Final Assessment – Project Work

Fundamentals of Data Science

Group Project

Student Profile Management System

Submitted By:

Raunak Kr. Singh

Nishchal Ranjitkar

Shirish Maharjan

The British college, Kathmandu

Cybersecurity and Digital Forensics

Kathmandu, Nepal

Date: 15th April,2025

Final Assessment – Project Work

Student Profile Management System – Report

Student Profile Management System - Overview

1. Introduction

The Student Profile Management System is a Python program designed to manage user registrations, student and admin functionalities, grades, extracurricular activities (ECA), and more. The system consists of classes, functions, and methods to perform various operations related to user management and data handling.

2. Program Components

The program consists of several key components:

a. Classes

1. **User Class**: Represents a basic user with attributes like username, role, email, password, phone number, grade, and ECA.
2. **Admin Class**: Inherits from User class and adds admin-specific functionalities like user registration, user search, grade management, ECA management, etc.
3. **Student Class**: Inherits from User class and represents a student with additional attributes like grades and ECAs.

b. Functions

1. **create_empty_files()**: Creates empty files for users, grades, ECAs, and students if they don't exist.
2. **login()**: Handles user login functionality for both admins and students.
3. **modify_student_record()**: Allows admins to modify student records, including adding grades, ECAs, and updating profiles.
4. **display_student_info()**: Displays detailed information about a student.
5. **admin_actions()**: Handles actions performed by an admin, such as user registration, search, modification, deletion, viewing users, etc.
6. **display_users_txt()**: Displays user information from the users.txt file.
7. **display_users_csv()**: Displays user information from the users.csv file.
8. **main()**: Entry point of the program, orchestrates the flow of the system.

3. Program Flow

1. **Initialization**: The program initializes required modules like colorama, Figlet for ASCII art, and sets up necessary configurations.
2. **Class Definitions**: Defines the User, Admin, and Student classes with respective attributes and methods.
3. **File Handling**: Implements file handling operations like creating empty files, reading user data from files, saving user data, etc.

4. **User Registration/Login**: Allows users (admins or students) to register, log in, and authenticate.
5. **Admin Functionalities**: Admins can perform actions like registering users, searching for users, modifying student records, deleting users, viewing user lists, etc.
6. **Student Functionalities**: Students can view their profile, update details, view ECAs, view grades, and perform other related actions.
7. **Dashboard Integration**: Integrates with a dashboard for admin functionalities.
8. **User Interface**: Provides a user-friendly interface using colorama for coloured output and ASCII art for aesthetic appeal.
9. **Error Handling**: Implements error handling mechanisms to catch and display errors during program execution.

4. Additional Notes

- The program uses external libraries like colorama, matplotlib, pyfiglet, and prettytable for enhanced functionality and visual presentation.
- It employs object-oriented programming principles for better code organization, reusability, and scalability.
- Various input validation checks are implemented to ensure data integrity and prevent runtime errors.
- The program offers detailed feedback and status messages using colored output for better user experience.

5. Conclusion

The Student Profile Management System is a comprehensive Python program that effectively manages user registrations, student records, grades, ECAs, and admin functionalities. It provides a robust framework for educational institutions to streamline their management processes.

Student Profile Management System – Detailed Report

```python
import pickle
from colorama import init, Fore, Style  # Import colorama for colored output
from matplotlib.pyplot import table
from pyfiglet import Figlet  # Import Figlet for ASCII art
from prettytable import PrettyTable  # Import PrettyTable library for tabular data
import os  # Import os module for file operations
import uuid  # Import uuid module for generating unique IDs
import csv  # Import CSV module for file operations
import sys

init(autoreset=True)  # Initialize colorama for auto-resetting color styles
f = Figlet(font='slant')  # Initialize Figlet with 'slant' font for ASCII art text
```

A. Import Function:

1. **pickle**: Serialize and deserialize Python objects for data persistence and transfer.
2. **colorama**: Simplify colored text output in the terminal with cross-platform support.
3. **matplotlib.pyplot.table**: Create tables within Matplotlib plots (not used directly in the provided code snippet).
4. **pyfiglet**: Generate ASCII art text using various font styles for decorative text.
5. **prettytable**: Create visually appealing ASCII tables for displaying structured data.
6. **os**: Perform file operations and interact with the operating system.
7. **uuid**: Generate universally unique identifiers (UUIDs) for creating unique IDs.
8. **csv**: Read and write CSV files, parse CSV data into Python data structures.
9. **sys**: Access system-specific variables and functions, handle program exit and command-line arguments.

These modules and functions collectively provide functionalities for file operations, data serialization, coloured output, ASCII art generation, table creation, unique ID generation, CSV file handling, and system-specific operations in your program.

B. Class User (Parent Class):

The provided code defines a Python class named User that represents a user in a system. Let's describe and comment on each function in the code:

__init__ Method:

Description: This method is the constructor for the User class. It initializes the attributes of a user object when an instance of the class is created.
Parameters:
username: The username of the user.
role: The role of the user (e.g., admin, student).
email: The email address of the user.
password: The password of the user.
phone_number: The phone number of the user.
grade: The grade of the user.
eca: The extracurricular activities (ECA) of the user.
Comments: Each parameter passed to the constructor is assigned to the corresponding attribute of the user object (self).
display_user_info Method:

Description: This method prints the user information, including username, role, email, student ID, phone number, grade, and ECA.
Parameters: None (uses attributes of the user object).
Comments: It checks if the user has a grade and ECA before printing them.
create_empty_files Method:

Description: This method creates empty files (if they don't exist) for storing user data, grades, ECAs, and student information.
Parameters: None.
Comments: It iterates through a list of file names and checks if each file exists. If a file doesn't exist, it creates an empty file using open with write mode ('w').

Overall, the User class encapsulates user-related functionalities such as initialization, displaying user information, and creating empty files for data storage.

Please note that the code snippet references self.student_id in the display_user_info method, but student_id is not an attribute defined in the __init__ method. If student_id is intended to be used, it should be added as an attribute in the __init__ method.

C. Class Admin(User):

Admin that inherits from the **User** class. The **Admin** class includes methods for registering users, viewing user data, finding users, adding grades, deleting users, saving ECA and grade details, managing student records, and logging in as an admin.

a. __init__:

The __init__ method initializes the attributes of an Admin object by calling the constructor of the superclass (User class).
It sets up the initial state of an Admin object with the provided parameters.
The print("Admin Initialized") statement is for informational purposes, indicating that an Admin object has been successfully initialized.

b. Def Register_user:

```python
def register_user(self, username, role, email, password, phone_number, grades=None, eca=None):
    try:
        # Generate a unique user ID
        new_user_id = str(uuid.uuid4())[:8]

        # Create a list with user details
        new_user = [new_user_id, username, role, email, password, phone_number]

        # Add grades and ECA if provided
        if grades is not None:
            new_user.append(','.join(map(str, grades)))
        else:
            new_user.append('')
        if eca is not None:
            new_user.append(','.join(map(str, eca)))
        else:
            new_user.append('')

        # Append the user details to a CSV file (e.g., users.csv)
        with open("users.csv", "a") as file:
            writer = csv.writer(file)
            writer.writerow(new_user)
        print(Fore.GREEN + f"User registered successfully with ID: {new_user_id}")
        return new_user_id
    except Exception as e:
        print(Fore.RED + f"Error registering user: {e}")

```

This **register_user** method in the provided code snippet is part of a class, likely a subclass of the **User** class, such as **Admin**. Below is a description of the program:

1. **Method Signature**:
    - Name: **register_user**
    - Parameters:
        - **self**: Represents the instance of the class.
        - **username**: Username of the user being registered.
        - **role**: Role of the user (e.g., admin, student).
        - **email**: Email address of the user.
        - **password**: Password for the user account.
        - **phone_number**: Phone number of the user.
        - **grades=None**: Optional parameter representing the grades of the user.
        - **eca=None**: Optional parameter representing extracurricular activities (ECA) of the user.
2. **Functionality**:
    - Generates a unique user ID using **uuid.uuid4()** and truncates it to 8 characters.
    - Creates a list **new_user** containing user details such as ID, username, role, email, password, and phone number.
    - Appends grades and extracurricular activities to **new_user** if provided.
    - Writes the **new_user** details to a CSV file named "users.csv".
    - Prints a success message with the user's ID if registration is successful.
    - Handles exceptions and prints error messages if any issues occur during registration.
3. **Major content of this function**:
    - **str(uuid.uuid4())[:8]**: Generates a unique user ID using UUID (Universally Unique Identifier) and truncates it to 8 characters.
    - **new_user**: Represents a list containing user details to be written to the CSV file.
    - **if grades is not None** and **if eca is not None**: Checks if grades and extracurricular activities are provided before appending them to **new_user**.
    - Writing to CSV: Uses the **csv.writer** to write the **new_user** details to a CSV file in append mode.
    - Exception Handling: Wraps the registration process in a try-except block to handle any potential errors and print corresponding error messages.
    - Success Message: Prints a success message with the user's ID if registration is successful.

    Overall, this method facilitates the registration of a new user by collecting their details, generating a unique ID, and saving the information to a CSV file. It provides flexibility by allowing the inclusion of grades and extracurricular activities if available.

c. Def View_user:

```
def view_users(self, search_term=None):
    with open('users.txt', 'r') as file:
        lines = file.readlines()

    if search_term:
        found_users = []
        for line in lines:
            user_data = line.strip().split(',')
            if search_term.lower() in user_data[0].lower():  # Check if search_term is in username
                found_users.append(user_data)
        if not found_users:
            print("No matching users found.")
        else:
            print("Matching Users:")
            table = PrettyTable(['Username', 'Role', 'Email', 'Password', 'Hash', 'Phone Number'])
            for user in found_users:
                table.add_row(user)
            print(table)
    else:
        if not lines:
            print("No users found.")
        else:
            print("All Users:")
            table = PrettyTable(['Username', 'Role', 'Email', 'Password', 'Hash', 'Phone Number'])
            for line in lines:
                user_data = line.strip().split(',')
                table.add_row(user_data)
            print(table)
```

The **view_users** method in the provided code is designed to display user information based on a search term. Here's a description of the code:

1. **Method Signature**:
   - Name: **view_users**
   - Parameters:
     - **self**: Represents the instance of the class.
     - **search_term=None**: Optional parameter representing the term to search for in usernames.
2. **Functionality**:
   - Opens the "users.txt" file in read mode and reads all lines into the **lines** list.
   - Checks if a **search_term** is provided:
     - If **search_term** is not None, it searches for matching usernames in the **lines** list and creates a list of found users (**found_users**).
     - If no matching users are found (**found_users** is empty), it prints "No matching users found." Otherwise, it prints the details of the matching users in a formatted table using **PrettyTable**.
   - If no **search_term** is provided (or **search_term** is an empty string), it checks if there are any users in the **lines** list:
     - If there are no users (**lines** is empty), it prints "No users found."
     - If there are users, it prints the details of all users in a formatted table using **PrettyTable**.
3. **Comments on the Code**:
   - File Handling: Uses **open** to read the content of the "users.txt" file and store it in the **lines** list.
   - Search Term Check: Determines whether a search term is provided or not.
   - Username Matching: Checks if the **search_term** (converted to lowercase) is present in any usernames (also converted to lowercase) from the file.
   - PrettyTable Usage: Utilizes **PrettyTable** to create a visually appealing table for displaying user information.

- Printing Results: Prints the matching users' information or a message indicating no users or matching users were found.

Overall, this method allows the user to view user information based on a search term. If a search term is provided, it displays matching users; otherwise, it displays all users or indicates if there are no users.

d. Def Find_User:

```
@staticmethod
    def find_user(self, search_term):
        # Logic to search for a user by username or email in the users.txt file
        with open('users.txt', 'r') as file:
            lines = file.readlines()
            found_users = []
            for line in lines:
                user_data = line.strip().split(',')
                if search_term.lower() in [user_data[0].lower(), user_data[2].lower()]:
                    found_users.append(user_data)
            return found_users
```

1. **@staticmethod**:
   - Indicates that the method doesn't require access to instance attributes and can be called directly from the class.
2. **def find_user(self, search_term):**:
   - Defines a static method named **find_user** that takes **self** (although it's not used, as it's a static method) and **search_term** as parameters.
3. **with open('users.txt', 'r') as file:**:
   - Opens the 'users.txt' file in read mode and ensures it's properly closed after the block of code inside.
4. **lines = file.readlines()**:
   - Reads all lines from the file into a list called **lines**.
5. **found_users = []**:
   - Initializes an empty list to store the found users.
6. **for line in lines:**:
   - Iterates through each line in the list of lines from the file.
7. **user_data = line.strip().split(',')**:
   - Strips any leading/trailing whitespace from the line and splits it into user data using a comma as the delimiter, creating a list of user attributes.
8. **if search_term.lower() in [user_data[0].lower(), user_data[2].lower()]:**:
   - Checks if the search term (converted to lowercase) is in either the username (index 0) or email (index 2) of the user data.
9. **found_users.append(user_data)**:
   - If the search term matches, adds the user data (a list) to the list of found users.
10. **return found_users**:
    - Returns the list of found users, which may be empty if no matches were found.

Overall, this method reads user data from a file, checks if the provided search term matches any usernames or emails, and returns a list of users that match the search criteria. It's designed to be used statically without relying on instance attributes.

e. Def Find_user_By_Username:

Defines a method named find_user_by_username that takes self and username_to_search as parameters.

- with open('users.txt', 'r') as file:

  Opens the 'users.txt' file in read mode and ensures it's properly closed after the block of code inside.
- **lines = file.readlines():**

  Reads all lines from the file into a list called lines.
- **for line in lines::**

  Iterates through each line in the list of lines from the file.
- **user_data = line.strip().split(','):**

  Strips any leading/trailing whitespace from the line and splits it into user data using a comma as the delimiter, creating a list of user attributes.
- **if user_data[0] == username_to_search::**

  Checks if the username (at index 0 of user_data) matches the username_to_search parameter.

  return user_data:

  If the username matches, returns the user data (a list) containing all user attributes.
- **return None:**

  If the user with the specified username is not found after iterating through all lines, returns None to indicate that no user was found with that username.

Overall, this method reads user data from a file, specifically searches for a user by their username, and returns the user data if a match is found. If no match is found, it returns None to indicate that the user does not exist in the file.

f. Def Add_grade:

```
    def add_grade(self, user_id, course, score):
        try:
            grades = self.load_grades()     # Load grades from the grades file
          user = self.find_user_by_id(int(user_id))# Find the user by their ID

          if len(course) > 5 or int(score) < 0 or int(score) > 100:         # Check if the course name is not
too long and the score is within the valid range
                raise ValueError

            elif user is None:                  # Check if the user exists
                print("User does not exist.")

            else:
                user.add_grade(course, score)          # Add the grade for the user and update the grades
dictionary
                grades[str(user.get_ID())][course] = score
        # Write the updated grades back to the grades file using pickle
        with open(self.GRADES_FILENAME, 'wb') as file:
          pickle.dump(grades, file)

        print(f'Successfully added grade for {user.get_name()} ({user.get_username()}).')         #
Print a success message

    except FileNotFoundError:
      print("Grades database could not be found.")

    except KeyError:
      print("That user ID was not found.")

    except Exception as e:
      print(e)
```

Defines a method named **add_grade** that takes **self**, **user_id**, **course**, and **score** as parameters.

1. **try:**:
    - Starts a try-except block to handle potential exceptions.
2. **grades = self.load_grades()**:
    - Calls the **load_grades** method to load grades from a file and assigns them to the **grades** variable.
3. **user = self.find_user_by_id(int(user_id))**:
    - Calls the **find_user_by_id** method to find the user by their ID and assigns the user object to the **user** variable.
4. **if len(course) > 5 or int(score) < 0 or int(score) > 100:**:
    - Checks if the length of the course name is greater than 5 or if the score is less than 0 or greater than 100.
5. **raise ValueError**:
    - Raises a **ValueError** if the conditions in the previous line are met.
6. **elif user is None:**:
    - Checks if the **user** object is **None**, indicating that the user does not exist.
7. **print("User does not exist.")**:
    - Prints a message indicating that the user does not exist.
8. **user.add_grade(course, score)**:
    - Calls the **add_grade** method of the **user** object to add a grade for the user.
9. **grades[str(user.get_ID())][course] = score**:
    - Updates the **grades** dictionary with the new grade for the user.
10. **with open(self.GRADES_FILENAME, 'wb') as file:**:
    - Opens the grades file in binary write mode to write the updated grades using pickle.
11. **pickle.dump(grades, file)**:
    - Uses pickle to dump (serialize) the **grades** dictionary into the file.
12. **print(f'Successfully added grade for {user.get_name()} ({user.get_username()}).')**:
    - Prints a success message indicating that the grade was successfully added for the user.

13. **except FileNotFoundError:**:
    - Handles the exception if the grades database file is not found.
14. **except KeyError:**:
    - Handles the exception if the user ID is not found in the grades.
15. **except Exception as e:**:
    - Handles any other exceptions and prints the exception message.

g. Def Delet_user:

```
def delete_user(self, username):
    with open('users.txt', 'r') as file:     # Open the 'users.txt' file in read mode to read all lines
        lines = file.readlines()

    with open('users.txt', 'w') as file:      # Open the 'users.txt' file in write mode to overwrite with updated data
        for line in lines:                       # Iterate through each line in the file
            user_data = line.strip().split(',')       # Split the line into user data using ',' as the delimiter

            if user_data[0] != username:              # Check if the username in the current line does not match the username to be deleted
                file.write(line)
        # Print a success message indicating that the user was deleted successfully
    print(Fore.GREEN + f"User '{username}' deleted successfully!")
  # Functions to save ECA and grade details to respective files
```

1.
    - Defines a method named **delete_user** that takes **self** and **username** as parameters.
2. **with open('users.txt', 'r') as file:**:
    - Opens the 'users.txt' file in read mode to read all lines and automatically closes it after the block.
3. **lines = file.readlines()**:
    - Reads all lines from the file and stores them in the **lines** list.
4. **with open('users.txt', 'w') as file:**:
    - Opens the 'users.txt' file in write mode to overwrite with updated data and automatically closes it after the block.
5. **for line in lines:**:
    - Iterates through each line in the **lines** list.
6. **user_data = line.strip().split(',')**:
    - Strips any leading or trailing whitespace from the line and then splits it into user data using ',' as the delimiter.

7. **if user_data[0] != username:**:
    - Checks if the username in the current line does not match the username to be deleted.
8. **file.write(line)**:
    - If the usernames do not match, writes the line back to the file, effectively keeping it in the file.
9. **print(Fore.GREEN + f"User '{username}' deleted successfully!")**:
    - Prints a success message indicating that the user with the specified username was deleted successfully.

h. Save Eca:

```
def save_eca(eca, new_user_id=None):
        try:
            # Open the eca.txt file in append mode
            with open("eca.txt", "a") as file:
                # Write the ECA data for the new user ID to the file
                writer = csv.writer(file)
                writer.writerow([new_user_id] + eca)

            print(Fore.GREEN + f"ECA saved successfully for new user ID:
{new_user_id}")
        except Exception as e:
            print(Fore.RED + f"Error saving ECA: {e}")
```

@staticmethod: This decorator indicates that the method is a static method, meaning it can be called on the class itself rather than an instance of the class. Static methods don't have access to instance attributes and methods.

def save_eca(eca, new_user_id=None): This defines the static method save_eca, which takes two parameters: eca (presumably a list of ECA data) and new_user_id (an optional parameter, defaulting to None).

with open("eca.txt", "a") as file: This opens the "eca.txt" file in append mode, allowing data to be added to the end of the file without overwriting existing content.

writer = csv.writer(file): This creates a CSV writer object using the opened file, which will be used to write data in CSV format.

writer.writerow([new_user_id] + eca): This writes a new row to the CSV file. The row consists of the new_user_id followed by the elements of the eca list. It concatenates [new_user_id] with eca using the + operator.

print(Fore.GREEN + f"ECA saved successfully for new user ID: {new_user_id}"): This prints a success message with the newly saved user ID if the saving process is successful. It uses Fore.GREEN to print the text in green color using a library like colorama.

except Exception as e: This catches any exceptions that occur during the file operation and prints an error message with the exception details.

Overall, this code snippet handles saving ECA data to a file in CSV format and provides feedback on the success or failure of the operation. However, it's important to note that the specific usage of Fore.GREEN suggests that this code snippet might be part of a larger application that includes colorized console output, which may require additional libraries such as colorama or termcolor.

i. Def Save_Grade:

```
def save_grades(grades, student_id=None):
        try:
            # Open the grades.txt file in append mode
            with open("grades.txt", "a") as file:
                # Write the grade data for the student ID to the file
                writer = csv.writer(file)
                writer.writerow([student_id] + grades)
            print(Fore.GREEN + f"Grades saved successfully for student ID:
{student_id}")
        except Exception as e:
            print(Fore.RED + f"Error saving grades: {e}")
```

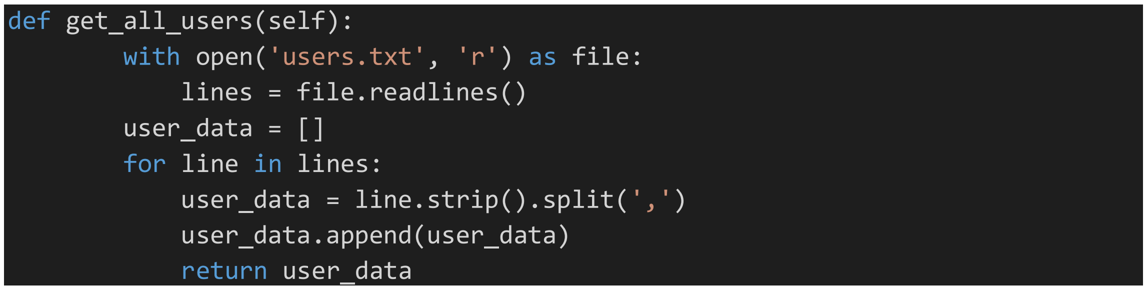
- This program defines a function named save_grades that is intended to save grade data for a student to a file named "grades.txt" in append mode. It uses the csv module to handle writing data to the file in CSV format.

  def save_grades(grades, student_id=None):: This defines the **save_grades** function with two parameters: **grades** (presumably a list of grades) and **student_id** (an optional parameter, defaulting to **None** if not provided).

- **try:**: This starts a try-except block to handle potential exceptions during file operations.
- **with open("grades.txt", "a") as file:**: This opens the "grades.txt" file in append mode (**"a"**), ensuring that data is added to the end of the file.
- **writer = csv.writer(file)**: This creates a CSV writer object to write data to the opened file.

- **writer.writerow([student_id] + grades)**: This writes a new row to the CSV file. The row consists of the **student_id** followed by the elements of the **grades** list. It concatenates **[student_id]** with **grades** using the + operator.
- **print(Fore.GREEN + f"Grades saved successfully for student ID: {student_id}")**: This prints a success message in green color if the grades are saved successfully. It uses **Fore.GREEN** from the **colorama** library to colorize the output.
- **except Exception as e:**: This catches any exceptions that occur during the file operation.
- **print(Fore.RED + f"Error saving grades: {e}")**: This prints an error message in red if there's an exception during the saving process. It includes the specific exception (**e**) to provide more information about the error.

j. Def get_all_user:



```
def get_all_users(self):
        with open('users.txt', 'r') as file:
            lines = file.readlines()
        user_data = []
        for line in lines:
            user_data = line.strip().split(',')
            user_data.append(user_data)
            return user_data
```

- **def get_all_users(self):**: This defines a method named **get_all_users** inside a class. The **self** parameter suggests it's part of a class, but the full class definition is not shown in this snippet.
- **with open('users.txt', 'r') as file:**: This opens the file 'users.txt' in read mode ('r'). The **with** statement ensures that the file is properly closed after reading, even if an exception occurs.
- **lines = file.readlines()**: This reads all lines from the file and stores them as a list in the variable **lines**. Each line is represented as a string element in the list.
- **user_data = []**: This initializes an empty list named **user_data**. This list will be used to store user information retrieved from the file.
- **for line in lines:**: This starts a loop that iterates through each line in the **lines** list.
- **user_info = line.strip().split(',')**: For each line, it removes leading and trailing whitespace characters (such as spaces and newline characters) using **strip()**, then splits the line using a comma (',') as the delimiter, creating a list of user information.
- **user_data.append(user_info)**: This appends the **user_info** list (containing user data from one line) to the **user_data** list. This accumulates user information from all lines in the file.
- **return user_data**: This returns the **user_data** list. However, there's a potential issue here: the **return** statement is inside the loop. This means that the function will return after processing the first line, which might not be the intended behavior. Depending on the desired logic, the **return** statement might need to be placed outside the loop to return all user data from the file, not just the first line.

k. Def Login:

```python
def login(self, username, password):
        self.username = username
        self.password = password
        print(Fore.CYAN + f"\n{f.renderText('Student Profile Management System')}")
        while True:
            print(Fore.YELLOW + "\n1. Admin Login")
            print("2. Student Login")
            print("3. Exit")
            choice = input(Fore.CYAN + "Enter your choice: ")
            if choice == '1':
                admin_username = input("Enter admin username: ")
                admin_password = input("Enter admin password: ")
                # Check if the admin credentials are valid
                admin_valid = False
                with open('users.txt', 'r') as file:
                    lines = file.readlines()
                    for line in lines:
                        user_data = line.strip().split(',')
                        if user_data[0] == admin_username and user_data[3] == admin_password and user_data[
                            1] == 'admin':
                            admin_valid = True
                            break
                if admin_valid:
                    admin = Admin(admin_username, 'admin')
                    print(Fore.GREEN + f"Login successful! Welcome, {admin_username}")
```

This code appears to be part of a login system for a Student Profile Management System.

1. **Instance Attributes Setting**: The **login** method sets the **username** and **password** attributes of the instance (**self**) using the values passed as parameters.
2. **Title Display**: It prints the title "Student Profile Management System" in cyan color using ASCII art. This is done using the **renderText** method of an object **f** (not defined in the provided snippet).
3. **Login Menu Loop**: It enters a while loop that displays the login menu options and prompts the user to enter their choice.
4. **Admin Login Section**: If the user chooses option 1 for admin login, it prompts for admin credentials, reads user data from a file ('users.txt'), and checks if the provided admin username, password, and user type ('admin') match any entry in the file.
5. **Admin Login Validation**: It iterates through each line in 'users.txt' and compares the entered admin credentials with each line's data. If a match is found, it sets **admin_valid** to **True**.
6. **Admin Login Success**: If **admin_valid** is **True**, it assumes there's an **Admin** class (not shown) and creates an instance of **Admin**, printing a success message.

Note: The code snippet provided doesn't include handling for options 2 (Student Login) and 3 (Exit), which could be added similarly to the admin login section. Additionally, error handling for invalid choices or incorrect credentials could be implemented for a more robust login system.

l. def modify record:

```python
def modify_student_record(student_to_modify):
        print(Fore.CYAN + f"\n{f.renderText('Modify Student Record')}")
        if student_to_modify is None:
            print(Fore.RED + "Error: Invalid student object.")
            return
        # Now you can safely access the username attribute
        username_to_search = student_to_modify.username
        found_student = None
        students = Admin.load_student_details()
        for student in students:
            if student.username == username_to_search:
                found_student = student
                break
        if found_student:
            print(Fore.GREEN + "Student found!")
            while True:
                print(Fore.YELLOW + "\n1. Add Grade\n2. Add ECA Record\n3.
Update Profile\n4. Back")
                choice = input(Fore.CYAN + "Enter your choice: ")
                if choice == '1':
                    grade = input("Enter grade: ")
                    try:
                        grade = int(grade)
                        found_student.add_grade(grade)
                        Admin.save_grade(found_student.username, grade)
                    except ValueError:
                        print(Fore.RED + "Error: Please enter a valid Grade
(Numeric Value):")
                elif choice == '2':
                    eca_description = input("Enter ECA description: ")
                    found_student.add_eca(eca_description)
                    Admin.save_eca(found_student.username, eca_description)
                elif choice == '3':
                    new_username = input("Enter new username: ")
                    new_email = input("Enter new email: ")
                    new_phone_number = input("Enter new phone number: ")
                    found_student.update_profile(new_username, new_email,
new_phone_number)
                    Admin.save_student_info(found_student)
                    print(Fore.GREEN + "Student details updated
successfully!")
                elif choice == '4':
                    break
                else:
                    print(Fore.RED + "Invalid choice.")
        else:
            print(Fore.RED + "Error: Student not found.")
```

This code defines a function **modify_student_record** that allows for modifications to a student's record within a Student Profile Management System.

1. **Title Display**: It prints the title "Modify Student Record" in cyan color using ASCII art (presumably through the **renderText** method of an object **f**).
2. **Input Validation**: It checks if the **student_to_modify** parameter is valid and exits the function if it's not.
3. **Finding Student**: It searches for the student to modify by matching the provided student's username with usernames in the system.
4. **Modification Options**: It presents a menu of modification options (add grade, add ECA record, update profile, go back) and handles user input to perform the chosen action.
5. **Handling Choices**: For each choice, it interacts with the **found_student** object and performs the corresponding action, such as adding a grade, adding an ECA record, updating the profile, or going back.
6. **Error Handling**: It includes error handling, such as checking if a grade input is a valid numeric value and providing appropriate error messages.
7. **Success Messages**: It prints success messages when actions like updating the profile or saving data are successful.

Note: The code assumes the existence of methods like **add_grade**, **add_eca**, **update_profile**, and **save_*** (e.g., **save_grade**, **save_eca**, **save_student_info**) within the **Admin** class, but these methods are not shown in the provided snippet.

D. **Class Student(User):**
The provided code snippet defines a class named **Student**, which inherits from another class **User.**

class Student(User):: This line defines a new class **Student** that inherits from the **User** class. This means that the **Student** class will have access to all attributes and methods defined in the **User** class.

i. **Def __INIT_:**

```
class Student(User):
    def __init__(self, username, role, email_address, password,
phone_number, grade, eca):
        super().__init__(username, role, email_address, password,
phone_number, grade, eca)
        self.grades = grade if isinstance(grade, list) else [grade]
        self.eca = eca
        self.phone_number = phone_number
```

.

1. **super().__init__(username, role, email_address, password, phone_number, grade, eca)**: This line calls the constructor of the parent class (**User**) to initialize attributes common to all users, such as username, role, email address, password, etc.

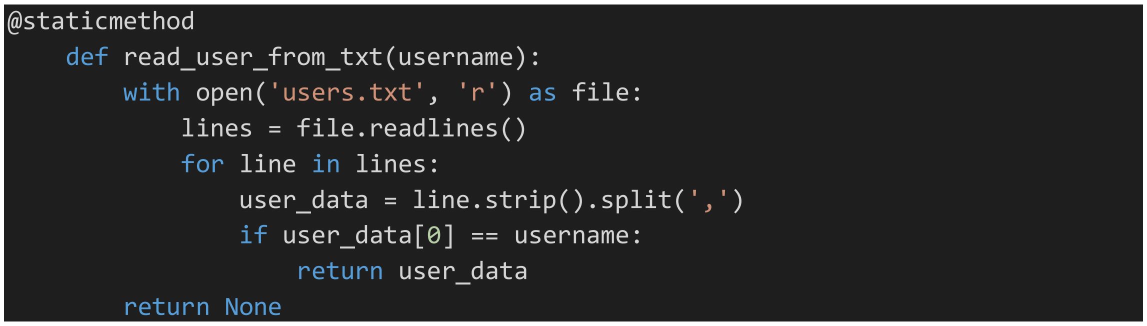
2. **self.grades = grade if isinstance(grade, list) else [grade]**: This line initializes the **grades** attribute of the student. It checks if **grade** is already a list; if not, it converts it to a list containing the single grade provided. This ensures that **grades** is always stored as a list, even if only one grade is given.

3. **self.eca = eca**: This line initializes the **eca** attribute of the student, which typically stands for extracurricular activities. It stores the provided list of extracurricular activities.

4. **self.phone_number = phone_number**: This line initializes the **phone_number** attribute of the student, storing the provided phone number.

Overall, this class sets up the basic structure for a **Student** object, inheriting common attributes from the **User** class and adding specific attributes related to grades, extracurricular activities, and phone number for a student.

ii. Def Display_user_info(self):



```
@staticmethod
    def read_user_from_txt(username):
        with open('users.txt', 'r') as file:
            lines = file.readlines()
            for line in lines:
                user_data = line.strip().split(',')
                if user_data[0] == username:
                    return user_data
        return None
```

display_user_info method within the **Student** class. It takes **self** as a parameter, indicating it's an instance method.

1. **super().display_user_info()**: This line calls the **display_user_info** method of the parent class (**User**) using **super()**. It displays the common user information such as username, role, email address, etc.
2. **print(f"Grades: {','.join([str(g) for g in self.grades]) or 'No Grades'}")**: This line prints the grades information of the student. It uses a list comprehension to convert each grade to a string, joins them with commas using **','.join(...)**, and handles the case where there are no grades (prints "No Grades" in that case).
3. **print(f"ECA: {self.eca}" if self.eca else "No ECA")**: This line prints the extracurricular activities (ECA) information of the student if available. It checks if **self.eca** is not empty (truthy value), and if so, it prints the ECA; otherwise, it prints "No ECA" to indicate that no extracurricular activities are available.

Overall, this method complements the **__init__** constructor of the **Student** class by providing a way to display detailed information about a student, including their grades and extracurricular activities. It also leverages inheritance by first displaying common user information via the **super().display_user_info()** call.

iii. **def read_user_from_txt:**

This code snippet defines a static method **read_user_from_txt** within an unspecified class. This method reads user data from a text file named "users.txt" and returns the data corresponding to a given username.

1. **@staticmethod**: This decorator indicates that the method **read_user_from_txt** is a static method, meaning it's bound to the class rather than an instance of the class. It can be called without creating an instance of the class.
2. **def read_user_from_txt(username):**: This line defines the static method **read_user_from_txt**, which takes a **username** parameter.
3. **with open('users.txt', 'r') as file:**: This line opens the file "users.txt" in read mode ('r') using a context manager (**with** statement), ensuring that the file is properly closed after reading, even if an exception occurs.
4. **lines = file.readlines()**: This reads all lines from the file and stores them in the **lines** list.
5. **for line in lines:**: This iterates through each line in the file.
6. **user_data = line.strip().split(',')**: This line strips leading and trailing whitespace from the line and splits it by commas to extract user data. The user data is stored in the **user_data** list.
7. **if user_data[0] == username:**: This checks if the first element of **user_data** (which should be the username) matches the given **username**.
8. **return user_data**: If a match is found, this line returns the **user_data**.
9. **return None**: If the given **username** is not found in the file, this line returns **None**.

Overall, this method allows reading user data from a text file based on a given username. If the username is found, it returns the corresponding user data; otherwise, it returns **None**.

iv. **Def Read_user_from_csv:**

```
def read_user_from_csv(username):
      with open('users.csv', 'r') as file:
          reader = csv.reader(file)
          headers = next(reader)  # Skip the headers
          for user in reader:
              if user[1] == username:
                  return user
      return None
```

1. The function **read_user_from_csv** takes a single parameter **username**, which is the username of the user whose data needs to be retrieved.

2. It opens the CSV file "users.csv" in read mode using a context manager (**with** statement).

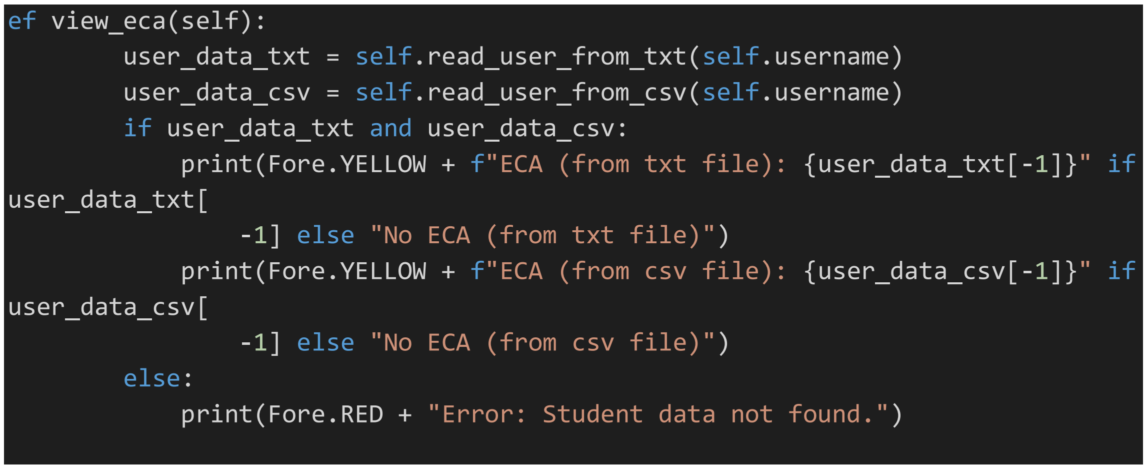
3. It creates a CSV reader object using **csv.reader(file)**, which allows reading the CSV file row by row.

4. It skips the headers (first row) of the CSV file using **next(reader)** to move the reader to the next row after reading the headers.

5. It iterates through each row (user data) in the CSV file using a **for** loop.
6. Inside the loop, it checks if the username in the current row (index 1, assuming the username is the second column) matches the given **username**.
7. If a match is found, it returns the user data for that row.
8. If no match is found after checking all rows, it returns **None** to indicate that the username was not found in the CSV file.

In summary, this function provides a way to extract user data from a CSV file based on a given username. It's useful for retrieving user information stored in a structured format like CSV.

v. Def View_eca(self):



```
ef view_eca(self):
        user_data_txt = self.read_user_from_txt(self.username)
        user_data_csv = self.read_user_from_csv(self.username)
        if user_data_txt and user_data_csv:
            print(Fore.YELLOW + f"ECA (from txt file): {user_data_txt[-1]}" if
user_data_txt[
                -1] else "No ECA (from txt file)")
            print(Fore.YELLOW + f"ECA (from csv file): {user_data_csv[-1]}" if
user_data_csv[
                -1] else "No ECA (from csv file)")
        else:
            print(Fore.RED + "Error: Student data not found.")
```

This code snippet defines a method **view_eca** within an unspecified class. The purpose of this method is to display the extracurricular activities (ECA) of a student, using data obtained from both a text file ("users.txt") and a CSV file ("users.csv").

1. The method **view_eca** first calls two helper methods **read_user_from_txt** and **read_user_from_csv** to retrieve user data from the text and CSV files, respectively, based on the current username.
2. It then checks if user data was successfully retrieved from both sources (**user_data_txt** and **user_data_csv**).
3. If data is available from both sources, it prints the extracurricular activities (ECA) retrieved from each file. It checks if the last item of each user data list (**[-1]**) exists, indicating the presence of ECA data. If ECA data is present, it prints it in yellow color; otherwise, it prints "No ECA".
4. If user data was not found in either file, it prints an error message in red indicating that student data was not found.

Overall, this method provides a way to view extracurricular activities associated with a student, leveraging data stored in both text and CSV files.

vi. Def View_grade(self):

```
def view_grades(self):
        user_data_txt = self.read_user_from_txt(self.username)
        user_data_csv = self.read_user_from_csv(self.username)
        if user_data_txt and user_data_csv:
            grades_txt = user_data_txt[-2].split(',') if user_data_txt[-2]
else []
            grades_csv = user_data_csv[-2].split(',') if user_data_csv[-2]
else []
            print(Fore.YELLOW + f"Grades (from txt file): {',
'.join(grades_txt) or 'No Grades (from txt file)'}")
            print(Fore.YELLOW + f"Grades (from csv file): {',
'.join(grades_csv) or 'No Grades (from csv file)'}")
        else:
            print(Fore.RED + "Error: Student data not found.")
```

1. The method **view_grades** first calls two helper methods **read_user_from_txt** and **read_user_from_csv** to retrieve user data from the text and CSV files, respectively, based on the current username.
2. It then checks if user data was successfully retrieved from both sources (**user_data_txt** and **user_data_csv**).
3. If data is available from both sources, it extracts the grades from the user data lists obtained from the text and CSV files.
4. It prints the grades retrieved from each file. It joins the list of grades with commas using **', '.join(grades_txt)** and **', '.join(grades_csv)** respectively. If no grades are available, it prints "No Grades".
5. If user data was not found in either file, it prints an error message in red indicating that student data was not found.

Overall, this method provides a way to view grades associated with a student, leveraging data stored in both text and CSV files.

vii. Def Update_Profile:

```
 def update_profile(self, new_username, new_email, new_phone_number):
        self.username = new_username
        self.email = new_email
        self.phone_number = new_phone_number
        print(Fore.GREEN + "Student details updated successfully!")
    # Methods to add grade and ECA details
        def add_grade(self, grade):
            self.grades.append(grade)
            print(Fore.GREEN + f"Grade added: {grade}")

        def view_eca(self):
```

```
        print(Fore.YELLOW + f"ECA: {self.eca}")
```

1. **update_profile method**:
   - This method takes three parameters: **new_username**, **new_email**, and **new_phone_number**.
   - It updates the attributes **username**, **email**, and **phone_number** of the current object with the new values passed as parameters.
   - After updating the profile, it prints a success message indicating that the student details have been updated successfully.
2. **add_grade method**:
   - This method takes a single parameter **grade**.
   - It appends the provided **grade** to the list of grades (**self.grades**) of the current object.
   - After adding the grade, it prints a success message indicating that the grade has been added.
3. **view_eca method**:
   - This method does not take any parameters.
   - It simply prints the extracurricular activities (**self.eca**) associated with the current object.
   - This method seems to be incomplete as it only prints the extracurricular activities but does not offer any functionality to modify or add new extracurricular activities.

viii. Def create_empty_file:

```
def create_empty_files():
    file_names = ['users.txt', 'grades.txt', 'eca.txt', 'students.txt']
    for file_name in file_names:
        if not os.path.exists(file_name):
            with open(file_name, 'w') as file:
                pass
```

This function, create_empty_files(), is designed to create empty text files if they don't already exist. Let's break down its functionality:

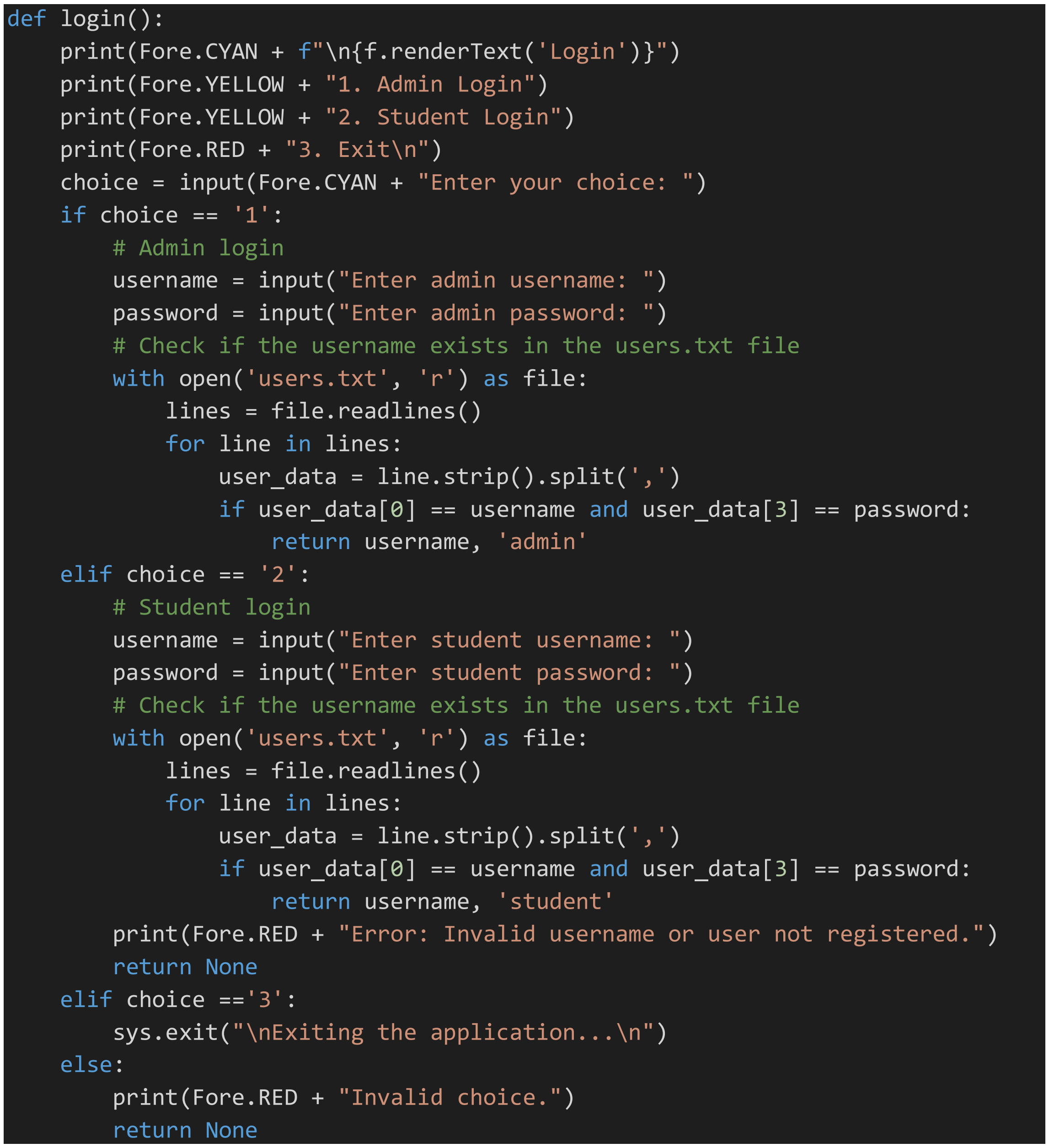
Initialization: It begins by defining a list of file names (file_names) representing the files to be created if they don't exist. These files include 'users.txt', 'grades.txt', 'eca.txt', and 'students.txt'.

File Creation Loop: It iterates through each file name in the file_names list.

Existence Check: For each file name, it checks whether the file already exists using os.path.exists(). If the file doesn't exist, it proceeds to create it.

File Creation: Upon confirming that the file doesn't exist, it opens the file in 'write' mode ('w') using a context manager (with statement). The pass statement is used as a placeholder within the context manager block, ensuring that the file is created empty.

ix. **Def Login():**



```python
def login():
    print(Fore.CYAN + f"\n{f.renderText('Login')}")
    print(Fore.YELLOW + "1. Admin Login")
    print(Fore.YELLOW + "2. Student Login")
    print(Fore.RED + "3. Exit\n")
    choice = input(Fore.CYAN + "Enter your choice: ")
    if choice == '1':
        # Admin login
        username = input("Enter admin username: ")
        password = input("Enter admin password: ")
        # Check if the username exists in the users.txt file
        with open('users.txt', 'r') as file:
            lines = file.readlines()
            for line in lines:
                user_data = line.strip().split(',')
                if user_data[0] == username and user_data[3] == password:
                    return username, 'admin'
    elif choice == '2':
        # Student login
        username = input("Enter student username: ")
        password = input("Enter student password: ")
        # Check if the username exists in the users.txt file
        with open('users.txt', 'r') as file:
            lines = file.readlines()
            for line in lines:
                user_data = line.strip().split(',')
                if user_data[0] == username and user_data[3] == password:
                    return username, 'student'
        print(Fore.RED + "Error: Invalid username or user not registered.")
        return None
    elif choice =='3':
        sys.exit("\nExiting the application...\n")
    else:
        print(Fore.RED + "Invalid choice.")
        return None
```

This login() function is designed to provide a simple login interface for an application

1. **User Interface**:

    - It prints a login title in cyan color using ASCII art.

- It displays three options: Admin Login, Student Login, and Exit, each with a corresponding number.

2. **User Input**:
    - It prompts the user to enter their choice by typing the corresponding number.
    - It reads the user's input choice using **input()**.
3. **Admin Login**:
    - If the user chooses '1' for Admin Login, it prompts the user to enter the admin username and password.
    - It reads the username and password entered by the user.
    - It checks if the entered username and password match any admin credentials stored in the 'users.txt' file. If a match is found, it returns the username and 'admin' role.
4. **Student Login**:
    - If the user chooses '2' for Student Login, it prompts the user to enter the student username and password.
    - It reads the username and password entered by the user.
    - It checks if the entered username and password match any student credentials stored in the 'users.txt' file. If a match is found, it returns the username and 'student' role.
    - If no match is found, it prints an error message indicating invalid credentials and returns **None**.
5. **Exit**:
    - If the user chooses '3' to Exit, the application exits using **sys.exit()** with a message.
6. **Invalid Choice**:
    - If the user enters an invalid choice, it prints an error message and returns **None**.

x. Def Modify_student_record:

```
def modify_student_record(student, student_to_modify=None):
    print(Fore.CYAN + f"\n{f.renderText('Modify Student Record')}")
    print(Fore.YELLOW + "\n1. Add Grade\n2. Add ECA Record\n3. Update 
Profile\n4. Back")
    choice = input(Fore.CYAN + "Enter your choice: ")
    if choice == '1':
        grade = input("Enter grade: ")
        try:
            grade = int(grade)
            found = False
            with open('users.txt', 'r') as file:
                for line in file:
                    user_data = line.strip().split(',')
                    if user_data is not None and len(user_data) > 0 and
student_to_modify is not None and hasattr(
                            student_to_modify, 'username') and user_data[0] ==
student_to_modify.username:
                        # Your code here
                        found = True
                        student_to_modify.add_grade(grade)
                        Admin.save_grade(student_to_modify.username, grade)
                        print(Fore.GREEN + f"Grade added successfully for
{student_to_modify.username}.")
                        break
            if not found:
                print(Fore.RED + "Error: Student not found.")
        except ValueError:
            print(Fore.RED + "Error: Please enter a valid Grade (Numeric
Value):")
    elif choice == '2':
        eca_description = input("Enter ECA description: ")
        if student_to_modify is not None:
            student_to_modify.add_eca(eca_description)
            Admin.save_eca(student_to_modify.username, eca_description)
        else:
            print(Fore.RED + "Error: Student not found.")
    elif choice == '3':
        new_username = input("Enter new username: ")
        new_email = input("Enter new email: ")
        new_phone_number = input("Enter new phone number: ")
        if student_to_modify is not None:
            student_to_modify.update_profile(new_username, new_email,
new_phone_number)
            Admin.save_student_info(student_to_modify)
            print(Fore.GREEN + "Student details updated successfully!")
        else:
            print(Fore.RED + "Error: Student not found.")
    elif choice == '4':
        return
    else:
        print(Fore.RED + "Invalid choice.")
# Function to display student information
```

This function, **modify_student_record**, appears to be designed to modify the records of a student.

1. **User Interface**:
    - It prints a title "Modify Student Record" in cyan color using ASCII art.
    - It displays options for modifying a student's record: Add Grade, Add ECA Record, Update Profile, and Back.
2. **User Input**:

- It prompts the user to enter their choice by typing the corresponding number.
- It reads the user's input choice using **input()**.

3. **Handling User Choices**:
    - It checks the user's choice and executes corresponding actions accordingly.
4. **Add Grade**:
    - If the user chooses '1' to Add Grade, it prompts the user to enter a grade.
    - It attempts to convert the input grade to an integer.
    - It then checks if the student to modify is found in the 'users.txt' file based on their username.
    - If found, it adds the grade to the student using **student_to_modify.add_grade(grade)** and saves the grade using **Admin.save_grade()**. It prints a success message.
    - If the student is not found, it prints an error message.
5. **Add ECA Record**:
    - If the user chooses '2' to Add ECA Record, it prompts the user to enter an ECA description.
    - It checks if the student to modify exists.
    - If found, it adds the ECA description to the student using **student_to_modify.add_eca(eca_description)** and saves it using **Admin.save_eca()**.
    - If the student is not found, it prints an error message.
6. **Update Profile**:
    - If the user chooses '3' to Update Profile, it prompts the user to enter a new username, email, and phone number.
    - It checks if the student to modify exists.
    - If found, it updates the student's profile using **student_to_modify.update_profile()** and saves the updated information using **Admin.save_student_info()**.
    - If the student is not found, it prints an error message.
7. **Back**:
    - If the user chooses '4', it returns without any further action.
8. **Invalid Choice**:
    - If the user enters an invalid choice, it prints an error message.

This function seems to depend on the existence of **Admin** class methods like **save_grade**, **save_eca**, and **save_student_info**, as well as methods like **add_grade**, **add_eca**, and **update_profile** of the **student_to_modify** object. Also, it assumes that the **student_to_modify** parameter is an object with a **username** attribute.

xi. Display student_info(Student):

```python
# Function to display student information
def display_student_info(student):
    print(Fore.CYAN + f"\n{f.renderText(' !! Student Information !! ')}")
    # Create a PrettyTable instance with the required columns
    table = PrettyTable(['Username', 'Role', 'Email', 'Grades', 'ECA', 'Phone 
Number'])
    # Add the student's information as a row in the table
    table.add_row([
        student.username,
        student.role,
        student.email,
        ', '.join(map(str, student.grades)) if student.grades else 'N/A',
        student.eca if student.eca else 'N/A',
        student.phone_number
    ])
    # Print the table
    print(table)
    print(Fore.YELLOW + f"\nUsername: {student.username}")
    print(Fore.LIGHTYELLOW_EX +f"Role: {student.role}")
    print(f"Email: {student.email}")
    # Assuming student_id is accessible through student object
    print(f"Grades: {student.grades}")
    print(f"ECA: {student.eca}")
    print(f"Phone Number: {student.phone_number}")
```

1. This function, **`display_student_info`**, is intended to display detailed information about a student.

   User Interface:

   - It prints a title "!! Student Information !!" in cyan color using ASCII art.

2. **Creating PrettyTable**:

   - It creates a **PrettyTable** instance with the required columns: Username, Role, Email, Grades, ECA, and Phone Number.

3. **Adding Student Information**:

   - It adds the information of the student object passed as a parameter to the PrettyTable as a row.
   - It accesses various attributes of the student object such as **username**, **role**, **email**, **grades**, **eca**, and **phone_number** to populate the table.

4. **Printing the Table**:

   - It prints the populated PrettyTable, displaying the student's information in a structured format.

5. **Additional Information**:

- After printing the table, it also prints individual attributes of the student such as username, role, email, grades, ECA, and phone number in a formatted manner.

xii. Def Admin_Actions_Admin(admin):

```
def admin_actions(admin):
    while True:
        print(Fore.YELLOW + "\n1. Register a new user\n2. search for user \n3.
Modify student record\n4. Delete student record\n5. View Users\n6. Go to
Dashboard\n7. Log oUt")
        choice = input(Fore.CYAN + "Enter your choice: ")
        if choice == '1':
            student_id  = input("Enter Student ID: ")
            new_username = input("Enter new username: ")
            new_role = input("Enter role (admin/student): ")
            new_email = input("Enter email: ")
            new_password = input("Enter password: ")
            new_phone_number = input("Enter phone number: ")
            grades_input = input("Enter grades : ")
            eca_input = input("Enter ECA description: ")
            # Split the grades and eca_input strings into lists
            grades = grades_input.split(',')
            eca = eca_input.split(',')
            admin.register_user(new_username, new_role, new_email,
new_password, new_phone_number, grades, eca)

            new_user_id=admin.register_user(new_username,new_role,new_email,ne
w_password,new_phone_number)
            print(Fore.GREEN + ' !!  registration Successful  !! With ID:
(new_student_id)  ')

        elif choice == '2':
            # Search for a user
            search_type = input(Fore.LIGHTRED_EX +"Search by username (u) or
email (e): ").lower()
            if search_type == 'u':
                username_to_search = input(Fore.LIGHTBLUE_EX +"Enter username
to search: ")
                found_user = admin.find_user_by_username(username_to_search)
                if found_user:
                    # Create a PrettyTable instance
                    table = PrettyTable(['Username', 'Role', 'Email', 'Phone
Number'])  # Add other fields as needed
                    # Add the found user's information to the table
                    table.add_row(found_user[:4])  # Consider only the
relevant fields for display
                    # Print the table
                    print(table)
                else:
                    print(Fore.RED + "No matching users found.")

        elif choice == '3':
        # Modify student record
            username_to_modify = input(Fore.BLUE +"Enter username to modify
record: ")
            student_to_modify =
admin.find_user_by_username(username_to_modify)
            if student_to_modify:
                modify_student_record(student_to_modify)
            else:
                print(Fore.RED + "Error: Student not found.")

        elif choice == '4':
            # Delete a user record
            username_to_delete = input(Fore.LIGHTBLUE_EX +"Enter username to
delete: ")
            admin.delete_user(username_to_delete)

        elif choice == '5':
            # View registered users
            display_users_txt()
            display_users_csv()
            print(Fore.YELLOW + '-----!!-- ShoWinG LiSt Of Users:--!!----- ')

            users = admin.get_all_users()

            table = PrettyTable(['Username', 'Role', 'Email', 'ECA',
'Grade'])  # Define table columns
            for user in users:
                # Assuming user is a list with at least 5 elements (username,
role, email, eca, grade)
                if len(user) >= 5:
                    table.add_row(
                        user[:5])  # Add only the first 5 elements to the
table (username, role, email, eca, grade)

            print(table)  # Print the table with user information
        elif choice == '6':
            #go to Dashboard
            os.system('python dashboard.py')
            dashboard=Dashboard(admin)
            dashboard.start()

        elif choice == '7':
            print(Fore.GREEN + "Log oUt...")
            break
        else:
            print(Fore.RED + "SORRY!! YOU HAVE Entered Invalid choice. Please
try again.")
```

This **admin_actions** function appears to be a part of an administrative interface, allowing an administrator to perform various actions within a system.

1. **User Interface**:
    - It displays a menu of options for the administrator, including Register a new user, Search for user, Modify student record, Delete student record, View Users, Go to Dashboard, and Log out.
2. **Handling User Choices**:
    - It prompts the administrator to enter their choice.
    - It reads the administrator's input choice using **input()**.
3. **Registering a New User**:
    - If the administrator chooses '1', it prompts the administrator to enter details for registering a new user, including username, role, email, password, phone number, grades, and ECA.
    - It then calls the **register_user** method of the **admin** object to register the new user.
4. **Searching for a User**:
    - If the administrator chooses '2', it prompts the administrator to enter the search type (by username or email) and the search query.
    - It then calls the **find_user_by_username** method of the **admin** object to search for the user.
    - If found, it displays the user's information using PrettyTable.
5. **Modifying a Student Record**:
    - If the administrator chooses '3', it prompts the administrator to enter the username of the student whose record needs to be modified.
    - It then calls the **find_user_by_username** method of the **admin** object to find the student.
    - If found, it calls the **modify_student_record** function to modify the student's record.
6. **Deleting a User Record**:
    - If the administrator chooses '4', it prompts the administrator to enter the username of the user whose record needs to be deleted.
    - It then calls the **delete_user** method of the **admin** object to delete the user's record.
7. **Viewing Registered Users**:
    - If the administrator chooses '5', it displays the list of registered users using PrettyTable.

8. **Go to Dashboard**:
    - If the administrator chooses '6', it navigates to the dashboard by executing a separate Python script.
9. **Logging Out**:
    - If the administrator chooses '7', it logs out of the system and breaks out of the loop.
10. **Invalid Choice**:
    - If the administrator enters an invalid choice, it prints an error message.

This function interacts with various methods of the **admin** object, including **register_user**, **find_user_by_username**, and **delete_user**. Additionally, it interacts with the **modify_student_record** function.

xiii. Def Display_user.txt:

```
def display_users_txt():
    with open('users.txt', 'r') as file:
        lines = file.readlines()
    if not lines:
        print(" Users Not found in users.txt.")
        return

    # Create a PrettyTable for users from users.txt
    table_txt = PrettyTable(['Username', 'Role', 'Email', 'Phone Number'])
    for line in lines:
        user_data = line.strip().split(',')
        if len(user_data) >= 4:  # Check if the line has at least 4 values
            table_txt.add_row(
                user_data[:4])  # Add only the first 4 values to the table
(Username, Role, Email, Phone Number)
        else:
            print(f"Ignoring invalid user data: {user_data}")

    print(Fore.LIGHTCYAN_EX +"------------------------!!--Users from
users.txt:--!!--------------------------")
    print(table_txt)
```

his **display_users_txt** function is designed to display information about users stored in a text file named 'users.txt'.

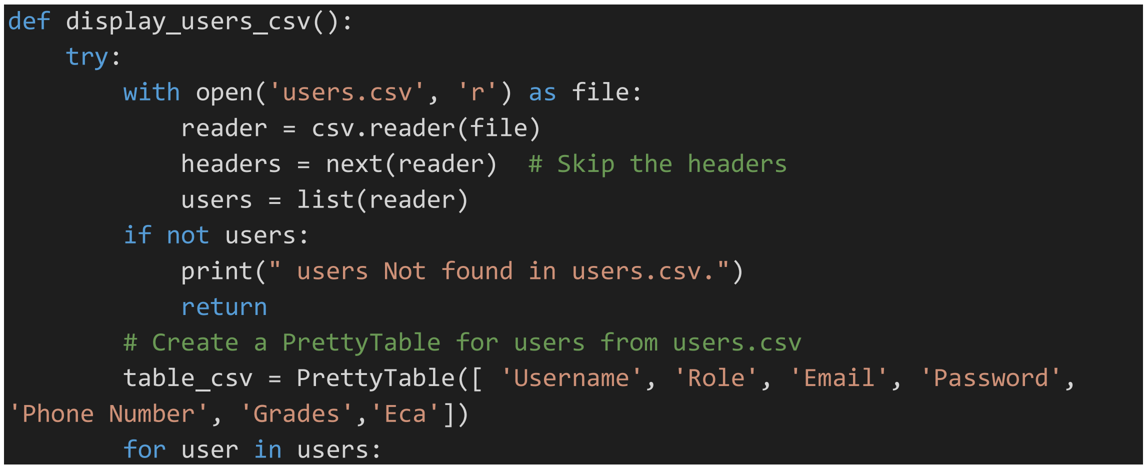
1. **Reading User Data from 'users.txt'**:
    - It opens the 'users.txt' file in read mode using a context manager (**with** statement).
    - It reads all the lines from the file using **readlines()**.
2. **Checking for Empty File**:

- It checks if there are any lines read from the file. If not, it prints a message indicating that no users are found in the file.

3. **Creating PrettyTable**:
    - It creates a **PrettyTable** instance with the required columns: Username, Role, Email, and Phone Number.
4. **Populating the Table**:
    - It iterates through each line read from the file.
    - It splits each line into user data using the comma (',') as the delimiter.
    - It checks if the line has at least 4 values (Username, Role, Email, and Phone Number).
    - If valid user data is found, it adds the first 4 values to the PrettyTable.
5. **Displaying the Table**:
    - It prints a heading indicating that the displayed users are from 'users.txt'.
    - It prints the populated PrettyTable containing user information.
6. **Ignoring Invalid User Data**:
    - If the line does not contain at least 4 values, it prints a message indicating that the user data is invalid.

This function provides a convenient way to visualize user data stored in 'users.txt' using PrettyTable. It also handles cases where the user data is incomplete or invalid.

xiv. Def Display_user.csv:



```
def display_users_csv():
    try:
        with open('users.csv', 'r') as file:
            reader = csv.reader(file)
            headers = next(reader)  # Skip the headers
            users = list(reader)
        if not users:
            print(" users Not found in users.csv.")
            return
        # Create a PrettyTable for users from users.csv
        table_csv = PrettyTable([ 'Username', 'Role', 'Email', 'Password',
'Phone Number', 'Grades','Eca'])
        for user in users:
            # Ensure each row has exactly 7 values
            if len(user) == 7:
                table_csv.add_row(user)
            else:
                print(f"Ignoring invalid row: {user}")
        print("-----------------!!--Users Lists  from users.csv:--!!----------
------------")
        print(table_csv)
    except FileNotFoundError:
        print("users.csv not found.")
```

This **display_users_csv** function is designed to display information about users stored in a CSV file named 'users.csv'.

1. **Opening and Reading the CSV File**:
    - It opens the 'users.csv' file in read mode using a context manager (**with** statement).
    - It creates a CSV reader object to read the contents of the file.
    - It skips the header row using **next(reader)** to move the reader's pointer to the next row containing actual data.
    - It reads all remaining rows into a list named **users**.
2. **Checking for Empty File**:
    - It checks if the **users** list is empty. If so, it prints a message indicating that no users are found in the file.
3. **Creating PrettyTable**:
    - It creates a **PrettyTable** instance with the required columns: Username, Role, Email, Password, Phone Number, Grades, and ECA.
4. **Populating the Table**:
    - It iterates through each row in the **users** list.
    - It adds each row to the PrettyTable.
    - It ensures that each row has exactly 7 values before adding it to the table. If a row has a different number of values, it prints a message indicating that the row is invalid.
5. **Displaying the Table**:
    - It prints a heading indicating that the displayed users are from 'users.csv'.
    - It prints the populated PrettyTable containing user information.
6. **Handling File Not Found Error**:
    - If the 'users.csv' file is not found, it prints a message indicating that the file is not found.

This function provides a convenient way to visualize user data stored in 'users.csv' using PrettyTable. It also handles cases where the user data is incomplete or invalid and the file is not found.

E. Def Main():

```
def main():
    display_users_txt()
    display_users_csv()
    create_empty_files()
    while True:
        user_data = login()
        if user_data:
            username, role = user_data
            if role == 'admin':
                admin = Admin('', 'admin', '', '', '', '', '')
                admin_actions(admin)
            elif role == 'student':
                student = Student(username, role, '', '', [], '', '')  #
Initialize student object
                print(Fore.GREEN + "Login successful! ---!--Welcome--!--,
Student", username)

                # Display student details
                display_student_info(student)

                while True:
                    # Student options menu
                    print(
                        Fore.YELLOW + "\n1. Update Your profile\n2. View Your
ECA details\n3. View Your Examination Grades\n4. LoGoUt")
                    choice = input(Fore.CYAN + "Enter your choice: ")

                    if choice == '1':
                        # Update student profile
                        new_username = input("Enter new username: ")
                        new_email = input("Enter new email: ")
                        new_phone_number = input("Enter new phone number: ")
                        student.update_profile(new_username, new_email,
new_phone_number)
                        print(Fore.GREEN + " !!Congratulations!! ---Your
details Has BeEn UpDatEd successfully--")

                    elif choice == '2':
                        # View ECA details
                        student.view_eca()

                    elif choice == '3':
                        # View examination grades
                        student.view_grades()



                    elif choice == '4':
                        print(Fore.GREEN + "---ThAnK YoU--- Exiting...To---
LoGin Page---")
                        break

                    else:
                        print(Fore.RED + "Invalid choice. Please try again.")

            else:
                print(Fore.RED + " SoRrY YoU HaVe SeLeCtEd WrOnG NuMbeR.
Exiting...")
```

The **main** function serves as the entry point of the program.

1. **Display User Data and Create Empty Files**:
    - It calls **display_users_txt()** and **display_users_csv()** functions to display user information from text and CSV files, respectively.
    - It calls **create_empty_files()** to create empty files if they don't exist.
2. **User Authentication and Role-Based Actions**:
    - It enters a loop for user authentication using the **login()** function.
    - Upon successful login, it determines the user's role (admin or student).
    - If the user is an admin, it creates an **Admin** object and calls **admin_actions(admin)** to perform administrative tasks.
    - If the user is a student, it creates a **Student** object, prints a login success message, displays the student's information using **display_student_info(student)**, and enters a loop for student-specific actions.
3. **Student Actions Menu**:
    - Inside the student loop, it displays a menu for student-specific actions such as updating profile, viewing ECA details, viewing examination grades, and logging out.
    - Based on the student's choice, it performs the corresponding action.
4. **Exiting the Program**:
    - It provides an option for users to exit the program gracefully.

5. **Handling Invalid Inputs**:
    - It prints error messages for invalid user inputs.

This function orchestrates the flow of the program, handles user authentication, and provides interfaces for both administrators and students to interact with the system.

```
# Entry point of the program
if __name__ == "__main__":
    main()
```

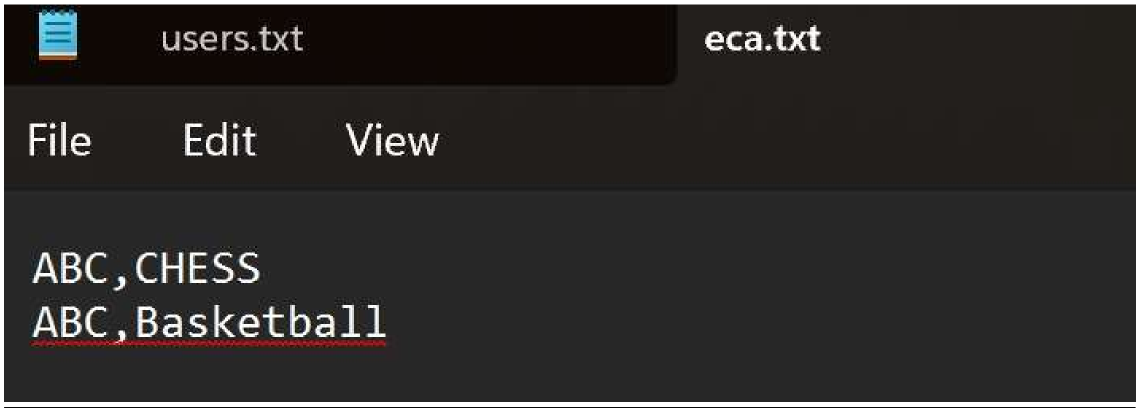
The **if __name__ == "__main__":** block serves as the entry point of the program.

1. It checks if the current script is being run as the main program (**__name__** is equal to **"__main__"**).
2. If it is the main program, it calls the **main()** function to start the execution of the program.

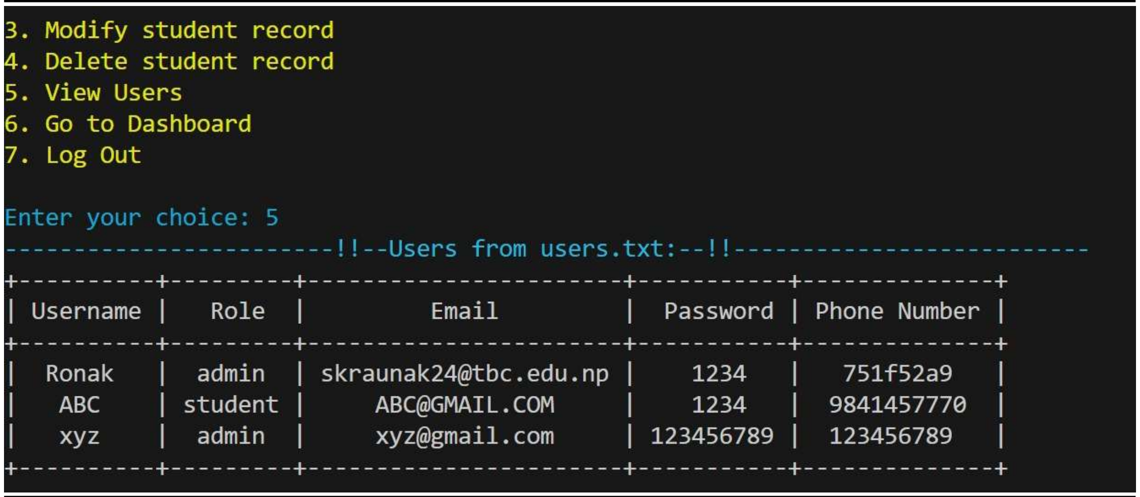
This block ensures that the **main()** function is executed only when the script is run directly, not when it's imported as a module in another script.

This ensures that the **main()** function is called when the script is executed directly from the command line or by other means. If the script is imported as a module elsewhere, the **main()** function won't be executed automatically.

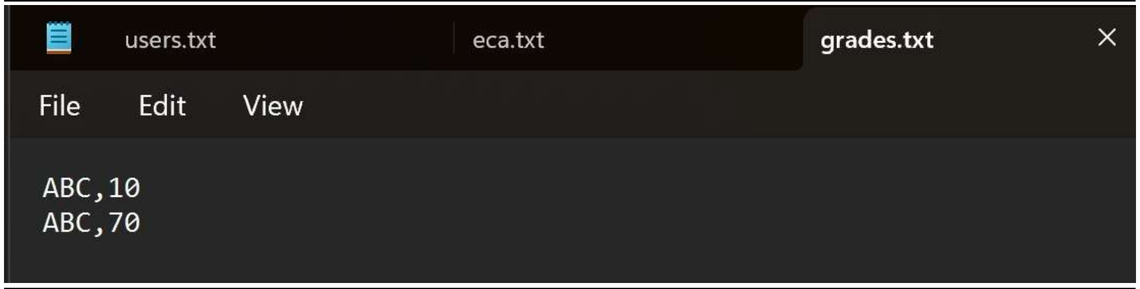
OUTPUTS OF STUDENT PROFILE MANAGEMENT SYSTEM



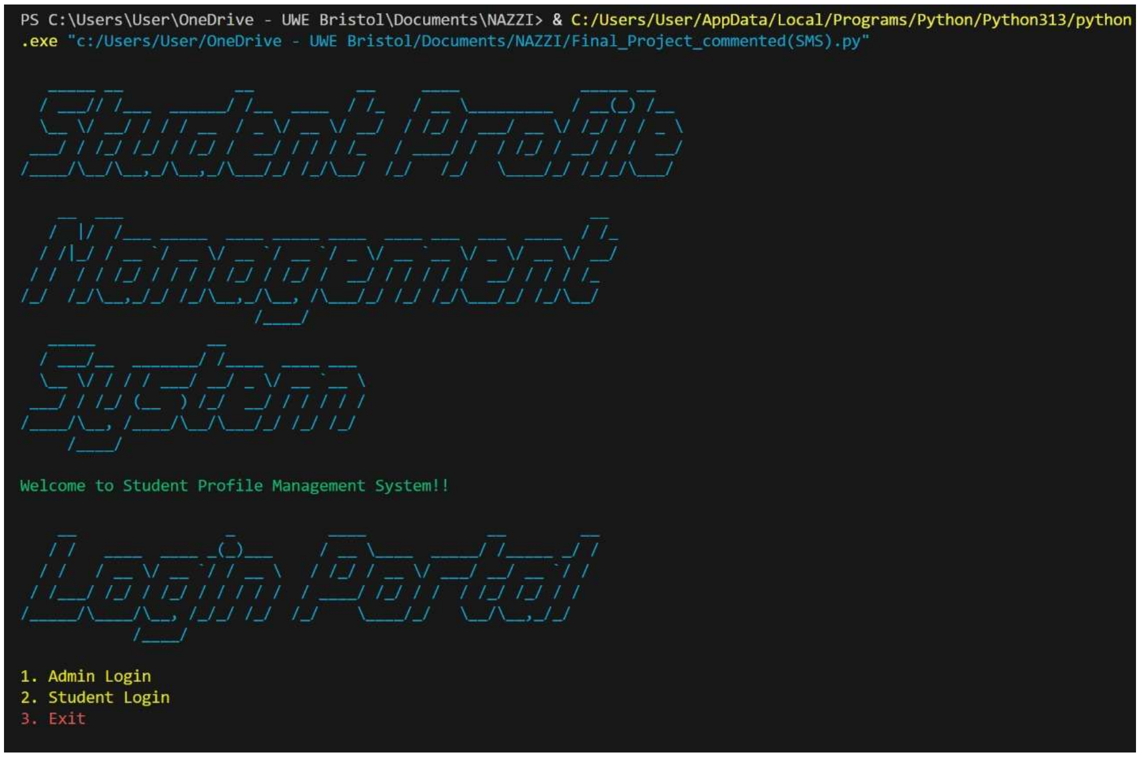
```
ABC,CHESS
ABC,Basketball
```



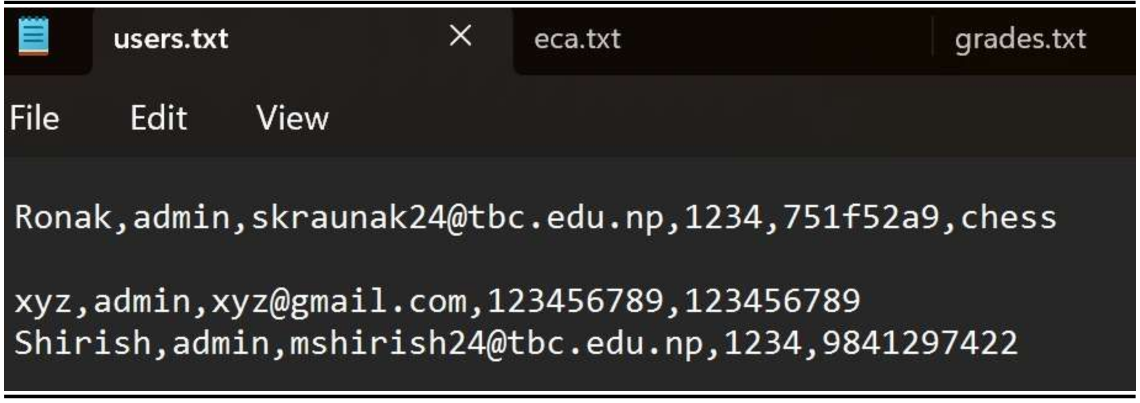
```
3. Modify student record
4. Delete student record
5. View Users
6. Go to Dashboard
7. Log Out

Enter your choice: 5
------------------------!!--Users from users.txt:--!!-------------------------
+----------+---------+-----------------------+-----------+-------------+
| Username |  Role   |         Email         |  Password | Phone Number |
+----------+---------+-----------------------+-----------+-------------+
|  Ronak   |  admin  | skraunak24@tbc.edu.np |    1234   |   751f52a9   |
|   ABC    | student |     ABC@GMAIL.COM     |    1234   |  9841457770  |
|   xyz    |  admin  |     xyz@gmail.com     | 123456789 |  123456789   |
+----------+---------+-----------------------+-----------+-------------+
```



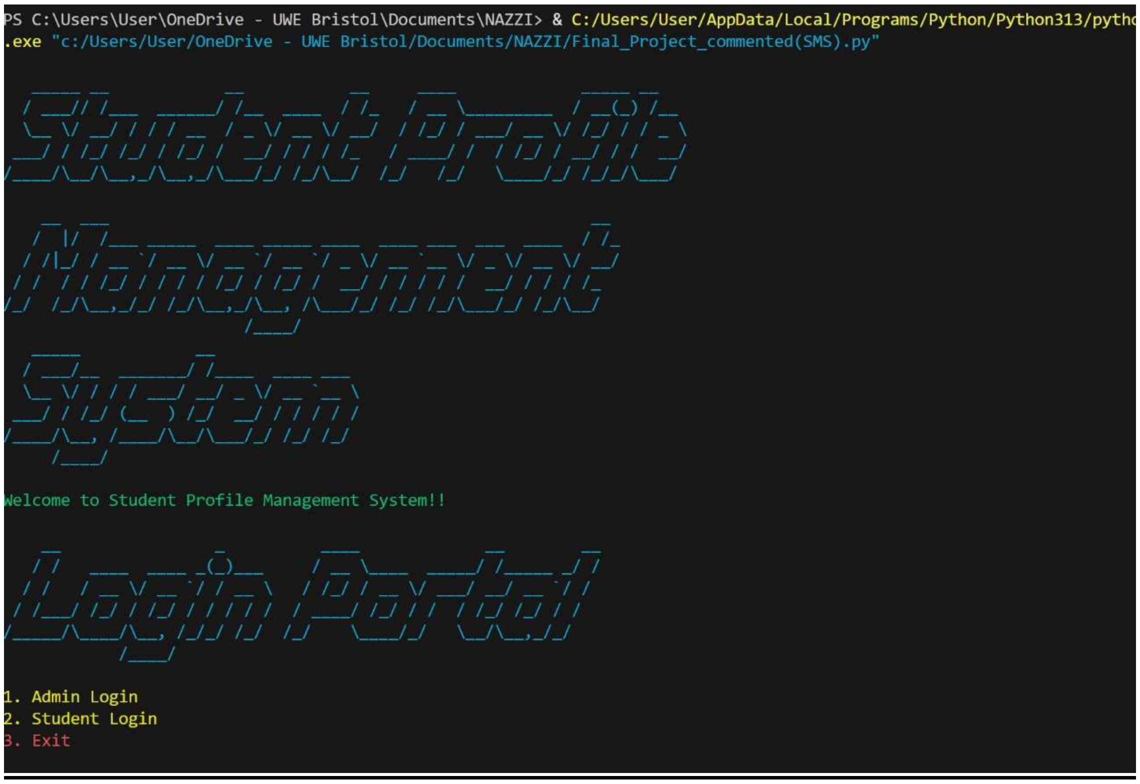
```
ABC,10
ABC,70
```



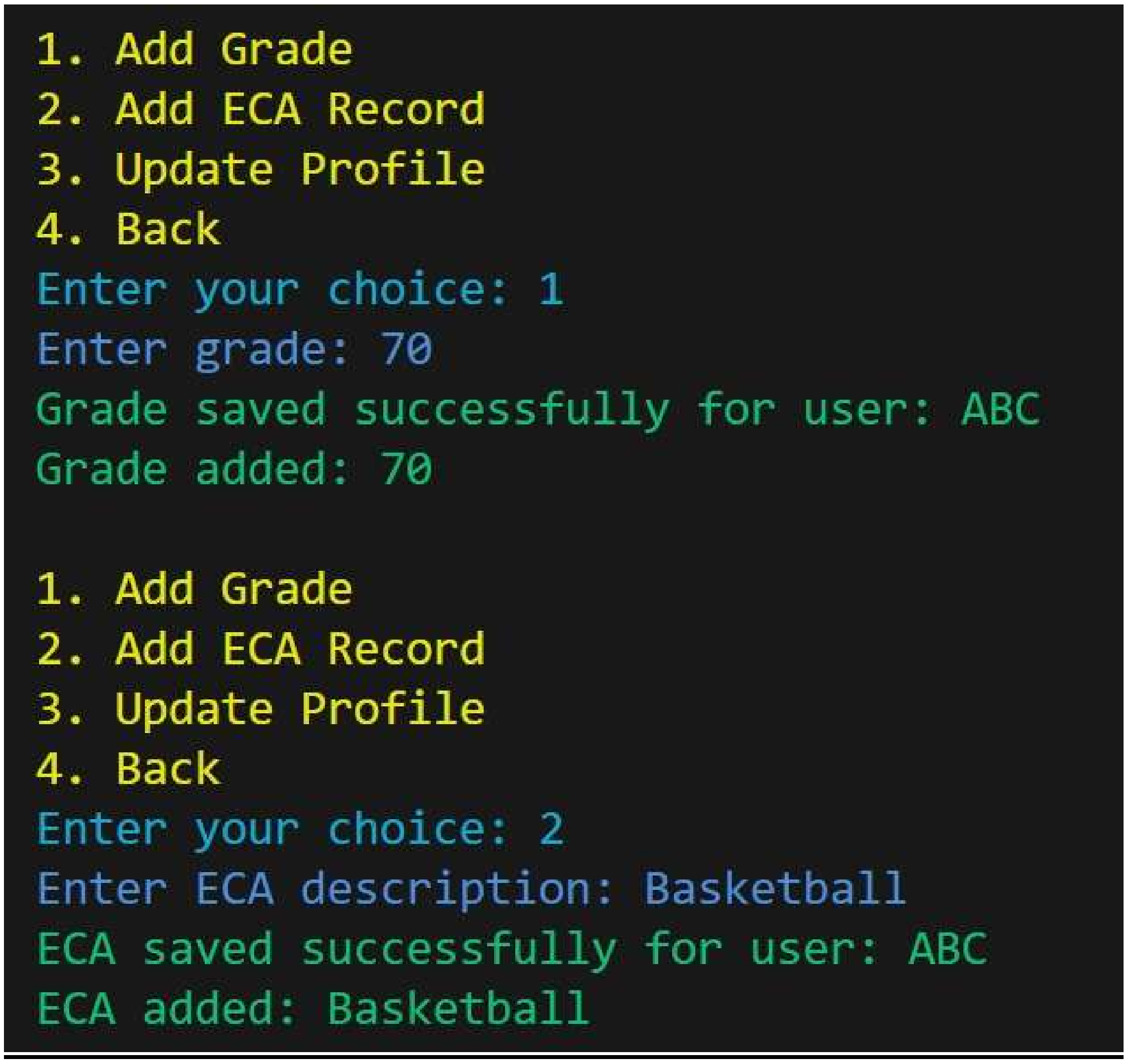
```
PS C:\Users\User\OneDrive - UWE Bristol\Documents\NAZZI> & C:/Users/User/AppData/Local/Programs/Python/Python313/python
.exe "c:/Users/User/OneDrive - UWE Bristol/Documents/NAZZI/Final_Project_commented(SMS).py"

   _____ __            __           __     ____             _____ __
  / ___// /___  ______/ /__  ____  / /_   / __ \_________  / __(_) /__
  \__ \/ __/ / / / __  / _ \/ __ \/ __/  / /_/ / ___/ __ \/ /_/ / / _ \
 ___/ / /_/ /_/ / /_/ /  __/ / / / /_   / ____/ /  / /_/ / __/ / /  __/
/____/\__/\__,_/\__,_/\___/_/ /_/\__/  /_/   /_/   \____/_/ /_/_/\___/


    __  ___                                                  __
   /  |/  /___ _____  ____ _____ ____  ____ ___  ___  ____  / /_
  / /|_/ / __ `/ __ \/ __ `/ __ `/ _ \/ __ `__ \/ _ \/ __ \/ __/
 / /  / / /_/ / / / / /_/ / /_/ /  __/ / / / / /  __/ / / / /_
/_/  /_/\__,_/_/ /_/\__,_/\__, /\___/_/ /_/ /_/\___/_/ /_/\__/
                         /____/
   _____            __
  / ___/__  _______/ /____  ____ ___
  \__ \/ / / / ___/ __/ _ \/ __ `__ \
 ___/ / /_/ (__  ) /_/  __/ / / / / /
/____/\__, /____/\__/\___/_/ /_/ /_/
     /____/

Welcome to Student Profile Management System!!

    __                _          ____             __        __
   / /   ____  ____ _(_)___     / __ \____  _____/ /_____ _/ /
  / /   / __ \/ __ `/ / __ \   / /_/ / __ \/ ___/ __/ __ `/ /
 / /___/ /_/ / /_/ / / / / /  / ____/ /_/ / /  / /_/ /_/ / /
/_____/\____/\__, /_/_/ /_/  /_/    \____/_/   \__/\__,_/_/
            /____/

1. Admin Login
2. Student Login
3. Exit
```



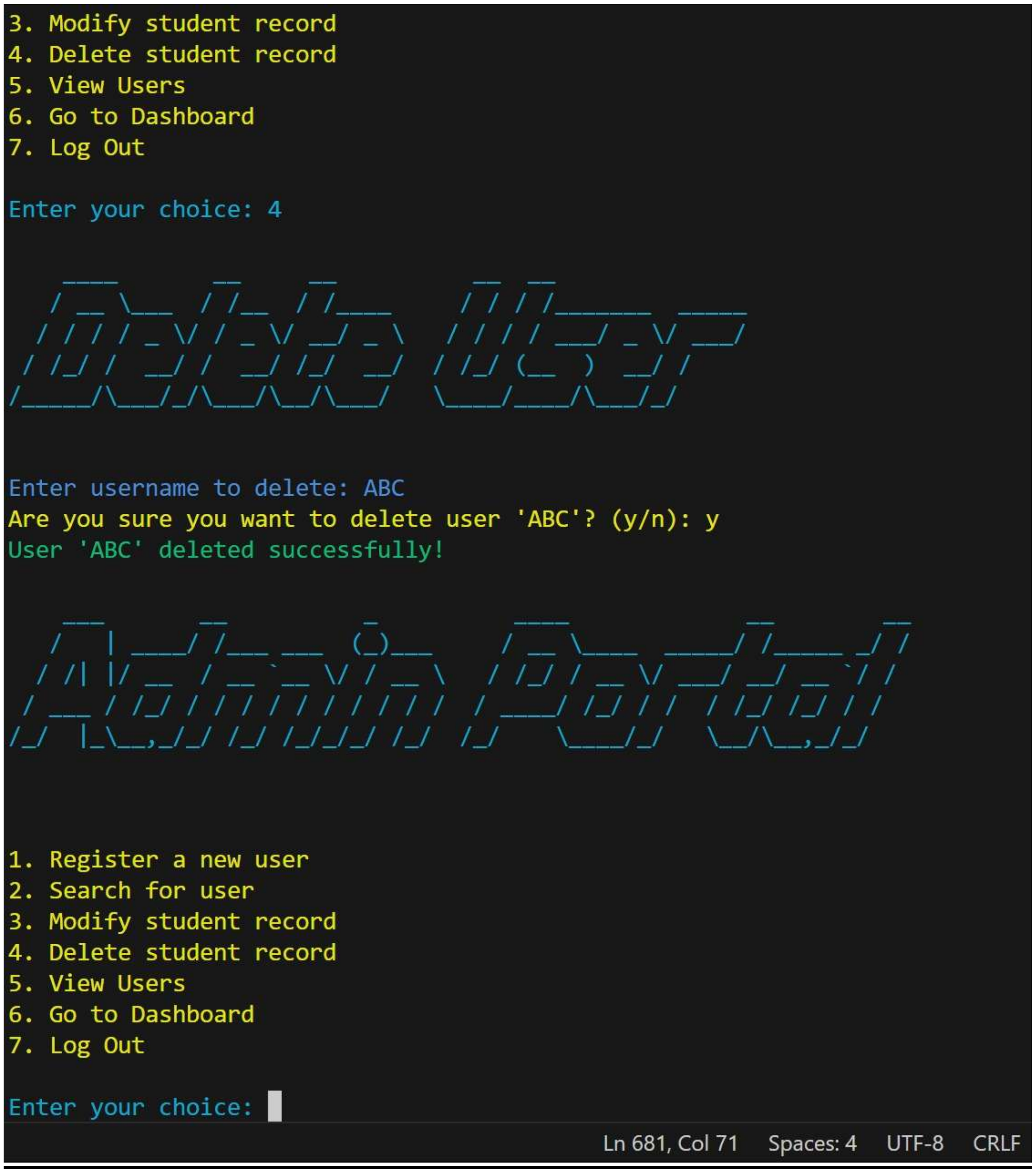
```
Ronak,admin,skraunak24@tbc.edu.np,1234,751f52a9,chess

xyz,admin,xyz@gmail.com,123456789,123456789
Shirish,admin,mshirish24@tbc.edu.np,1234,9841297422
```



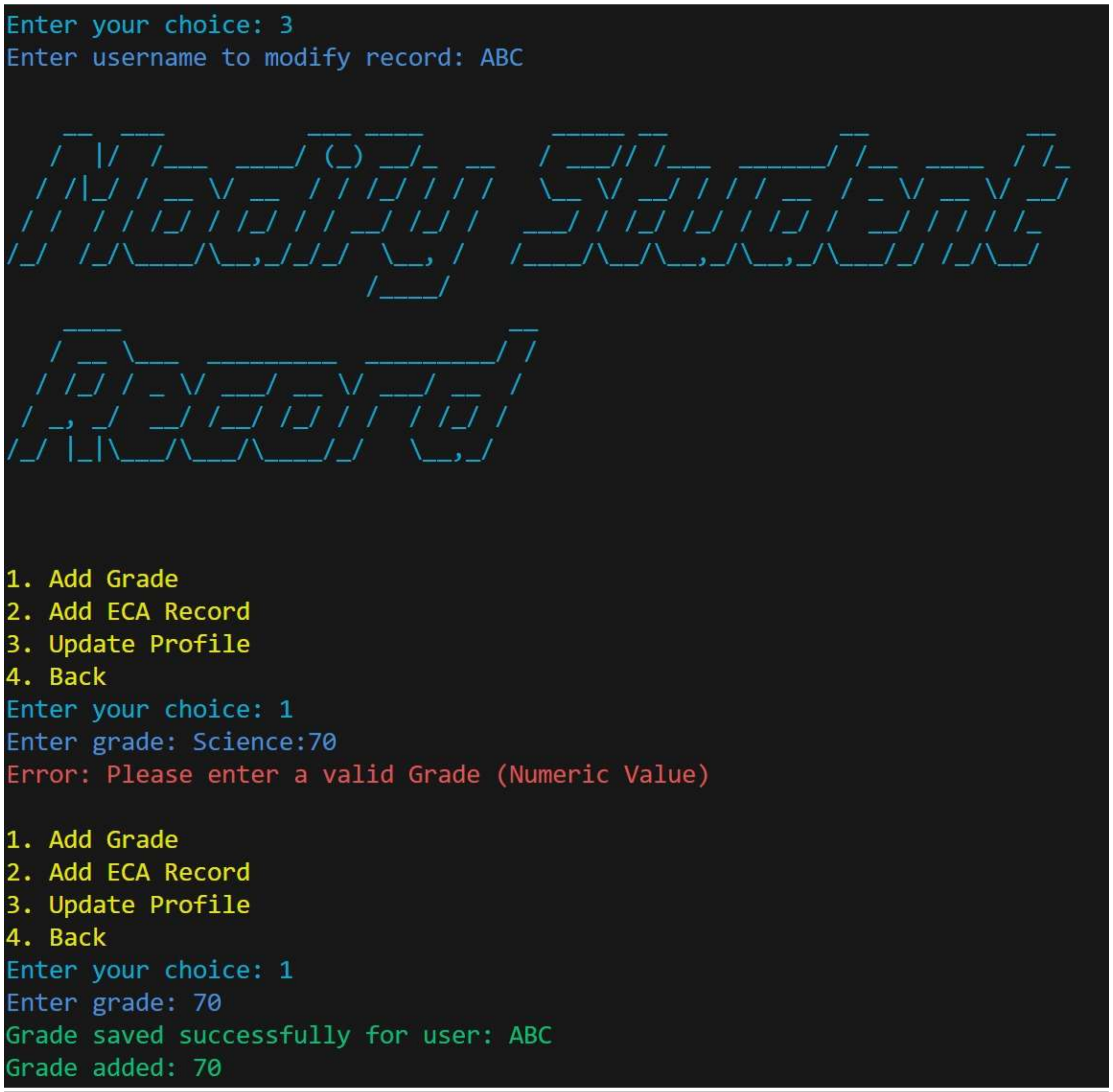
```
PS C:\Users\User\OneDrive - UWE Bristol\Documents\NAZZI> & C:/Users/User/AppData/Local/Programs/Python/Python313/pytho
.exe "c:/Users/User/OneDrive - UWE Bristol/Documents/NAZZI/Final_Project_commented(SMS).py"

   _____ __            __           __     ____             _____ __
  / ___// /___  ______/ /__  ____  / /_   / __ \_________  / __(_) /__
  \__ \/ __/ / / / __  / _ \/ __ \/ __/  / /_/ / ___/ __ \/ /_/ / / _ \
 ___/ / /_/ /_/ / /_/ /  __/ / / / /_   / ____/ /  / /_/ / __/ / /  __/
/____/\__/\__,_/\__,_/\___/_/ /_/\__/  /_/   /_/   \____/_/ /_/_/\___/

    __  ___                                                  __
   /  |/  /___ _____  ____ _____ ____  ____ ___  ___  ____  / /_
  / /|_/ / __ `/ __ \/ __ `/ __ `/ _ \/ __ `__ \/ _ \/ __ \/ __/
 / /  / / /_/ / / / / /_/ / /_/ /  __/ / / / / /  __/ / / / /_
/_/  /_/\__,_/_/ /_/\__,_/\__, /\___/_/ /_/ /_/\___/_/ /_/\__/
                         /____/
   _____            __
  / ___/__  _______/ /____  ____ ___
  \__ \/ / / / ___/ __/ _ \/ __ `__ \
 ___/ / /_/ (__  ) /_/  __/ / / / / /
/____/\__, /____/\__/\___/_/ /_/ /_/
     /____/

Welcome to Student Profile Management System!!

    __                _          ____             __        __
   / /   ____  ____ _(_)___     / __ \____  _____/ /_____ _/ /
  / /   / __ \/ __ `/ / __ \   / /_/ / __ \/ ___/ __/ __ `/ /
 / /___/ /_/ / /_/ / / / / /  / ____/ /_/ / /  / /_/ /_/ / /
/_____/\____/\__, /_/_/ /_/  /_/    \____/_/   \__/\__,_/_/
            /____/

1. Admin Login
2. Student Login
3. Exit
```



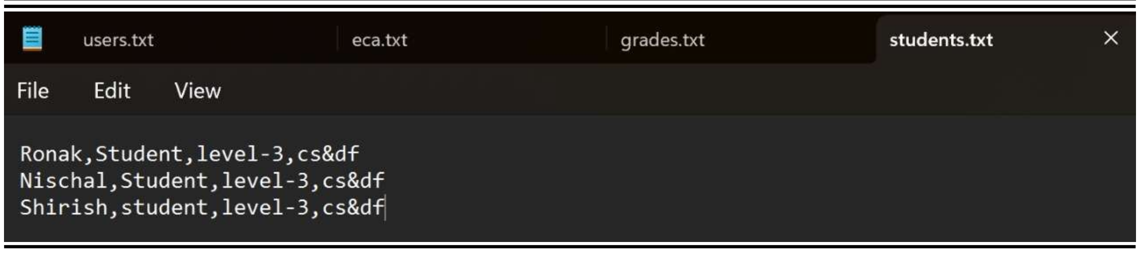
```
1. Add Grade
2. Add ECA Record
3. Update Profile
4. Back
Enter your choice: 1
Enter grade: 70
Grade saved successfully for user: ABC
Grade added: 70

1. Add Grade
2. Add ECA Record
3. Update Profile
4. Back
Enter your choice: 2
Enter ECA description: Basketball
ECA saved successfully for user: ABC
ECA added: Basketball
```



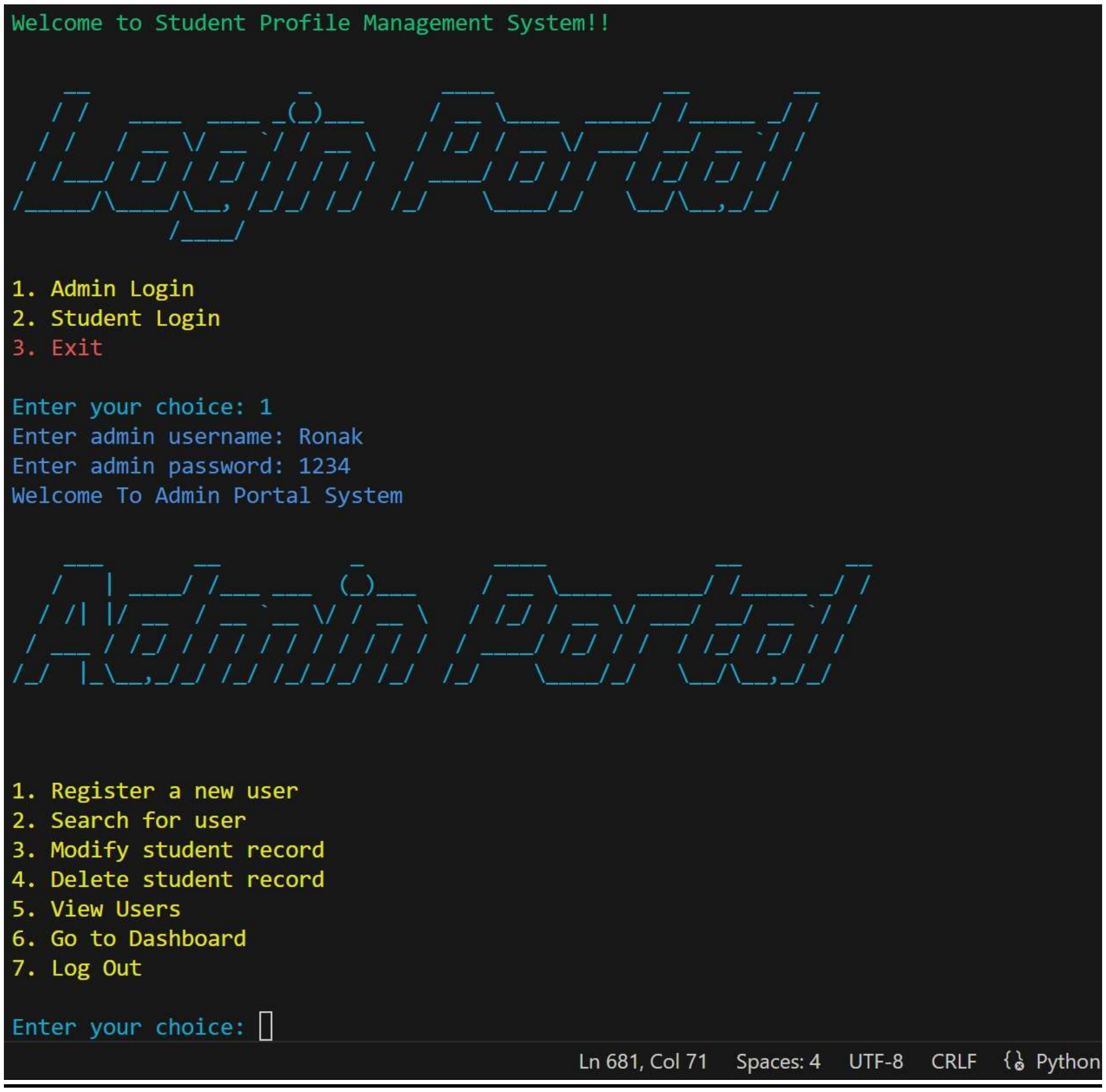
```
3. Modify student record
4. Delete student record
5. View Users
6. Go to Dashboard
7. Log Out

Enter your choice: 4


    ____       __     __          __  __
   / __ \___  / /__  / /____     / / / /_______  _____
  / / / / _ \/ / _ \/ __/ _ \   / / / / ___/ _ \/ ___/
 / /_/ /  __/ /  __/ /_/  __/  / /_/ (__  )  __/ /
/_____/\___/_/\___/\__/\___/   \____/____/\___/_/


Enter username to delete: ABC
Are you sure you want to delete user 'ABC'? (y/n): y
User 'ABC' deleted successfully!


    ___       __          _          ____             __        __
   /   | ____/ /___ ___  (_)___     / __ \____  _____/ /_____ _/ /
  / /| |/ __  / __ `__ \/ / __ \   / /_/ / __ \/ ___/ __/ __ `/ /
 / ___ / /_/ / / / / / / / / / /  / ____/ /_/ / /  / /_/ /_/ / /
/_/  |_\__,_/_/ /_/ /_/_/_/ /_/  /_/    \____/_/   \__/\__,_/_/



1. Register a new user
2. Search for user
3. Modify student record
4. Delete student record
5. View Users
6. Go to Dashboard
7. Log Out

Enter your choice: 
```



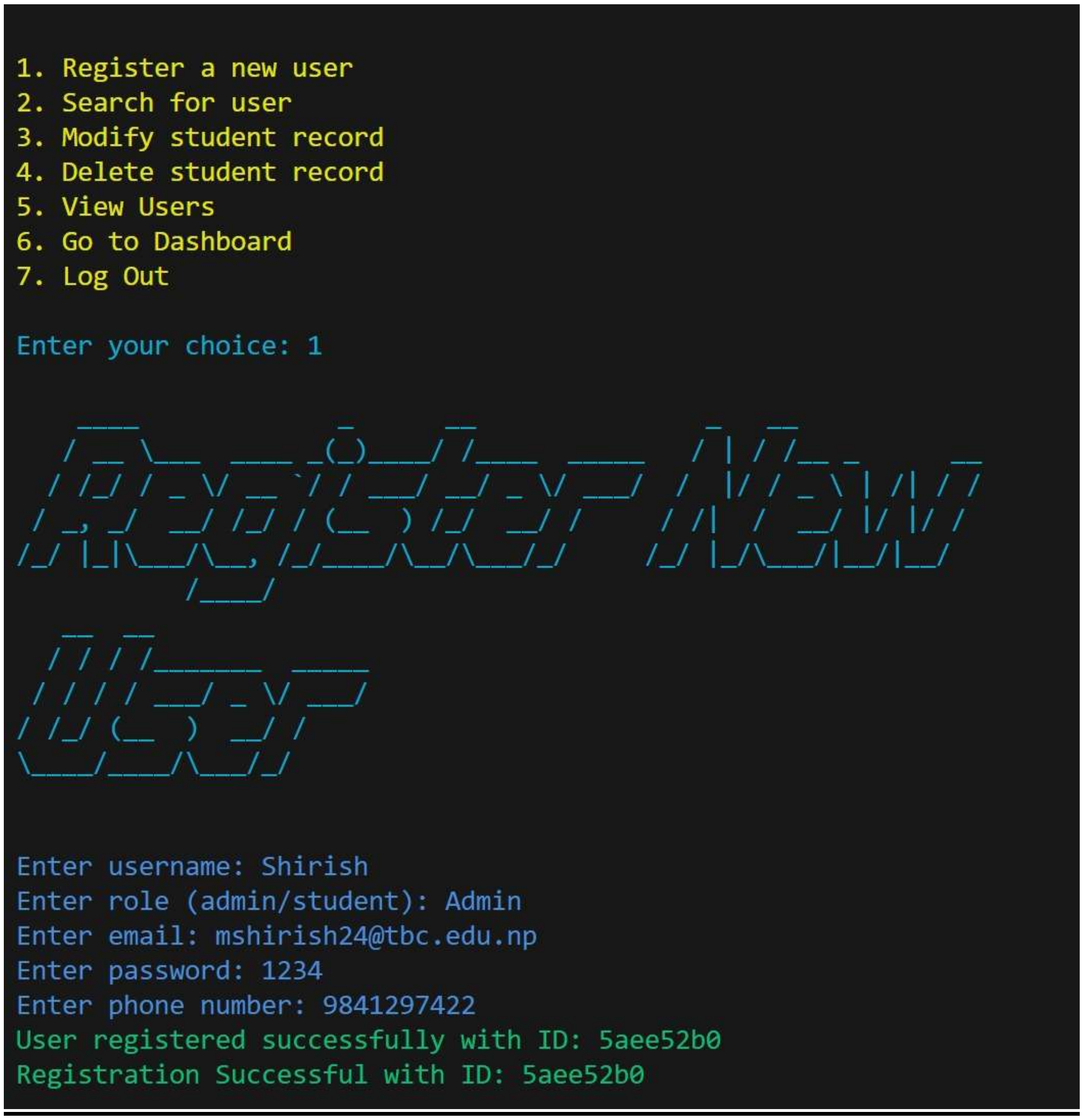
```
Enter your choice: 3
Enter username to modify record: ABC

    __  ___          ___ ____         _____ __            __           __ 
   /  |/  /___  ____/ (_) __/_  __   / ___// /___  ______/ /__  ____  / /_
  / /|_/ / __ \/ __  / / /_/ / / /   \__ \/ __/ / / / __  / _ \/ __ \/ __/
 / /  / / /_/ / /_/ / / __/ /_/ /   ___/ / /_/ /_/ / /_/ /  __/ / / / /_  
/_/  /_/\____/\__,_/_/_/  \__, /   /____/\__/\__,_/\__,_/\___/_/ /_/\__/  
                         /____/                                           
    ____                            __
   / __ \___  _________  _________/ /
  / /_/ / _ \/ ___/ __ \/ ___/ __  / 
 / _, _/  __/ /__/ /_/ / /  / /_/ /  
/_/ |_|\___/\___/\____/_/   \__,_/   


1. Add Grade
2. Add ECA Record
3. Update Profile
4. Back
Enter your choice: 1
Enter grade: Science:70
Error: Please enter a valid Grade (Numeric Value)

1. Add Grade
2. Add ECA Record
3. Update Profile
4. Back
Enter your choice: 1
Enter grade: 70
Grade saved successfully for user: ABC
Grade added: 70
```



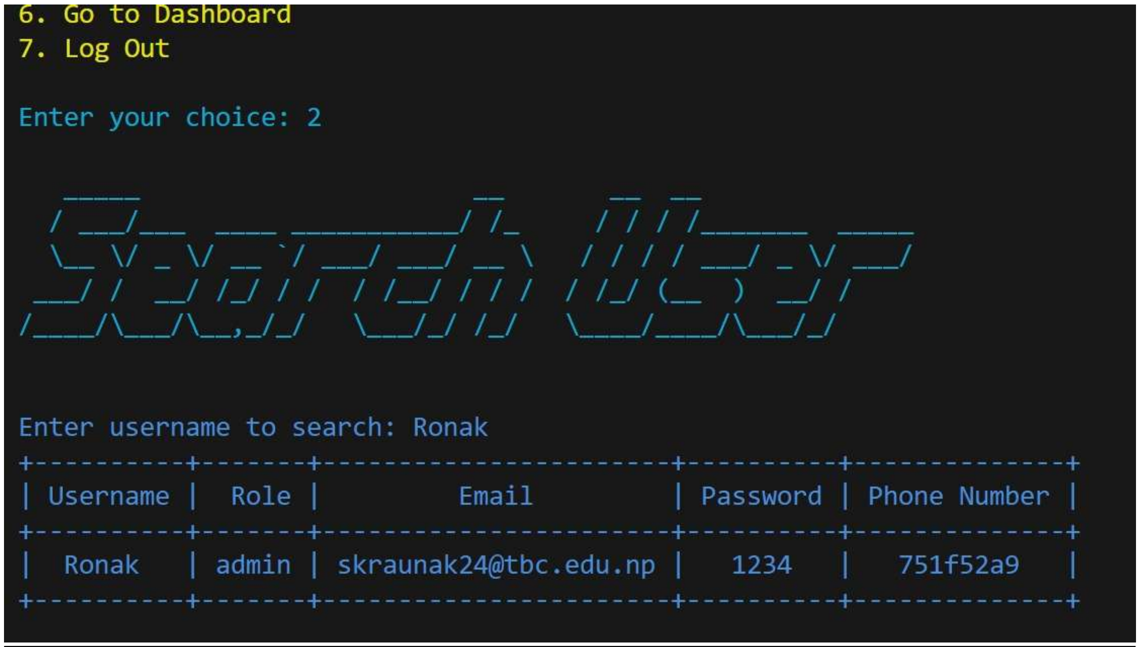
```
Ronak,Student,level-3,cs&df
Nischal,Student,level-3,cs&df
Shirish,student,level-3,cs&df
```



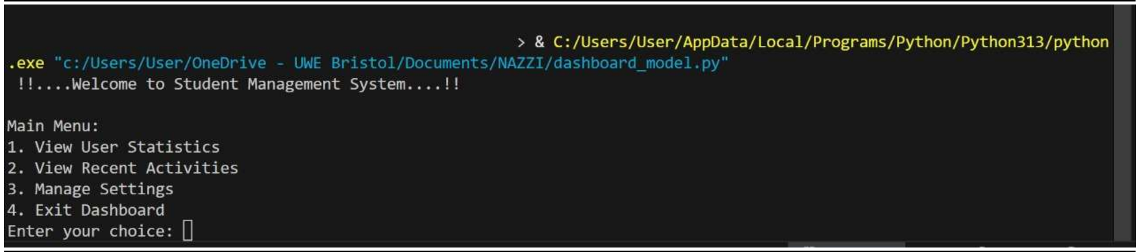
```
Welcome to Student Profile Management System!!

    __                _          ____             __        __
   / /   ____  ____ _(_)___     / __ \____  _____/ /_____ _/ /
  / /   / __ \/ __ `/ / __ \   / /_/ / __ \/ ___/ __/ __ `/ /
 / /___/ /_/ / /_/ / / / / /  / ____/ /_/ / /  / /_/ /_/ / /
/_____/\____/\__, /_/_/ /_/  /_/    \____/_/   \__/\__,_/_/
            /____/

1. Admin Login
2. Student Login
3. Exit

Enter your choice: 1
Enter admin username: Ronak
Enter admin password: 1234
Welcome To Admin Portal System

    ___       __          _          ____             __        __
   /   | ____/ /___ ___  (_)___     / __ \____  _____/ /_____ _/ /
  / /| |/ __  / __ `__ \/ / __ \   / /_/ / __ \/ ___/ __/ __ `/ /
 / ___ / /_/ / / / / / / / / / /  / ____/ /_/ / /  / /_/ /_/ / /
/_/  |_\__,_/_/ /_/ /_/_/_/ /_/  /_/    \____/_/   \__/\__,_/_/



1. Register a new user
2. Search for user
3. Modify student record
4. Delete student record
5. View Users
6. Go to Dashboard
7. Log Out

Enter your choice: 
```



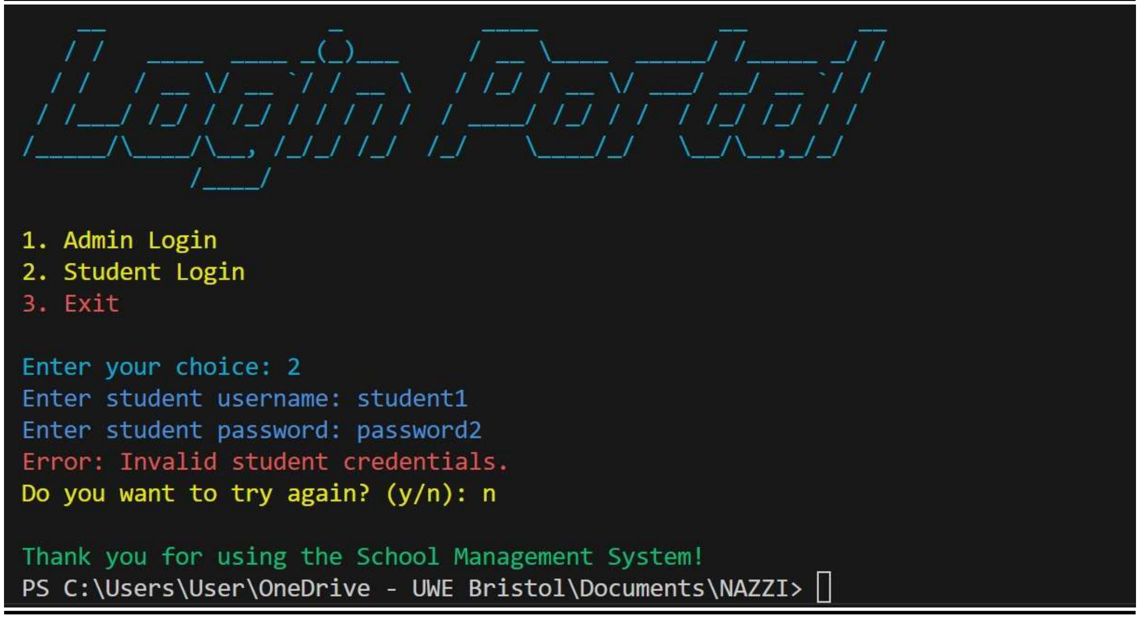
```
1. Register a new user
2. Search for user
3. Modify student record
4. Delete student record
5. View Users
6. Go to Dashboard
7. Log Out

Enter your choice: 1

    ____            _      __               _   __
   / __ \___  ____ _(_)____/ /____  _____   / | / /__ _      __
  / /_/ / _ \/ __ `/ / ___/ __/ _ \/ ___/  /  |/ / _ \ | /| / /
 / _, _/  __/ /_/ / (__  ) /_/  __/ /     / /|  /  __/ |/ |/ /
/_/ |_|\___/\__, /_/____/\__/\___/_/     /_/ |_/\___/|__/|__/
           /____/

   __  __
  / / / /_______  _____
 / / / / ___/ _ \/ ___/
/ /_/ (__  )  __/ /
\____/____/\___/_/


Enter username: Shirish
Enter role (admin/student): Admin
Enter email: mshirish24@tbc.edu.np
Enter password: 1234
Enter phone number: 9841297422
User registered successfully with ID: 5aee52b0
Registration Successful with ID: 5aee52b0
```



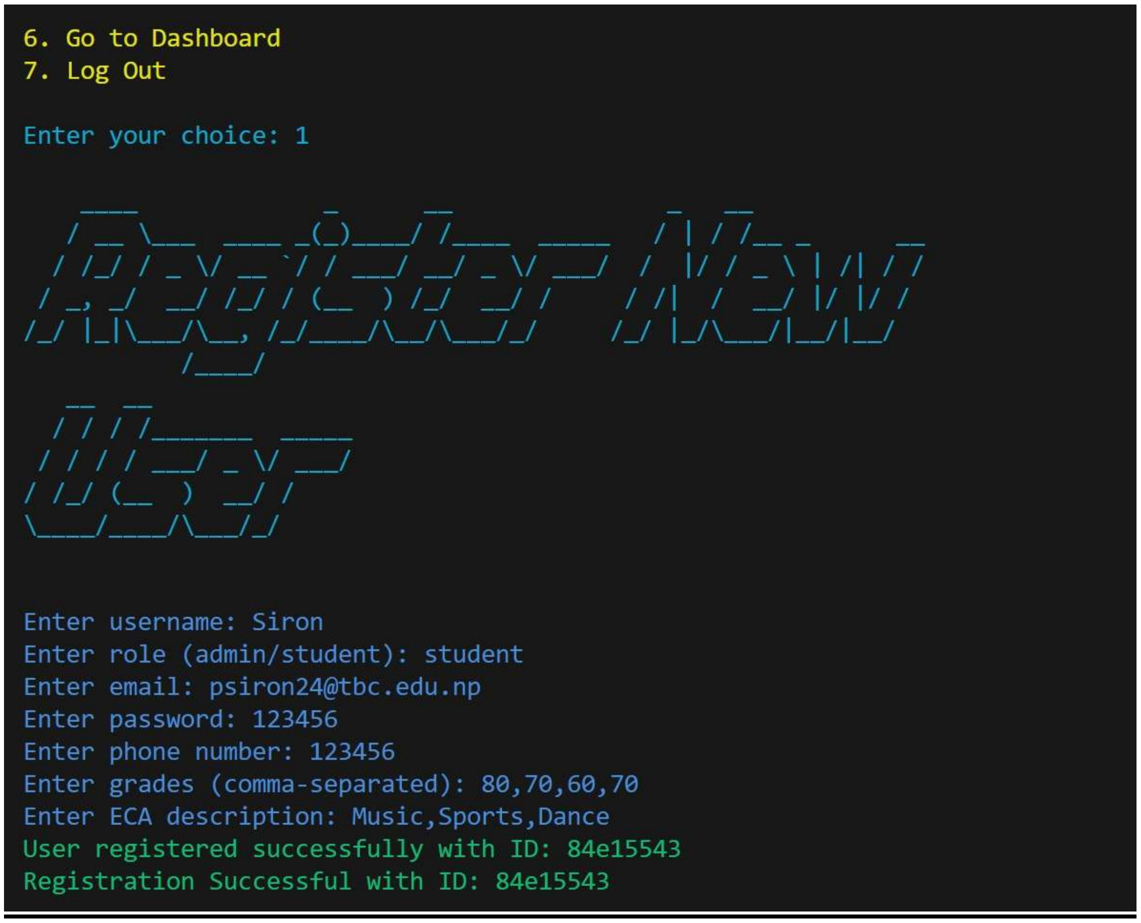
```
6. Go to Dashboard
7. Log Out

Enter your choice: 2

   _____                      __       __  __
  / ___/___  ____ ___________/ /_     / / / /_______  _____
  \__ \/ _ \/ __ `/ ___/ ___/ __ \   / / / / ___/ _ \/ ___/
 ___/ /  __/ /_/ / /  / /__/ / / /  / /_/ (__  )  __/ /
/____/\___/\__,_/_/   \___/_/ /_/   \____/____/\___/_/


Enter username to search: Ronak
+----------+-------+----------------------+----------+--------------+
| Username | Role  |        Email         | Password | Phone Number |
+----------+-------+----------------------+----------+--------------+
|  Ronak   | admin | skraunak24@tbc.edu.np |   1234   |   751f52a9   |
+----------+-------+----------------------+----------+--------------+
```



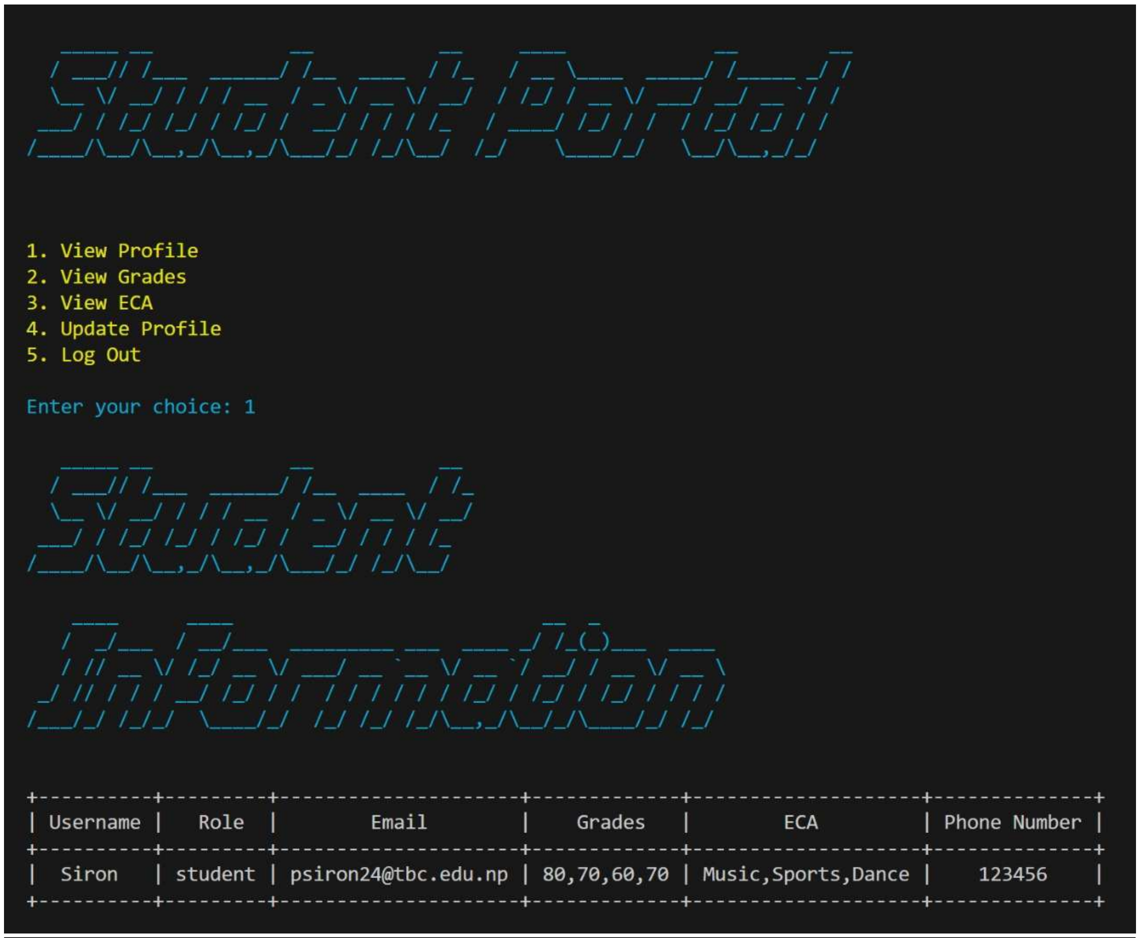
```
                                                   > & C:/Users/User/AppData/Local/Programs/Python/Python313/python
.exe "c:/Users/User/OneDrive - UWE Bristol/Documents/NAZZI/dashboard_model.py"
 !!....Welcome to Student Management System....!!

Main Menu:
1. View User Statistics
2. View Recent Activities
3. Manage Settings
4. Exit Dashboard
Enter your choice:
```



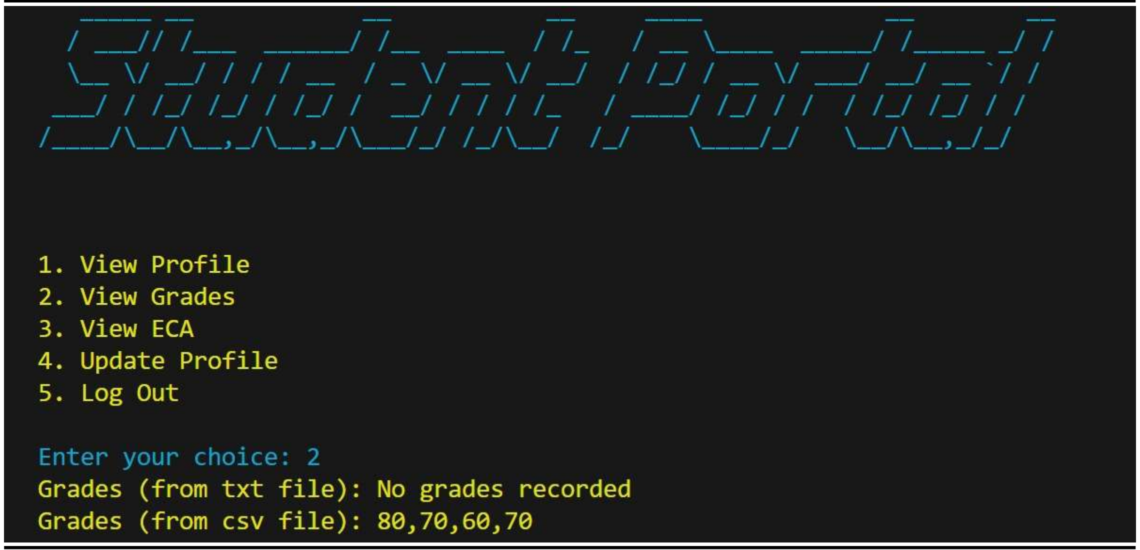
```
    __                _          ____             __        __
   / /   ____  ____ _(_)___     / __ \____  _____/ /_____ _/ /
  / /   / __ \/ __ `/ / __ \   / /_/ / __ \/ ___/ __/ __ `/ /
 / /___/ /_/ / /_/ / / / / /  / ____/ /_/ / /  / /_/ /_/ / /
/_____/\____/\__, /_/_/ /_/  /_/    \____/_/   \__/\__,_/_/
            /____/

1. Admin Login
2. Student Login
3. Exit

Enter your choice: 2
Enter student username: student1
Enter student password: password2
Error: Invalid student credentials.
Do you want to try again? (y/n): n

Thank you for using the School Management System!
PS C:\Users\User\OneDrive - UWE Bristol\Documents\NAZZI>
```



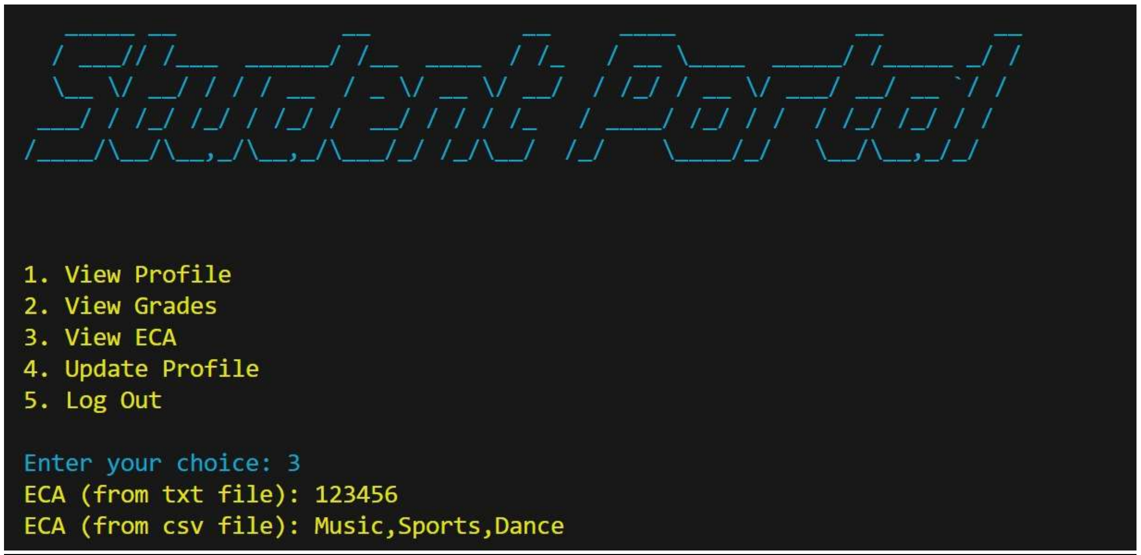
```
6. Go to Dashboard
7. Log Out

Enter your choice: 1

    ____             _      __               _   __
   / __ \___  ____ _(_)____/ /____  _____   / | / /__ _      __
  / /_/ / _ \/ __ `/ / ___/ __/ _ \/ ___/  /  |/ / _ \ | /| / /
 / _, _/  __/ /_/ / (__  ) /_/  __/ /     / /|  /  __/ |/ |/ /
/_/ |_|\___/\__, /_/____/\__/\___/_/     /_/ |_/\___/|__/|__/
           /____/
   __  __
  / / / /_______  _____
 / / / / ___/ _ \/ ___/
/ /_/ (__  )  __/ /
\____/____/\___/_/


Enter username: Siron
Enter role (admin/student): student
Enter email: psiron24@tbc.edu.np
Enter password: 123456
Enter phone number: 123456
Enter grades (comma-separated): 80,70,60,70
Enter ECA description: Music,Sports,Dance
User registered successfully with ID: 84e15543
Registration Successful with ID: 84e15543
```



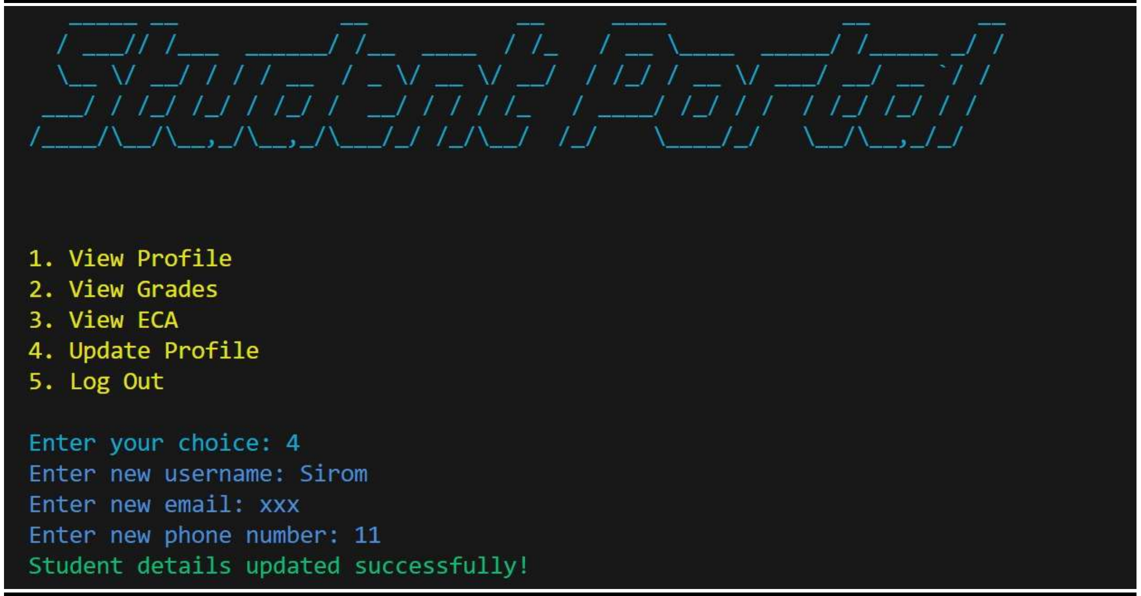
```
   _____ __            __           __     ____             __        __
  / ___// /___  ______/ /__  ____  / /_   / __ \____  _____/ /_____ _/ /
  \__ \/ __/ / / / __  / _ \/ __ \/ __/  / /_/ / __ \/ ___/ __/ __ `/ /
 ___/ / /_/ /_/ / /_/ /  __/ / / / /_   / ____/ /_/ / /  / /_/ /_/ / /
/____/\__/\__,_/\__,_/\___/_/ /_/\__/  /_/    \____/_/   \__/\__,_/_/



1. View Profile
2. View Grades
3. View ECA
4. Update Profile
5. Log Out

Enter your choice: 1

   _____ __            __           __
  / ___// /___  ______/ /__  ____  / /_
  \__ \/ __/ / / / __  / _ \/ __ \/ __/
 ___/ / /_/ /_/ / /_/ /  __/ / / / /_
/____/\__/\__,_/\__,_/\___/_/ /_/\__/

    ____      ____                           __  _
   /  _/___  / __/___  _________ ___  ____ _/ /_(_)___  ____
   / // __ \/ /_/ __ \/ ___/ __ `__ \/ __ `/ __/ / __ \/ __ \
 _/ // / / / __/ /_/ / /  / / / / / / /_/ / /_/ / /_/ / / / /
/___/_/ /_/_/  \____/_/  /_/ /_/ /_/\__,_/\__/_/\____/_/ /_/


+----------+---------+--------------------+------------+--------------------+--------------+
| Username |  Role   |       Email        |   Grades   |        ECA         | Phone Number |
+----------+---------+--------------------+------------+--------------------+--------------+
|  Siron   | student | psiron24@tbc.edu.np | 80,70,60,70 | Music,Sports,Dance |    123456    |
+----------+---------+--------------------+------------+--------------------+--------------+
```



```
   _____ __            __           __     ____             __        __
  / ___// /___  ______/ /__  ____  / /_   / __ \____  _____/ /_____ _/ /
  \__ \/ __/ / / / __  / _ \/ __ \/ __/  / /_/ / __ \/ ___/ __/ __ `/ /
 ___/ / /_/ /_/ / /_/ /  __/ / / / /_   / ____/ /_/ / /  / /_/ /_/ / /
/____/\__/\__,_/\__,_/\___/_/ /_/\__/  /_/    \____/_/   \__/\__,_/_/



1. View Profile
2. View Grades
3. View ECA
4. Update Profile
5. Log Out

Enter your choice: 2
Grades (from txt file): No grades recorded
Grades (from csv file): 80,70,60,70
```



```
   _____ __            __           __     ____             __        __
  / ___// /___  ______/ /__  ____  / /_   / __ \____  _____/ /_____ _/ /
  \__ \/ __/ / / / __  / _ \/ __ \/ __/  / /_/ / __ \/ ___/ __/ __ `/ /
 ___/ / /_/ /_/ / /_/ /  __/ / / / /_   / ____/ /_/ / /  / /_/ /_/ / /
/____/\__/\__,_/\__,_/\___/_/ /_/\__/  /_/    \____/_/   \__/\__,_/_/



1. View Profile
2. View Grades
3. View ECA
4. Update Profile
5. Log Out

Enter your choice: 3
ECA (from txt file): 123456
ECA (from csv file): Music,Sports,Dance
```



```
   _____ __            __           __     ____             __        __
  / ___// /___  ______/ /__  ____  / /_   / __ \____  _____/ /_____ _/ /
  \__ \/ __/ / / / __  / _ \/ __ \/ __/  / /_/ / __ \/ ___/ __/ __ `/ /
 ___/ / /_/ /_/ / /_/ /  __/ / / / /_   / ____/ /_/ / /  / /_/ /_/ / /
/____/\__/\__,_/\__,_/\___/_/ /_/\__/  /_/    \____/_/   \__/\__,_/_/



1. View Profile
2. View Grades
3. View ECA
4. Update Profile
5. Log Out

Enter your choice: 4
Enter new username: Sirom
Enter new email: xxx
Enter new phone number: 11
Student details updated successfully!
```