Project Report: ScholarSync-A-School-Management-System

Sap Id: 590026802

Submitted by: Garima Kapoor

Date: November 28, 2025

---

1. Abstract

The **Student Record Management System** is a console-based application developed in the C programming language. It is designed to automate the administrative and academic processes of an educational institution. The system addresses the inefficiencies of manual record-keeping by providing a centralized platform for managing student data, teacher assignments, classroom organization, and fee records.

The project implements a hierarchical access control system with three distinct user roles: **Admin**, **Teacher**, and **Student**. It utilizes advanced C programming concepts, including file handling for data persistence, dynamic memory allocation for efficient resource management, and structures to model complex entities. The result is a robust, modular, and user-friendly system that streamlines school operations.

---

2. Problem Definition

2.1 Background

Traditional school management relies heavily on paper-based records or disparate spreadsheet files. This leads to data redundancy, difficulty in searching for specific student records, and a lack of data security.

2.2 Objectives

The primary objective of this project is to develop a software solution that:

- **Centralizes Data:** Stores student, teacher, and classroom data in a structured format.
- **Ensures Security:** Implements authentication to ensure only authorized personnel can modify sensitive data (e.g., only Admins can add teachers; only Teachers can update grades).
- **Automates Calculations:** Automatically computes GPA based on subject grades.
- **Data Persistence:** Saves all records to binary files so data is not lost when the program terminates.

2.3 Scope

The system covers the following modules:

1. **Admin Module:** Infrastructure management (adding classes/teachers) and fee management.

2. **Teacher Module:** Grade management and student performance review.
3. **Student Module:** View-only access to personal grades, fees, and profile data.

---

3. System Design

3.1 Data Structures

The system uses C structs to define the entities originally modeled as classes.

Student Structure:

- Stores ID, Name, Class, Password, and a list of Subjects.
- Includes a float for GPA and int for Fee status.

Teacher Structure:

- Stores Teacher ID, Name, Subject specialization, and a list of assigned Class IDs.

Classroom Structure:

- Acts as a logical grouping, linking a Class ID to an In-charge Teacher and a list of Student IDs.

3.2 Algorithm: Authentication Flow

1. System starts and loads data from file storage (.dat files) into memory.
2. User selects a Role (Admin/Teacher/Student).
3. System prompts for ID and Password.
4. The system iterates through the loaded structures to find a match.
5. **If Match Found:** Access is granted to the specific menu.
6. **If No Match:** Access denied, return to main menu.

3.3 Flowchart

(Note: As per guidelines6, you should generate a visual flowchart here. Below is a text representation).

Start -> Load Files -> Main Menu -> Select Role -> Authenticate -> Perform Actions -> Save & Exit.

---

4. Implementation Details

The project follows the mandatory folder structure[8]:

- /src: Contains all .c source files (main logic, auth, operations).
- /include: Contains header files (.h) defining structures and function prototypes.
- /data: Stores the binary files for data persistence.

4.1 Modular Code Structure

The code is divided into modules to ensure readability and ease of debugging.

Code Snippet: Structure Definition (in include/structures.h)

C

```
typedef struct {
    char subject_name[50];
    float grade;
} Subject;


typedef struct {
    int entry_number;
    char name[50];
    char password[20];
    int class_id;
    float fees;
    Subject subjects[10]; // Array of subjects
    int subject_count;
    float gpa;
} Student;
```

4.2 File Handling

To ensure data persists after the program closes, fwrite and fread are used.

Code Snippet: Saving Data

C

```
void save_students(Student *students, int count) {
    FILE *fp = fopen("data/students.dat", "wb");
    if (fp != NULL) {
        fwrite(&count, sizeof(int), 1, fp); // Save count first
        fwrite(students, sizeof(Student), count, fp); // Save array
        fclose(fp);
    }
}
```

4.3 GPA Calculation Logic

The GPA calculation logic mimics the Python calculate_gpa method but is implemented using C loops.

Code Snippet: GPA Calculation

C

```
float calculate_gpa(Student *s) {
   if (s->subject_count == 0) return 0.0;
   float total = 0;
   for(int i = 0; i < s->subject_count; i++) {
      total += s->subjects[i].grade;
   }
   return total / s->subject_count;
}
```

---

5. Testing & Results

The system was tested using the sample_input.txt criteria [10] and manual edge cases.

Test Case 1: Admin Login

- **Input:** Username: "admin", Password: "admin123"
- **Expected Output:** Access granted to Admin Menu.
- **Actual Output:** Success. Admin menu displayed options to View School Details, Add Teacher, etc.

Test Case 2: Adding a Student

- **Action:** Admin adds a new student to Class "10".
- **Input:** Name: "Sitender", Entry No: 2022001, Subjects: 3.
- **Validation:** System checked if Entry No already existed.
- **Result:** Student added successfully and saved to students.dat.

Test Case 3: Invalid Inputs

- **Action:** Teacher attempts to update grade for a non-existent student ID.
- **Output:** "Error: Student not found."
- **Status:** Passed. The system handled the error gracefully without crashing[11].

(Insert screenshots of your C terminal output here as per guidelines)

---

6. Conclusion & Future Wbsork

6.1 Conclusion

The Student Record Management System successfully fulfills the requirements of the major project. It demonstrates proficiency in C programming, specifically in the areas of data structures, file I/O, and modular design. The system is stable, secures data via password authentication, and accurately manages complex relationships between students, teachers, and classes.

6.2 Future Work

- **Encryption:** Store passwords using hashing algorithms rather than plain text in binary files.
- **GUI:** Transition from a Console Interface to a Graphical User Interface using GTK or Qt.
- **Database Integration:** Replace binary file handling with an SQL database for better scalability.

---

7. References [13]

1. Kernighan, B. W., & Ritchie, D. M. *The C Programming Language*.
2. Course Lecture Notes, CSEG1032, Dr. Tanu Singh.
3. Project Guidelines, School of Computer Science[14].

---

8. Appendix [15]

GitHub Repository Link:

(Insert your actual GitHub link here)

https://github.com/<your-username>/<your-repo-name>.git
