**Project Report**

**Student Management System**

**Fundamentals of Data Science**



**Submitted by:**

**Name:** Pratyush Shakya (24071230),

Prashish Sapkota (24071222),

Nabin Karki (24071149)

**Section:** C

**Date:** 05/01/2025

**Contents**

[Introduction 2](#_Toc195646704)

[Objective 3](#_Toc195646705)

[System Design 4](#_Toc195646706)

[Implementation 6](#_Toc195646707)

[Sample Output 8](#_Toc195646708)

[Testing 14](#_Toc195646709)

[Challenges Faced 15](#_Toc195646710)

[Conclusion 16](#_Toc195646711)

# Introduction

The Student Profile Management System is designed to handle student personal information, examination grades, and extracurricular activities through a role-based login system. The system differentiates between admin and student and gives the data accordingly. It provides access for admins for all the data of students, like deleting a student, adding a student, and maintaining grading, viewing analytics where whereas students can only update their profile, view personal details, view grades, and view extracurricular activities.

File structure:

1. Users.txt
2. Grades.txt
3. Eca.txt
4. Passwords.txt

# Objective

1. Student data management (Personal details, grades, extracurricular activities)
2. Role-Based Access Control

* Admin can add, edit, or delete student records, and analyze academic performance.
* Student can update personal information (e.g., view their grades and activities.

1. Secure and Scalable system

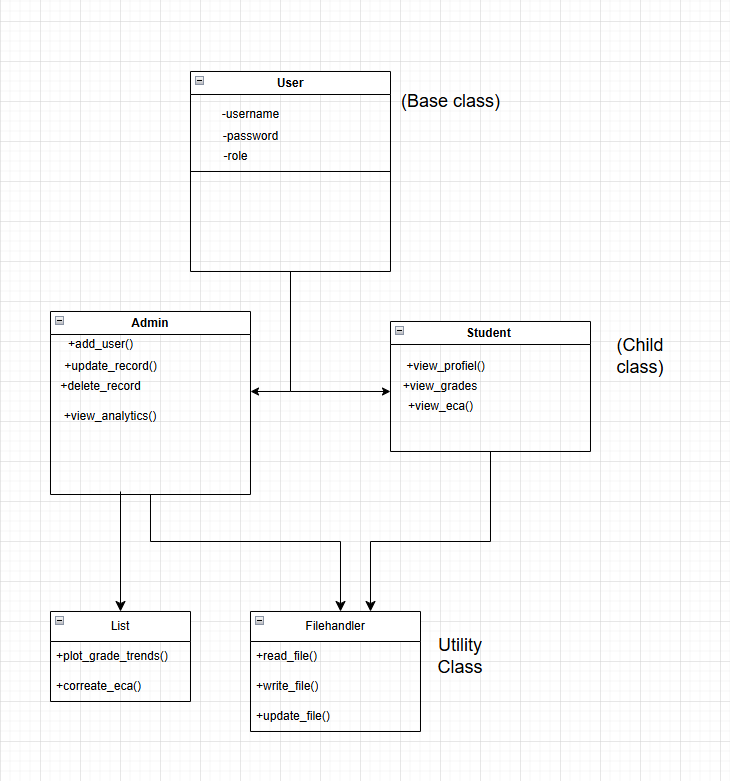
* Login authentication if it is viewed by an admin or a student.
* File-based storage for simplicity and learning purposes.
* Modular design using OOP to allow future upgrades.

1. User-Friendly Interface

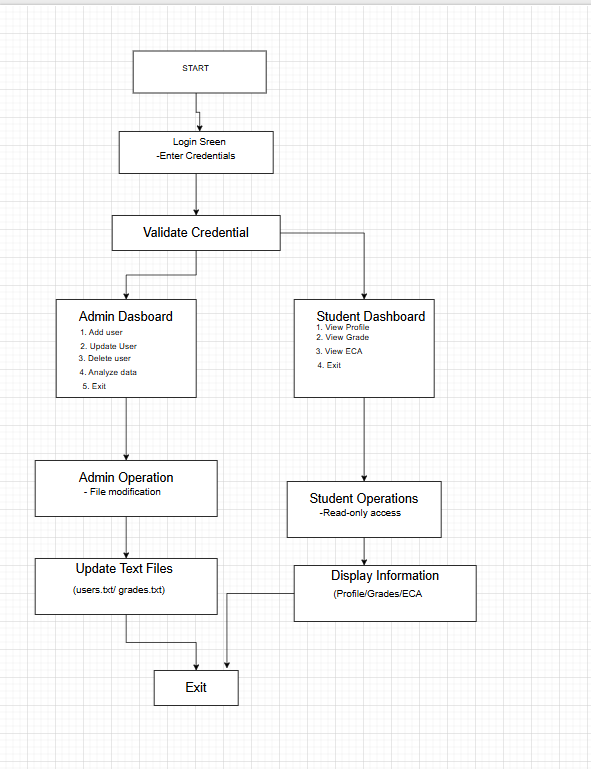
* Menu-driven navigation in the terminal for ease of use.
* Clear error messages.

# System Design

Class diagram



System Flow



# Implementation

**project/**

**├── main.py # Entry point**

**├── auth.py # Authentication module**

**├── admin.py # Admin operations**

**├── student.py # Student operations**

**├── file\_handler.py # File operations**

**├── users.txt # User records**

**├── grades.txt # Grade records**

**├── eca.txt # Activity records**

**└── passwords.txt # Login credentials**

All four features of OOP, including file handling, are implemented in this program.

1. **Classes and Objects:**

* User (Base class)

It contains different attributes like user\_id, name, and role, and it is also known as a parent class for **admin** and **student.**

* Admin (Derived Class)

Inherits from **User,** and different methods are implemented like **add\_user(), update\_grades(),** and **delete\_user().**

* Student (Derived class)

It is also inherited from User and provides student-specific methods (**view\_grades(),** **update\_profile())**

1. **Inheritance**

* **Admin** and **Student** inherit from **User**, reducing code duplication.

1. **Encapsulation:**

* Private methods **(\_load\_data())** handle file operations internally.
* Public methods (**view\_grades(), update\_profile()**) expose only necessary functionality.

1. **Polymorphism:**

* **Load\_user()** returns either an Admin or Student object based on role.

1. **File handling:**

* Text files (users.txt, grades.txt, eca.txt, and passwords.txt) store data.
* FileHandler class provides reusable methods: **read\_file(),** **write\_file()** and **update\_file().**

1. **Role-Based Access Control:**

* Admin Features: add, update, delete student records, and generate average grades.
* Student Features: View personal grades, ECA.

1. **Exception Handling:**

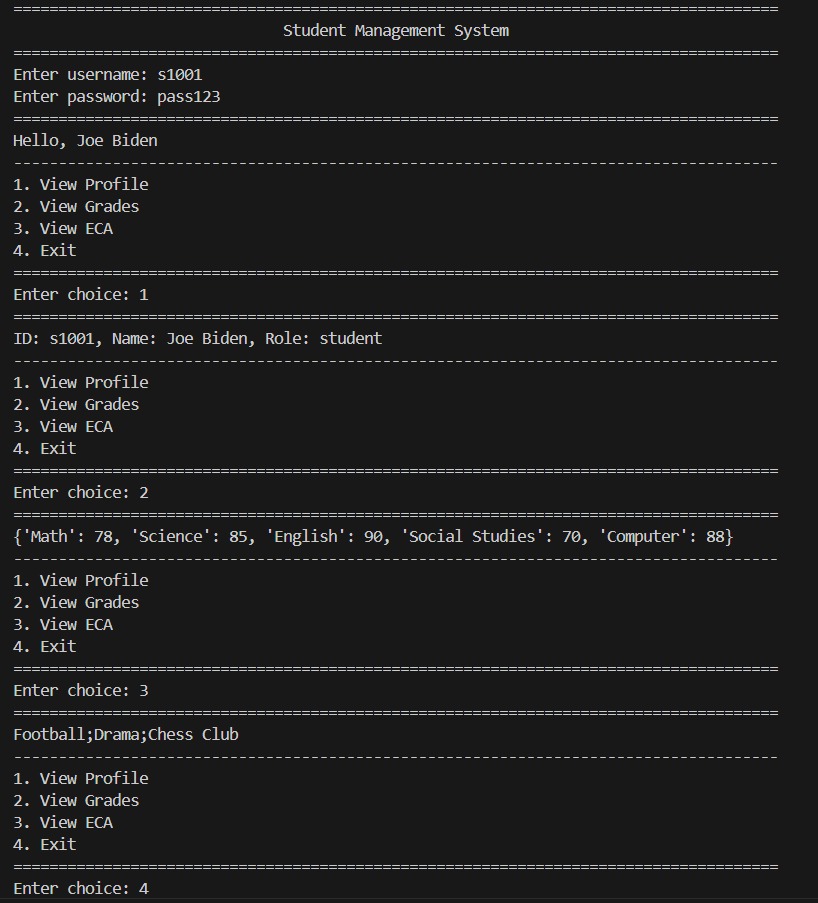
* File operations (try-except blocks) prevent crashes if any file is missing.
* Input Validation ensures correct data entry (e.g., grades must be 0-100).

1. **Modular Design:**

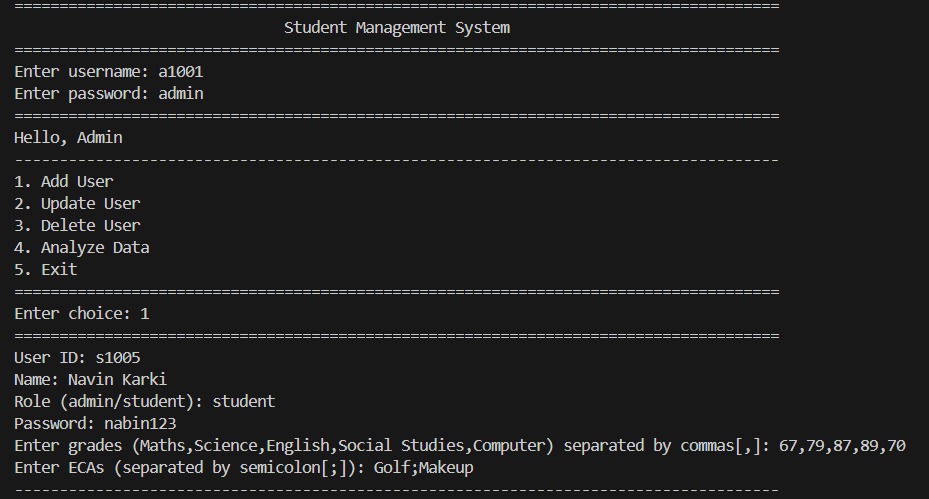
* Separate modules (auth.py, admin.py, student.py) improve maintainability.

# Sample Output

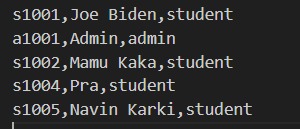
Output to view the profile of the student, including the **username** and **password** entered by the student. Therefore, students can view grades, view their profile, and view ECA.



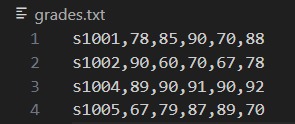
Output as admin so that the admin can add users, update users, delete users, analyze data, and Exit.



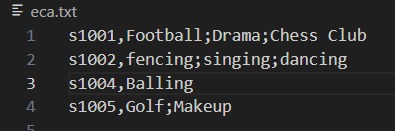
User.txt takes the userid, name, and role.



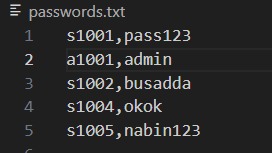
Grades.txt takes the userid, grades of math, science, English, social studies, and computer.



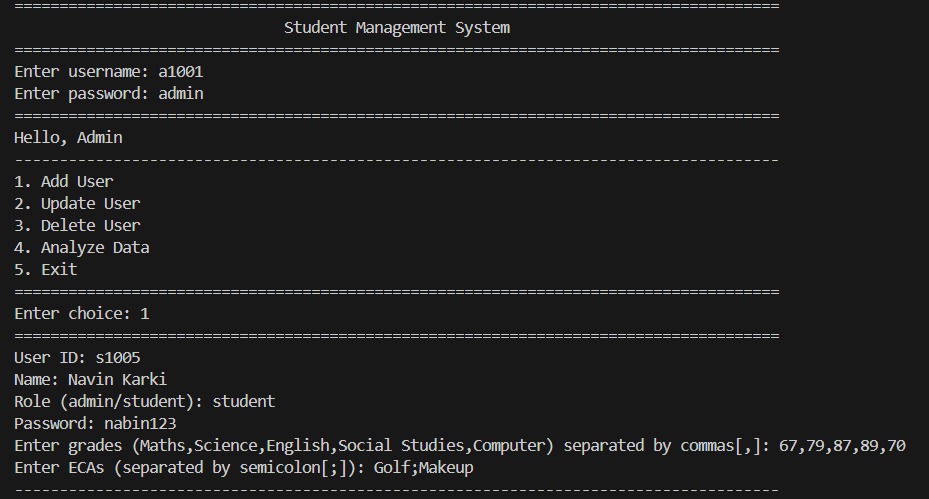
Eca.txt takes the userid and takes the data from activities.



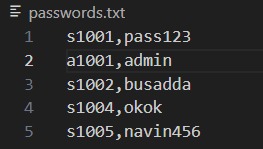
Passwords.txt takes the userid and password.

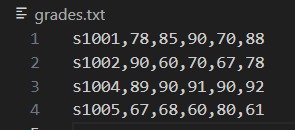


Update done by admin

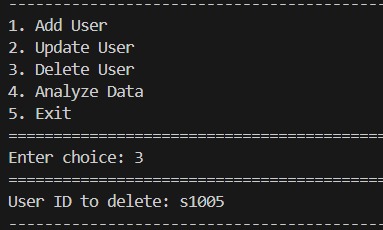


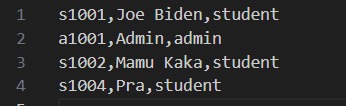
Update the new grades and the new password of Navin Karki.



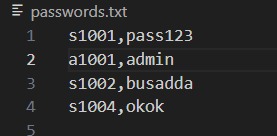


Deleting the data of a student by using the ID, and if it matches the ID, then it deletes all the data related to the student.

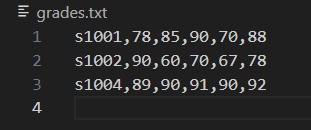




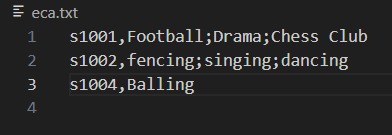
After deleting, the password and userid of **Navin Karki** get deleted.



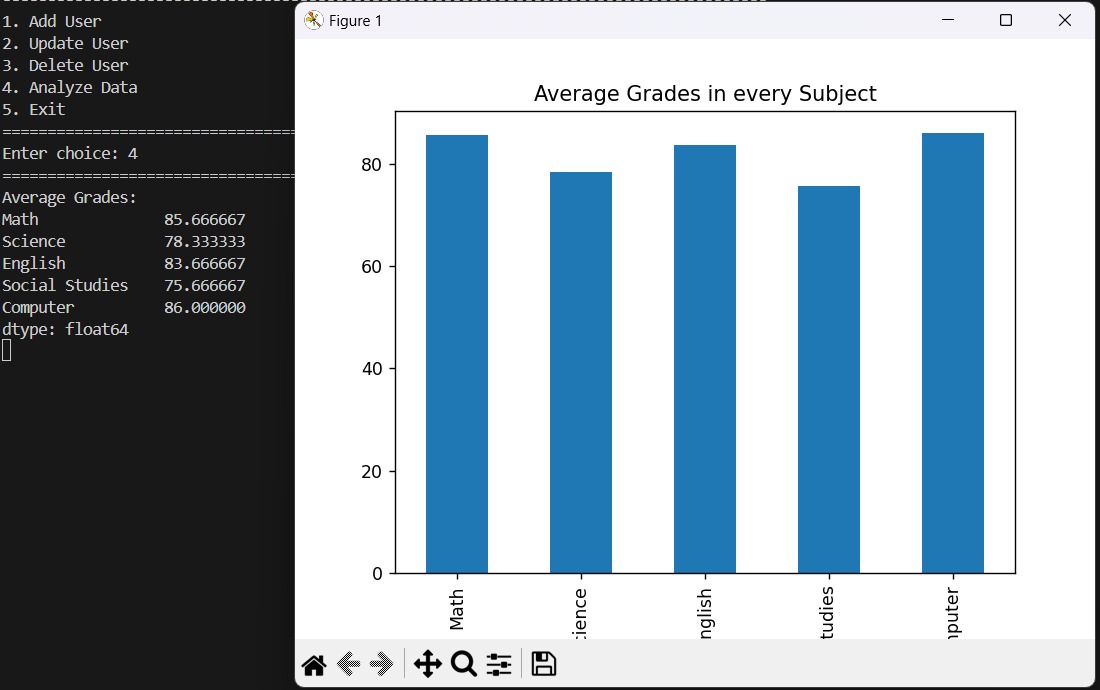
In **grades.txt**, the userid and grades get deleted.



In eca.txt, the userid and activities of a specific student get deleted.



Analyzing the average grades of total students



# Testing

**Unit testing:**

* Checked **User.viewprofile()** displays info correctly (e.g., ID, Name, Role).
* Verified **Student.viewgrades()** returns correct subjects and grades.
* Confirmed **Student.vieweca()** shows the right activities.

**Integration testing**

* Testing admin functions (add, update, delete) – all data updated properly.
* Made sure the login system correctly identifies admin and student roles.
* Verified data analysis plots display without errors.

**User Acceptance Testing:**

* Adding a student (e.g., s1004, Prashish Sapkota) – data saved correctly.
* Tested student dashboard – profile, grades, and ECA show as expected.
* Checked bar and scatter plots- clear and accurate.

# Challenges Faced

**File-Based Storage:**

* Risk of data corruption from multiple file accesses.
* Needed strong error handling for file read/write issues.

**Data Consistency:**

* Keeping all files (e.g., users.txt, grades.txt) in sync during updates/deletions was tricky.
* Created updatefile() and deletefile() functions to manage consistency.

S**calability:**

* Text files are not efficient for handling large amounts of data.
* Performance may slow down with many records.

**Security:**

* Passwords are stored in plaintext (passwords.txt), which is unsafe.
* Minimal input validation could lead to incorrect or bad data being saved.

# Conclusion

Hence, using OOP principles like inheritance, encapsulation, polymorphism, file handling for persistent storage, modular design, error handling, and role-based access control helps to build a student management system that demonstrates core concepts of programming and practical software development. Furthermore, we can extend the database and GUI for real-world use.