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 design	Chen Li
10/20	1.0	Adding initial data table design	Chen Li
10/24	1.0	Adding initial test case design	Chen Li
10/28	1.0	Adding Class UML diagram	Chen Li

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.txt, courses.txt, schedules.txt, waitlists.txt). Each file will be formatted to ensure easy parsing and loading of data:

- **User Class**:

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

Test Cases

1. **Save User**: Verify `User.csv` includes a new row with the correct information after saving.
2. **Load User**: Confirm that `loadUser` returns the correct `User` object for an existing userID.
3. **Authentication**: Ensure `isAuthenticated` reflects login/logout status accurately.

---

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

Test Cases

1. **Save Student**: Ensure a new entry in `Student.csv` with the correct course IDs.
2. **Load Student**: Validate that `loadStudent` correctly restores a `Student` object.
3. **Course Enrollment**: Confirm `enrolledCourses` and `waitlistedCourses` lists match data in `Student.csv`.

3. Faculty Class

Table Fields in Faculty.csv

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

Test Cases

1. **Save Faculty**: Confirm data in `Faculty.csv` after saving a faculty member.
2. **Load Faculty**: Verify that `loadFaculty` retrieves the correct courses for a faculty member.
3. **Course Assignment**: Check `assignedCourses` reflects correctly after updating in `Faculty.csv`.

4. 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)

Test Cases

1. **Save Course**: Verify `Course.csv` correctly saves course attributes.
2. **Load Course**: Ensure `loadCourse` retrieves the course data accurately.
3. **Enrollment Limits**: Confirm `enrollmentLimit` matches value stored in `Course.csv`.

---

5. 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

Test Cases

1. **Save Schedule**: Check that `Schedule.csv` saves each field correctly.
2. **Load Schedule**: Ensure `loadSchedule` accurately returns schedule data.

3. **Time Formatting**: Verify start and end times conform to expected format.

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.

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

4. **saveWaitlist**: Writes the waitlist information for each course to a row in `Waitlist.csv`. Each row includes `courseID`, `studentIDs` (semicolon-separated), and `maxWaitlist`.
5. **loadWaitlist**: Reads from `Waitlist.csv` to load the waitlist for a specific course.
6. **addStudent**: Adds a student ID to the waitlist if the `maxWaitlist` limit has not been reached.
7. **removeStudent**: Removes a student ID from the waitlist if they drop their spot or are promoted to enrollment.
8. **getWaitlistPosition**: Returns the position of a specific student in the waitlist.

Test Cases for Waitlist Class

1. **Save Waitlist**:

    - **Description**: Save a new waitlist entry to `Waitlist.csv`.
    - **Expected Result**: A new row is added in `Waitlist.csv` with the correct `courseID`, `studentIDs`, and `maxWaitlist` values.

2. **Load Waitlist**:

- **Description**: Load an existing waitlist from `Waitlist.csv`.
- **Expected Result**: The correct `Waitlist` object is returned for a given `courseID`.

3. **Add Student to Waitlist**:

   - **Description**: Add a student to the waitlist if it has not reached its maximum capacity.
   - **Expected Result**: The student ID is added to `studentIDs`, and the method returns `true`.
   - **Edge Case**: Attempt to add a student when the waitlist is full.
   - **Expected Result**: The method returns `false`.

4. **Remove Student from Waitlist**:

   - **Description**: Remove a student from the waitlist if they decide not to wait or are enrolled.
   - **Expected Result**: The student ID is removed from `studentIDs`, and the method returns `true`.
   - **Edge Case**: Attempt to remove a student who is not on the waitlist.
   - **Expected Result**: The method returns `false`.

5. **Get Waitlist Position**:

   - **Description**: Retrieve the position of a specific student on the waitlist.
   - **Expected Result**: The correct 1-based position of the student is returned.
   - **Edge Case**: Attempt to get the position of a student who is not on the waitlist.
   - **Expected Result**: The method returns `-1` or an indication of non-existence.

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

1. Login Screen

Layout: BorderLayout

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

2. Role-Based Dashboards

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.

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.

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.
        - **JTextField**: Course ID, Course Name
        - **JComboBox**: 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.

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.

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.

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

2.1 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.

2.2 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.

2.3 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.

2.4 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.

2.5 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.

2.6 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.

2.7 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.

2.8 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 enrolled when a slot opens.
    - **Expected Result**: Student is moved from waitlist to enrolled.

2.9 GUI Validation

6.1 Login Functionality

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

6.2 Course Registration

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

6.3 Drop Course

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

6.4 Schedule Conflicts

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

6.5 Syllabus Management

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

6.6 Data Persistence

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

6.7 Waitlist Management

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

6.8 GUI Responsiveness

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

8. 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.

9. 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.

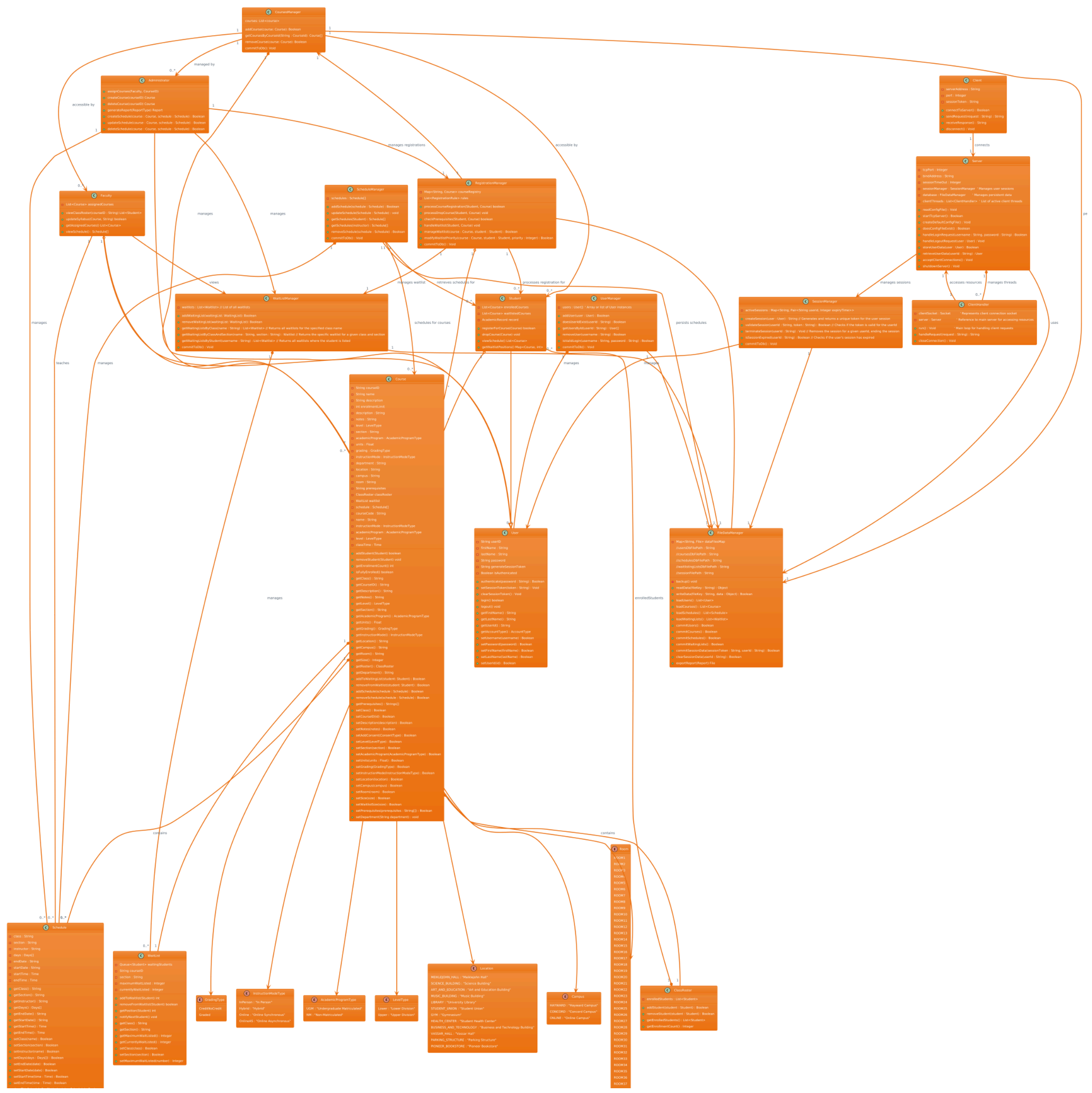
10. Error Handling

- **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.

11. Future Enhancements

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

12. Class UML Diagrams


CoursesManager
courses: List<course>
addCourse(course: Course): Boolean
getCoursesByCourseId(String : CourseId): Course[]
removeCourse(course: Course): Boolean
commitToDb(): Void
managed by
accessible by
Administrator
assignCourses(Faculty, CourseID)
createCourse(courseID) Course
deleteCourse(courseID) Course
generateReport(ReportType) Report
createSchedule(course : Course, schedule : Schedule) : Boolean
updateSchedule(course : Course, schedule : Schedule) : Boolean
deleteSchedule(course : Course, schedule : Schedule) : Boolean
manages registrations
accessible by
Faculty
List<Course> assignedCourses
viewClassRoster(courseID : String) List<Student>
updateSyllabus(Course, String) boolean
getAssignedCourses() List<Course>
viewSchedule() : Schedule[]
ScheduleManager
schedules : Schedule[]
addSchedule(schedule : Schedule) : Boolean
updateSchedule(Schedule : Schedule) : void
getSchedules(Student) : Schedule[]
getSchedules(instructor) : Schedule[]
removeSchedule(schedule : Schedule) : Boolean
commitToDb() : Void
RegistrationManager
Map<String, Course> courseRegistry
List<RegistrationRule> rules
processCourseRegistration(Student, Course) boolean
processDropCourse(Student, Course) void
checkPrerequisites(Student, Course) boolean
handleWaitlist(Student, Course) void
manageWaitlist(course : Course, student : Student) : Boolean
modifyWaitlistPriority(course : Course, student : Student, priority : Integer) : Boolean
commitToDb() : Void
Client
serverAddress : String
port : Integer
sessionToken : String
connectToServer() : Boolean
sendRequest(request : String) : String
receiveResponse() : String
disconnect() : Void
connects
Server
tcpPort : Integer
bindAddress : String
sessionTimeOut : Integer
sessionManager : SessionManager ' Manages user sessions
database : FileDataManager ' Manages persistent data
clientThreads : List<ClientHandler> ' List of active client threads
readConfigFile() : Void
startTcpServer() : Boolean
createDefaultConfigFile() : Void
doesConfigFileExist() : Boolean
handleLoginRequest(username : String, password : String) : Boolean
handleLogoutRequest(user : User) : Void
storeUserData(user : User) : Boolean
retrieveUserData(userId : String) : User
acceptClientConnections() : Void
shutdownServer() : Void
manages sessions
accesses resources
manages threads
uses
pe
SessionManager
activeSessions : Map<String, Pair<String userId, Integer expiryTime>>
createSession(user : User) : String // Generates and returns a unique token for the user session
validateSession(userId : String, token : String) : Boolean // Checks if the token is valid for the userId
terminateSession(userId : String) : Void // Removes the session for a given userId, ending the session
isSessionExpired(userId : String) : Boolean // Checks if the user's session has expired
commitToDb() : Void
ClientHandler
clientSocket : Socket ' Represents client connection socket
server : Server ' Reference to main server for accessing resources
run() : Void ' Main loop for handling client requests
handleRequest(request : String) : String
closeConnection() : Void
manages
manages
views
manages waitlist
retrieves schedules for
processes registration for
schedules for courses
persists schedules
WaitListManager
waitlists : List<Waitlist> // List of all waitlists
addWaitingList(waitingList: WaitingList): Boolean
removeWaitingList(waitingList: WaitingList): Boolean
getWaitingListsByClass(name : String) : List<Waitlist> // Returns all waitlists for the specified class name
getWaitingListByClassAndSection(name : String, section : String) : Waitlist // Returns the specific waitlist for a given class and section
getWaitingListsByStudent(username : String) : List<Waitlist> // Returns all waitlists where the student is listed
commitToDb() : Void
Student
List<Course> enrolledCourses
List<Course> waitlistedCourses
AcademicRecord record
registerForCourse(Course) boolean
dropCourse(Course) void
viewSchedule() List<Course>
getWaitlistPositions() Map<Course, int>
UserManager
users : User[] ' Array or list of User instances
addUser(user : User) : Boolean
doesUserIdExist(userId : String) : Boolean
getUsersById(userId : String) : User[]
removeUser(username : String) : Boolean
isValidLogin(username : String, password : String) : Boolean
commitToDb() : Void
manages
teaches
manages
manages
Course
String courseID
String name
String description
int enrollmentLimit
description : String
notes : String
level : LevelType
section : String
academicProgram : AcademicProgramType
units : Float
grading : GradingType
instructionMode : InstructionModeType
department : String
location : String
campus : String
room : String
String prerequisites
ClassRoster classRoster
WaitList waitlist
schedule : Schedule[]
courseCode : String
name : String
instructionMode : InstructionModeType
academicProgram : AcademicProgramType
level : LevelType
classTime : Time
addStudent(Student) boolean
removeStudent(Student) void
getEnrollmentCount() int
isFullyEnrolled() boolean
getClass() : String
getCourseID() : String
getDescription() : String
getNotes() : String
getLevel() : LevelType
getSection() : String
getAcademicProgram() : AcademicProgramType
getUnits() : Float
getGrading() : GradingType
getInstructionMode() : InstructionModeType
getLocation() : String
getCampus() : String
getRoom() : String
getSize() : Integer
getRoster() : ClassRoster
getDepartment() : String
addToWaitingList(student: Student) : Boolean
removeFromWaitlist(student: Student) : Boolean
addSchedule(schedule : Schedule) : Boolean
removeSchedule(schedule : Schedule) : Boolean
getPrerequisites() : Strings[]
setClass() : Boolean
setCourseID(id) : Boolean
setDescription(description) : Boolean
setNotes(notes) : Boolean
setAddConsent(ConsentType) : Boolean
setLevel(LevelType) : Boolean
setSection(section) : Boolean
setAcademicProgram(AcademicProgramType) : Boolean
setUnits(units : Float) : Boolean
setGrading(GradingType) : Boolean
setInstructionMode(InstructionModeType) : Boolean
setLocation(location) : Boolean
setCampus(campus) : Boolean
setRoom(room) : Boolean
setSize(size) : Boolean
setWaitlistSize(size) : Boolean
setPrerequisites(prerequisites : String[]) : Boolean
setDepartment(String department) : void
User
String userID
firstName : String
lastName : String
String password
String generateSessionToken
Boolean isAuthenicated
authenticate(password : String) : Boolean
setSessionToken(token : String) : Void
clearSessionToken() : Void
login() boolean
logout() void
getFirstName() : String
getLastName() : String
getUserId() : String
getAccountType() : AccountType
setUsername(username) : Boolean
setPassword(password) : Boolean
setFirstName(firstName) : Boolean
setLastName(lastName) : Boolean
setUserId(id) : Boolean
enrolledStudents
FileDataManager
Map<String, File> dataFilesMap
//usersDbFilePath : String
//coursesDbFilePath : String
//schedulesDbFilePath : String
//waitlistingListsDbFilePath : String
//sessionFilePath : String
backup() void
readData(fileKey : String) : Object
writeData(fileKey : String, data : Object) : Boolean
loadUsers() : List<User>
loadCourses() : List<Course>
loadSchedules() : List<Schedule>
loadWaitingLists() : List<Waitlist>
commitUsers() : Boolean
commitCourses() : Boolean
commitSchedules() : Boolean
commitWaitingLists() : Boolean
commitSessionData(sessionToken : String, userId : String) : Boolean
clearSessionData(userId : String) : Boolean
exportReport(Report) File
contains
contains
Schedule
class : String
section : String
instructor : String
days : Days[]
endDate : String
startDate : String
startTime : Time
endTime : Time
getClass() : String
getSection() : String
getInstructor() : String
getDays() : Days[]
getEndDate() : String
getStartDate() : String
getStartTime() : Time
getEndTime() : Time
setClass(name) : Boolean
setSection(section) : Boolean
setInstructor(name) : Boolean
setDays(days : Days[]) : Boolean
setEndDate(date) : Boolean
setStartDate(date) : Boolean
setStartTime(time : Time) : Boolean
setEndTime(time : Time) : Boolean
WaitList
Queue<Student> waitingStudents
String courseID
section : String
maximumWaitListed : Integer
currentlyWaitListed : Integer
addToWaitlist(Student) int
removeFromWaitlist(Student) boolean
getPosition(Student) int
notifyNextStudent() void
getClass() : String
getSection() : String
getMaximumWaitListed() : Integer
getCurrentlyWaitListed() : Integer
setClass(class) : Boolean
setSection(section) : Boolean
setMaximumWaitListed(number) : Integer
GradingType
CreditNoCredit
Graded
InstructionModeType
InPerson : "In Person"
Hybrid : "Hybrid"
Online : "Online Synchronous"
OnlineAS : "Online Asynchronous"
AcademicProgramType
UGM : "Undergraduate Matriculated"
NM : "Non-Matriculated"
LevelType
Lower : "Lower Division"
Upper : "Upper Division"
Location
MEIKLEJOHN_HALL : "Meiklejohn Hall"
SCIENCE_BUILDING : "Science Building"
ART_AND_EDUCATION : "Art and Education Building"
MUSIC_BUILDING : "Music Building"
LIBRARY : "University Library"
STUDENT_UNION : "Student Union"
GYM : "Gymnasium"
HEALTH_CENTER : "Student Health Center"
BUSINESS_AND_TECHNOLOGY : "Business and Technology Building"
VASSAR_HALL : "Vassar Hall"
PARKING_STRUCTURE : "Parking Structure"
PIONEER_BOOKSTORE : "Pioneer Bookstore"
Campus
HAYWARD : "Hayward Campus"
CONCORD : "Concord Campus"
ONLINE : "Online Campus"
Room
ROOM1
ROOM2
ROOM3
ROOM4
ROOM5
ROOM6
ROOM7
ROOM8
ROOM9
ROOM10
ROOM11
ROOM12
ROOM13
ROOM14
ROOM15
ROOM16
ROOM17
ROOM18
ROOM19
ROOM20
ROOM21
ROOM22
ROOM23
ROOM24
ROOM25
ROOM26
ROOM27
ROOM28
ROOM29
ROOM30
ROOM31
ROOM32
ROOM33
ROOM34
ROOM35
ROOM36
ROOM37
ClassRoster
enrolledStudents : List<Student>
addStudent(student : Student) : Boolean
removeStudent(student : Student) : Boolean
getEnrolledStudents() : List<Student>
getEnrollmentCount() : Integer