DATA STRUCTURES AND ALGORITHMS

CS202M

PROJECT REPORT

INSTITUTE SEAT ALLOCATION SYSTEM FOR NEW ADMITS

Submitted by:

Shubhang S. Galagali — **231MT049**

Ashish Manash — **231CH010**

Submitted to:

Prof. Annappa B.

Professor,

Department of Computer Science and Engineering,

NITK Surathkal.

National Institute of Technology Karnataka [NITK], Surathkal.

NH 66, Srinivasnagar, Surathkal, Mangalore, Karnataka- 575025.

INSTITUTE SEAT ALLOCATION SYSTEM FOR NEW ADMITS

Team Members:

Name:	Shubhang S. Galagali	Ashish Manash
Roll No.:	231MT049	231CH010
Mobile No.:	94820-31368	91429-81741
E-mail ID:	shubgalagali.231mt049@nitk.edu.in	ashishmanash.231ch010@nitk.edu.in

Source Code starts from Page-12

1. Problem Statement: (taken as was in the project proposal submitted)

'Develop a program which handles the task of seat allotment to students in an institute, according to the student's priority, among a list of participating institutes, accepting admissions through a common entrance examination.'

2. Introduction:

This Institute Seat Allocation System is designed to manage the allocation of seats for students based on their preferences, ranks, and available seats across institutes based on a common entrance examination criterion. This system is an attempt to simulate a real-world application of an admission process in educational institutions where students are ranked and seats are allocated based on the obtained ranks.

3. Primary Objectives:

The primary objectives of the project are:

- To develop a role-based access system for secure handling of users including Admin, Institute, and Student roles.
- To create a student and institute database, allowing for efficient data management and retrieval.
- To maintain a priority queue structure for effective sorting based on student ranks.
- To implement a seat allocation mechanism that respects student rank and preferences.
- To build a functional, interactive dashboard for each user role (Admin, Institute, and Student).

4. Design Overview: The project is structured into various components that support role management, data storage, rank-based sorting, preference handling, and allocation processes.

Each component is outlined below:

4.1 Defining Roles and Permissions

The project defines roles with specific permissions:

- **Admin**: Has full access to the system, including viewing all user data, institutes, and final seat allotments.
- **Institute**: Can view students admitted to the institute under different branches, along with student ranks.
- **Student**: Can register, login, and view their details, preferences, and allocated seats.

4.2 User and Student Database

- **User Database**:
  Uses a dictionary data structure to manage user credentials (username and password) for different roles.
- **Student Database**:
  Stores information on students' ranks and preferences in a python dictionary. Preferences are stored as a list of institute-branch pairs, and random rank assignments simulate a real entrance exam scenario.

Note: In python, Dictionaries are preferred for database-like operations for several key reasons:

- Key-Value pairs provide natural mapping for real-world data.
- Hash table implementation implies memory efficiency, no need of contiguous memory like arrays.
- Unique keys prevent duplicates
- Easy to maintain references, add/remove fields.
- O(1) time complexity in the average case.

4.3 Student Preferences and Rank Generation

The snippet generates:

- **Ranks**: Unique, randomized ranks for each student, simulating a real-world rank distribution scenario.
- **Preferences**: Realistic preferences for each student, with students allowed to select a minimum of 3 and a maximum of 5 choices among different institutes and branches.

Note: 4.3 was undertaken due to lack of a real database of students and their choices.

4.4 Priority Queue Implementation for Rank-Based Sorting

A priority queue organizes students based on ranks, ensuring that higher-ranked students are processed first during seat allocation. This implementation maintains the sorted order of students in ascending rank, with lower ranks receiving higher priority.

4.5 Seat Allocation System

The seat allocation system considers each student's preferences in order. For each student in the priority queue:

- The system checks the seat availability in the preferred institute and branch.
- If a seat is available, the student is allocated to that branch in that institute, and the seat count is decremented.

This approach prioritizes high-ranked students' choices and allocates seats based on their preferences as much as possible.

4.6 Dashboard Functions

The project implements three different dashboards, allowing users to interact with the system based on their role:

- **Admin Dashboard**: Displays all student data, institute details, and final seat allotments – all using Python dictionaries.
- **Institute Dashboard**: Shows allocated students for the respective institute, organized by branch and student rank, in a python dictionary.
- **Student Dashboard**: Displays the student's ID, rank, preferences, and allocated seat, if any.

---

5. Implementation:

The following outlines the project's core implementation aspects.

5.1 Role-Based Login System

Only users with valid credentials can access their respective dashboards. 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.

5.2 Registration System for New Users

A registration function enables new student users to register in the system. 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.

5.3 Priority Queue for Student Sorting

The priority queue is implemented as a list-based insertion sort, allowing rank-based organization. Each student ID and rank pair is enqueued, maintaining ascending order of ranks to ensure efficient allocation processing.

PriorityQueue Class:

- Initialization (_init_): Initializes an empty list (queue) to store student data.
- Check if Empty (is_empty): Returns True if the queue is empty, allowing easy checking of the queue's state.
- Enqueue Method (enqueue): Takes student_id and rank as parameters. Inserts a student into the queue while maintaining sorted order based on rank. This is achieved by iterating through the queue to find the correct position and then using list.insert() to add the student.
- Dequeue Method (dequeue): Removes and returns the student with the highest priority (lowest rank).
- If the queue is empty, it returns None; otherwise, it pops the first element from the list.

5.4 Seat Allocation Logic

Seats are defined for each participating institute and branch, with seat capacities initialized to 10 per branch. There are 5 participating institutes, all of them being engineering institutions, namely, IITB, IITM, NITC, NITK, and NITT. Each institute is assumed to have 2 branches, namely EC (Electronics and Communication Engineering) and ME (Mechanical Engineering. The allocation algorithm processes students in priority order, assigning seats to the highest-ranking students based on their top available preferences. The priority queue gives the ID of the student having the highest priority, i.e., numerically the lowest rank in the queue, that student's preferences are fetched from the students' database, and the algorithm for allocation runs until either the student is allocated a seat as per his ordered preference or exists if either preferences are exhausted or all seats of each preference is already filled.

5.5 Dashboard Functionality

Each role has specific access permissions:

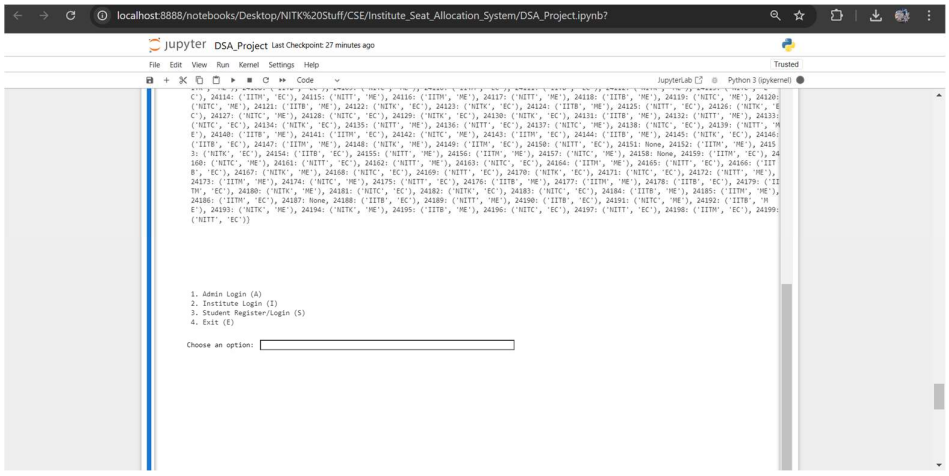
- Admin Dashboard: Accesses the entire student database, participating institutes and final seat allotments. This is done by calling and printing the dictionary that holds the student details, and the final seat allotment results.
- Institute Dashboard: Views admitted students for specific institutes. This is achieved by creating a new dictionary from the seat allotments dictionary where only those students who have obtained the respective institute are filtered using the institute username and displayed.
- Student Dashboard: Allows students to view their ranks, preferences, and allocated seats. This is implemented by calling the specific items held in the dictionary under the key represented by the respective student usernames.

Note: **Student database** – It has been 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 the minimum rank is 1, and maximum rank is 300 (numerically), with valid student IDs going from 24101 upto 24400.

---

4. Testing: PTO

All testcases were run locally on Jupyter Notebooks, on a Lenovo-IdeaPad-Laptop.



Testcases showcasing Registering a new student, ID outside of 24101 to 24199 initialised by default (above) and Logging into that newly registered account.

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

Choose an option:  S
Enter student username:  24200
Welcome Student! Create a new password for your account!
Enter student password:  24200
Please enter your rank:  255
Please enter your preferences!  [('NITK', 'EC'), ('NITT', 'ME')]
Registration successful!

Kindly re-login to access your account!
```

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

Choose an option:  S
Enter student username:  24200
Enter student password:  24200

Login Successful!


 Student dashboard

Student ID: 24200,
 Student Rank: 255,

 Student Preferences: [('NITK', 'EC'), ('NITT', 'ME')],
 Allocated Seat: None
```

Testcase showing logging in using the Admin role – All students data, participating institutes and the final seat allotments are displayed.

Testcases showing New Student Registration – Invalid Rank and Exit Option.

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

Choose an option:  S
Enter student username:  24230
Welcome Student! Create a new password for your account!
Enter student password:  24230
Please enter your rank:  500
Invalid rank. Rank should be between 1 and 300.

Kindly re-login to access your account!
```

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

Choose an option:  E
Exiting...
```

Testcases showing Logging in as a student and accessing allotted seat (above) and Invalid Password (below).

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

Choose an option:  S
Enter student username:  24116
Enter student password:  24116

Login Successful!


 Student dashboard

Student ID: 24116,
 Student Rank: 266,

 Student Preferences: [('IITM', 'ME'), ('IITM', 'EC'), ('IITM', 'ME'), ('NITC', 'ME')],
 Allocated Seat: ('IITM', 'ME')
```

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

Choose an option:  S
Enter student username:  24168
Enter student password:  24160
Invalid username or password!! Please try again.
```

Testcases showing Login of Student and accessing his allotted seat (upper portion) and Invalid Password for Institute Login (below).

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

Choose an option:  S
Enter student username:  24101
Enter student password:  24101

Login Successful!


 Student dashboard

Student ID: 24101,
 Student Rank: 247,

 Student Preferences: [('NITT', 'ME'), ('NITT', 'EC'), ('IITB', 'EC'), ('NITC', 'EC')],
 Allocated Seat: ('NITT', 'EC')
```

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

Choose an option:  I
Enter institute username:  NITK
Enter institute password:  NITT
Invalid username or password!! Please try again.
```

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

Choose an option:  I
Enter institute username:  NITK
Enter institute password:  NITK
Login successful!

 Institute dashboard

{'ME': [(24102, 156), (24104, 261), (24107, 126), (24112, 139), (24148, 192), (24167, 299), (24180, 32), (24193, 179), (24194, 151)], 'EC': [(24122,
56), (24123, 61), (24126, 71), (24129, 147), (24130, 215), (24134, 148), (24145, 214), (24153, 124), (24170, 53), (24182, 284)]}
```

Testcase showing successful Institute Login and Institute Dashboard (above).

Testcases showing invalid Role choice, successful institute login (IITM) and invalid Admin Password, successful institute login (IITB) (in respective order below).

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

Choose an option:  IITM
Invalid choice.
```

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

Choose an option:  I
Enter institute username:  IITM
Enter institute password:  IITM
Login successful!

 Institute dashboard

{'EC': [(24103, 210), (24110, 298), (24114, 185), (24141, 65), (24143, 43), (24149, 5), (24159, 227), (24179, 190), (24186, 20), (24198, 292)], 'M
E': [(24116, 266), (24147, 119), (24152, 132), (24156, 31), (24164, 24), (24173, 18), (24177, 107), (24185, 170)]}
```

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

Choose an option:  A
Enter admin username:  admin
Enter admin password:  dsapro
Invalid username or password!! Please try again.
```

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

Choose an option:  I
Enter institute username:  IITB
Enter institute password:  IITB
Login successful!

 Institute dashboard

{'EC': [(24105, 33), (24108, 246), (24111, 144), (24146, 13), (24154, 123), (24166, 175), (24178, 191), (24188, 300), (24190, 101)], 'ME': [(24118,
6), (24121, 208), (24124, 42), (24131, 200), (24140, 184), (24144, 23), (24176, 141), (24184, 78), (24192, 212), (24195, 112)]}
```

SOURCE CODE FOR THE PROJECT

GitHub Link: https://github.com/galava-shubhang/Institute_Seat_Allocation_System/

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:

```python
class Role:
    ADMIN = "admin"
    INSTITUTE = "institute"
    STUDENT = "User is a student"
```

[4] Python

2. Creating a User Database:

```python
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.
```

```python
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),
}
```

Python

3. Creating a Student Database:

```python
# 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}"
```

Python

```python
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):
```

```python
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()
```

Python

```python
'''
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
```

Python

```python
'''
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
```

```python
'''
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()
```

Python

4. Backend Logic for User Login:

```python
    '''
    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)
[14]        if user and user.password == password and user.role == role:
```

```python
    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
```

Python

5. Backend Logic for New User Registration Process:

```python
'''
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
            return " "
        preferences = input("Please enter your preferences! ")
            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 " "
```

16]

Pyt

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

```python
'''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):
```

18]

```python
# 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:

```python
'''
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}
}
```

```python
# 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)
```

Python

```python
# 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)
```

```python
        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
```

Python

8. Defining Functions for Dashboard of various User Roles.

Edit Cell (Enter)

```python
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 " "
```

```python
    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 " "
```

Python

```python
# 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
```

```python
    # 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 " "
```

```python
# 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[stua
                    return " "
```

Python

9. Main Execution Function:

```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:
```

```python
while flag:

    # Asking the user, the role to be assigned.

    print("\n\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: ").strip()
        admin_password = input("Enter admin password: ").strip()
        if login(admin_username.lower(), admin_password, Role.ADMIN):
            print("\n Admin dashboard \n")
            print(admin_ddashboard())
```

```python
    #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 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!")
```

```
                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.")
```

Python

```
# Calling the main function to run the project code.

main()
```

Python

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