College Database Management System Documentation

May 21, 2025

1 Introduction

The College Database Management System is a C++ application designed to manage academic records for a college, including students, courses, enrollments, faculty, projects, and user authentication. The system supports three user roles: Admin, Faculty, and Student, each with specific permissions. Key features include student and course management, course enrollment and grade updates, project assignments, and user authentication.

This documentation provides a comprehensive overview of the system's architecture, class hierarchy, functionalities, and instructions for cloning, building, and testing the application with custom data. It includes class hierarchy and architecture diagrams and detailed explanations for developers and users.

2 System Architecture

2.1 Class Hierarchy

The system is organized into several classes, with inheritance used for student types. Below are two diagrams: one for the inheritance hierarchy of the `Student` class and another for the system architecture showing associations managed by `CollegeDatabase`. The diagrams are scaled to fit within page bounds.

2.1.1 Student Inheritance Hierarchy

The `Student` class is an abstract base class with four derived classes representing different student types.

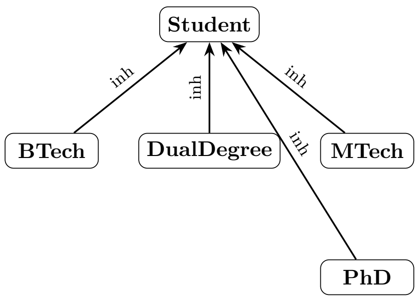

Figure 1: Student Class Inheritance Hierarchy

2.1.2 System Architecture

The `CollegeDatabase` class manages all entities, interacting with other classes through associations. The `UserManager` manages `User` objects for authentication.

Class Descriptions:

- **Student**: Abstract base class for students, with attributes `studentID`, `name`, `email`, `cgpa`, `projectGuide`, `assignedProject`, and `totalCredits`. Includes virtual methods `getStudentType()` and `display()`.

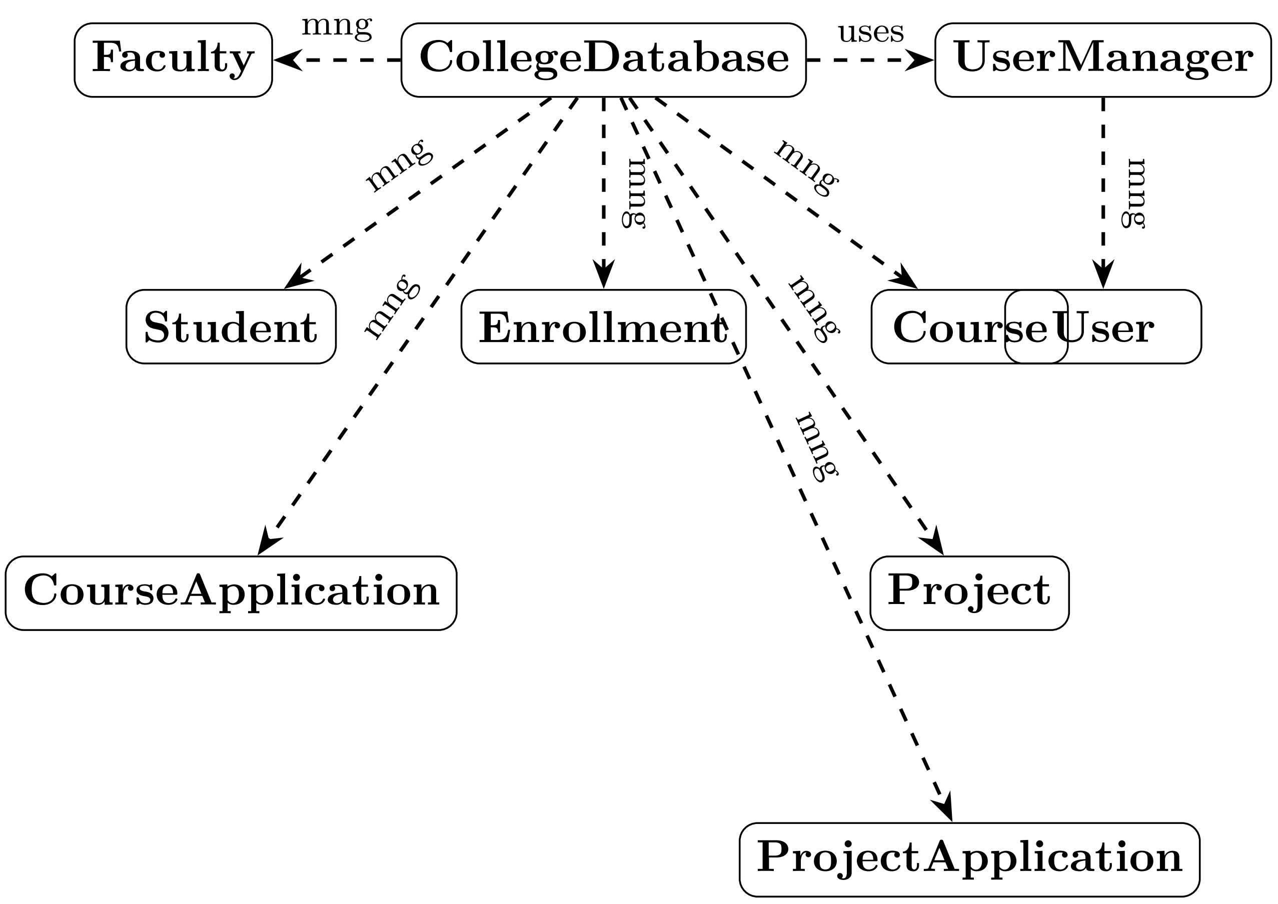

Figure 2: System Architecture and Associations

- **BTech**, **DualDegree**, **MTech**, **PhD**: Derived classes implementing specific student types, overriding `display()` to show type-specific information.
- **Course**: Represents a course with `courseID`, `name`, `credits`, `allocationType`, and `capacity`.
- **Enrollment**: Manages student-course enrollment with `studentID`, `courseID`, and `grade`.
- **CourseApplication**: Represents a student's application for a course with `studentID`, `courseID`, `priority`, and `applicationOrder`.
- **Faculty**: Represents a faculty member with `facultyID`, `name`, `isPermanent`, and lists of current/past courses and advisees.
- **Project**: Represents a project with `projectID`, `title`, `description`, `type`, `facultyID`, and `vacancies`.
- **ProjectApplication**: Represents a student's application for a project with `studentID`, `projectID`, and `applicationOrder`.
- **User**: Represents a user with `username`, `id`, `role`, and `password`.
- **UserManager**: Manages user authentication and sessions.
- **CollegeDatabase**: Central class managing all entities and operations, including vectors of students, courses, enrollments, etc.

2.2 Key Functionalities

The system supports the following functionalities, categorized by user role:

2.2.1 Admin Functionalities

- **User Management**:
  - Add a new user (`addUser`).
  - Log in/out (`login`, `logout`).
- **Student Management**:
  - Add a student (`addStudent`).
  - Update student details (`updateStudent`).
  - Delete a student (`deleteStudent`).
  - View all students (`displayStudents`).
- **Course Management**:
  - Add a course (`addCourse`).
  - Update course details (`updateCourse`).
  - Delete a course (`deleteCourse`).
  - View all courses (`displayCourses`).
  - Allocate courses to students (`allocateCourses`).
- **Enrollment Management**:
  - Enroll a student in a course (`enrollStudent`).
  - Update a student's grade, updating `totalCredits` and CGPA (`updateGrade`).
  - Delete an enrollment (`deleteEnrollment`).
  - View all enrollments (`displayEnrollments`).
- **Faculty Management**:
  - Add a faculty member (`addFaculty`).
  - Assign a course to a faculty (`assignCourseToFaculty`).
  - Assign a project advisee (`applyForProject`).
  - View all faculties (`displayFaculties`).
- **Project Management**:
  - Add a project (`addProject`).
  - Delete a project (`deleteProject`).
  - View all projects (`displayProjects`).
  - View all project applications (`displayProjectApplications`).
  - Assign projects to students (`assignProjects`).

2.2.2 Faculty Functionalities

- View all students, courses, enrollments, applications, faculties, projects, and project applications.
- Assign a course to themselves (`assignCourseToFaculty`).
- Move a course to past courses (`moveCourseToPast`).
- Update a student's grade for their assigned courses (`updateGrade`).
- Assign a project advisee (`applyForProject`).
- Add a project (`addProject`).

2.2.3 Student Functionalities

- View their information (`displayStudentInfo`).
- View all faculties, projects, and their enrolled courses (`displayStudentCourses`).
- Apply for a course (`applyForCourse`).
- Apply for a project (`applyForProject`).

Grade Update Logic: When a grade is updated (`updateGrade`), the student's `totalCredits` and CGPA are updated:

- If the enrollment has no prior grade, `temp = totalCredits * CGPA; temp += course.credits * grade; totalCredits += course.credits; CGPA = temp / totalCredits`, where grade values are S=10, A=9, B=8, C=7, D=6, E=5, P=4, F=0.
- If the enrollment has a prior grade, `temp = totalCredits * CGPA; temp -= course.credits * pastGrade; temp += course.credits * newGrade; CGPA = temp / totalCredits`

3 Setup and Installation

3.1 Prerequisites

- **Operating System**: Windows, Linux, or macOS.
- **Compiler**: C++17 compatible (e.g., GCC, Clang, MSVC).
- **CMake**: Version 3.10 or higher.
- **Git**: For cloning the repository.

3.2 Cloning the Repository

Clone the repository using:

```
git clone <repository_url>
cd college-database
```

3.3 Directory Structure

```
college-database/
  include/
     CollegeDatabase.hpp
     Course.hpp
     CourseApplication.hpp
     Enrollment.hpp
     Faculty.hpp
     Project.hpp
     ProjectApplication.hpp
     Student.hpp
     UserManager.hpp
  src/
     CollegeDatabase.cpp
     Course.cpp
     CourseApplication.cpp
     Enrollment.cpp
     Faculty.cpp
```

```
        Project.cpp
        ProjectApplication.cpp
        Student.cpp
        UserManager.cpp
    main.cpp
    CMakeLists.txt
```

3.4 Building the Project

1. Create a build directory:

```
mkdir build
cd build
```

2. Run CMake and build:

```
cmake ..
cmake --build .
```

3. The executable (`college_database` or `college_database.exe`) will be in the `build` or `build/Debug` directory.

4 Usage

4.1 Running the Application

Run the executable:

```
./college_database  # Linux/macOS
.\Debug\college_database.exe  # Windows
```

4.2 Initial Data

The `main.cpp` initializes the system with:

- **Users**: Admin (`admin/admin123`), Faculty (`smith/smith123`, `jones/jones123`), Students (`john/john123`, `jane/jane123`, `alice/alice123`, `bob/bob123`).
- **Students**: John (BTech, ID 1), Jane (DualDegree, ID 2), Alice (MTech, ID 3), Bob (PhD, ID 4).
- **Courses**: CS101 (3 credits), MATH201 (4 credits), PHYS301 (3 credits), CHEM401 (3 credits).
- **Enrollments**: John and Jane in CS101, Alice in MATH201, Bob in PHYS301.
- **Faculty**: Dr. Smith (ID 101, permanent), Dr. Jones (ID 102, non-permanent).
- **Projects**: BTP001, DDP001, RP001 (all by Dr. Smith).

4.3 Main Menu

The application starts with a login menu:

- Option 1: Log in with a username and password.
- Option 2: Exit.

After login, role-specific menus are displayed (Admin, Faculty, Student).

5 Testing with Custom Data

To test the system with your own data, modify `main.cpp` or use the Admin menu to add entities interactively.

5.1 Modifying `main.cpp`

Edit the initial data setup in `main.cpp` (lines after `CollegeDatabase db;`):

```
db.login("admin", "admin123");
db.addUser("newadmin", 999, Role::Admin, "newpass");
db.addStudent("BTech", 100, "New Student", "new@university.com", 8.0);
db.addCourse("CS999", "New Course", 4, AllocationType::CGPA, 10);
db.addFaculty(200, "Dr. New", true);
db.addProject("BTP002", "New Project", "Description", ProjectType::BTP, 200, 2);
db.enrollStudent(100, "CS999");
db.assignCourseToFaculty(200, "CS999");
db.logout();
```

Rebuild and run the application to test the new data.

5.2 Using the Admin Menu

Log in as `admin/admin123` and use the Admin menu options:

- Option 1: Add users.
- Option 2: Add students.
- Option 6: Add courses.
- Option 11: Enroll students.
- Option 12: Update grades.
- Option 16: Add faculty.
- Option 20: Add projects.

Example: Add a student and enroll them:

- Option 2: Enter type (`BTech`), ID (`100`), name (`Test Student`), email (`test@university.com`), CGPA (`8.0`).
- Option 11: Enroll student ID `100` in course `CS999`.
- Option 12: Update grade for student ID `100`, course `CS999`, grade `A`.

5.3 Testing Key Functionalities

- **Grade Update**:
  - As Admin: Use option 12 to set a grade (e.g., student ID 1, course CS101, grade A).
  - As Faculty: Log in as `smith`, use option 7 (ensure Dr. Smith is assigned to the course).
  - Verify: Log in as the student (e.g., `john`), use option 4 to check the grade, and option 1 to check CGPA and `totalCredits`.
  - Expected: For grade A in CS101 (3 credits), CGPA = 10, `totalCredits` = 3 (if new grade).
- **Course Allocation**: As Admin, use option 10 after adding course applications (option 3 as a student).
- **Project Assignment**: As Admin, use option 24 after adding project applications (option 6 as a student).

5.4 Verifying Grade and CGPA Updates

The `updateGrade` function updates `totalCredits` and CGPA:

- **New Grade**: Increments `totalCredits` by course credits.
- **CGPA Formula**: `temp = totalCredits * CGPA; temp += course.credits * grade; CGPA = temp / totalCredits`.
- **Grade Values**: A=10, A-=9, B=8, B-=7, C=6, C-=5, D=4, F=0.

Test by updating grades and checking student info:

```
Login as admin/admin123
Option 12: Student ID 1, Course CS101, Grade A
Login as john/john123
Option 1: Check CGPA (10), totalCredits (3)
Option 4: Check grade (A)
```

6 Troubleshooting

- **Permission Errors**: Ensure faculty are assigned to courses (Admin option 17). Admins bypass permission checks for `updateGrade`.
- **No Output**: Check console settings. Debug messages in `updateGrade` indicate execution steps.
- **Grades Not Updating**: Verify `Enrollment::setGrade` in `Enrollment.cpp`. Ensure enrollment exists (Admin option 14).
- **Build Errors**: Ensure C++17 support and all header/source files are present.
- **Diagram Bounds**: If diagrams still exceed margins, reduce `scalebox` to 0.7 or adjust `node distance` to 1cm.

7 Extending the System

To extend the system:

- Add new student types by creating derived classes from `Student`.
- Implement persistent storage (e.g., file I/O or database).
- Add new allocation types in `allocateCourses`.
- Enhance `updateGrade` to support grade history.

8 Conclusion

The College Database Management System provides a robust framework for managing academic records. By following the setup, usage, and testing instructions, users can customize and test the system with their data. The updated class hierarchy diagrams are now properly scaled, and the custom input mechanism allows flexible diagram design.