**Student Management System Report**

# **Task:**

The task is to develop a Student Management System that allows tutors to manage students, courses, and grades. The system should allow tutors to enter new students, see a student's courses, show all users, add student grades, see students on a specified module, see students under 40% grade average, and show students not yet graded. Students should also be able to log in to view their grades.

# **Algorithm:**

Define the initial data structure for users, courses, and assessments.

Define functions for entering a new user, seeing courses, adding grades, logging in, getting students for a specified module, getting students under 40% grade average, and getting ungraded students.

Implement a while loop for user login and verification.

If the user is a student, display their grades.

If the user is a tutor, display the main menu with options for managing students, courses, and grades.

Perform the selected action based on the tutor's choice.

Repeat steps 3-6 until the tutor chooses to exit the system.

# **Flow Chart:**

Diagram, schematic

Description automatically generated

# **Technical Overview:**

## **Variables:**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Data Type** | **Description** |
| username | string | Unique identifier chosen by a user to access their account. |
| password | string | Secret code chosen by a user to protect their account. |
| name | string | Full name of the user |
| student\_status | string | Indicates whether the user is a student or not |
| courses | dictionary | Contains information about each course |
| course\_title | string | The name of a course |
| assessments | list | A list of assessments in a course |
| grades | list | A list of grades for each assessment |
| Weightings | list | A list of weightings for each assessment |
| Users | list | A list of dictionaries containing information about all users |

## **Functions:**

|  |  |  |  |
| --- | --- | --- | --- |
| Function Name | Arguments | Return Type | Description |
| enterUser() | None | None | Allows the tutor to enter a new student and their course information. |
| getUser() | userName: str, users: list | tuple | Returns the record of the specified student identified by userName from the list of users. The tuple contains a dictionary of the user's information and a boolean value indicating whether the user was found. |
| seeCourses() | None | None | Displays the courses taken by a specified student, identified by their username entered through user input. |
| addGrades() | None | None | Allows the tutor to add grades to a student's assessments by identifying the student through user input and inputting grades for each assessment. |
| login() | None | dict | Allows the user to log in to the system by entering their login name and password through user input. Returns a dictionary containing the user's information if the login is successful. |
| get\_students() | None | list | Retrieves a list of the names of students enrolled in a specified module, identified through user input. |
| get\_students\_under\_40() | None | list | Retrieves a list of the names of students who have an average grade below 40%. |
| get\_students\_not\_graded() | None | list | Retrieves a list of the names of students who have ungraded assessments. |
| display\_grade() | student\_id: int | None | Displays the grades for the specified student identified by student\_id entered through user input. |

## **Data Structures:**

**Dictionary**: Used for storing user, course, and assessment information.

**List**: Used for storing a collection of user dictionaries.

## **Testing:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Input | Expected Output | Actual Output | Comments |
| seeCourses() - valid student | student's username: "johndoe" | list of courses taken by the student | list of courses taken by the student | The function successfully displayed the expected list of courses taken by the student. |
| seeCourses() - invalid student | student's username: "invalidname" | "invalidname is not a registered student." | "invalidname is not a registered student." | The function successfully displayed the expected message. |
| addGrades() - valid student | student's username: "johndoe", grades for assessments | updated dictionary of student's course information | updated dictionary of student's course information | The function successfully added the inputted grades to the specified student's assessments. |
| addGrades() - invalid student | student's username: "invalidname", grades for assessments | "user not found" | "user not found" | The function successfully displayed the expected message. |
| get\_students() | module code: "CS101" | list of students enrolled in the module | list of students enrolled in the module | The function successfully retrieved and returned the expected list of students. |
| get\_students\_under\_40() | None | list of students with an average grade below 40% | list of students with an average grade below 40% | The function successfully retrieved and returned the expected list of students. |
| get\_students\_not\_graded() | None | list of students with ungraded assessments | list of students with ungraded assessments | The function successfully retrieved and returned the expected list of students. |
| display\_grade() - valid student | student's username: "johndoe" | dictionary of the student's grades | dictionary of the student's grades | The function successfully displayed the expected dictionary of the student's grades. |
| display\_grade() - invalid student | student's username: "invalidname" | None | None | The function successfully returned None as the specified student was not found. |

# **Summary:**

The Student Management System allows tutors to manage students, courses, and grades effectively. The system provides options for adding new students, viewing student courses, adding grades, and generating reports based on specified criteria. The code has been organized into functions for better modularity and reusability. The data structures used are dictionaries and lists, which provide a simple and efficient way to store and manipulate data. The system has been tested to ensure it meets the desired requirements and provides a user-friendly experience for both students and tutors.

# **Code:**

import copy

import json

import pickle

courses = {'FC308': {'Ass 1': {'Grade': -1, 'weighting': 0.1},

'Ass 2': {'Grade': -1, 'weighting': 0.4},

'Exam': {'Grade': -1, 'weighting': 0.5}},

'FC315': {'Ass 1': {'Grade': -1, 'weighting': 0.4},

'Ass 2': {'Grade': -1, 'weighting': 0.6}},

'FC311': {'Ass 1': {'Grade': -1, 'weighting': 0.1},

'Ass 2': {'Grade': -1, 'weighting': 0.4},

'Exam': {'Grade': -1, 'weighting': 0.5}},

'EXTPRJ': {'Ass 1': {'Grade': -1, 'weighting': 0.4},

'Ass 2': {'Grade': -1, 'weighting': 0.6}},

'FC300': {'Ass 1': {'Grade': -1, 'weighting': 0.1},

'Ass 2': {'Grade': -1, 'weighting': 0.5},

'Presentation': {'Grade': -1, 'weighting': 0.4}}}

data\_path = 'data.txt'

# parse the data file and create a dictionary

def parse\_data(filename):

userdata = []

# read the data file

with open(filename, 'r') as file:

for line in file.readlines():

# user data

user = {

'User Name': line.split(",")[0],

'Password': line.split(",")[1],

'Name': line.split(",")[2],

'Student': line.split(",")[3],

'Courses': {}

}

# add the student to the courses dictionary

userdata.append(user)

for course\_grades in line.split(",")[4:]:

course, grades = course\_grades.split('-', 1)

grades = grades.split('|')

assignments = {}

# grade assignment list

for grade in grades:

assignment, grade\_value = grade.split(':')

assignments[assignment.strip()] = {

'Grade': int(grade\_value),

'weighting': float(courses[course][assignment.strip()]['weighting'])

}

# add the assignments to the course

user['Courses'][course] = assignments

# return the dictionary

return userdata

# write the data to the file

def write\_data(filename, users):

with open(filename, 'w') as f:

for i, user in enumerate(users):

# write the user data to the file

user\_data = [users[i]['User Name'], users[i]['Password'],

users[i]['Name'], str(user['Student'])]

for course, assignments in user['Courses'].items():

# write the course data to the file

grades = '|'.join(

[f"{assignment}:{grade['Grade']}" for assignment, grade in assignments.items()])

user\_data.append(f"{course}-{grades}")

# write the user data to the file

f.write(','.join(user\_data) + '\n')

# read the data file

users = parse\_data(data\_path)

# Enter the new student name

def enterUser():

while True:

newCourse = {}

newUser = copy.deepcopy(users[0])

newUser['User Name'] = input('Enter user name: ')

user\_exists = False

for user in users:

if user['User Name'] == newUser['User Name']:

user\_exists = True

break

# if user\_exists

if user\_exists:

print('User already exists Please choose another user name')

continue

else:

# password

newUser['Password'] = input('Enter password: ')

user\_input = ''

while user\_input not in ['S', 'T']:

user\_input = input('(S)tudent/(T)utor?: ').upper()

# if user\_input not in ['S', 'T

if user\_input == 'S':

newUser['Student'] = True

elif user\_input == 'T':

newUser['Student'] = False

else:

# Invalid input

print(

'Invalid input. Please enter either "S" for student or "T" for tutor.')

continue

while True:

# enter name

try:

name = input('Enter name: ')

if len(name) < 3:

raise ValueError

newUser['Name'] = name

break

# if name is not valid

except ValueError:

print('Please enter a valid name with at least 3 characters.')

continue

newUser['Courses'] = {}

while True:

course = input(

'Enter course title (Type \'end\' when finished): ')

# if course is lower than end

if course.lower() != 'end':

newCourse[course] = {}

while True:

# enter assessment

assessment = input(

'Enter assessment title (Type \'end\' when finished): ')

if assessment.lower() != 'end':

while True:

try:

# enter grade

grade = int(

input('Enter grade for ' + assessment + ': '))

break

# if grade is not valid

except ValueError:

print('Please enter a valid number.')

continue

while True:

try:

# enter weighting

weighting = float(

input('Enter weighting for ' + assessment + ': '))

break

# if weighting is not valid

except ValueError:

print('Please enter a valid number.')

continue

newCourse[course][assessment] = {

'Grade': grade, 'weighting': weighting}

else:

break

else:

break

# add new course

newUser['Courses'] = newCourse

users.append(newUser)

write\_data(data\_path, users)

print('User added successfully!')

if input('Enter another user (y/n)? ').lower() == 'n':

break

# Enter the new student name

def getUser(userName, users):

user = {}

userFound = False

for u in users:

if u['User Name'] == userName and u['Student']:

user = u

userFound = True

break

if not userFound:

print(f"No student user found with user name '{userName}'")

return user, userFound

# see courses and grades for a student

def seeCourses():

while True:

# enter user name

userName = input(

'Enter student user name (Type "end" when finished): ')

if userName == 'end':

break

user, userFound = getUser(userName, users)

if userFound:

# Check if user has any courses

courses = user['Courses']

for course in courses:

if course in courses:

print(f"Course: {course}")

for a in courses[course]:

# print out the assessment name, grade and weighting

print(

f"{a}: Grade={courses[course][a]['Grade']}, weighting={courses[course][a]['weighting']}")

score = 0

weight = 0

# calculate the weighted score

for a in courses[course]:

score += courses[course][a]['Grade'] \* \

courses[course][a]['weighting']

weight += courses[course][a]['weighting']

weighted\_score = score / weight

print(f"Accumulated score:{weighted\_score}\n")

else:

print(f"{userName} is not a registered student.")

def addGrades():

while True:

print('Add Grades')

# enter user name

userName = input('Enter student user name: (or "end" to quit): ')

if userName == 'end':

break

# get user and check if user exists

user, userFound = getUser(userName, users)

if userFound:

# enter grades

for c in user['Courses']:

for a in user['Courses'][c]:

while True:

try:

# enter updated grade for assessment

print(f'enter grade for {a} on {c}: ', end='')

g = int(input())

break

# if grade is not valid

except ValueError:

print('Please enter a valid number.')

continue

user['Courses'][c][a]['Grade'] = g

# write data to file

write\_data(data\_path, users)

print('Grades updated successfully!')

print(user['Courses'])

else:

print('user not found')

# Login function

def login():

attempts = 3

while True:

if attempts == 0:

print('Maximum attempts reached. Program will now exit.')

quit()

userFound = False

currentUser = {}

# Get user input

logName = input('Enter login name: ')

logPass = input('Enter Login password: ')

# Check if user exists

for k, i in enumerate(users):

if users[k]['User Name'] == logName and users[k]['Password'] == logPass:

currentUser = users[k]

userFound = True

break

# If user exists, break loop

if userFound:

break

else:

attempts -= 1

print('Invalid username or password. Please try again. {attempts} attempts left.\n'.format(

attempts=attempts))

return currentUser

# Get students in a module

def get\_students():

module = input('Enter module code: ')

students = []

for user in users:

# if user is student and module code is in user's courses

if user['Student'] and module in user['Courses']:

students.append(user['Name'])

return students

# Student unger 40 grade

def get\_students\_under\_40():

students = []

for user in users:

if user['Student']:

total = 0

for course in user['Courses']:

for assessment in user['Courses'][course]:

total += user['Courses'][course][assessment]['Grade']

if total / len(user['Courses']) < 40:

students.append(user['Name'])

return students

# Student graded

def get\_students\_not\_graded():

students = []

for user in users:

# if user is student

if user['Student']:

ungraded\_count = 0

for course in user['Courses']:

for assessment in user['Courses'][course]:

if user['Courses'][course][assessment]['Grade'] == -1:

ungraded\_count += 1

break

# if user has ungraded assessment

if ungraded\_count > 0:

students.append(user['Name'])

return students

# Display grade

def display\_grade(current\_user):

if current\_user['Student'] == "True":

print('Displaying Grades:\n')

user = current\_user

courses = user['Courses']

for course in courses:

print(f"Course: {course}")

# print out the assessment name, grade and weighting

for a in courses[course]:

print(

f"{a}: Grade={courses[course][a]['Grade']}, weighting={courses[course][a]['weighting']}")

score = 0

weight = 0

# calculate the weighted score

for a in courses[course]:

score += courses[course][a]['Grade'] \* \

courses[course][a]['weighting']

weight += courses[course][a]['weighting']

weighted\_score = score / weight

print(f"Accumulated score:{weighted\_score}\n")

else:

# If user is not a student

print('Access denied. Only students are allowed to view grades.')

def logout():

print('Logging out...')

print('Goodbye!')

quit()

while True:

curUser = login()

if curUser['Student'] == "True":

while True:

print()

# Student Grade Menu

print('---- Student Grade ----')

print('''

Press 1 to see your grades.

Press 2 to Logout

''')

choice = input()

# See grades

if choice == '1':

display\_grade(curUser)

# Logout

elif choice == '2':

logout()

else:

while True:

print()

# Student Management System Menu

print('---- Student Management System ----')

print('''

Press 1 to enter new student.

Press 2 to see a students courses.

Press 3 to show all users

press 4 to add student grades

press 5 to see Students on a specified module

press 6 to see students under 40% grade average

press 7 to show students not yet graded

press 8 to Logout

''')

choice = input()

# Enter new student

if choice == '1':

enterUser()

# See a students courses

elif choice == '2':

seeCourses()

# Show all users

elif choice == '3':

print(json.dumps(users, indent=4))

# Add student grades

elif choice == '4':

addGrades()

# See students on a specified module

elif choice == '5':

print(get\_students())

# See students under 40% grade average

elif choice == '6':

print(get\_students\_under\_40())

# See students not yet graded

elif choice == '7':

# See students not yet graded

print(get\_students\_not\_graded())

elif choice == '8':

logout()