Course Registration System Specification

I. System Introduction

The Course Registration System is developed to facilitate online course enrollment for students, grade management for lecturers, and data administration for system administrators.

II. Supplementary Specification

Supportability

None

III. Functional Requirements

3.1 Student Role

- Login: Access the system using Student ID and password.
- Password Recovery: Reset forgotten passwords via Email or OTP (One-Time Password).
- Logout: Securely exit the system.
- View Available Courses: View the list of currently open courses and their respective credit hours.
- Course Registration: Select and add courses to the registration list.
- Drop/Cancel Course: Remove a registered course within the designated registration period.
- View Registered Courses: Review the list of successfully registered courses.

3.2 Lecturer Role

- View Class List: Access and view the list of students enrolled in their assigned classes.
- Grade Management: Update and input grades for students.

3.3 Admin Role

- Course Management: Create, update, or delete course information (Add, Edit, Delete).
- Section Management: Manage class sections and set maximum enrollment capacity for each class.
- Student Management: Manage student records, including adding, updating, or removing student information.

- Open Registration Portal: Enable the system for students to begin registering for courses.
- Close Registration Portal: Disable the registration feature once the period ends.
- Lecturer Management: Manage faculty records, including adding, updating, or removing lecturer information.

IV. Non-Functional Requirements

4.1 Security

- Authorization: The system must ensure appropriate access rights. Students may only register for courses and view their own grades; Lecturers may only input grades for the classes they teach; Admins have full system administration privileges.

4.2 Performance and Speed

- Response Time: Basic operations (such as clicking the Register button or searching for courses) must receive a response within a maximum of 2 seconds.
- Load Capacity: The system must be able to handle at least 500 concurrent users during peak registration periods without crashing or data loss.

4.3 Stability and Reliability

- Availability: The system must ensure 24/7 operation throughout the course registration period (achieving 99.9% uptime).
- Data Integrity: In the event of a network error during registration, the system must ensure data consistency (i.e., the registration is either fully successful or not executed at all, adhering to SQL Transaction properties).

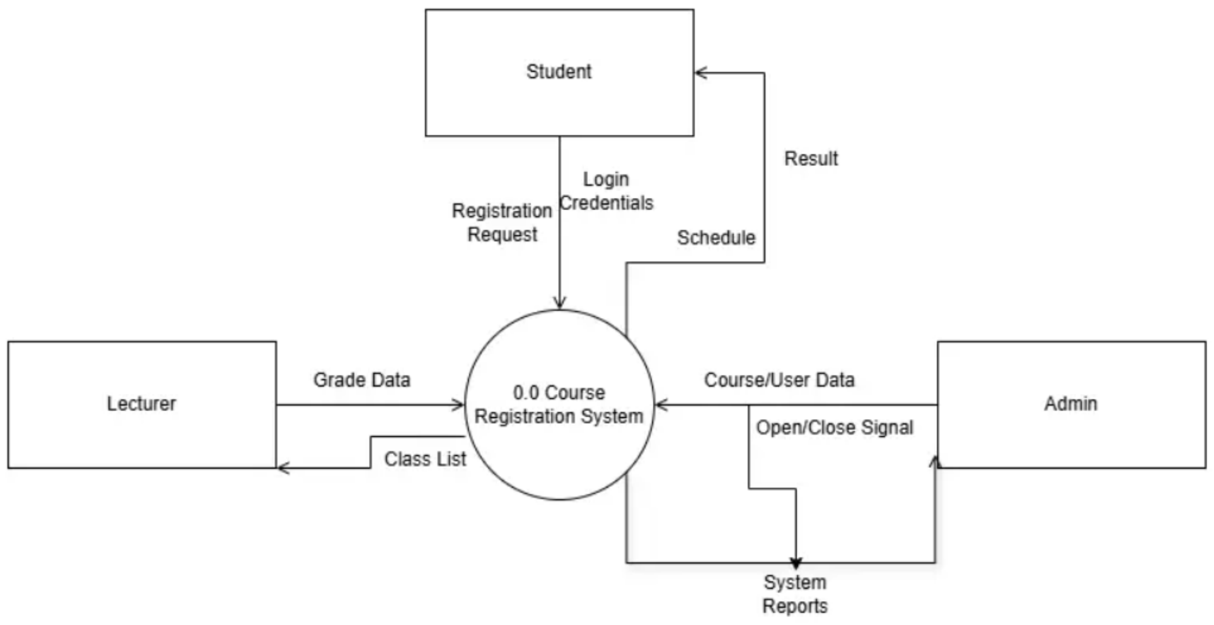
4.4 Usability

- Intuitive Interface: The interface should be simple with a clear layout so that students can perform registration independently without needing complex instruction manuals.
- Error Messages: Error messages (such as "Invalid ID or Password!") must be displayed in clear.

V. Data Flow Diagram

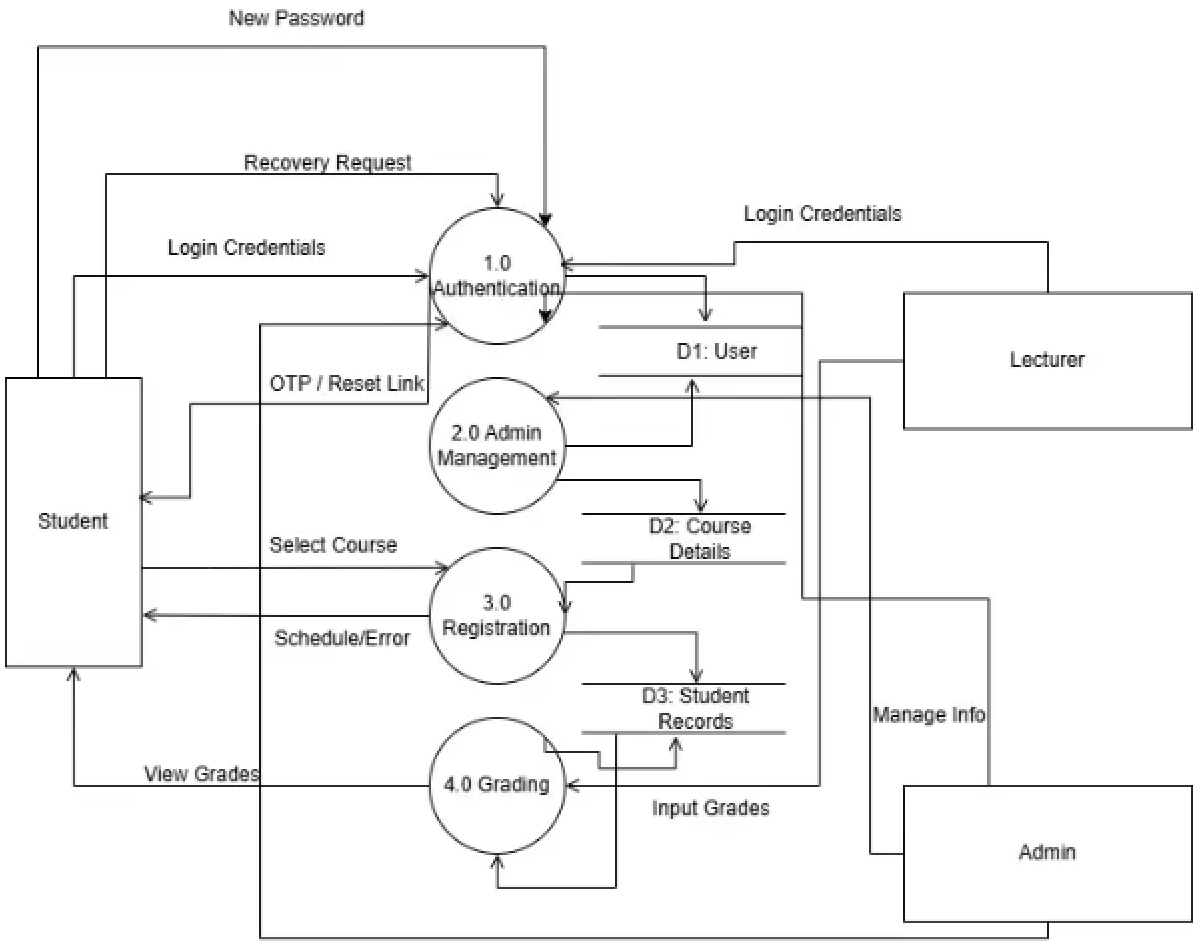
This section describes in detail how data moves and transforms within the system using Data Flow Diagrams (DFD).

5.1 DFD Level 0



- **Description:** The Course Registration System acts as a central processing unit, interacting with three main actors:
- **Student:** Submits login information and course registration requests. The system returns registration results and timetables.
- **Lecturer:** Submits grade data and receives a list of students in the class.
- **Admin:** Submits course settings, signals to open/close registration periods, and receives summary reports (System Reports).

5.2.DFD Level 1



Description of Processing Steps:

1.0 Authentication:

- Receive login information from Students, Lecturers, and Admins.
- Research data in repository **D1:User to verify identity**.
- Support password recovery via OTP/Reset Link.

2.0 Admin Management:

Allow Admins to add/edit/delete course information and save it to repository D2:Course Details.

Manage user information and update it in repository D1:User.

3.0 Registration:

- This is the most important process. Receive "Select Course" requests from students.
- Check course information from repository **D2:Course Details** (check class size, schedule).
- If valid, save the registration result to repository **D3:Student Records** and return a confirmation message.

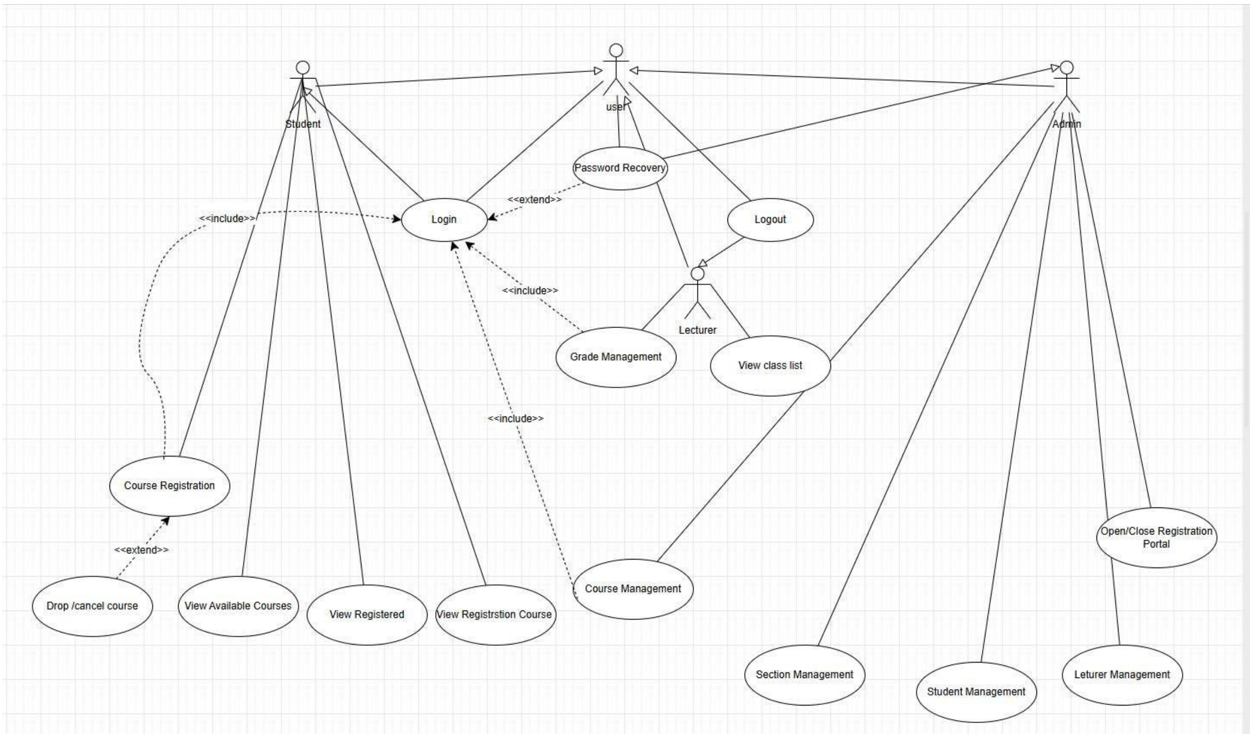
4.0 Grading (Score Processing):

- Receive grade data from instructors (Input Grades).
- Update grades in student records in the D3: Student Records database.
- Data Stores: The system uses MongoDB to organize the following data stores:
- **D1: User**: Stores user accounts, passwords (hash), and user roles.
- **D2: Course Details**: Stores course and section information.
- **D3: Student Records**: Stores registration history and grade reports.

VI. Use case Diagram

This section provides a high-level view of the system's functional requirements, illustrating the interactions between the actors (users) and the system's use cases.

6.1 Diagram Overview



6.2. Actors Description The system involves three primary actors:

- **Student:** Users who access the system to browse courses, register for classes, and view their academic schedules.
- **Lecturer:** Faculty members responsible for managing their assigned classes and inputting student grades.
- **Admin:** Superusers with full privileges to manage system data (courses, users) and control the registration timeline.

6.3. Use Case Specifications The following is a summary of key use cases depicted in the diagram:

A. General Use Cases (All Actors)

- **Login:** All users must authenticate before accessing specific features.
- **Password Recovery:** Extends the *Login* use case. Allows users to reset credentials via OTP if they forget their password.
- **Logout:** Ends the user session securely.

B. Student Use Cases

- **View Available Courses:** Browsing the list of open sections.
- **Course Registration:** The core function where students enroll in a class.

- **Drop/Cancel Course:** Extends *Course Registration*. Allows students to remove a course if the registration period is still open.
- **View Registered Courses:** Displays the student's current timetable.

C. Lecturer Use Cases

- **View Class List:** Displays students enrolled in a specific section.
- **Grade Management:** Allows lecturers to input or update grades for students.

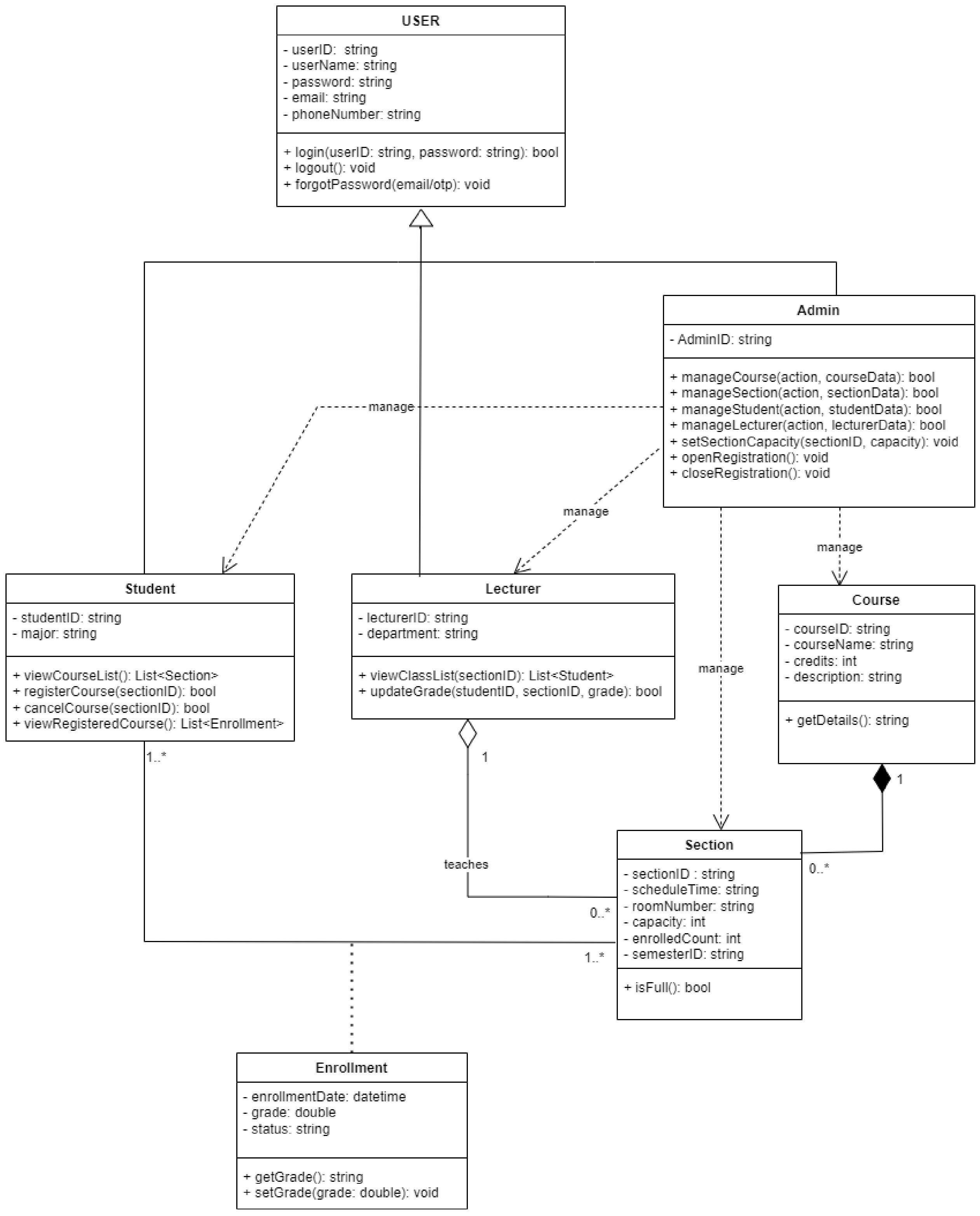
D. Admin Use Cases

- **Manage Courses & Sections:** Includes adding, editing, or deleting course details and setting class capacities.
- **Manage Users:** CRUD operations for Student and Lecturer accounts.
- **Open/Close Registration Portal:** Controls the global status of the registration period.

VII. Class Diagram

7.1. Overview

The Class Diagram describes the static logical structure of the Course Registration System, illustrating the main classes, their attributes, methods, and relationships. The system supports three primary user roles: **Student**, **Lecturer**, and **Admin**, each inheriting from a common abstract class **User**. The design follows object-oriented principles to ensure modularity, reusability, and maintainability.


USER
- userID: string
- userName: string
- password: string
- email: string
- phoneNumber: string
+ login(userID: string, password: string): bool
+ logout(): void
+ forgotPassword(email/otp): void
Admin
- AdminID: string
+ manageCourse(action, courseData): bool
+ manageSection(action, sectionData): bool
+ manageStudent(action, studentData): bool
+ manageLecturer(action, lecturerData): bool
+ setSectionCapacity(sectionID, capacity): void
+ openRegistration(): void
+ closeRegistration(): void
manage
manage
manage
manage
Student
- studentID: string
- major: string
+ viewCourseList(): List<Section>
+ registerCourse(sectionID): bool
+ cancelCourse(sectionID): bool
+ viewRegisteredCourse(): List<Enrollment>
Lecturer
- lecturerID: string
- department: string
+ viewClassList(sectionID): List<Student>
+ updateGrade(studentID, sectionID, grade): bool
Course
- courseID: string
- courseName: string
- credits: int
- description: string
+ getDetails(): string
1..*
1
1
teaches
0..*
1..*
0..*
Section
- sectionID : string
- scheduleTime: string
- roomNumber: string
- capacity: int
- enrolledCount: int
- semesterID: string
+ isFull(): bool
Enrollment
- enrollmentDate: datetime
- grade: double
- status: string
+ getGrade(): string
+ setGrade(grade: double): void

7.2. User Class Hierarchy

7.2.1. User (Abstract Class) The User class is an abstract superclass representing common information and behaviors shared by all system users.

Attributes:

- userId: String – Unique identifier for the user.
- userName: String – Full name of the user.
- email: String – Email address used for communication.
- password: String – Hashed user password.

Methods:

- login(userId, password): Boolean – Authenticates the user.
- logout(): Void – Logs the user out of the system.
- forgotPassword(): Void – Supports password recovery via Email or OTP.

7.2.2. Student Class Inherits from User, representing students who use the system to register for courses.

Attributes:

- studentID: String – Unique student identifier.
- major: String – Student's academic major.

Methods:

- viewCourseList(): List<Section> – Displays all open course sections.
- registerCourse(sectionID): Boolean – Registers the student for a section.
- dropCourse(sectionID): Boolean – Cancels a registered course.
- viewRegisteredCourses(): List<Enrollment> – Displays current timetable.

7.2.3. Lecturer Class Inherits from User, representing faculty members responsible for teaching and grading.

Attributes:

- lecturerID: String – Unique lecturer identifier.
- department: String – Academic department.

Methods:

- viewClassList(sectionID): List<Student> – Views students enrolled in a class.
- inputGrade(studentID, sectionID, grade): Boolean – Inputs or updates grades.

7.2.4. Admin Class Inherits from User, manages system configuration.

Attributes:

- adminId: String – Unique identifier.

Methods:

- manageCourse(), manageUsers(): Boolean – CRUD operations for entities.
- openRegistration(), closeRegistration(): Void – Controls the registration period.

7.3. Course and Section Management

7.3.1. Course Class Stores general information about academic courses (e.g., "Introduction to Programming").

- **Attributes:** courseId (String), courseName (String), credits (Integer).
- **Methods:** getDetails(): String.

7.3.2. Section Class Represents a specific instance of a course (e.g., "Intro to Programming - Monday Morning").

- **Attributes:** sectionID, scheduleTime, roomNumber, maxCapacity (Integer), currentEnrolled (Integer).
- **Methods:** isFull(): Boolean – Checks if capacity is reached.

7.4. Enrollment Class (Association Class) Records the Many-to-Many relationship between Student and Section.

Attributes:

- enrollmentDate: DateTime
- grade: Double
- status: String (Registered/Dropped)

Methods: updateGrade(newGrade): Void.

7.5. Class Relationships Summary

- **Inheritance:** Student, Lecturer, Admin inherit from User.
- **Composition:** A Course is composed of multiple Section instances (1 course has 0..* sections).
- **Association:** Admin manages Student, Lecturer, Course. Lecturer teaches Sections.
- **Association Class:** The relationship between Student and Section is managed by the Enrollment class to store specific data like Grades.

7.6. Conclusion The Class Diagram accurately models the functional requirements, effectively separating concerns between User Management, Course Management, and Academic Records. It serves as the blueprint for the backend logic implementation in Python.

VIII. Data Model

The system utilizes **MongoDB** as the primary database management system. The data is organized into four main collections: users, courses, registrations, and settings. Below is the detailed schema design represented in JSON format.

8.1 Collection: users

This collection stores profile and authentication data for all system actors (Students, Lecturers, and Admins). The role field determines the access permissions.

```
{

  "_id": ObjectId("659d1234..."),

  "id": "sv2021001",

  "password": "hashed_password_123",

  "name": "Nguyen Van A",

  "role": "student",
  "email": "sv2021001@uni.edu.vn",
  "phone": "0909123456",
  "dob": "2003-05-20",
  "department": "Software Engineering"
}
```

8.2 Collection: courses

This collection manages course sections. Each document represents a specific class section available for registration, including schedule and capacity details.

```
{
  "_id": ObjectId("659d5678..."),
  "id": "CS101",
  "name": "C++ Programming",
  "credits": 3,
  "slots": 50,
  "current": 45,
  "lecturer_id": "gv001",
  "room": "C201",
  "schedule": "Mon 07:30 - 11:30",
  "description": "Introduction to OOP with C++",
  "semester": "Spring 2026"
}
```

8.3 Collection: registrations

This collection functions as an associative entity to record student enrollments. It links a specific Student (users) to a specific Course (courses) and stores the academic result.

```
{
  "_id": ObjectId("659d9999..."),
  "student_id": "sv2021001",
  "course_id": "CS101",
  "enrollment_date": "2026-01-06",
  "grade": 8.5
}
```

8.4 Collection: settings

This collection stores system-wide configurations, used to control features like opening or closing the registration portal.

```
{
  "_id": ObjectId("659d0000..."),
  "key": "registration_open",
  "value": "1"
}
```

IX. Interface Design Description

This section provides a detailed visualization of the user interface for the Course Registration System. The design adheres to the usability requirements specified in Section 4.4, ensuring an intuitive, clean, and responsive experience for Students, Lecturers, and Admins.

9.1 Authentication Interfaces

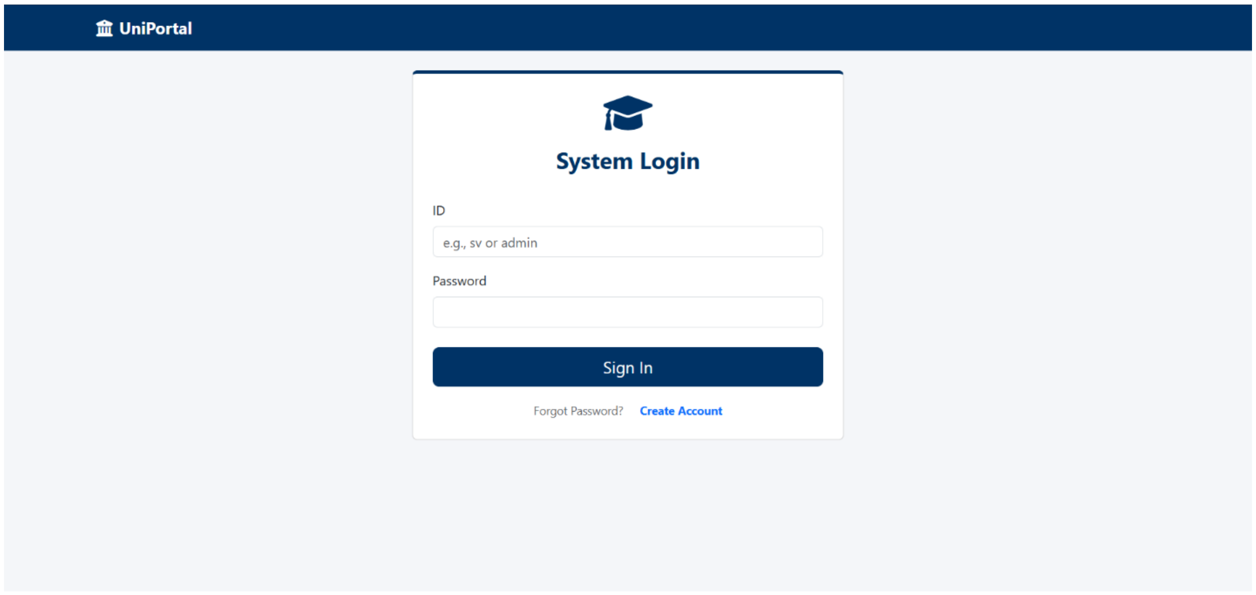
These interfaces are common to all users and serve as the entry point to the system.

9.1.1 System Login

The login screen is designed with a minimalist layout to minimize distraction. It requires users to authenticate using their unique ID and Password.

Key Elements:

- Input Fields: User ID (e.g., student ID or admin username) and Password.
- Action Button: "Sign In" triggers the authentication process (Functional Requirement 1.0).
- Navigation Links: Quick access to "Forgot Password?" and "Create Account" for account management.



9.1.2 Create Account

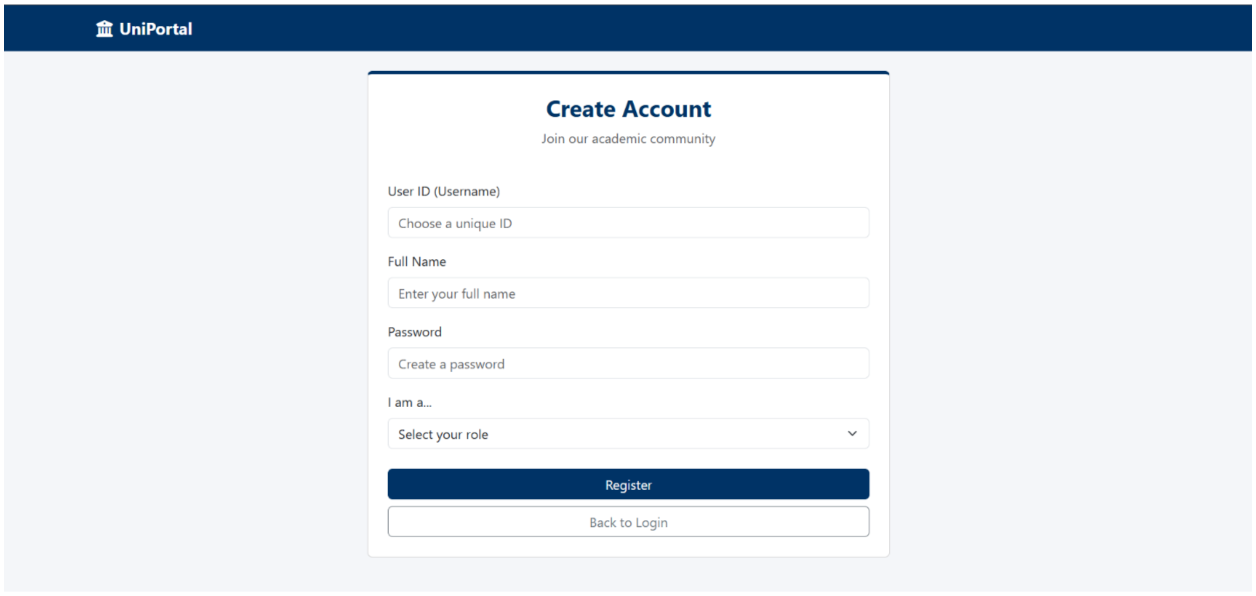
- This interface serves a critical role in system testing and data integrity. It acts as a validation checkpoint to ensure that any new entity entering the system (Student, Lecturer, or Admin) possesses all mandatory attributes defined in the Class Diagram (userId, userName, password, role, email, phoneNumber).

Purpose: To verify that the system correctly captures and instantiates a new User object with full information before committing data to the database.

Key Fields:

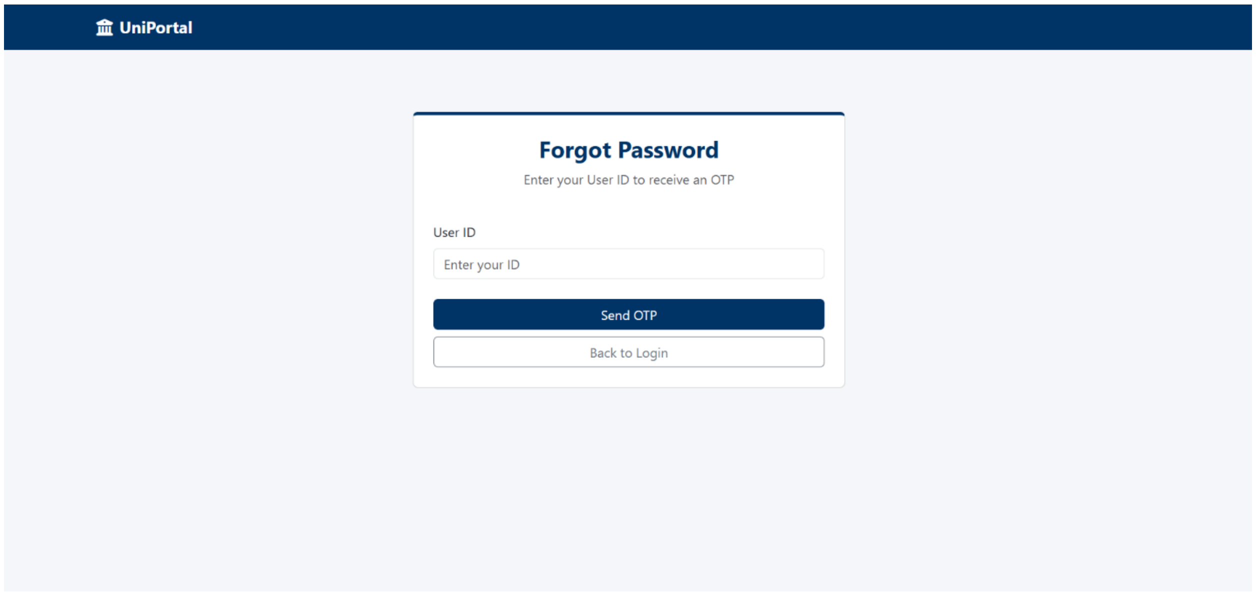
- User ID: Must be unique.
- Full Name: Display name for the dashboard.

- Password: Input for creating secure credentials.
- Email: User email address
- PhoneNumber: User phone number
- Role Selection: A dropdown menu to strictly categorize the user (Student/Lecturer/Admin), ensuring correct authorization levels upon login.



9.1.3 Forgot Password

- Designed to handle account recovery securely via OTP, reducing administrative overhead.
- Workflow: The user enters their User ID. The system validates the ID and sends an OTP to the registered email/phone.
- Key Elements: Clear instruction text ("Enter your User ID to receive an OTP") and a "Send OTP" button.



9.2. Student Interface (Dashboard and Registration)

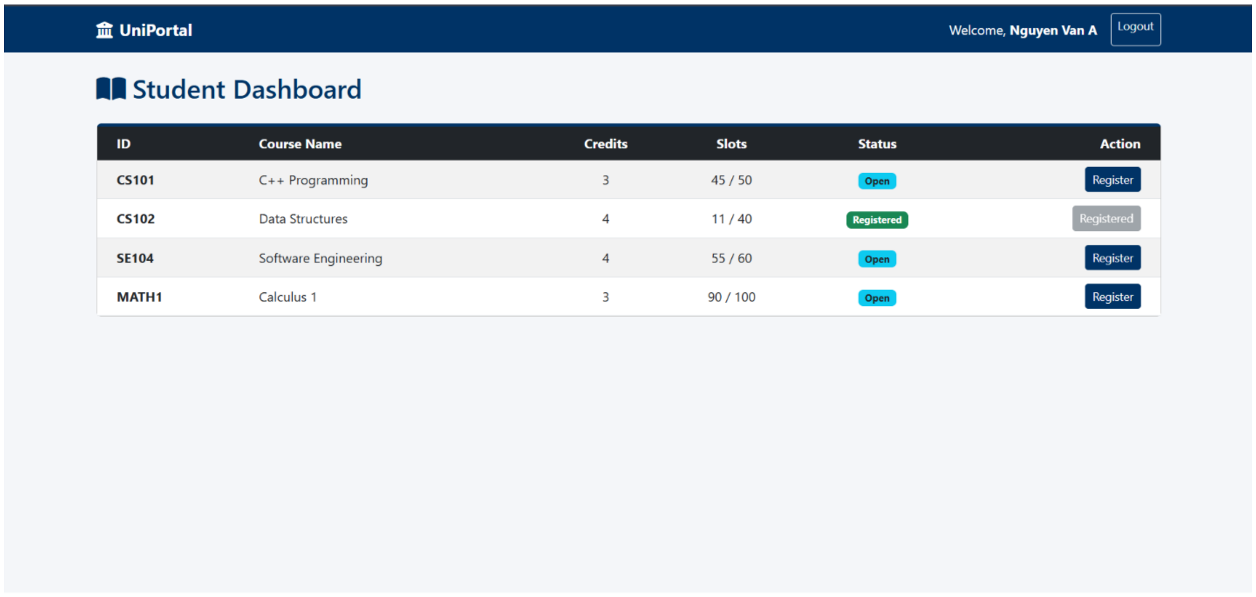
- Upon successful login, students are directed to the Student Dashboard. This interface aggregates course data and registration status into a single view.
- Student Dashboard
- Layout: A tabular view displaying the list of available courses (from the Course and Section classes).

Data Columns:

- ID & Course Name: clearly identifies the subject (e.g., CS101 - C++ Programming).
- Credits: Indicates academic weight.
- Slots: Real-time capacity tracking (Current Enrolled / Max Capacity).
- Status: Visual badges (e.g., "Open" in Blue, "Registered" in Green) provide immediate feedback on course availability.

Action Buttons:

- Register: Enabled for "Open" courses with available slots.
- Registered (Disabled): Indicates the student is already enrolled, preventing duplicate entries.
- (Note: A "Drop" button would appear for registered courses during the valid period).



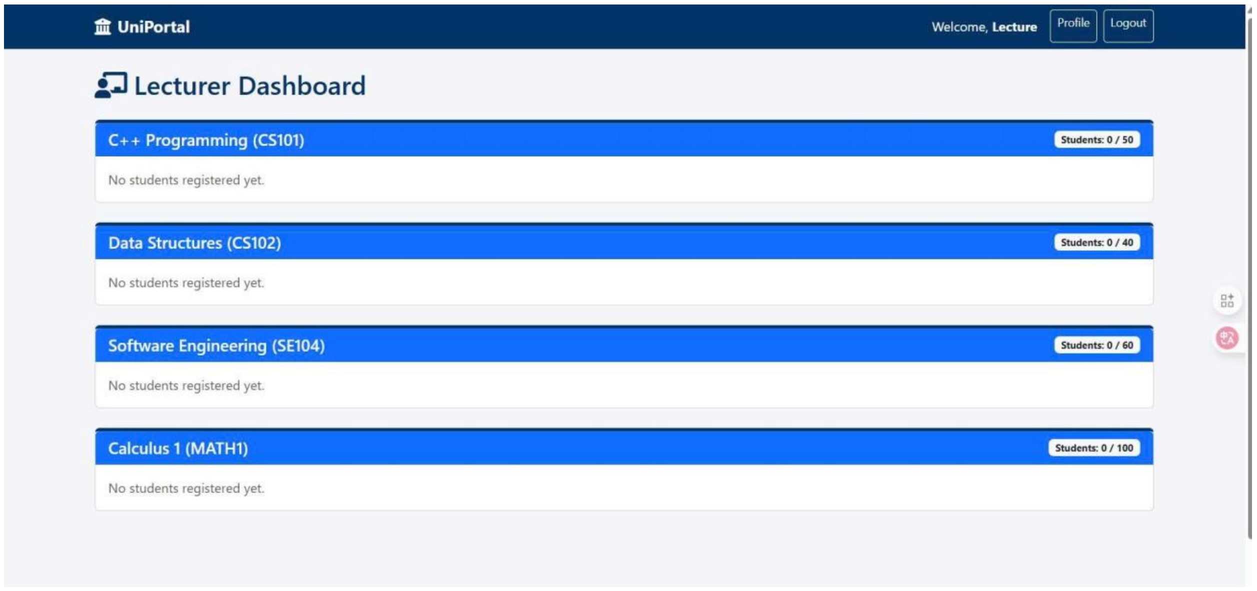
UniPortal Welcome, **Nguyen Van A** Logout

Student Dashboard

ID	Course Name	Credits	Slots	Status	Action
CS101	C++ Programming	3	45 / 50	Open	Register
CS102	Data Structures	4	11 / 40	Registered	Registered
SE104	Software Engineering	4	55 / 60	Open	Register
MATH1	Calculus 1	3	90 / 100	Open	Register

9.3. Lecturer Interface (Conceptual Design)

- Class List View: A dashboard similar to the Student view, but listing the classes the lecturer is teaching. Clicking a class expands the list of enrolled students.
- Grade Input Form: A data entry table allowing the lecturer to input or update numerical grades for each student ID. This interface connects directly to the Enrollment class in the Data Model to update the grade attribute.



UniPortal Welcome, **Lecture** Profile Logout

Lecturer Dashboard

C++ Programming (CS101) Students: 0 / 50

No students registered yet.

Data Structures (CS102) Students: 0 / 40

No students registered yet.

Software Engineering (SE104) Students: 0 / 60

No students registered yet.

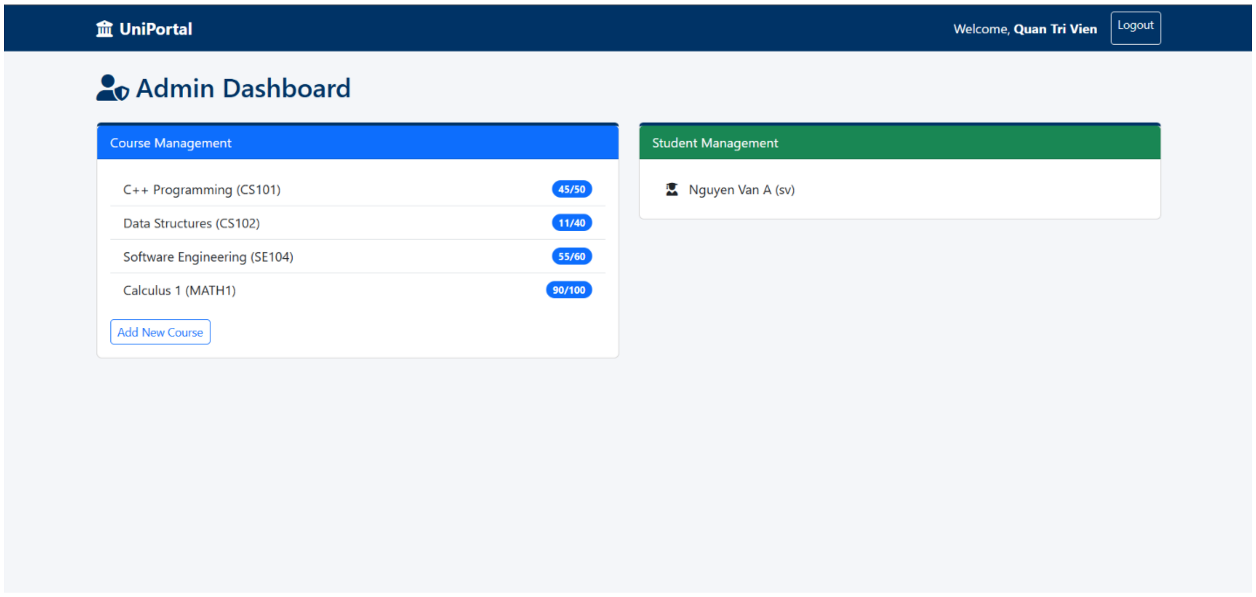
Calculus 1 (MATH1) Students: 0 / 100

No students registered yet.

9.4. Admin Interface (Conceptual Design)

- Management Dashboard: A sidebar navigation menu allowing access to "Course Management," "User Management," and "System Settings."

- Portal Control: A prominent toggle switch or button set to "Open Registration" or "Close Registration." This master control determines whether the "Register" buttons on the Student Dashboard are active or disabled.
- CRUD Forms: Detailed forms for adding new courses (Course ID, Name, Credits) and setting section capacities.



Admin Dashboard

Course Management

C++ Programming (CS101) 45/50
Data Structures (CS102) 11/40
Software Engineering (SE104) 55/60
Calculus 1 (MATH1) 90/100

Student Management

Nguyen Van A (sv)