exceptions:

- Required data files are corrupted or unavailable
- System lacks sufficient memory to generate large reports

Related Use Cases:

- View Class Roster
- Generate Waitlist Report

8.13 Search Courses

Use Case ID: UC-SC-001 Use Case Name: Search Courses Relevant Requirements: 3.1.2.3 Primary Actor: Student/Faculty/Administrator

Pre-conditions:

- User is logged into the system

- Course catalog is available

Post-conditions:

- Search results are displayed to the user

Basic Flow or Main Scenario:

1. User navigates to course search function
2. System displays search interface with various criteria options
3. User enters search criteria (course code, name, department)
4. User initiates search
5. System queries course catalog data
6. System displays matching courses
7. User can select a course for more details

Extensions or Alternate Flows:

- 6a. No courses match criteria:
 1. System displays "No results found" message
 2. Suggests broadening search criteria
- 6b. Too many results:
 1. System suggests refining search criteria
 2. Provides option to view all results

Exceptions:

- Search function fails due to data access error
- Invalid search criteria format

Related Use Cases:

- View Course Catalog
- Register Course

8.14 View Waitlist

Use Case ID: UC-VW-001 Use Case Name: Administrator View Waitlist

Relevant Requirements: 3.1.5.4 Primary Actor: Administrator

Pre-conditions:

- Administrator is logged into the system
- At least one course has an active waitlist

Post-conditions:

- Waitlist information is displayed to the administrator

Basic Flow or Main Scenario:

1. Administrator navigates to the waitlist management section
2. System displays a list of courses with active waitlists
3. Administrator selects a specific course
4. System retrieves the waitlist data for the selected course
5. System displays detailed waitlist information including:
 - Student names and IDs
 - Timestamp of waitlist entry
 - Current position in waitlist

Extensions or Alternate Flows:

- 2a. No active waitlists:
 1. System displays message indicating no current waitlists
- 3a. Administrator applies filters:
 1. Administrator selects filtering criteria (e.g., department, course level)
 2. System updates display with filtered results

Exceptions:

- System cannot access waitlist data file
- Waitlist data is corrupted or incomplete

Related Use Cases:

- Generate Waitlist Report
- Modify Course Waitlist

8.15 Generate Waitlist Report

Use Case ID: UC-GWR-001 Use Case Name: Generate Waitlist Report Relevant Requirements: 3.1.6.2, 3.1.6.3 Primary Actor: Administrator

Pre-conditions:

- Administrator is logged into the system
- Waitlist data is available in the system

Post-conditions:

- Waitlist report is generated and available for export

Basic Flow or Main Scenario:

1. Administrator navigates to the reporting section

2. System displays available report types
3. Administrator selects "Waitlist Report"
4. Administrator configures report parameters:
 - Time period
 - Specific courses or departments
 - Include historical waitlist movement
5. Administrator selects output format (CSV, plain text)
6. System generates the report
7. System displays report preview
8. Administrator exports or prints the report

Extensions or Alternate Flows:

- 6a. Large report generation:
 1. System displays progress bar
 2. Administrator can cancel report generation
- 7a. Administrator requests data visualization:
 1. System generates graphs showing waitlist trends
 2. Graphs are included in the exported report

Exceptions:

- Report generation fails due to data access error
- Insufficient system resources for large reports

Related Use Cases:

- View Waitlist
- Generate Enrollment Report

8.16 Modify Course Waitlist

Use Case ID: UC-MCW-001 Use Case Name: Modify Course Waitlist Relevant Requirements: 3.1.5.5 Primary Actor: Administrator

Pre-conditions:

- Administrator is logged into the system
- Course waitlist exists
- Student records are accessible

Post-conditions:

- Course waitlist is updated
- Affected students are notified of changes

Basic Flow or Main Scenario:

1. Administrator navigates to waitlist management section
2. System displays list of courses with waitlists
3. Administrator selects a course
4. System displays current waitlist for the course
5. Administrator chooses to add or remove a student
6. If adding: a. Administrator searches for student by ID or name b. System verifies student eligibility c. Administrator selects position for new entry
7. If removing: a. Administrator selects student from waitlist b. Administrator confirms removal
8. System updates waitlist
9. System generates notifications for affected students

Extensions or Alternate Flows:

- 6b. Student already on waitlist:
 1. System displays warning message
 2. Administrator can cancel or move student
- 7a. Administrator moves student position:
 1. Administrator selects new position
 2. System reorders waitlist accordingly

Exceptions:

- Student record not found
- System fails to update waitlist file

Related Use Cases:

- View Waitlist
- Generate Waitlist Report

8.17 View Personal Waitlist Position

Use Case ID: UC-VWP-001 Use Case Name: View Personal Waitlist Position

Relevant Requirements: 3.1.5.2 Primary Actor: Student

Pre-conditions:

- Student is logged into the system
- Student is on at least one course waitlist

Post-conditions:

- Student views their current waitlist positions

Basic Flow or Main Scenario:

1. Student navigates to "My Waitlists" section
2. System retrieves student's waitlist data
3. System displays list of courses where student is waitlisted:
 - Course name and code
 - Current position on waitlist
 - Estimated time/chance of enrollment
 - Option to remove self from waitlist
4. Student can select a specific course for more details
5. System displays detailed waitlist information for selected course

Extensions or Alternate Flows:

- 2a. No active waitlists:
 1. System displays message indicating student is not on any waitlists
 2. Provides link to course registration
- 4a. Student removes self from waitlist:
 1. System prompts for confirmation
 2. Upon confirmation, removes student and updates waitlist

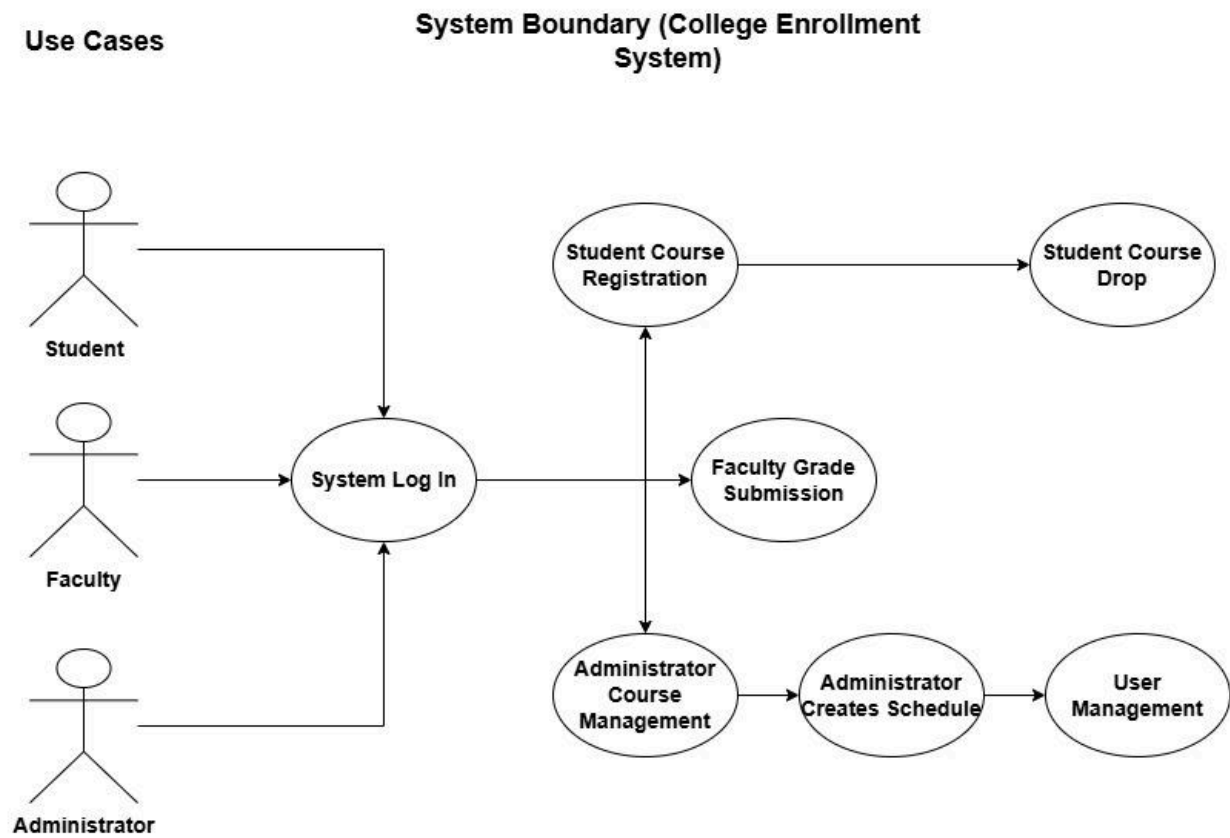
Exceptions:

- System cannot access waitlist data
- Waitlist position calculation error

Related Use Cases:

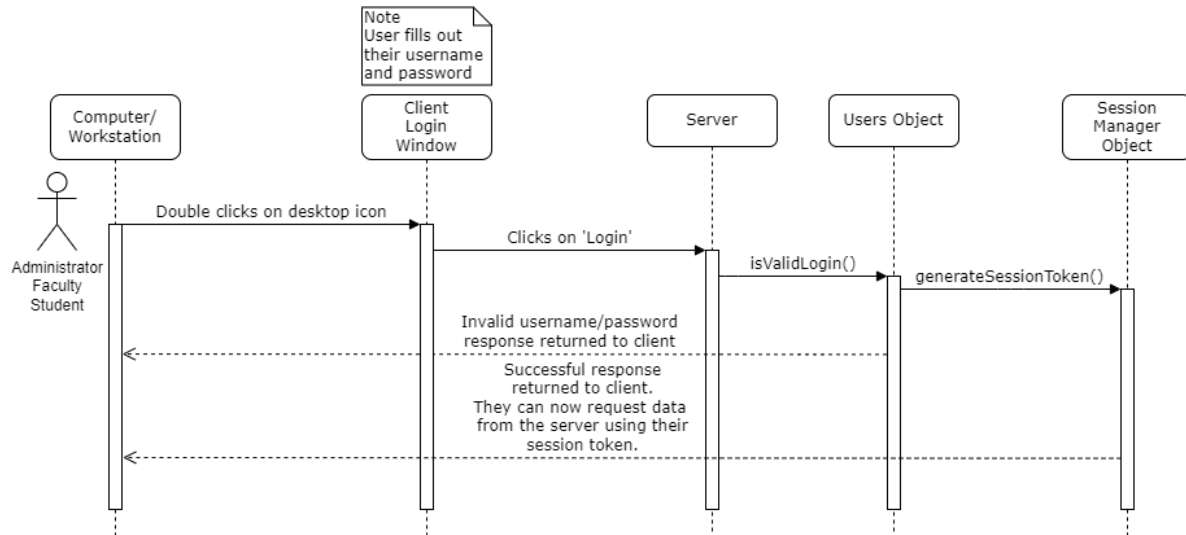
- Register for a Course
- View Course Catalog

8.2 Use Case Diagram

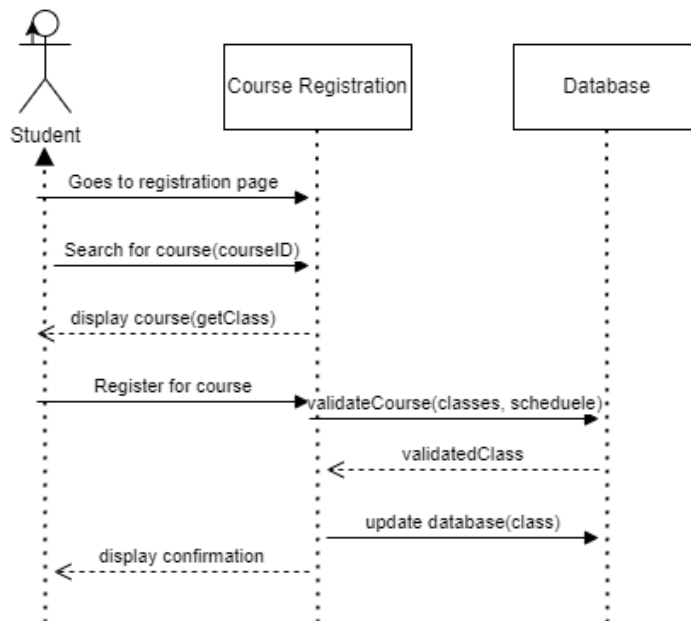


9. Sequence Diagrams

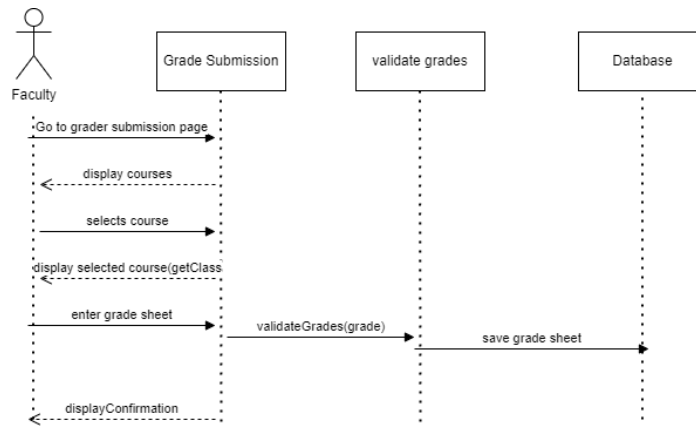
9.1 System Login



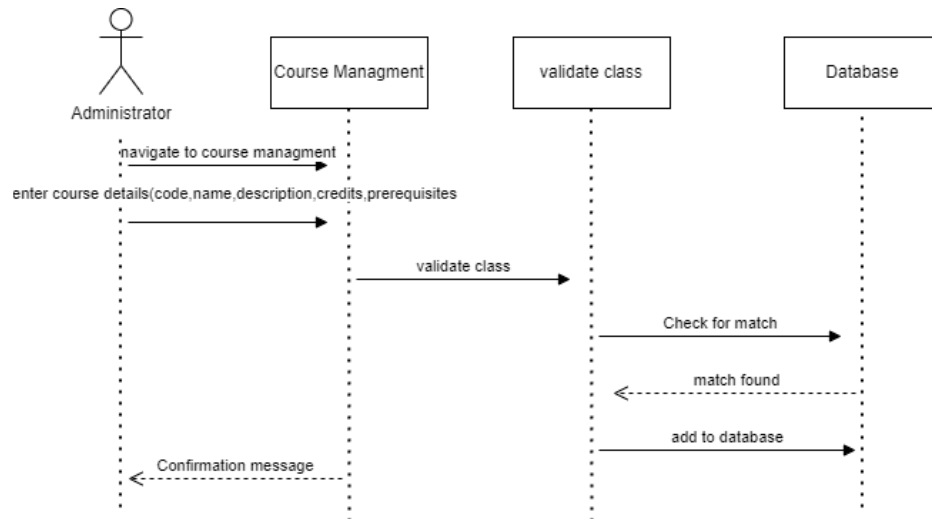
9.2 Student Course Registration



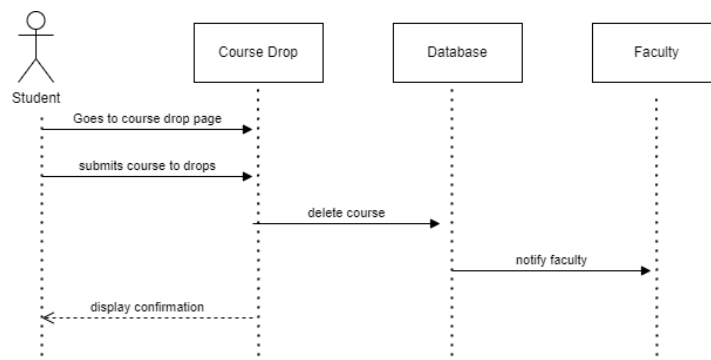
9.3 Faculty Grade Submission



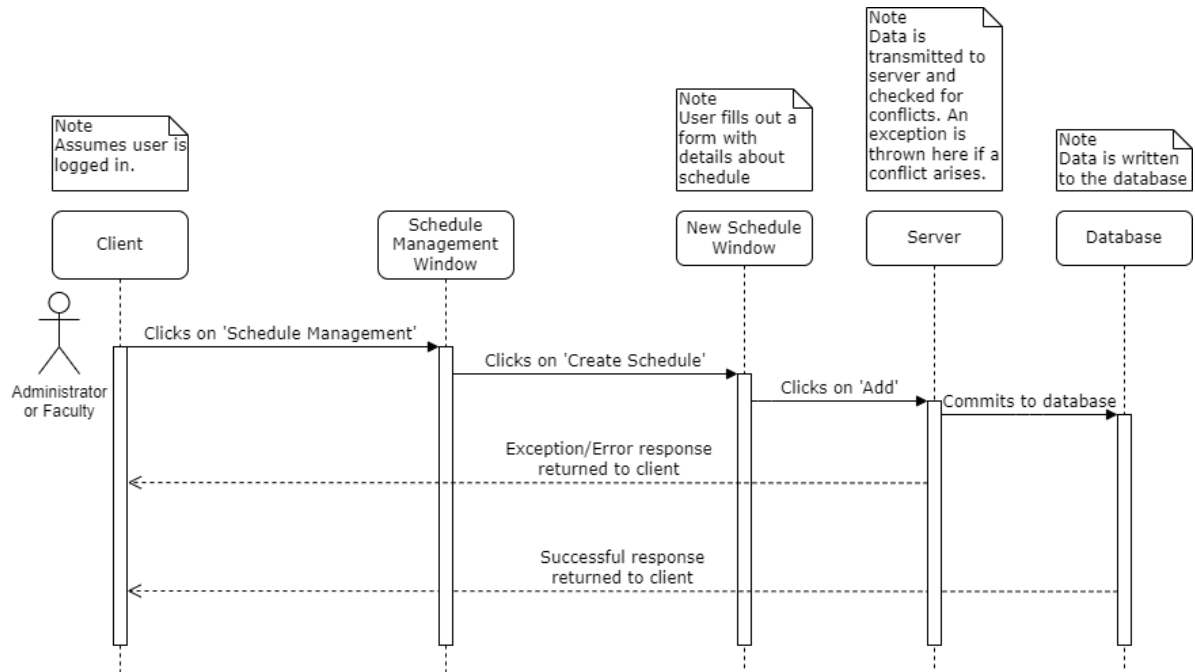
9.4 Administrator Course Management



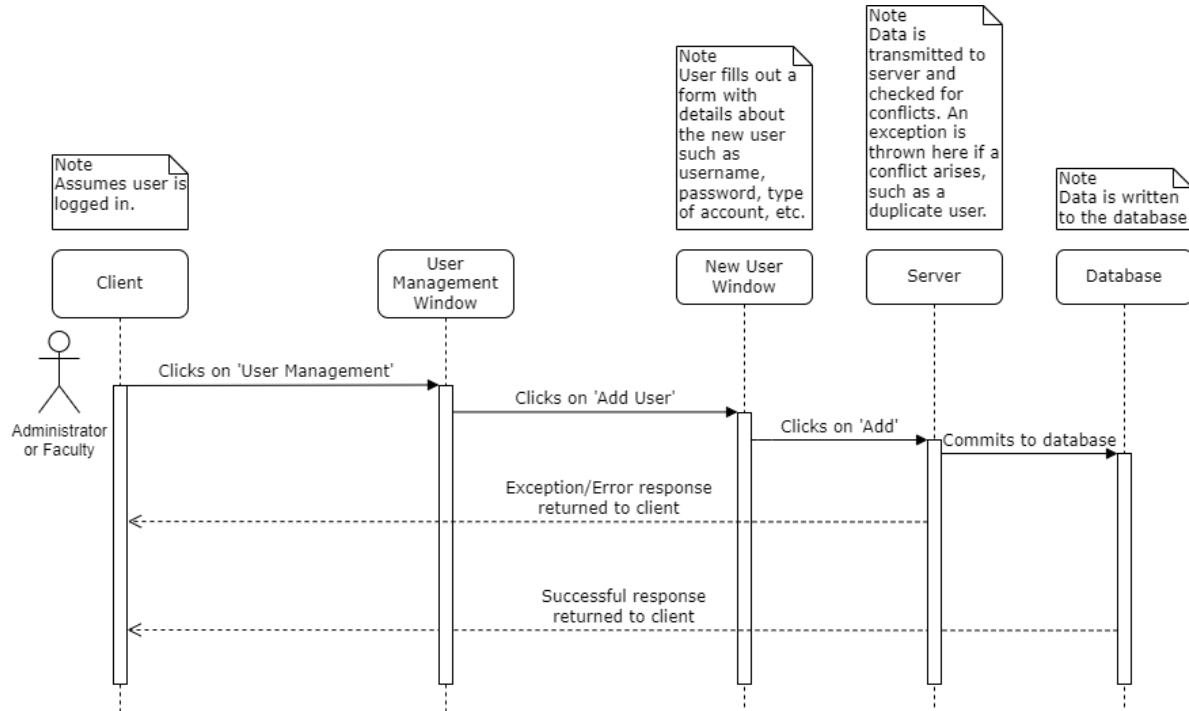
9.5 Student Course Drop



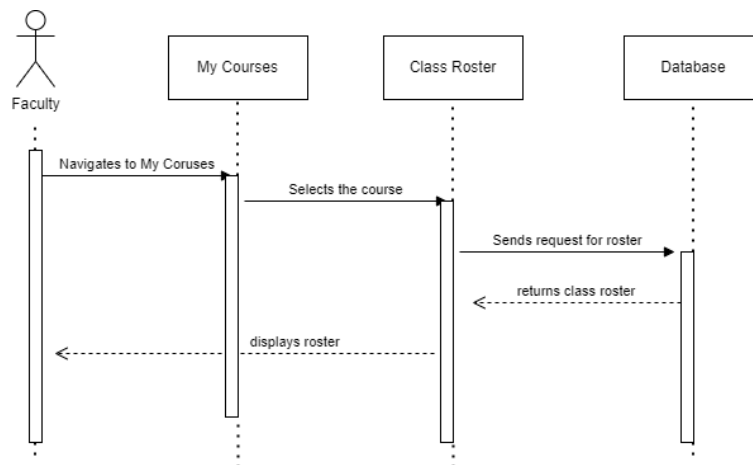
9.6 Administrator Creates Schedule



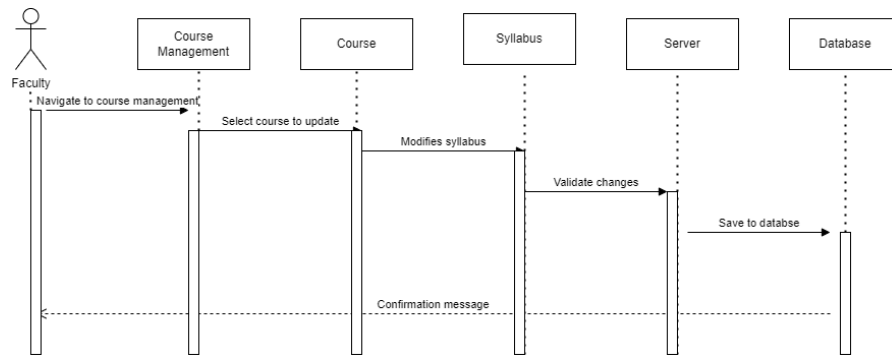
9.7 User Management



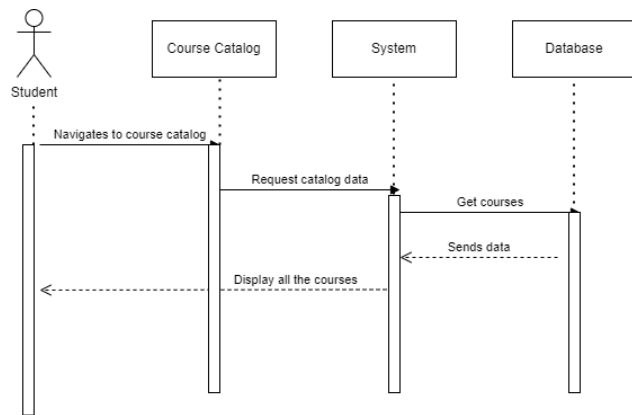
9.8 View Class Roster



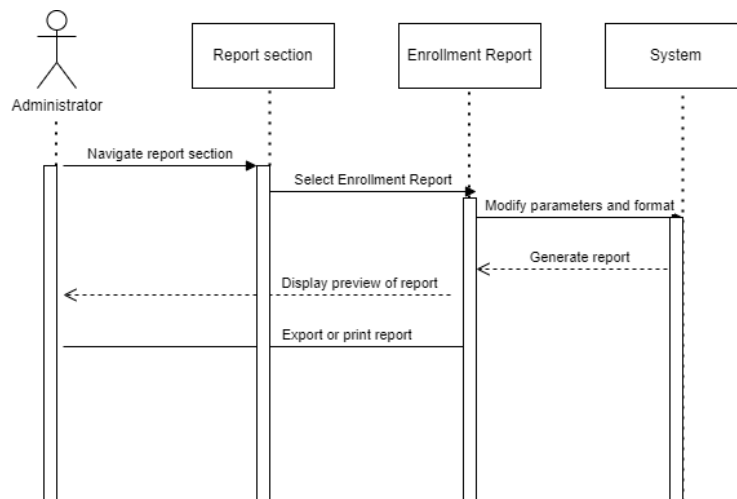
9.9 Update Course Syllabus



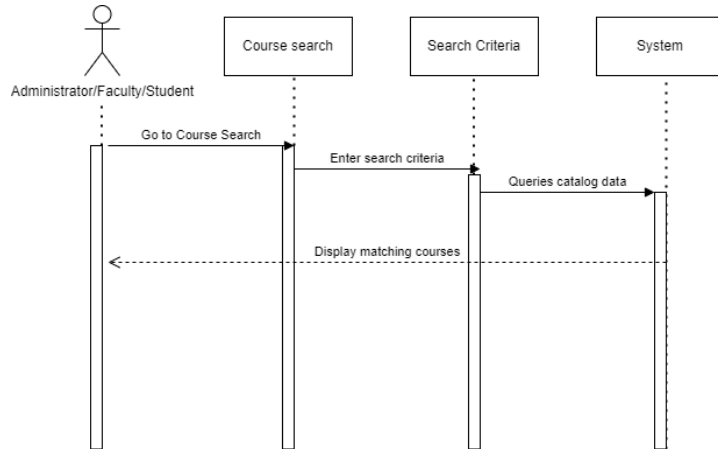
9.10 View Course Catalog



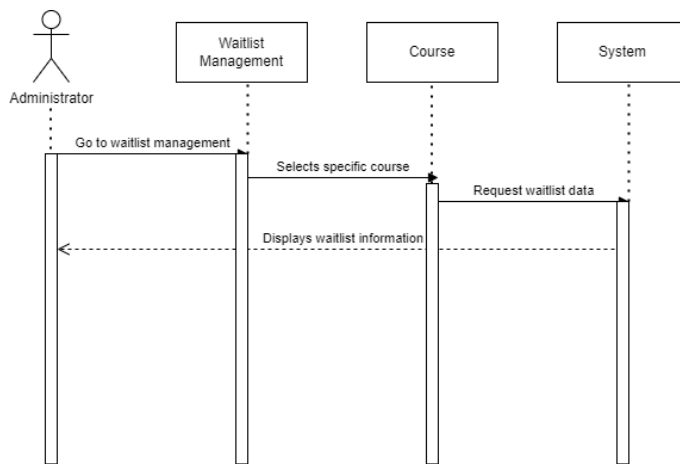
9.11 Generate Enrollment Report



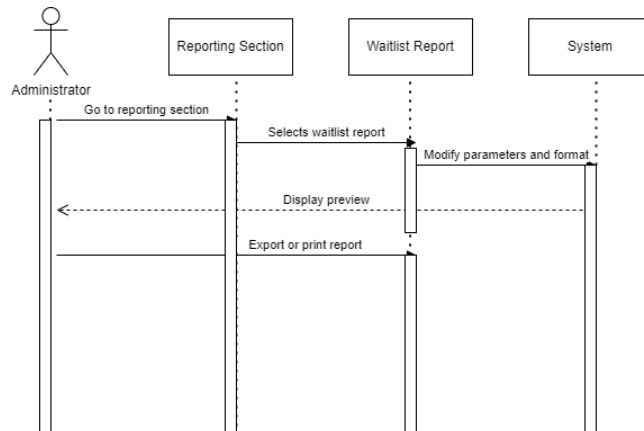
9.12 Search Courses



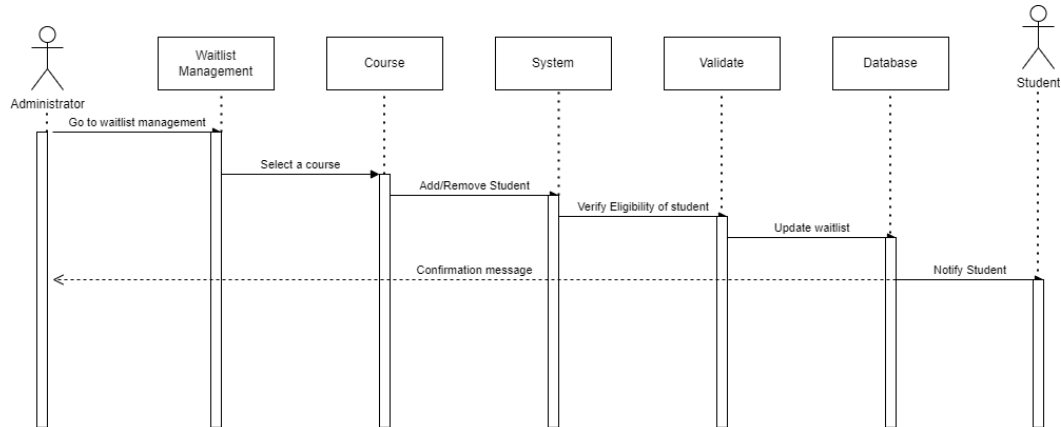
9.13 View Waitlist



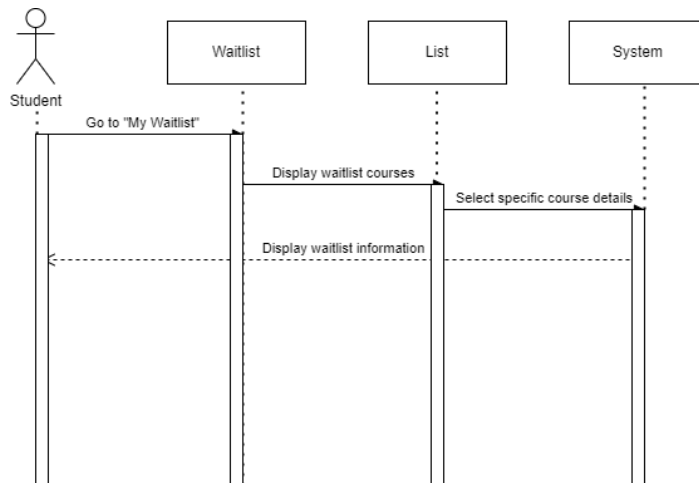
9.14 Generate Waitlist Report



9.15 Modify Course Waitlist



9.16 View Personal Waitlist Position



10. Data Integrity and Security

- **Immediate Persistence:** Data changes are immediately saved to prevent data loss.
- **Encryption:** User passwords will be hashed and salted.
- **Session Management:** Uses `SessionManager` to track user sessions with expiration times.

11. Concurrency and Synchronization

- **Concurrency:** `FileDataManager` will use synchronized methods for reading/writing files to prevent data corruption during concurrent access.
- **Multi-threaded Server:** If expanded for network use, the `Server` class will manage multiple connections using client handlers for each user.

12. Error Handling

- **File I/O Errors:** Any read/write operations will be enclosed in try-catch blocks, logging errors for system admins.
- **User Feedback:** All errors (e.g., login failures, schedule conflicts) will display user-friendly messages on the GUI.

13. Future Enhancements

- **Network-based Data Storage:** Integration with a central database for remote access.
- **Enhanced Reporting:** Additional reporting features, such as enrollment trends

14. Class UML Diagrams

https://github.com/chenAndrew4/CS401ClassEnrollmentSystem/blob/main/401_UML_00.png

