DSA Minor Project - Institute Seat Allocation System For New Admits

---

Team Member Details:

Member 1 -
Name: Shubhang S. Galagali
Roll No.: 231MT049
Mobile No.: 94820-31368
E-mail ID: shubgalagali.231mt049@nitk.edu.in

Member 2 -
Name: Ashish Manash
Roll No.: 231CH010
Mobile No.: 91429-81741
E-mail ID: ashishmanash.231ch010@nitk.edu.in

---

1. Defining Roles and Permissions for Users:

In [77]:
```
class Role:
    ADMIN = "admin"
    INSTITUTE = "institute"
    STUDENT = "User is a student"
```

2. Creating a User Database:

In [79]:
```
class User:
    def __init__(self, username="", password="", role=""):
        """

        Args:
          username:
          password:
          role:
        """
        self.username = username
        self.password = password
        self.role = role

# User database (in-memory for simplicity)
# Objects under the class User are created, wherein we have assigned each participating institute an username, and the same as the password.

users = {
    "admin": User("admin", "DSAPROJECTCS202M", Role.ADMIN),
    "NITK": User("NITK", "NITK", Role.INSTITUTE),
    "NITT": User("NITT", "NITT", Role.INSTITUTE),
    "NITC": User("NITC", "NITC", Role.INSTITUTE),
    "IITB": User("IITB", "IITB", Role.INSTITUTE),
    "IITM": User("IITM", "IITM", Role.INSTITUTE),
}
```

3. Creating a Student Database:

In [81]:
```
# Define the Student class
class Student:
    """

    Attributes:
      username:
      password:
      rank:
      preferences:
      allocated:
      role:
    """
    def __init__(self, username, password, rank, preferences, role):
        self.username = username
        self.password = password
        self.rank = rank
        self.preferences = preferences  # List of tuples like [(institute, branch), ...]
        self.allocated = None  # To store allocated seat as (institute, branch)
        self.role = role
    # Define a method to return student details.
    def __repr__(self):
        return f"{self.username},{self.password},{self.rank},{self.preferences},{self.role}"
```

In [82]:
```
import random
# We are using the random command to generate random ranks and preferences.


'''
Student database - currently initialised for simplicity in a python dictionary,but it can
be taken as input from  the user during registration and can be stored in a database, as in a .csv file.

Currently, the username and the password of the student is initialised to their registration number, starting from 24101 upto 24199.
Also we have assumed 300 students took up the common entrance examination, so minimum rank is 1, and maximum rank is 300 (numerically).
'''

ranks = []
def generate_ranks():
    while(len(ranks) != 300):
        for i in range(1, 301):
            rank = random.randint(1,300)
            if rank in ranks: # Ensuring all elements of 'ranks' are unique.
                continue
            else:
                ranks.append(rank)

    #print(len(ranks))
    #print(len(set(ranks))) # Cross-checking if all elements of 'ranks' are unique.

    return ranks

ranks = generate_ranks()
```

In [83]:
```
'''
Writing a fucntion to generate preferences for each student (doing this due to lack of database management).
Thus hadnling of invalid preferences is taken care of in the backend logic.
'''

def generate_realistic_preferences():
    institutes = ["NITK", "NITT", "NITC", "IITB", "IITM"]
    branches = ["EC", "ME"]  # EC: Electronics and Communication Engineering, ME: Mechanical Engineering.
    preferences = []
    no_of_preferences = random.randint(3,5)
    for _ in range(no_of_preferences): # We are allowing a minimum of 3 & a maximum of 5 preferences to each student.
        # Randomly select an institute and a branch
        institute = random.choice(institutes)
        branch = random.choice(branches)
        preferences.append((institute, branch))
    return preferences
```

In [84]:
```
'''
Using dictionary data structure to store the student details, and an advantage is hashing while using dictionaries.

We are assuming 99 students registred for the Institute Seat Allocation process
out of 300 students who gave the common entrance examination.

Thus the IDs/Usernames of students range from 24001 upto 24199.
'''

students = {}
def generate_student_data():

    for i in range(24101, 24200):

        j = i - 24101 ## j will be the index of the rank in the list 'ranks' which contains ranks from 1 to 300 in a random order.
        preferences=generate_realistic_preferences() # calling the 'generate_realistic_preferences()' function for each student.
        students[i] = Student(i, i, ranks[j], preferences, role=Role.STUDENT)

    return students
```

In [85]:
```
'''
Calling the 'generate_student_data()' function for each student, to assign ranks and preferences to each student
who has registered for the Institute Seat Allocation System.
'''

students = generate_student_data()
```

4. Backend Logic for User Login:

In [87]:
```
'''
Implementing a login function which checks if the username and password entered by the user matches the data stored/initialised
in the system under 'users' dictionary as objects of the 'User' class.
'''

def login(username, password, role):
    if role == Role.STUDENT:
        user = students.get(username)
        if user and user.password == password:
            print("\nLogin Successful!\n")
            return True
        print("Invalid username or password!! Please try again.")
        return False
    else:
        user = users.get(username)
        if user and user.password == password and user.role == role:
            print("Login successful!")
            return True
        print("Invalid username or password!! Please try again.")
        return False
```

5. Backend Logic for New User Registration Process:

In [89]:
```
'''
Implementing a register_user() function which creates a new object
in the system under 'users' dictionary as objects of the 'User' class, and initialises using the input given by the user.

Currently, this new registered user cannot participate in the allocation process, but in the next version of the project, we shall implement in such
a manner where using database management tools, new users can be allowed to register and participate in the allocation process.
'''



def register_user(username, password, role):
    if username not in students:
        rank = int(input("Please enter your rank: "))
        if rank < 1 or rank > 300:
            print("Invalid rank. Rank should be between 1 and 300.") #Checking if rank is valid as per assumptions made under Section 3.
            return " "
        preferences = input("Please enter your preferences! ")
        students[username] = Student(username, password, rank, preferences, role=Role.STUDENT)
        print("Registration successful!")
    else:
        print("Username already exists.")
    return " "
```

6. Implementation of a Priority Queue for sorting the students based on their rank:

In [91]:
```
'''Priority Queue to sort students based on ascending order of ranks'''

# Priority Queue implementation
```

```
class PriorityQueue:
    def __init__(self):
        self.queue = []

    def is_empty(self):
        return len(self.queue) == 0

    def enqueue(self, student_id, rank):
        # Insert student based on rank to maintain sorted order
        index = 0
        while index < len(self.queue) and self.queue[index][1] <= rank:
            index += 1
        self.queue.insert(index, (student_id, rank))

    def dequeue(self):
        # Remove the student with the highest priority (lowest rank)
        if not self.is_empty():
            return self.queue.pop(0)
        return None
```

7. Seat Allocation System Logic:

In [93]:
```
'''
Defined the seats for each institute and branch.
We have taken 5 Engineering institutions and have assumed 2 branches per institute and 10 seats per branch.

EC - Electronics and Communication Engineering, ME - Mechanical Engineering
'''

# Defiing seat capacities for institutes

seat_capacity = 10

institutes = {
    "NITK": {"EC": seat_capacity, "ME": seat_capacity},
    "NITT": {"EC": seat_capacity, "ME": seat_capacity},
    "NITC": {"EC": seat_capacity, "ME": seat_capacity},
    "IITB": {"EC": seat_capacity, "ME": seat_capacity},
    "IITM": {"EC": seat_capacity, "ME": seat_capacity}
}
```

In [94]:
```
# Initialize the priority queue and populate it with students based on their rank

priority_queue = PriorityQueue()
for student_id, student in students.items():
    priority_queue.enqueue(student_id, student.rank)
```

In [95]:
```
# Dictionary to store final seat allotments for each student
seat_allotments = {student_id: None for student_id in students}

# Seat allocation process
while not priority_queue.is_empty():
    student_id, _ = priority_queue.dequeue()
    student = students[student_id]
    preferences = student.preferences

    # Allocate seat based on preferences
    for institute, branch in preferences:
        if institutes[institute][branch] > 0:  # Check seat availability
            seat_allotments[student_id] = (institute, branch)
            students[student_id].allocated = (institute, branch)
            institutes[institute][branch] -= 1  # Reduce seat count
            break  # Stop after assigning the first available preference
```

8. Defining Functions for Dashboard of various User Roles.

In [97]:
```
def admin_ddashboard():
                # Admin dashboard - Can view the whole data stored in the system including students' passwords, Institute details
                # and the Final Seat Allotments of each student.

                print(f"\n Student Database - ADMIN access:\n{students}")
                print(f"\n\n List of participating institutes:\n IITB, \n IITM, \n NITC, \n NITK, \n NITT. \n")
                print(f"\n Final Seat Allotments of each student:\n{seat_allotments}")
                return " "
```

In [98]:
```
 # Institute dashboard contains the students admitted into the respective institute, under each branch with the respective rank of the student.

def institute_dashboard(institute_username):
                # Initialize the institute_allotments dictionary
                institute_allotments = {}

                # Populate the institute_allotments dictionary
                for student_id in seat_allotments.keys():
                  if seat_allotments[student_id] == None: continue
                  else:
                        institute, branch = seat_allotments[student_id]
                        if institute == institute_username:
                          # Get the rank for the student_id
                          rank = students[student_id].rank
                          # Create a institute_allotments entry for the branch if it doesn't exist
                          if branch not in institute_allotments:
                            institute_allotments[branch] = []
                          # Append the student_id and rank tuple to the list for the branch
                          institute_allotments[branch].append((student_id, rank))

                # Output the institute_allotments dictionary
                print(institute_allotments)
                return " "
```

In [99]:
```
# Student Dashboard shows the Student ID, the respective rank, filled preferences and the alotted seat.

def student_dashboard(student_username):
                      print(f'''Student ID: {student_username},\n Student Rank: {students[student_username].rank},
                      \n Student Preferences: {students[student_username].preferences},\n Allocated Seat: {students[student_username].allocated}''')
                      return " "
```

9. Main Execution Function:

In [101…
```python
#main()

def main():
    # We want our project interaction window to run till the user wants to exit the window.
    # Using while loop for that.
    flag = True  # As long as flag is True, the loop runs, thus the project window is active.
    while flag:

        # Asking the user, the role to be assigned.

        print("\n\n\n\n\n\n1. Admin Login (A)")
        print("2. Institute Login (I)")
        print("3. Student Register/Login (S)")
        print("4. Exit (E)")

        choice = input("\nChoose an option: ")

        #Asking the user to login using the correct credentials to give access to specific data of the program as per the role of the user.

        #The admin role:
        if choice in ['A', 'a']:
            admin_username = input("Enter admin username: ")
            admin_password = input("Enter admin password: ")
            if login(admin_username.lower(), admin_password, Role.ADMIN):
                print("\n Admin dashboard \n")
                print(admin_ddashboard())

        #The institute role:
        elif choice in ['I', 'i']:
            institute_username = input("Enter institute username: ")
            institute_password = input("Enter institute password: ")
            if login(institute_username, institute_password, Role.INSTITUTE):
                print("\n Institute dashboard \n")
                print(institute_dashboard(institute_username))



        #The student role:
        elif choice in ['S', 's']:
            student_username = int(input("Enter student username: "))
            if student_username > 24300 or student_username < 24101: #300 students took the exam, with IDs 24101 - 24400.
                print("Invalid username. Please try again.")
                continue
            if student_username in students:
                student_password = int(input("Enter student password: "))
                if login(student_username, student_password, Role.STUDENT):
                    print("\n Student dashboard \n")
                    print(student_dashboard(student_username))
            else:
                #If the student ID is not present in the database, add the details by creating a new user.
                print("Welcome Student! Create a new password for your account!")
                student_password = int(input("Enter student password: "))
                register_user(student_username, student_password, Role.STUDENT)
                print("\nKindly re-login to access your account!\n")

        #If the user is satified with the program and its results, the step step would be to exit the window:
        elif choice in ['E', 'e']:
            print("Exiting...")
            flag = False

        #If an input other than the given roles/exit is given by the user:
        else:
            print("Invalid choice.")
```

In [ ]:
```python
# Calling the main function to run the project code.

main()
```

```






1. Admin Login (A)
2. Institute Login (I)
3. Student Register/Login (S)
4. Exit (E)
```