**A red and blue logo

Description automatically generated**

**Module Title**

**Fundamentals of Data Science**

**Assessment Type**

**Documentation on Final Project**

**Year**

**2024-25**

**Student Name: Asim Nath Pant, Michelle Shrestha, Pragya Ghimire**

**UWE ID: 24071188, 24071238, 24071131**

**Assignment Due Date: April 15, 2025**

**Assignment Submission Date: April 15, 2025**

# Documentation on FODS Project: Student Profile Mangement System

The program we created is a simple student profile management system where the users are prompt to choose whether they are admin or students. If the user is student, the program will take the Student user to the student menu system, where they have dynamic options to choose. They are allowed to change their name and passwords only and can view their academic progress, extra activities they are involved in and their profile details.

If the user is Admin then, the program will open the Admin menu. Like Students admins have dynamic options to choose. They also have the feature to add, delete, change, and update the student profile. The overall students and the average grades in every subject’s graphs are also presented to the admin to view the student process. Along with this they can also view how ECA is effecting the students’ progress academically.

All the data are saved in the respective text file and the changes are done according to the changes which makes it easy to keep track of the current records and details of the students. It is a user-friendly program which is easy to use and maintain in low time and cost.

1. **Class Student: Base Class**

# Necessary Libraries

import matplotlib.pyplot as plt

import pandas as pd

# ---------------------------------- Class Definitions --------------------------

class Student: #Base Class

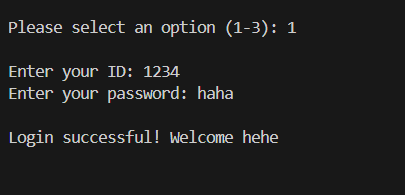
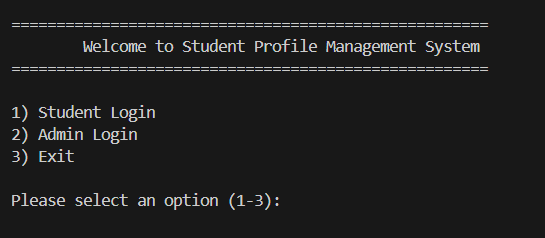
    def \_\_init\_\_(self, student\_id, name): #Constructor

        self.id = student\_id # Instance Variable

        self.name = name

        self.role = "student"

It defines the class. The student class is the main base class. It initializes new object using \_\_init\_\_ method (class constructor). The **self** parameter refers to the instance of the current class which allows us to assign values to the instance variable. The role of Student class is set into student.



1. **Student menu method:**

#-------------------------------- student menu ----------------------------------

    def student\_menu(self): #menu system for students

        while True: #loop to keep the menu open

            print("\n---------------------------------------------")

            print(f"Welcome, {self.name} (Student)")

            print("1) View Profile")

            print("2) Update Profile")

            print("3) View Grades")

            print("4) View ECA")

            print("5) Change Password")

            print("6) Logout")

            print("---------------------------------------------")

            choice = input("Enter your choice: ")

            #have dynamic options for students

            #calls the methods of the class based on the choice of the student

            if choice == "1":

                self.view\_profile()

            elif choice == "2":

                self.update\_profile()

            elif choice == "3":

                self.view\_grades()

            elif choice == "4":

                self.view\_eca()

            elif choice == "5":

                self.change\_password()

            elif choice == "6":

                print("Logging out...")

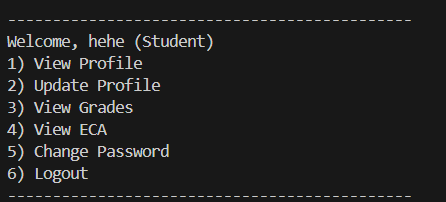
                break

            else:

                print("Invalid choice. Try again.")

The menu system opens if the log-in user is student. It calls the other methods of the class based on the choice of the user. There is dynamic option for student user to choose and to exit by choosing 6 they can log out of student menu system. If the user choose outside of the given option it’ll show Invalid choice and prompts users to choose from the given options.

**Output:**



1. **Student View Profile method:**

#---------------------------------- view profile --------------------------------

    def view\_profile(self):

        try:

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

                found\_profile = False #condition to check whether there is profile or not

                for line in file:

                    #using logical or so split the sentence

                    user\_id, name, role = line.strip().split("|") #split the line into three parts using delimiter

                    # compares the user\_id from the file with the currently logged\_in users ID

                    if user\_id == self.id:

                        print("\n-------- Your Profile --------")

                        print(f"ID     : {user\_id}")

                        print(f"Name   : {name}")

                        print(f"Role   : {role}")

                        print("----------------------------")

                        found\_profile = True   #  if it matches,it means we've found their profile

                        break

            if not found\_profile: #if profile not found after the loop

                print("Profile not found.")

        except FileNotFoundError:

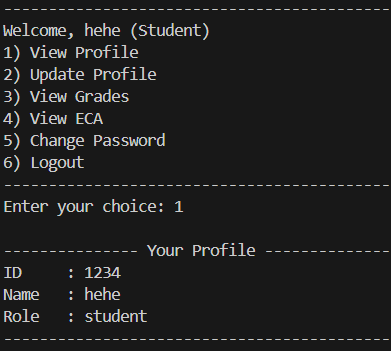
            print("Error: File not found") # if file not found, print this message

        except Exception as e:

            print(f"Error reading profile: {e}") # if any other error occurs, print this message

This method is used to show the student profile. The try and except, error handling is used to handle the errors efficiently. The with open statement it ensures that user.txt file is opened, used and closed properly without requiring manual closing. It sets the found\_profile into false as it acts as a condition to check whether there is profile or not later. It reads the file line by line, and removes white space and splits into three parts (user ID, name and role) with the sign of delimiter. If the user id in the text file is same as the logged in user id then it displays the user profile. The found\_profile is set True, as the logged in user is also in the existing txt file then it breaks the for loop. If the profile is not found then it prints the message of “Profile not found.” If the file is not found then it print error with the respective message. And if there’s other error then it prints error with the problem message.

**Output:**



**Student Update Profile:**

#--------------------------------- Update Profile-------------------------------

    def update\_profile(self): #for changing user name

        try:

            #for new name

            updated\_name = input("Enter your new name: ").strip()

            if not updated\_name: #if nothing is written

                print("Name cannot be empty.")

                return

            lines = [] #stores all lines from the file

            updated = False #to track the update status

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

                for line in file:

                    user\_id, name, role = line.strip().split("|") #splits into 3 parts using delimiter

                    if user\_id == self.id: #compares user\_id from file with logged in id(seld.id)

                        #adds the updated name into the line list

                        lines.append(f"{user\_id}|{updated\_name}|{role}\n")

                        updated = True #update successful

                        self.name = updated\_name  # update current session name too

                    else:

                        lines.append(line) #if user, just add the line as it is

            with open("users.txt", "w") as file:

                #writes all the lines in the file

                file.writelines(lines)

            if updated:

                print("Name updated successfully!")

            else:

                print("User not found.")

        except FileNotFoundError:

            print ("Error: File not found") # if file not found, print this message

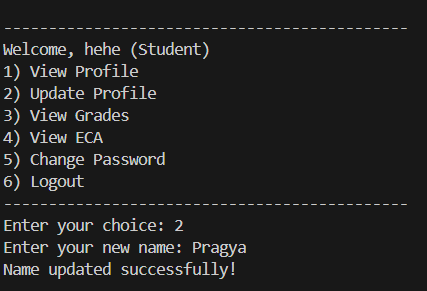
        except Exception as e:

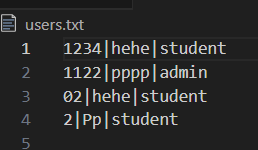
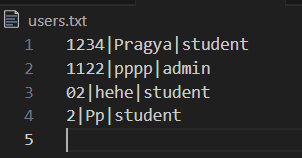
            print("Error updating profile:", e)

This method is used to change the student user name. The error handling is used in the method. It prompts user to enter the new name. If the new name is left empty then it prints the given message and returns back. With empty list data structure, it stores all the lines from the file which will be appended. The updated status is set False, it acts as a condition to know whether the name is updated or not. It opens the file in read mode and reads the line of the file. The user id, name and role is separated by delimiter. If the user id and logged in id (self.id) is same, it appends the user id, updated name and their role in the list. If the update was successful it sets the condition to true. It saves the user name into updated name.

Until the logged in ID is not found in the line it appends the line as it is in the list. Then it writes the updated name of the in the user.txt file, while other user’s id are remained same. If the update was successful it shows the given message else prints user not found. With error handling it prints the errors with respective problem.

**Output:**



1. **Student View Grades method:**

#---------------------------------- View Grades -----------------------------

    def view\_grades(self):

        #list of grades

        subjects = ["English", "Computer Fundamentals", "Programming", "Multimedia", "Database"]

        try:

            with open("grades.txt", "r") as file:

                #loops through each line in the file

                for line in file:

                    #strips spaces and newlines(.strip() and separate parts by logical or)

                    parts = line.strip().split("|")

                    #checks first line (user\_id) matches logged in users ID

                    if parts[0] == self.id: #if user\_id matches

                        print("\n------ Your Grades ------")

                        for i in range(1, 6): #loops through the grades corresponding to the subjects in the list

                            print(f"{subjects[i-1]}: {parts[i]}") #prints the respective grade from the file

                        print("-------------------------")

                        return

            print("No grades found.") # if users grade not found in the loop

        except FileNotFoundError:

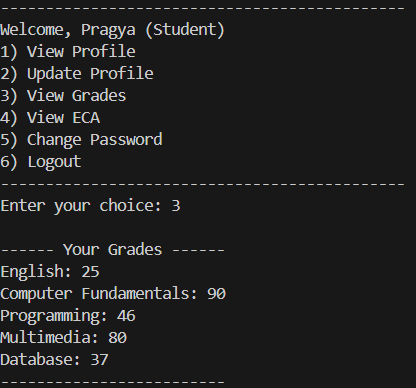
            print ("Error: File not found")

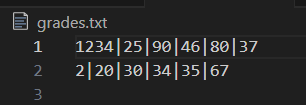
        except Exception as e:

            print("Error reading grades:", e)

This method is used to show the marks of the respective subject to the user. With error handling, it opens the grades.txt in read mode. It reads the file line and separates it in part by delimiter. The index 0 of part one is logged in user id. The for loop is run from second index to the 6th index (5th value). It prints the subject and its respective grades serially and return if the user id and logged in id is same and loops runs successfully. If the grades are not found then it prints the given message. With error handling it prints the errors with respective problem.

**Output:**





1. **Student View ECA method:**

#-------------------------------------- View ECA --------------------------------

    def view\_eca(self):

        try:

            with open("eca.txt", "r") as file:

                for line in file:

                    user\_id, activities = line.strip().split("|") #separates user\_id and activities

                    if user\_id == self.id: #Checks user\_id in the file with logged in user id

                        print("\n------ Your ECA ------")

                        print(f"Activities: {activities}") #prints the activities of the user

                        print("----------------------")

                        return

            print("No ECA activities found.") #if user's ECA not found in the loop

        except FileNotFoundError:

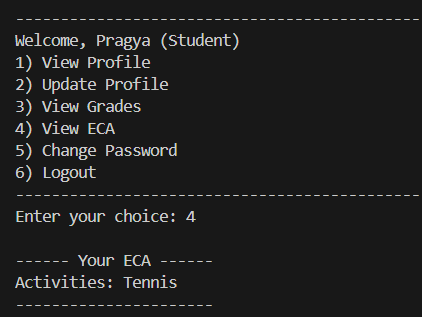
            print ("Error: File not found")

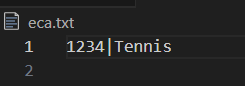
        except Exception as e:

            print("Error reading ECA data:", e)

This method is used to show the extra curriculum activates the user is involved in. With error handling, it opens the eca.txt in read mode. It reads the file line and separates the user id and activities by delimiter. If the logged in user id is present in user.txt then it shows the users ECA activities and returns back. If the user is not involved in ECA it prints the given message. With error handling it prints the errors with respective problem.

**Output:**





1. **Student Change Password method:**

#--------------------------------- Change Password -----------------------------

    def change\_password(self):

        try:

            with open("passwords.txt", "r") as file:

                lines = file.readlines() #reads all lines in the file

            updated\_lines = [] #to store modified data

            password\_changed = False #initializes the condition to check if password changed

            #ask input and removes whitspace

            current\_password = input("Enter your current password: ").strip()

            new\_password = input("Enter the new password: ").strip()

            confirm\_password = input("Confirm the new password: ").strip()

            if new\_password != confirm\_password: #if entered passwords do not match

                print("New password and confirm password do not match.")

                return

            for line in lines:

                user\_id, pwd = line.strip().split("|") #for each line in the file, splits user\_id and password

                if user\_id == self.id: #if user\_id matches logged in user id

                    if pwd != current\_password: #if entered current password is not correct

                        print("Current password is incorrect.")

                        return

                    updated\_lines.append(f"{user\_id}|{new\_password}\n")#updates the password in the file if pwd is correct

                    password\_changed = True #sets condition to True if password changed

                else:

                    updated\_lines.append(line) # if user\_id does not match, adds the line as it is

            with open("passwords.txt", "w") as file:

                file.writelines(updated\_lines)# writes the updated lines in the file

            if password\_changed:

                print("Password updated successfully!")

            else:

                print("Password update failed.") # if password not changed

        except FileNotFoundError:

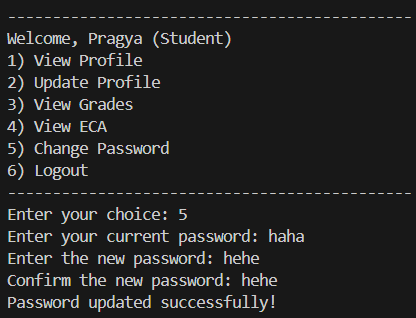
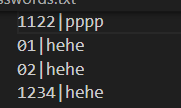
            print ("Error: File not found")

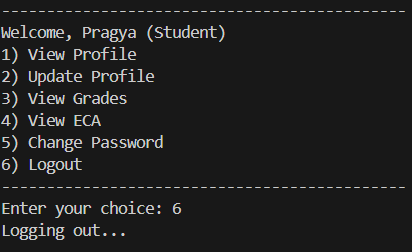
        except Exception as e:

            print("Error while changing password:", e)

The users can change the password through the help of this function. With error handling, it opens the passwords.txt in read mode. It reads the file line and separates the user id and activities by delimiter. The empty list (update\_lines) is created to store the modified data. It prompts user to enter the current password, new password and confirm the password. If the new and confirmation passwords are not same then it prints the given message and returns. And if the passwords are same then it goes through lines. It splits the user id and password. If the current password does not matches with the password set password (old pw) it shows incorrect password and returns. If the password matches with txt file it appends the user id and new password in the list and set the password-changed into True. Then it writes the updated lines (list) into the file and prints the successful message else prints failed message. With error handling it prints the errors with respective problem.

**Output:**





1. **Class Admin: Derived Class**

#------------------------------------- Admin ------------------------------------

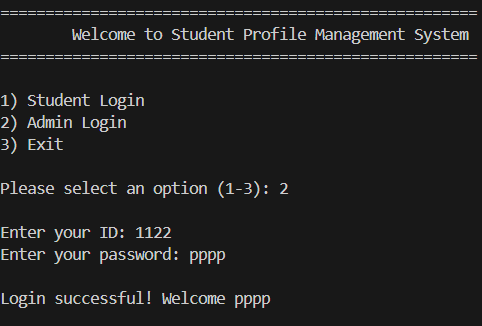
class Admin(Student): # Admin class inherits from Student class

    def \_\_init\_\_(self, admin\_id, name): # initializes admin\_id and name

        super().\_\_init\_\_(admin\_id, name) # calls the constructor of the parent class

        self.role = "admin" # sets the role of the admin as admin (in Student- student in Admin - admin)

It defines the class. The admin class the derived class,, it inherits from Student class. It initializes new object using \_\_init\_\_ method (class constructor). The **self** parameter refers to the instance of the current class, it initializes admin id and name. It calls the constructor of the parent class using super(). The role of the Admin class is set into admin.



1. **Admin menu method:**

#----------------------------------- Admin Menu ---------------------------------

   def admin\_menu(self): # menu system for Admins

        # print("Admin Menu")

        while True: # loop to keep the menu running

            print("\n================================================")

            print(f"Welcome, {self.name} (Admin)")

            print("1) Add Student")

            print("2) Delete Student")

            print("3) Edit Student Profile")

            print("4) Edit Student Grades")

            print("5) Edit Student ECA")

            print("6) Update Admin Username/Password")

            print("7) View Student Activities Analysis")

            print("8) Logout")

            print("================================================")

            choice = input("Enter your choice: ")

            # dynamic options for Admin

            #calls the methods of the class based on the choice of the admin

            if choice == "1":

                self.add\_student()

            elif choice == "2":

                self.delete\_student()

            elif choice == "3":

                self.update\_student\_profile()

            elif choice == "4":

                self.update\_student\_grades()

            elif choice == "5":

                self.update\_student\_eca()

            elif choice == "6":

                self.update\_admin\_credentials()

            elif choice == "7":

                self.show\_analytics()

            elif choice == "8":

                print("Logging out...")

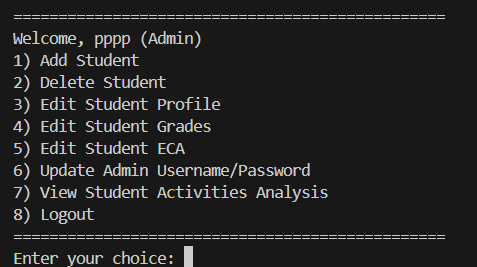
                break

            else:

                print("Invalid choice. Try again.") # if choice is not in the options

The menu system opens if the log-in user is Admin. It calls the other methods of the class based on the choice of the user. Additional method is added in Admin class. There is dynamic option for admin user to choose and to exit by choosing 8 they can log out of student menu system. The student average grade is also shown in bar graph and the comparative analysis using scatter plot. If the user choose outside of the given option it’ll show Invalid choice and prompts users to choose from the given options.

**Output:**



1. **Admin Add Students method:**

#-------------------------------- Add Students ----------------------------------

    def add\_student(self):

        try:

            # get the new student details from user

            new\_id = input("Enter new student ID: ").strip()

            new\_name = input("Enter student name: ").strip()

            new\_password = input("Enter student password: ").strip()

            # Check if student already exists

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

                for line in file:

                    # using "\_" unused variables are ignored

                    user\_id, \_, \_ = line.strip().split("|") # user\_id, name, password

                    if user\_id == new\_id: # if user id is alread in the file and is same as new id

                        print("Error: Student ID already exists.")

                        return

            # Add to users.txt

            with open("users.txt", "a") as file: # open file in append mode

                file.write(f"{new\_id}|{new\_name}|student\n") # write new student details to file (users.txt)

            # Add to passwords.txt

            with open("passwords.txt", "a") as file:

                file.write(f"{new\_id}|{new\_password}\n") # write new student password to file (passwords.txt)

            # Add to grades.txt with empty default grades

            with open("grades.txt", "a") as file:

                file.write(f"{new\_id}|0|0|0|0|0\n") # write new student grades to file (grades.txt)

            # Add to eca.txt with empty entry

            with open("eca.txt", "a") as file:

                file.write(f"{new\_id}|\n") # write new student eca entry to file (eca.txt)

            print("Student added successfully!")

        except FileNotFoundError:

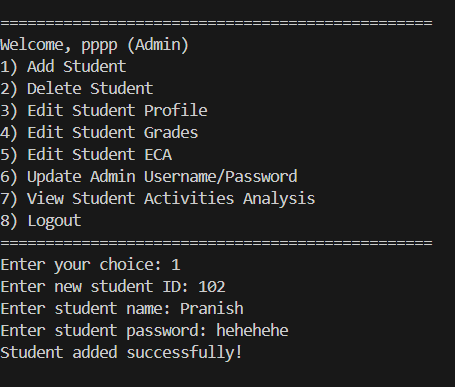
            print("Error: File not found.")

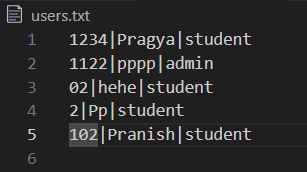
        except Exception as e:

            print("Error adding student:", e)

This method is used to add the new student id, name and password. It opens the file in read mode and reads the line of the file. It splits the id, name and password into three parts using delimiter. ( “-“ is used to ignore the unused variables.). If the given id is already in the file then it prints error message and returns back to the admin menu, else it its unique it appends the new student name, id, password, grades and eca activities in their respective files. After appending it prints successful message. With error handling it prints the errors with respective problem.

**Output:**





1. **Admin Delete Student method:**

#------------------------------------ Delete Student ----------------------------

    def delete\_student(self):

        target\_id = input("Enter student ID to delete: ").strip() # get student id from user which is to be deleted

        found = False # condition to check if student is found

    #----------------- Filter out the student from all files ----------------

        # helper function to filter out student id from file

        def filter\_out(filename): #Handles the delete process of studen id from the files (at a time)

            #nonlocal keyword is used to access outer function variable

            nonlocal found # gives acces to the found variable of its outer function (helps to modify found variable)

            try:

                with open(filename, "r") as f:

                    lines = f.readlines() # read all lines from file

                with open(filename, "w") as f: #overwrites its content

                    for line in lines:

                        if not line.startswith(target\_id + "|"): # if line does not start with target id

                            f.write(line) # write line back to the file

                        else:

                            found = True # if line starts with target ID it skips writing the line and deletes that line

            except:

                print(f"Could not access {filename}") # if file not accessable

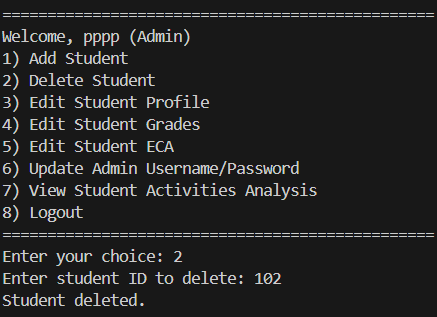
        for file in ["users.txt", "passwords.txt", "grades.txt", "eca.txt"]: # searches for the student id in all the files

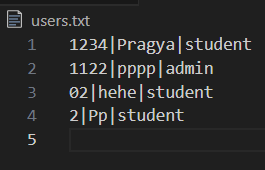
            filter\_out(file) # filter out (deletes) student id from each file

        print("Student deleted." if found else "Student ID not found.") # conditional expression to print student deleted or student id not found

This method allows Admin to delete the student’s details from the text file. It prompts the Admin to input the targeted student id to be deleted and sets the condition (found) to False, as it acts as a tracker to check the student id. It uses sub method for deleting the student details from all the text file. The Found (condition) is made non-local for accessing the outer variable of the method. It opens the files in read mode and reads all the lines from the file. Then it is opened in write mode to overwrite the contents and delete the targeted student details. It goes through every line and searches the targeted user id. If the user id is not found in the lines it writes the original line back to the file else it writes nothing in the file if its found. With error handling if the file could not be accessed it prints the respective error message. It go through all the text file and filters out (deletes) the targeted students details. Finally, it prints the deleted message if successful else unsuccessful message if ID not found using conditional expression.

**Output:**





1. **Admin Update Student Profile:**

#------------------------------- Update Student Profile -----------------------

    #allows updating the student names based on Student ID

    def update\_student\_profile(self):

        target\_id = input("Enter student ID to update: ").strip() # get student id from user which is to be updated

        new\_name = input("Enter new name: ").strip() # prompts new name from user

        updated = False # condition to check if student is updated

        try:

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

                lines = file.readlines() # read all lines from file and stores in list variable lines

            with open("users.txt", "w") as file: # reopens the .txt & overwrites its content

                for line in lines:

                    user\_id, name, role = line.strip().split("|") # removes whitespace and splits the line into three variables using delimiter

                    if user\_id == target\_id and role == "student": # if user id matches with target id and role is student

                        file.write(f"{user\_id}|{new\_name}|{role}\n") # updates the name in the file replacing old name

                        updated = True #sets true if updated successfully

                    else:

                        file.write(line) #writes the original line back to the file

            print("Name updated." if updated else "Student not found.") #uses conditional expression to check if updated or not

        except FileNotFoundError:

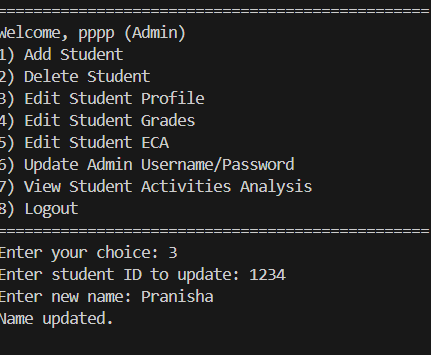
            print("Error File not found.")

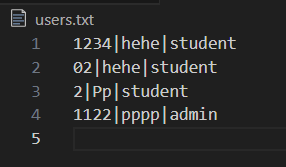
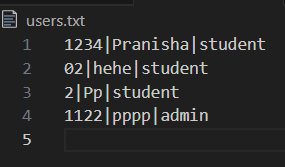
        except Exception as e:

            print("Error updating profile:", e)

This method allows Admin to update the student names based of the student id. It prompts the user to input the targeted student id and the new name of the student. The update status is set False for checking the condition. With try and except it handles the error handling efficiently. It reads the line of the file. Then opens the user.txt in write mode. The user id, name and role is separated by delimiter. If the user id matches the targeted id and the role is student then it changes the old name by new name and sets the status into True ,while other are remained unchanged or written same. It uses the conditional expression to print the message as per the updated status. With error handling it prints the errors with respective problem.

**Output:**





**.txt file after updating**

**.txt file without updating**

1. **Admin Update Students Grades method:**

#----------------------------- Update Student Grades ----------------------------

    def update\_student\_grades(self): # allows updating the student grades based on Student ID

        target\_id = input("Enter student ID: ").strip() # get student id from user which is to be updated

        # subjecta for which grade is to be updated

        subjects = ["English", "Computer Fundamentals", "Programming", "Multimedia", "Database"]

        new\_grades = [] # list to store new grades of student

        for subject in subjects: # iterates over each subject

            while True:

                try:

                    mark = int(input(f"Enter marks for {subject}: ")) # prompts user to enter marks for each subject

                    if 0 <= mark <= 100: # checks if marks are valid (between 0 and 100)

                        new\_grades.append(mark) # appends the marks to the list

                        break

                    else:

                        print("Enter a value between 0 and 100.")

                except:

                    print("Invalid input.")

        updated = False # condition to check if student is updated

        try:

            with open("grades.txt", "r") as file: #to access its contents

                lines = file.readlines() # read all lines from file and stores in list variable lines

            with open("grades.txt", "w") as file: # reopens the .txt & overwrites its content

                for line in lines:

                    if line.startswith(target\_id + "|"): # if line starts with target id followed by delimiter

                        file.write(f"{target\_id}|" + "|".join(map(str, new\_grades)) + "\n") #converts into strring and joins the using delimiter

                        updated = True #sets true if updated successfully

                    else:

                        file.write(line) #writes the original line back to the file

            print("Grades updated." if updated else "Student not found.") #uses conditional expression to check if updated or not

        except FileNotFoundError:

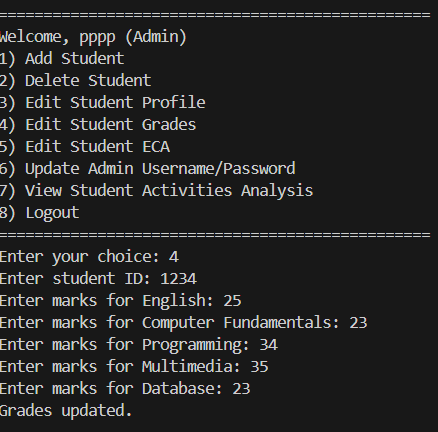
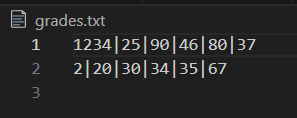
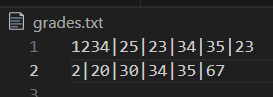
            print("Error File not found.")

        except Exception as e: # catches any exceptions that may occur

            print("Error updating grades:", e)

This method allows Admin to update the grades of the students respective to the subjects. It ask input from the Admin to enter the targeted user id and creates an empty list to store the grades of the students. It loops through the subjects and ask to enter the marks .If it’s in between 0-100 it appends the marks and break, if not it prints the given message. Then it stores the updated marks in the grade.txt file after reading the lines. It updates the marks of the targeted users and writes the other student’s marks as it was. Then it uses the conditional expression to print the message. With error handling it prints the errors with respective problem.

**Output:**



**.txt file after updating**

**.txt file without updating**

1. **Admin Update Student ECA method:**

--------------------------------- Update Student ECA ------------------------

    def update\_student\_eca(self):

        target\_id = input("Enter student ID: ").strip() # get student id from user which is to be updated

        new\_activities = input("Enter updated ECA activities (comma separated): ").strip() # get new eca activities from user and removes whitespace

        updated = False # condition to check if student is updated

        try:

            with open("eca.txt", "r") as file: #to access its contents

                lines = file.readlines() # read all lines from file and stores in list variable lines

            with open("eca.txt", "w") as file: # reopens the .txt and overwrites its content

                for line in lines:

                    if line.startswith(target\_id + "|"): # if line starts with target id followed by delimiter

                        file.write(f"{target\_id}|{new\_activities}\n") #writes the updated eca activities in the targeted id

                        updated = True #sets true if updated successfully

                    else:

                        file.write(line) #writes the original line back to the file

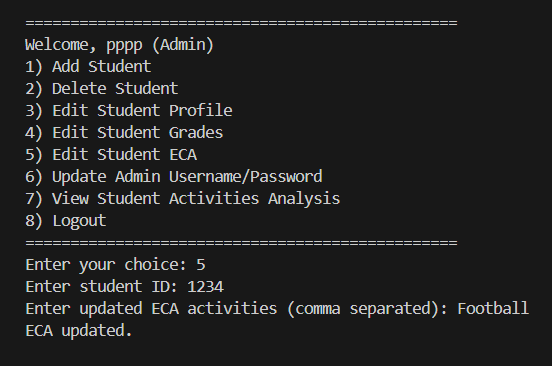
            print("ECA updated." if updated else "Student not found.") #uses conditional expression to check if updated or not

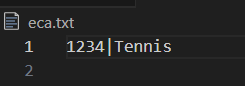
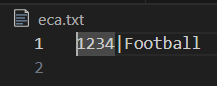
        except Exception as e:

            print("Error updating ECA:", e)

The Admin can update the Extra Curricular Activities of the students through Student ID. This method prompts the Admin to input the targeted student id and the new activities they are involved in. The update status is set False to check the condition. The file is opened in read mode to access its contents. And in write mode it searches the targeted user ID and writes the new activities they are involved in the eca.txt file. Then sets the update status to true (which means successful). It writes back the other students details as it is. With conditional expression it prints the message according to the status. With error handling it prints the errors with respective problem.

**Output:**





**.txt file after updating**

**.txt file without updating**

1. **Admin Update Credentials method:**

#----------------------------- User Admin Credentials --------------------------

    def update\_admin\_credentials(self): # method to update admin credentials

        # get new admin credentials from user

        new\_id = input("Enter new admin ID: ").strip()

        new\_name = input("Enter new admin name: ").strip()

        new\_password = input("Enter new password: ").strip()

        old\_id = self.id  # get current admin id

        # stores existing ID in the variable old\_id  before changing it

        # Update users.txt

        try:

            with open("users.txt", "r") as file: #to access its contents

                lines = file.readlines() # read all lines from file and stores in list variable lines

            with open("users.txt", "w") as file: # reopens the .txt and overwrites its content

              for line in lines:

                parts = line.strip().split("|") # splits the line into parts using delimiter

                if len(parts) != 3: # if line does not contain 3 parts (id,name,role)

                    file.write(line)  # Write it back unchanged

                    continue  # Skip to the next line

                #to ensure that the only the records of admin is updated

                user\_id, name, role = parts  # extracts from the current records (parts): user id, name, and role

                if user\_id == old\_id and role == "admin": # if current admin id matches with old id and role is admin

                    file.write(f"{new\_id}|{new\_name}|admin\n")  # updates the admin credentials

                    self.id = new\_id  #updates attributes of the admin object with new values

                    self.name = new\_name

                else:

                  file.write(line) # Write it back unchanged if record doesnot match

        except FileNotFoundError:

            print("Error File not found.")

        except Exception as e:

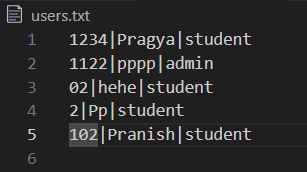
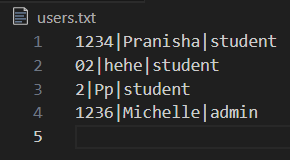
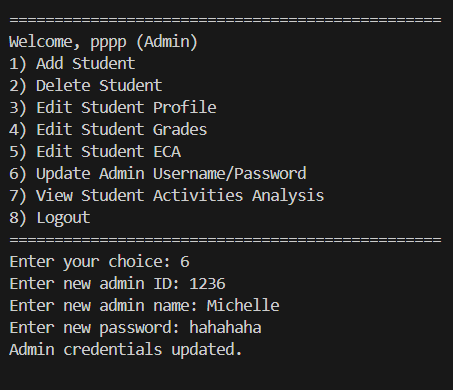
          print("Error updating user file:", e)

It allows the Admin to change the id, name and password. It is the additional feature for admin to add new id, name and password replacing the old one. Old id is assigned the logged in user id. Thrn in try except it opens the user.txt file in read mode to access the content of the file. Then it opens the file in write mode It separates the line in three parts (id, name and role) using delimiter. If one of them is missing it writes back as it and skips it to the next line. If the parts and user.txt matches and if the role is admin then it writes the new id and name in place of old into the file. It writes backs the other details except the targeted user back as it originally was. With error handling it prints the errors with respective problem.

**Update password in text file:**

It opens the password.txt file in read mode and reads all the lines of the files. Then it opens the .txt in write mode. It loops through all the lines and splits the user id and password using delimiter. If the user id in text file is not same as input user id then it writes it back without changing. If it matches then it overwrites the new user id and new password of the admin and keeping other user details back as it was. It prints the message if the update was successful. With error handling it prints the errors with respective problem.

**Output:**

****

**.txt file after updating**

**.txt file without updating**

1. **Admin View Analytics method:**

#----------------------- Show Students Activity Analytics -----------------------

 def show\_analytics(self):

        try:

            df\_grades = pd.read\_csv("grades.txt", sep="|", header=None) # reads grades.txt file into a pandas dataframe

            df\_grades.columns = ["ID", "English", "CompFund", "Programming", "Multimedia", "Database"] # assigns column names to the dataframe

            df\_grades.set\_index("ID", inplace=True) # sets ID as index of the dataframe

            df\_eca = pd.read\_csv("eca.txt", sep="|", header=None, names=["ID", "Activities"]) # reads eca file into a pandas dataframe

            df\_eca.set\_index("ID", inplace=True)

            # 1. Grade Trend (Average per subject)

            subject\_avg = df\_grades.mean() # calculates average of each subject

            subject\_avg.plot(kind='bar', title="Average Grades per Subject", ylabel="Marks", xlabel="Subjects") # plots a bar chart of average grades per subject

            plt.tight\_layout() # ensures labels fit within the figure area

            plt.show() # displays the plot

            # 2. ECA Impact

            df\_grades["Average"] = df\_grades.mean(axis=1) # calculates average of each student

            df\_eca["ECA\_Count"] = df\_eca["Activities"].fillna("").apply(lambda x: len(x.split(",")) if x else 0) # counts the number of activities each student has done

            df\_combined = df\_grades.join(df\_eca, how='left').fillna(0) # combines the two dataframes on ID and fills missing values with 0

            plt.scatter(df\_combined["ECA\_Count"], df\_combined["Average"], color='green') # plots a scatter plot of ECA count vs average grade

            plt.title("ECA Participation vs Academic Performance") # sets title of the plot

            plt.xlabel("Number of ECAs") # sets x-axis label

            plt.ylabel("Average Grade") # sets y-axis label

            plt.grid(True) # displays grid lines

            plt.tight\_layout() # ensures labels fit within the figure area

            plt.show()

            # 3. Performance Alerts (Below threshold)

            threshold = 40 # sets the threshold for performance alert

            alerts = df\_combined[df\_combined["Average"] < threshold] # identifies students with average grade below threshold

            if not alerts.empty: # if there are students with average grade below threshold

                print("\n Students Below Performance Threshold (Avg < 40):")

                for student\_id in alerts.index: # prints student IDs with average grade below threshold

                    print(f"- {student\_id}: Avg = {alerts.loc[student\_id, 'Average']:.2f}") # prints student ID and average grade

            else:

                print("\n All students are above the performance threshold.") # prints message if all students are above threshold

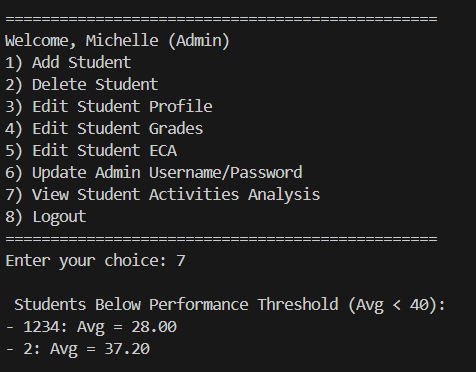
        except FileNotFoundError: # handles file not found error

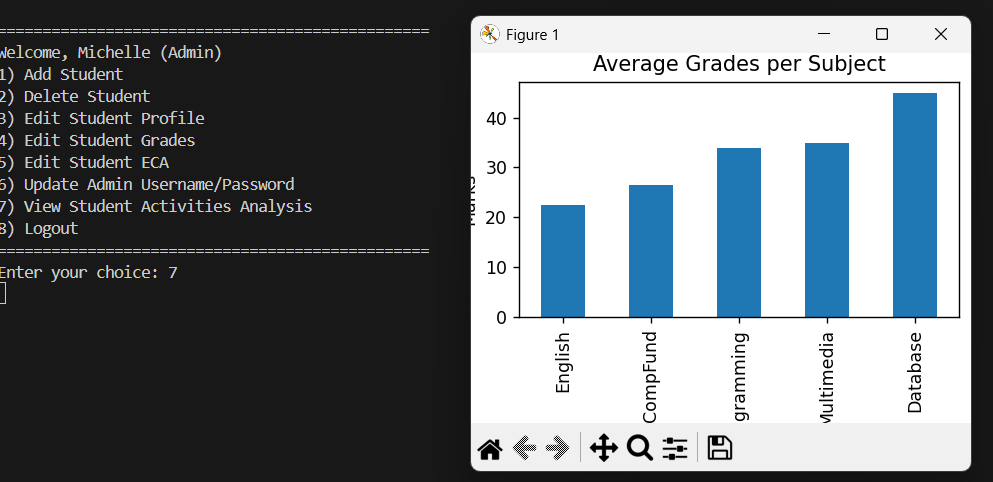
            print("Missing 'grades.txt' or 'eca.txt'. Please ensure files exist.") # prints error message

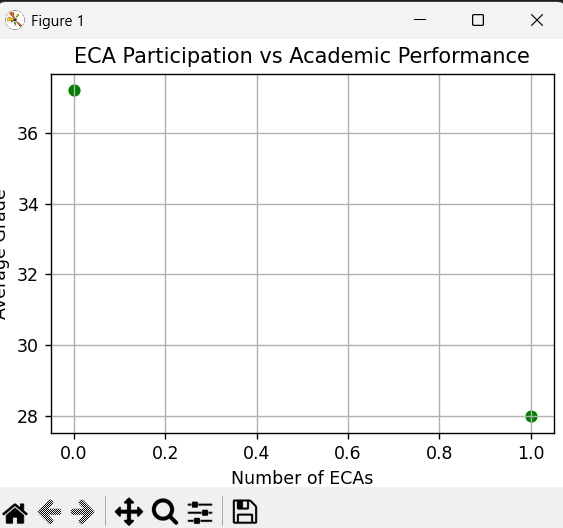
        except Exception as e:

            print("Analytics Error:", e)

This methods allows users to view the analytical report of the students through graphs. Df\_grades reads the panda grades.txt and eca.txt csv file separated by delimiter. Then it assigns the column names to the data frame and set ID as the index respectively. At first, it calculates the students average marks in each subjects. Then it plots it into a bar chart of average grades of every subject ensuring the labels fit within the figure area. It also calculates the average grade of each students and the number of eca activities they are involved in. Then combines them on id then sets the title, x-axis, y-axis and grid. It ensures all labels are fitted in the figure area and shows the plot. Finally, it sets the threshold for performance alert. If the students average grade is below threshold it prints the given message, it also show students id with the average grade. Else, if all students’ performance are above average, it shows the given message. With error handling it prints the errors with respective problem.

**Output:**

****

****

1. **Extra Utility Functions:**
2. **Load User function:**

           #----------------load users ------------------

def load\_users():

    users = {} # dictionary to store users

    try:

        with open("users.txt", "r") as file: # opens the file in read mode

            for line in file:

                user\_id, name, role = line.strip().split("|") # splits the line into parts using delimiter

                users[user\_id] = {"name": name, "role": role} # stores user id (key) and its details (value) in dictionary

    except FileNotFoundError:

        print("Error: 'users.txt' not found.")

    return users # returns the dictionary of users

It checks if the user.txt file exist or not. It creates the empty dictionary to store the user’s detail. It opens the user.txt file in read mode then go through the loop in the file line by line. It splits the user\_id, name and role using the delimiter and removes the white space. After splitting it adds the user id as key and the name and role as values in the user’s dictionary. If file is not found it prints the error message. It returns back the users dictionary storing the id with its values (name, role).

1. **Load Password function:**

             #-------------- load passwords ---------------

def load\_passwords():

    passwords = {} # dictionary to store passwords

    try:

        with open("passwords.txt", "r") as file:

            for line in file:

                user\_id, password = line.strip().split("|") # splits the line into parts using delimiter

                passwords[user\_id] = password # stores user id (key) and its password (value) in dictionary

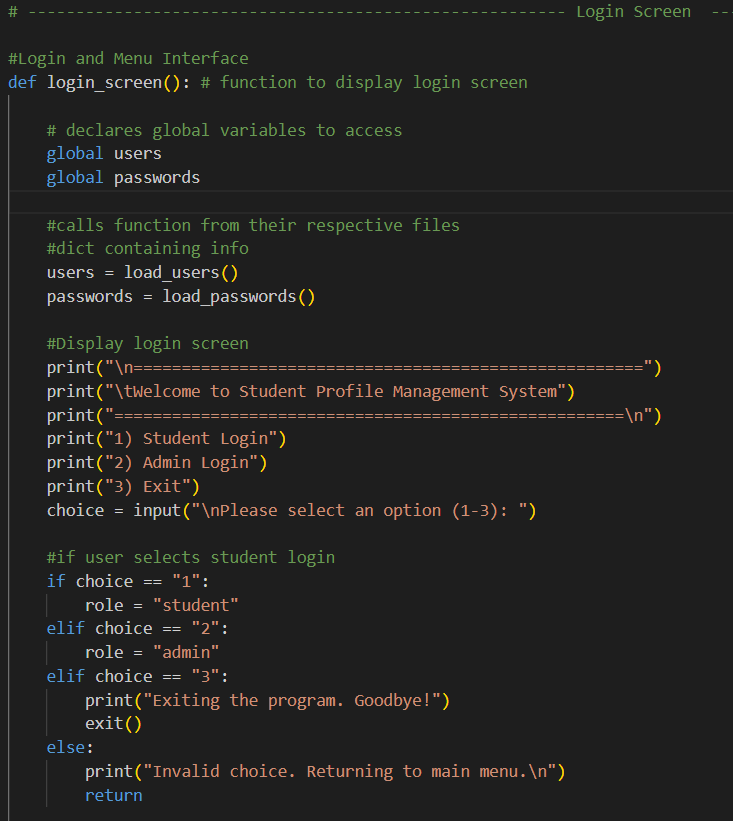
    except FileNotFoundError:

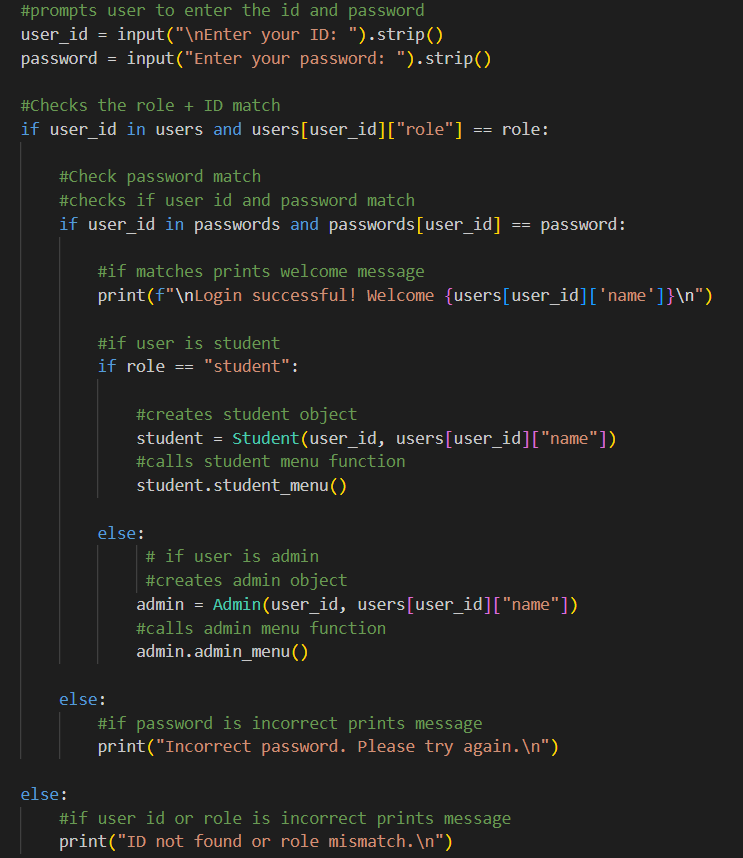
        print("Error: 'passwords.txt' not found.")

    return passwords # returns the dictionary of passwords

It checks if the password.txt file exist or not. It creates the empty dictionary to store the user’s password detail. It opens the password.txt file in read mode then go through the loop in the file line by line. It splits the user\_id, and password using the delimiter and removes the white space. After splitting it adds the user id as key and password as its values in the passwords dictionary. If file is not found it prints the error message. It returns back the passwords dictionary storing the id (key) and password (its value).

1. **Login Screen function:**





The login\_screen() function is responsible for logging users (either students or the admin) into the system. It first loads user data and passwords from text files using helper functions. Then, it displays a menu asking the user to choose between Student Login, Admin Login, or Exit.

Once the user selects a role, the function asks for their ID and password. It checks if the ID exists, matches the selected role, and if the password is correct. If everything is valid, the user is welcomed and directed to their respective menu: students go to student\_menu() and the admin goes to admin\_menu().

If the ID doesn’t exist, or the role doesn’t match, or the password is wrong, the program displays an appropriate error message and returns to the login screen. This function ensures that only valid users can access and interact with the system based on their roles.

1. **Main Function:**

# ----------------------------- Main Program Loop -------------------------------

def main(): # main function

    while True:

        login\_screen() # call login\_screen function to get into the program

The main () function acts as the primary logic of the program. It contains an infinite loop which executes the login interface until the user exits the overall program. It allows users to access the overall program.

1. **Entry point:**

# ------------------------ Entry Point of the Program ------------------------

if \_\_name\_\_ == "\_\_main\_\_": # This is the entry point of the program

    main() # call main function to start the program

The \_\_name\_\_ is a special build-in variable in Python which ensures that the script only runs the main function when the program is executed directly. This checks whether the script is being executed directly and runs the code if the condition is true. The conditional expression means the script is running as the main program. It calls the main function to start the program.