Joint Engineering Seat Allocation

UCS2201 – Fundamentals and Practice of Software Development

A PROJECT REPORT

Submitted By

Rohith V 3122225001111

Rosan D 3122225001112

Ram Prasath 3122225001103

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Kalavakkam – 603110

July 2023

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this project report titled "**Joint engineering seat allocation**" is the bonafide work of " Rohith V (3122225001111) , Rosan D (3122225001112) , Ram Prasath (3122225001103) " who carried out the project work in the UCS2201 – Fundamentals and Practice of Software Development during the academic year 2022-23.

Internal Examiner External Examiner

Date:

ABSTRACT

Our project aims to develop a software system for the engineering counselling and admission process in two sets of institutes, namely Indian Institutes of Technology (IITs) and National Institutes of Technology (NITs). The system will facilitate the efficient allocation of seats based on candidates' preferences and their ranks in the respective qualifying exams, JEE-Advanced and JEE-Main.

The software system will address the challenges of managing a large number of candidates and available seats across different branches in each institute. To ensure fairness and transparency, candidates will be allowed to provide a list of preferences, indicating their desired institutes and branches.

The admission process will consider the candidates' ranks in the qualifying exams, with each candidate having a specific rank in one or both merit lists. The system will allocate seats based on the candidates' preferences, availability of seats in each branch, and their ranks. The goal is to maximize satisfaction by assigning candidates to their preferred institutes and branches while adhering to the available seat limits.

By implementing this software system, we aim to streamline the counselling and admission process for engineering institutes. The system will automate the allocation process, reducing manual effort and minimizing errors. It will provide an intuitive user interface for candidates to submit their preferences and view their allocated seats. Additionally, administrators will have access to comprehensive reports and analytics to monitor the admission process and make informed decisions.

Overall, our project intends to create a reliable and efficient software system that simplifies the engineering counselling and admission process for IITs and NITs. By leveraging technology, we aim to enhance transparency, fairness, and the overall experience for both candidates and institute administrators.

TABLE OF CONTENTS

Content	Page Number
1. Problem Statement	5
2. Extended exploration of problem statement	6
3. Analysis using Data Flow Diagrams	11
4. Detailed Design	14
5. Description of each Module	19
6. Implementation	22
7. Validation through Detailed Testcases	36
8. Limitations of the solution provided	42
9. Observations from the Societal, Legal, Environmental and Ethical perspectives	44
10. Learning Outcomes	46
11. References	48

Problem Statement

Develop a **software system** for the **engineering counselling** and **admission** process for **two sets of institutes** (for example, say IITs and NITs) each having a set of **different branches**, each branch with a certain number of seats available. **Number of candidates** can be assumed as **5 times** the **total number of seats** available. Each candidate can provide a **list of preferences** where each preference is a **2-tuple**, (*institute*, *branch*). Admission to each set of institutes is based **on its own qualifying** exam (for example, **JEE-Advanced** and **JEE-Main**). Each candidate will have a **specific rank** in **one or both merit lists**.

Extended exploration of problem statement

Designing a software system for the engineering counselling and admission process for two sets of institutes (such as IITs and NITs) can be a complex task. In context of the original counselling we have tried to mimic the functionalities of the counselling window. We have tried to jot down the important components involved in the original JoSAA counselling through extensive research of the problem statement.

1. User Registration:

- ❖ Ideally, Candidates register with their personal details, contact information, and exam-specific information (such as JEE-Advanced or JEE-Main rank).
- ❖ System verifies and validates the registration details.
- ❖ But we employed a pre-existing database (student_info.csv) consisting of above-mentioned details to cut down the hassle of registering for large number of candidates.

2. Institute and Branch Setup:

- ❖ Admin sets up the list of participating institutes (IITs and NITs) and their respective branches.
- ❖ Each branch has a specific number of available seats.
- ❖ The system stores this information for further processing.
- ❖ We have created a similar database named seat_matrix.txt

3. Choice Filling:

- ❖ Ideally, Candidates provide their preferences by selecting institutes and branches.
- ❖ Preferences are stored in a prioritized list format.
- ❖ Candidates can modify their preferences until the specified deadline, due to knowledge constraints in replicating a similar method. We stick with an alterable choice list in the student_info.csv file.

4. Seat Allocation:

- ❖ Based on the merit list/rank of candidates, the system allocates seats in the participating institutes.
- ❖ The allocation process should consider the rank, preferences, and availability of seats.
- ❖ The allocation algorithm can be implemented using a variety of methods, but we have achieved the maximum efficiency out of the originally deployed Gale-Shapley Algorithm. We are successful in replicating the algorithm without any external bias like reservation, etc, but purely based on merit.

5. Seat Acceptance and Confirmation:

- ❖ Ideally, Candidates are notified about their seat allocation through the system. But, we failed to deploy them due to lack of knowledge in curl.h library and unavailability of the library in our environment.
- ❖ Candidates accept or reject the allocated seat within a specified timeframe. (Built on a primary assumption that student accepts whatever seat he gets , as we could replicate the confirmation and payment portal)
- ❖ If a candidate accepts a seat, it is considered confirmed and the seat is no longer available for further allocation.

Since, there is no Multiple Rounds deployed. Thereby, we simply display seat available post First Round of allotment.

6. Upgradation and Second Round:

- ❖ Originally, Candidates who have accepted seats can participate in subsequent rounds for better seat allocation if they are eligible for upgradation.
- ❖ The process involves releasing the previously accepted seat and re-allocating it to another candidate, while allocating a new seat to the candidate who was upgrading.
- ❖ The upgradation process continues until all eligible candidates are satisfied or until the specified rounds are completed.
- ❖ But failed to do so, due to lack of getting individual entries from the candidates after each round.

7. Reporting and Documentation:

- ❖ The system generates reports for each round, displaying allocated seats and candidates' choices.
- ❖ The system generates necessary documents, such as provisional admission letters, fee payment details, and joining instructions for the allocated institute.
- ❖ Involves Complex UI knowledge and can’t achieve in a terminal run program.

8. Waitlist Management:

- ❖ The system maintains a waitlist of candidates who have not been allocated seats but are eligible for subsequent rounds based on their preferences and ranks.

- ❖ As seats become available due to withdrawals or rejections, the system automatically allocates them to candidates on the waitlist.
- ❖ Entirely omitted as the concept of multi-round counselling isn't deployed here.

9. Withdrawal and Refunds:

- ❖ Candidates who wish to withdraw from the counselling process can do so within a specified timeframe.
- ❖ The system handles withdrawal requests and initiates the refund process for the appropriate candidates.
- ❖ The entire process of payment authentication and incorporation of time-based deadlines are omitted and are considered beyond the scope of problem statement.

10. Communication and Notifications:

- ❖ The system sends regular updates, notifications, and reminders to candidates regarding seat allocation, acceptance deadlines, and other important information.
- ❖ Email/SMS integration can be used to facilitate communication.
- ❖ Couldn't achieve real time updates due to lack of knowledge in curl.h library and unavailability of the library in our environment.

11. Admin Dashboard and Analytics:

- ❖ The system provides an admin dashboard to monitor and manage the entire counselling process.

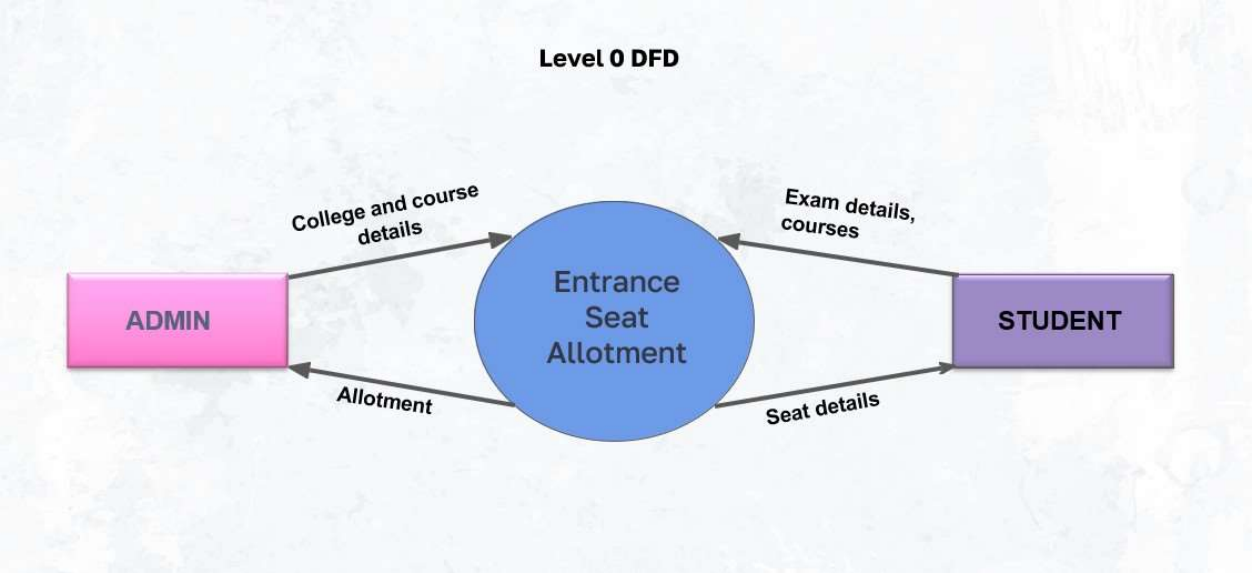
- ❖ Admins can view statistics, track progress, handle exceptions, and perform necessary administrative tasks.
- ❖ Have successfully incorporated important Admin roles.

12. Data Security and Privacy:

- ❖ The system ensures the security and privacy of candidates' personal information and exam-specific data.
- ❖ Access controls and encryption techniques can be implemented to protect sensitive data.
- ❖ Have introduced masking of passwords and repl.it encrypted Passwords using a in-built SECRETS feature .

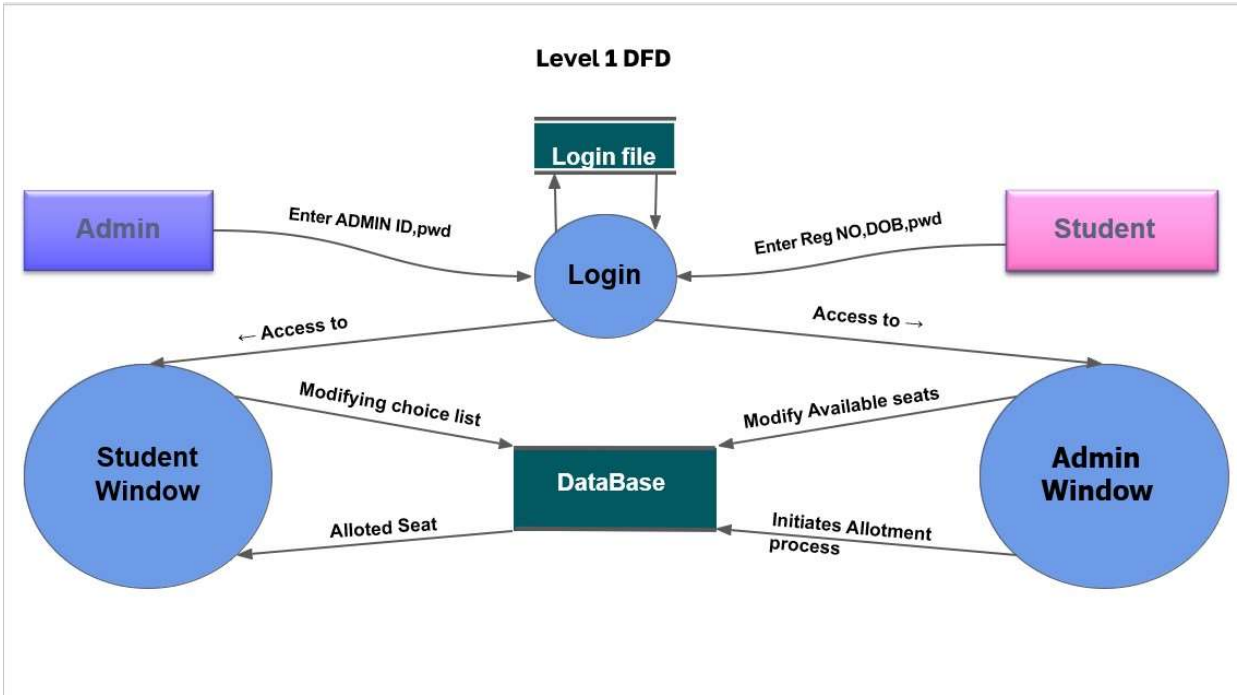
. Analysis using Data Flow Diagrams

Level 0



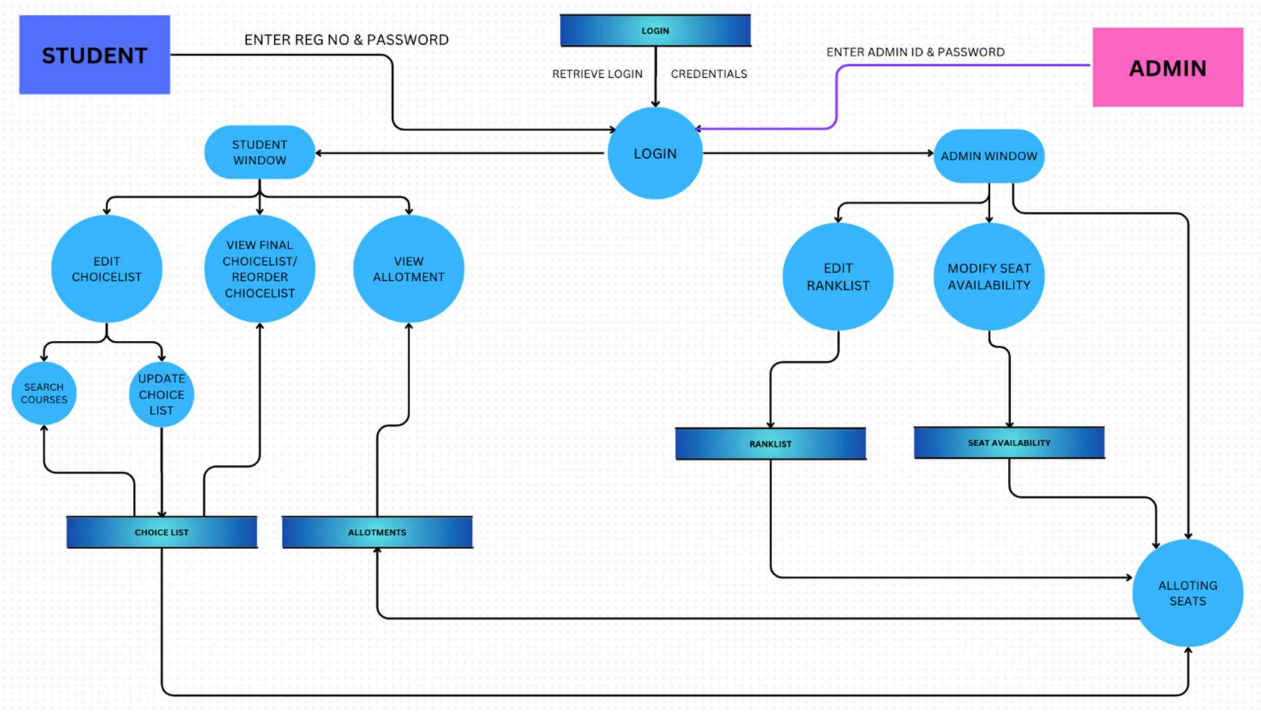
Level 0 DFD explains about the interaction between the two main entities the ADMIN and STUDENT with respect to Entrance seat Allotment Software. The admin feeds the system with College and Course details, Similarly the student feeds his exam details, student info and choice list of courses. As the result of the interaction Overall allotment is provided to Admin and Individual Allocated Seat is provided to the student.

Level 1



Level 1 DFD involves the Login Module, where the external entity such as ADMIN and STUDENT undergo initial verification of Login data. Post login, a successful entry by the student will provide access to the student window performing tasks like Modifying choice List through a common Database and similarly the Admin can Modify seat availability and Update rank List. Once, the admin initiates allotment a overall allotment result is stored in database and the student receives a allotment result.

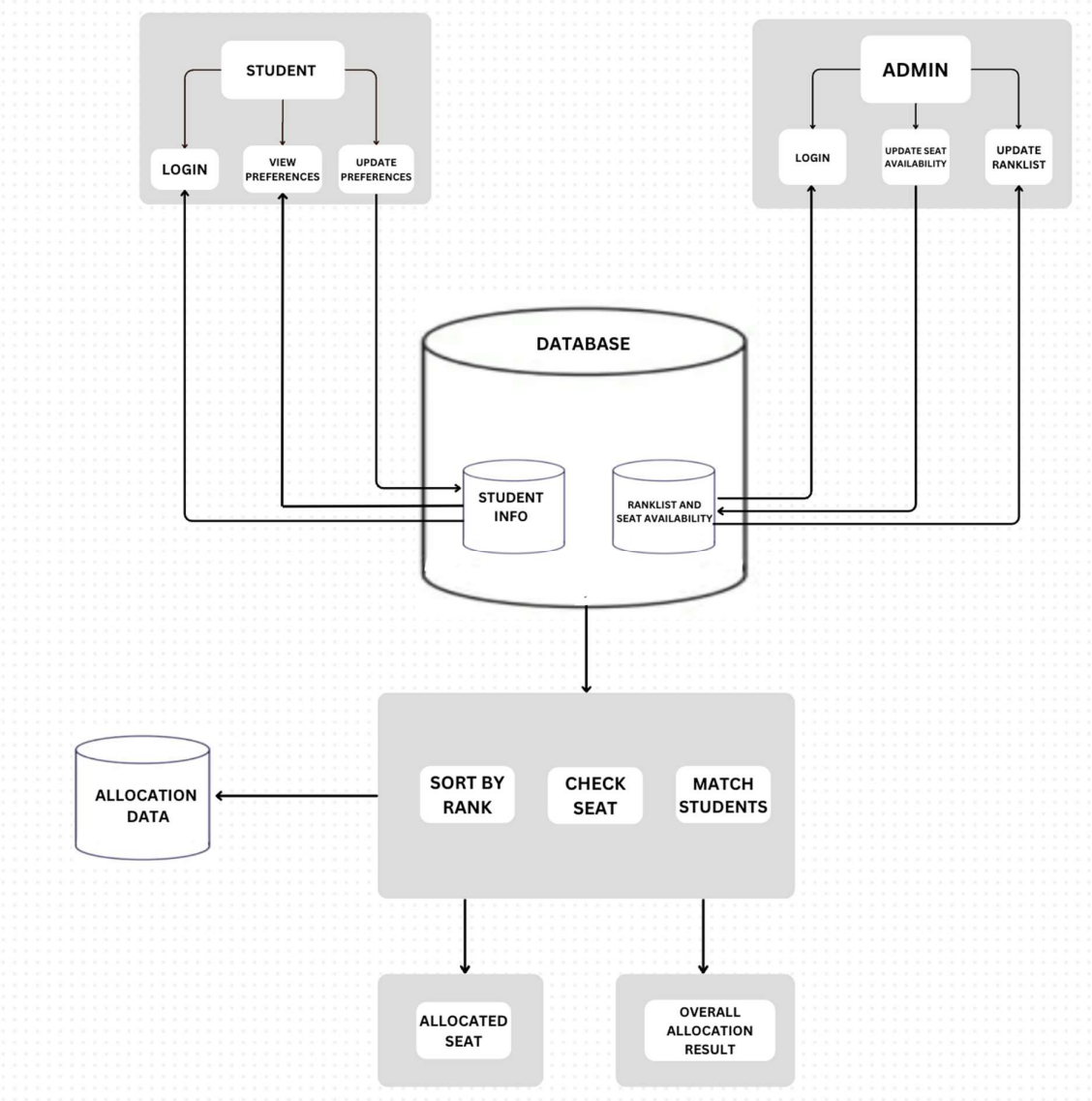
Level 2



Level 2 DFD involves the Login Module, where the external entity such as ADMIN and STUDENT undergo initial verification of Login data. Post verification the student is directed towards the student window where the student can perform functions like Edit Choice list, View/Re-order finalised choice list and view allotment. The courses are initially searched in order make further edits; courses are retrieved from choice list file which are later displayed on basis on user prompt. Edits in the choice list is updated to the Choice List data. The finalised choice list can be viewed with help of the data retrieved from choice list. And further swapping is done, and the choice list data is updated accordingly. Allotment result of the student is retrieved from Allotments data, marking the final outcome of student window. Whereas the post-login in the admin window the admin gets to Edit Rank list, Modify seat availability, Initiate allotment. The edited rank list is stored in the rank list data. And the updates made in the seat availability is also intricated in the Seat availability data. The allotting seats modules retrieves student's choices from the choice list data , and the merit is regulated with help of rank of the individual candidate stored in rank list data and with accordance to seat availability matching is achieved with help of Gale-Shapley algorithm and the Overall allotment result is stores in Allotments data.

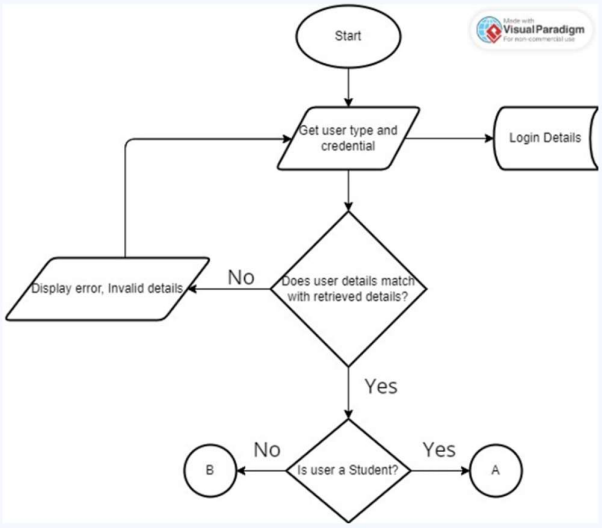
Detailed Design

Overall ARCHITECTURE diagram



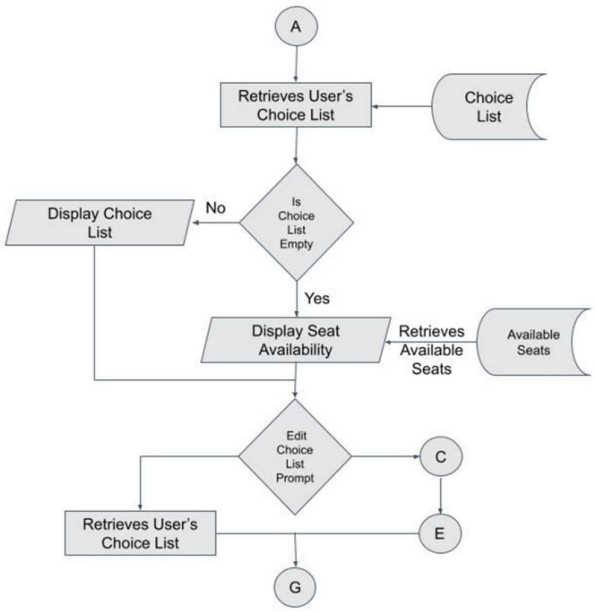
Flowchart for Individual Modules

Flowchart – Login Window



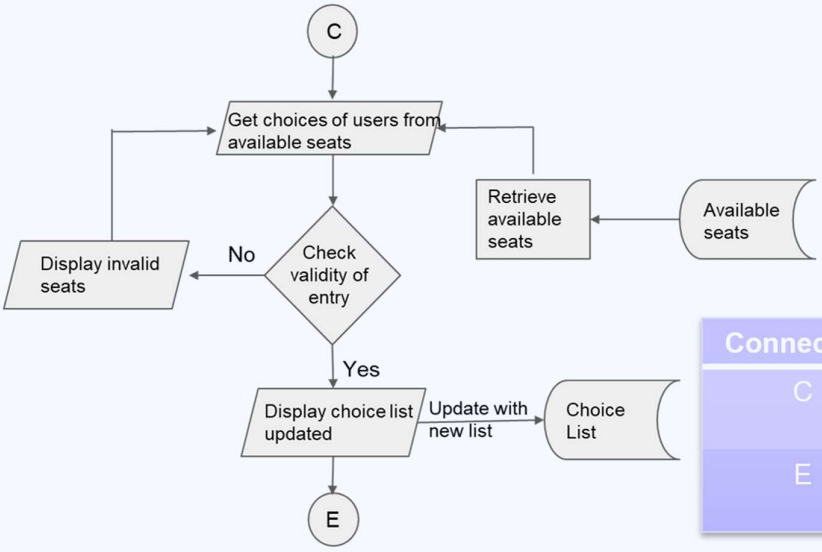
Connectors	Flow
A	To Student Window
B	To Admin Window

Flowchart – Student Window



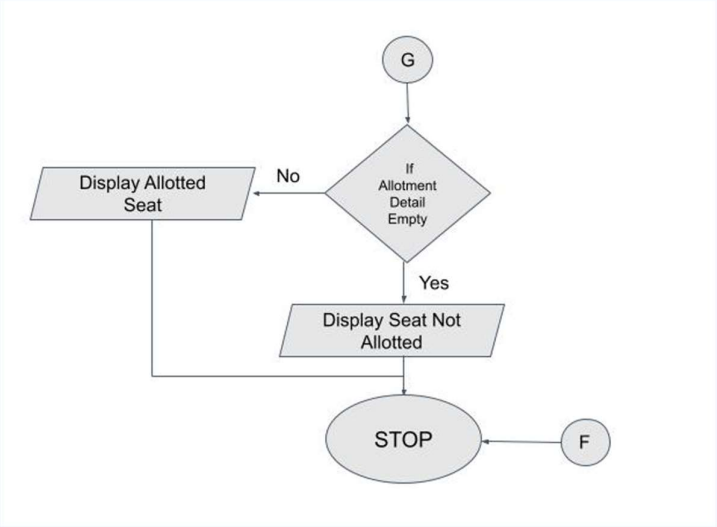
Connectors	Flow
A	From Login Window
C	To Editing Choice List
E	From Editing Choice List
G	View Allotment

Flowchart – Editing Choice List



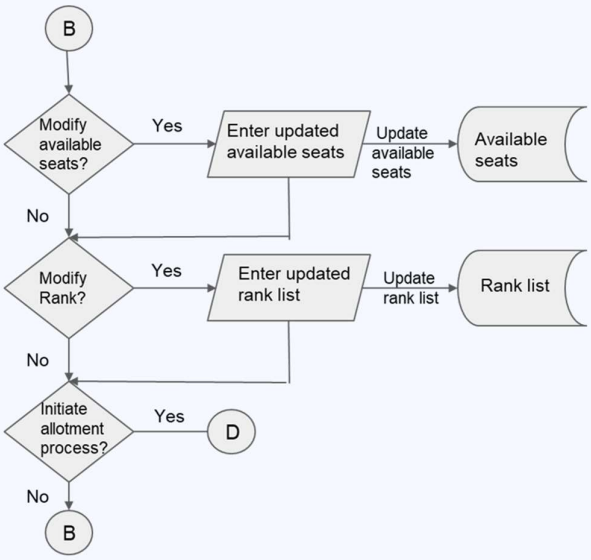
Connectors	Flow
C	From Student Window
E	To Student Window

Flowchart – Viewing Allotment



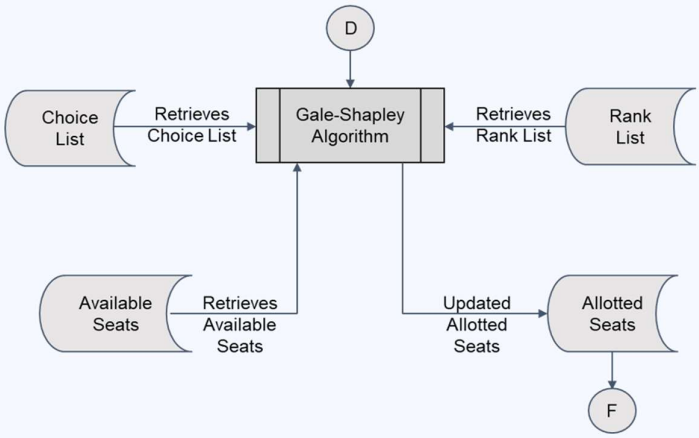
Connectors	Flow
G	From Student Window
F	From Algorithm

Flowchart – Admin Window

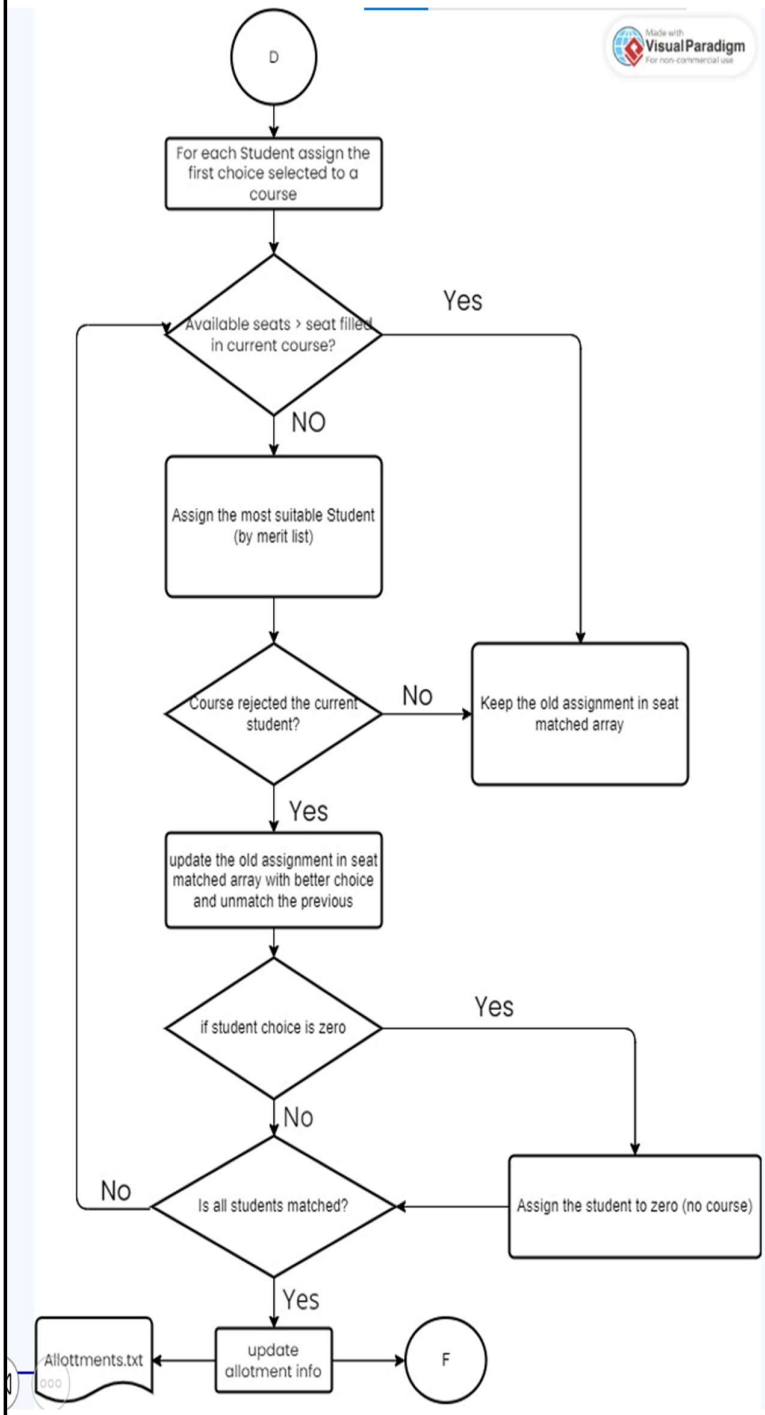


Connectors	Flow
B	From Login Window
D	To Algorithm

Flowchart - Algorithm



Connectors	Flow
D	From Admin Window
F	To Stop



Connectors	Flow
D	From Admin Window
F	To Stop

Module explanation

***Admin Window* :-**

Admin users can access the admin window through Login portal. It's a menu driven window which allows the users to access submodules such as modify seat availability, update rank list and initialize allotment process.

***Student Window* : -**

Students can access the student window through Login portal. It's a menu driven window which allows the users to access submodules such as Edit choice-list, View finalized choice-list and view allotment process.

Exit :-

The program terminates and any further engagement with the software requires a Re-run of the software in the terminal.

Update_choicelist:-

This module helps the student to perform operations such as add or remove choices. This module also ensures that student can only add IIT on their choice list only if they are eligible. All the changes made to the choice list are updated to the student_info.csv file.

View_choicelist :-

It displays the choice list of the current user. The choice list of the current student is initially retrieved from the current_stud structure. The choices are displayed with preference order and course details.

Display_course :-

This module helps the students to search for available courses. Currently this function supports two ways of searching. Students can either search courses by course name or by college category.

Reorder choice list:-

This module helps the students to reorder their choice list at the time of viewing their finalised choice list. All the changes made to the choice list will be reflected on student_info.csv file.

Check_course_Id:-

This module verifies if the input course id matches with the seat matrix. If the courses do not match, then it will throw an prompt stating the fault and the user can make corrections. This module is paired along with update_choicelist to ensure legal choices.

Edit_choicelist:-

This module acts as a menu which allows the user to use submodules such as update choicelist, search courses by name, search courses by college category. It also maintains the state of the program after every usage of sub module.

Update_ranklist:-

Users with admin privilege can use this module. It gets the file path of the new rank list. if the file path exists, it replaces new rank list with existing rank list. All the new read data will be automatically fetched to the structures having the rank data.

Modify_seat_availabilty:-

Users with admin privilege can use this module. It displays the current seat matrix of courses. Admins will be provided an option where they can increase or decrease available seats of a selected course. All the changes at the end will be reflected on seat_matrix.txt file.

Initialize allotment process:-

Users with admin privilege can use this module. Initially on execution, this module creates an array of course preferences (merit list) for every courses. Once the merit list for each course is created, Gale shapley algorithm begins. This algorithm guarantees that all students are either matched to a seat or left unalloted. Once the algorithm ends, allotted seats of every student is displayed on the terminal. This data is also stored in allotments.txt file for future references.

Implementation

Explanation of how the data is organized and the Rationale behind the selection of a particular language construct

A structure definition for storing information about students. Let's break down the components of the structure:

```
struct{
  int stu_id;
  char stu_name[20];
  int mains_rank;
  int adv_rank;
  int choices[COU_SIZE];
}stud_info[STUD_SIZE],current_stud,update_stud[STUD_SIZE];
```

`stu_id`: This is an integer variable representing the student ID.

`stu_name`: This is a character array of size 20, used to store the student's name.

`mains_rank` and `adv_rank`: These are integer variables representing the ranks of the student in JEE main and advanced exams, respectively.

`choices`: This is an integer array of size `COU_SIZE`, used to store the choices of courses selected by the student.

`stud_info`: This is an array of structures, `stud_info[STUD_SIZE]`, which allows you to store multiple instances of the `struct` type. It can hold information for multiple students.

`current_stud`: This is an individual instance of the `struct` type, used to store information about the current student.

`update_stud`: This is an array of structures, `update_stud[STUD_SIZE]`, similar to `stud_info`, used to store updated information for multiple students.

The above structure definition enables you to create instances of the structure to store information about students, including their ID, name, exam ranks, and choices of courses. The use of arrays allows for multiple students' data to be stored simultaneously.

A structure definition, for storing information about courses. Let's break down the components of the structure:

```
struct{
  int cou_id;
  char cou_name[15];
  char col_name[20];
  int preferences[STUD_SIZE];
  int avail_seat;
  int seat_filled;
}course_info[COU_SIZE];
```

`cou_id`: This is an integer variable representing the course ID.

`cou_name`: This is a character array of size 15, used to store the course name.

`col_name`: This is a character array of size 20, used to store the college name offering the course.

`preferences`: This is an integer array of size `STUD_SIZE`, used to store the preferences of students for this course. It indicates the order of preference of students who want to take this course.

`avail_seat`: This is an integer variable representing the number of available seats in the course.

`seat_filled`: This is an integer variable representing the number of seats filled in the course.

`course_info`: This is an array of structures, `course_info[COU_SIZE]`, which allows you to store multiple instances of the `struct` type. It can hold information for multiple courses.

The above structure definition enables you to create instances of the structure to store information about courses, including their ID, name, college name, student preferences, available seats, and filled seats. The use of arrays allows for multiple courses' data to be stored simultaneously.

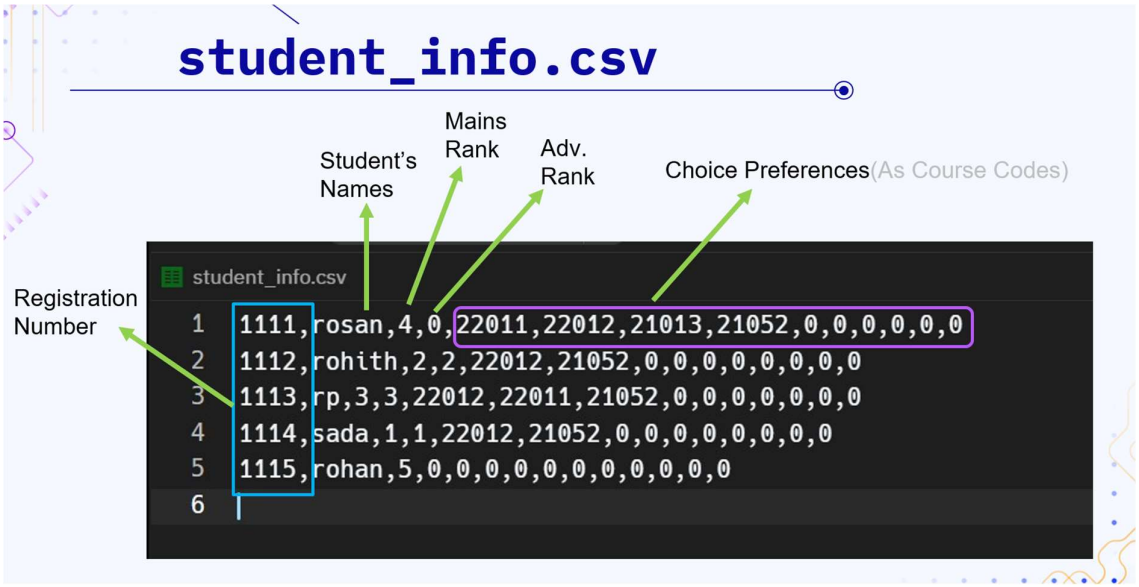
The purpose of using the "student_info.csv" file is to store and manage student information in a structured format. The file is in CSV (Comma-Separated Values) format, which is a common file format used for storing tabular data.

By using a CSV file, the program can read and write student information in a structured manner. Each line in the CSV file represents a student record, and the values for each student's attributes (such as student ID, name, ranks, and choices) are separated by commas.

The benefits of using a CSV file for storing student information include:

- Easy data exchange
- Human-readable format
- Structured data
- Simplicity

By utilizing the **"student_info.csv"** file, the program can read student information from the file, store it in the **`stud_info` array of structures**, perform operations on the data, and potentially update or write back the modified information to the file if required.



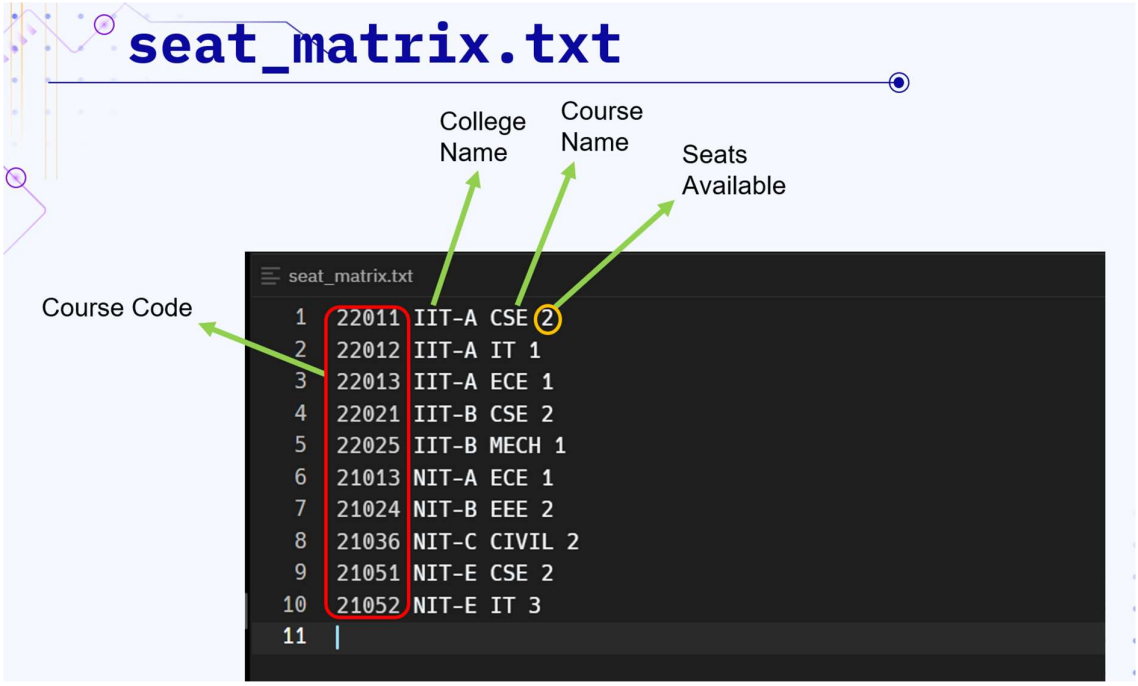
The purpose of using the "seat_matrix.txt" file is to store the seat matrix or availability of seats for different courses. The file contains information about the number of available seats and the number of seats filled for each course.

By storing this information in a separate file, the program can read and update the seat matrix as needed. The file is in a plain text format, where each line represents a course and its corresponding seat information.

The benefits of using a text file for storing seat matrix information include:

- Flexibility
- Human-readable
- Simplicity

By utilizing the "seat_matrix.txt" file, the program can retrieve the seat matrix information, track the availability and filling of seats for different courses, and potentially update the seat matrix data based on the choices and preferences of students. This allows for efficient management of available seats and allocation to students during the admission process.



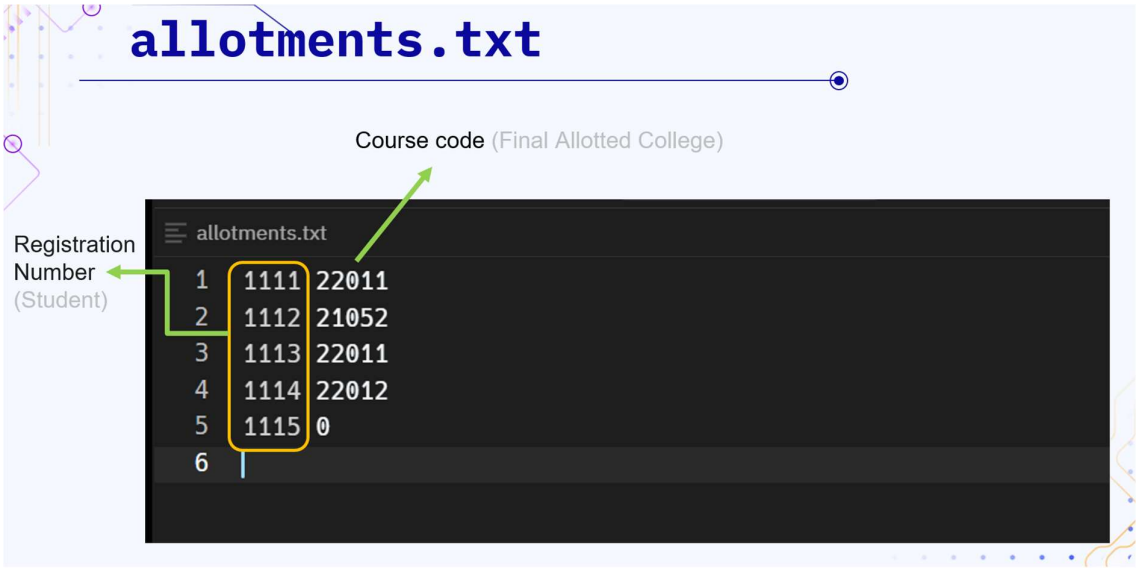
The purpose of using the "allotments.txt" file is to store the allotment details or the final course allocations for each student. The file contains information about which course has been allocated to each student based on their preferences and seat availability.

By storing this information in a separate file, the program can easily retrieve and update the allotment details as needed. The file is in a plain text format, where each line represents a student and their allocated course.

The benefits of using a text file for storing allotment information include:

- Persistence
- Compatibility
- Easy Sharing

By utilizing the "allotments.txt" file, the program can store and retrieve the final course allocations for each student, enabling easy access to the information and facilitating the management of student-course assignments.

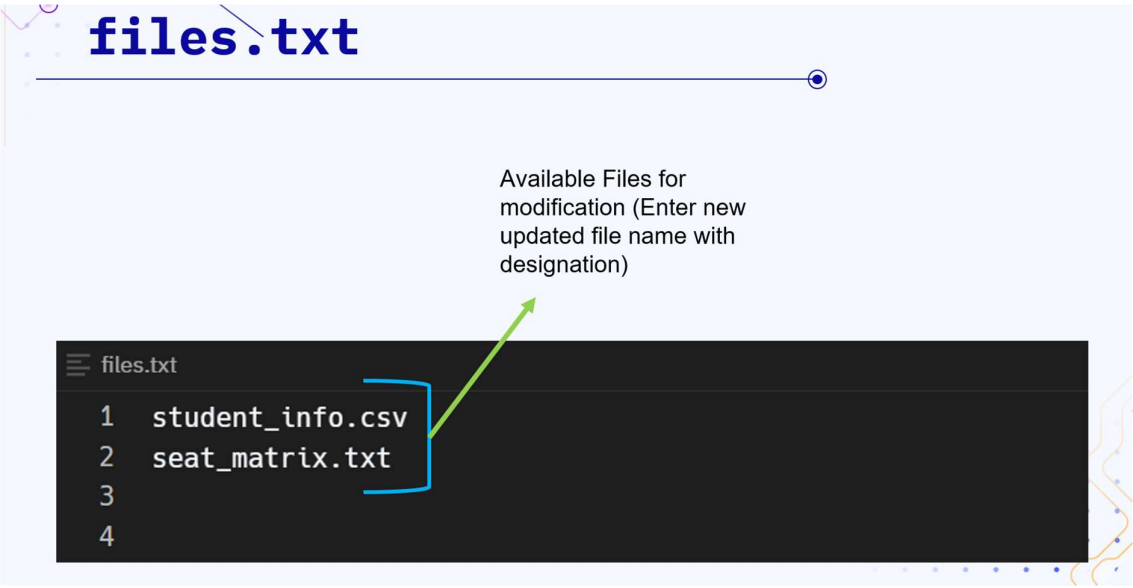


The "files.txt" file serves as a configuration file that stores the file names and their corresponding paths. This configuration file allows for easy replacement or modification of the file names and paths used in the program without having to modify the source code.

By storing the file names and paths in a separate configuration file like "files.txt," you can simply update the file names or paths within the configuration file instead of modifying the source code of your program. This approach adds flexibility and makes it easier to manage and maintain the file references used in your application.

For example, if you want to replace the "student_info.csv" or "seat_matrix.txt" file with a different file or change their directory locations, you can do so by updating the respective entries in the "files.txt" file. This eliminates the need to search through the source code to modify the file references manually.

Overall, using a configuration file like "files.txt" provides a convenient way to manage file names and paths in your program and allows for easy replacement or modification of files consisting rank , seat and student info without modifying the source code directly.



Explanation of any other libraries or APIs that have been used

`stdio.h`: This library provides input/output functions such as `printf` and `scanf`, used for standard input/output operations in C.

`math.h`: This library contains mathematical functions and constants, such as trigonometric functions (`sin`, `cos`, `tan`), logarithmic functions (`log`, `log10`), and mathematical constants (`M_PI` for pi).

`string.h`: This library provides various string manipulation functions, including functions for copying strings (`strcpy`, `strncpy`), concatenating strings (`strcat`, `strncat`), comparing strings (`strcmp`, `strncmp`), and finding substrings (`strstr`).

`stdlib.h`: This library includes functions related to memory allocation, random number generation, and various utility functions. It includes functions like `malloc` and `free` for dynamic memory allocation, `rand` and `srand` for random number generation, and `exit` for program termination.

`unistd.h`: This library provides access to various POSIX (Portable Operating System Interface) operating system services. It includes functions like `fork` for creating child processes, `exec` for executing a new program in the current process, `read` and `write` for file I/O operations, and `sleep` for suspending the program execution for a specified time.

`stdbool.h`: This library defines the Boolean data type in C. It provides the constants `true` and `false` for representing boolean values and the type `bool` for boolean variables.

`sys/ioctl.h`: This library provides functions and data structures for input/output control operations. It includes functions like `ioctl` for manipulating device parameters and `termios.h` for terminal I/O operations.

`termios.h`: This library defines functions, data types, and constants for terminal I/O operations. It includes functions like `tcgetattr` and `tcsetattr` for getting and setting terminal attributes, and constants like `ECHO` and `ICANON` for configuring terminal behavior.

User Interface Design

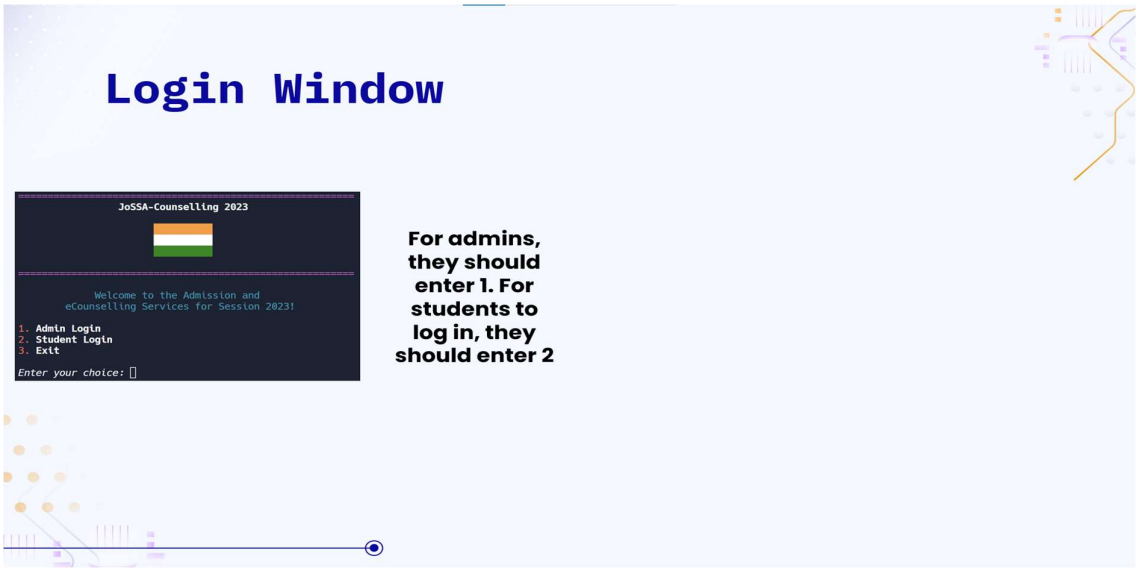


Figure 1: Initial Login Page

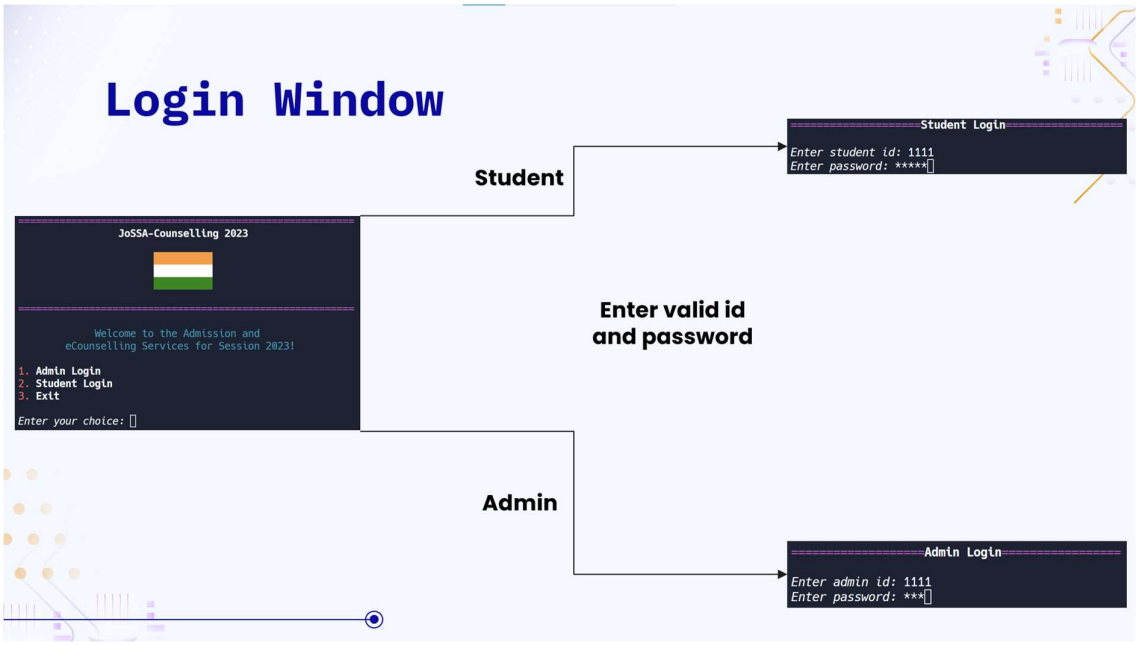


Figure 2: Password Verfication

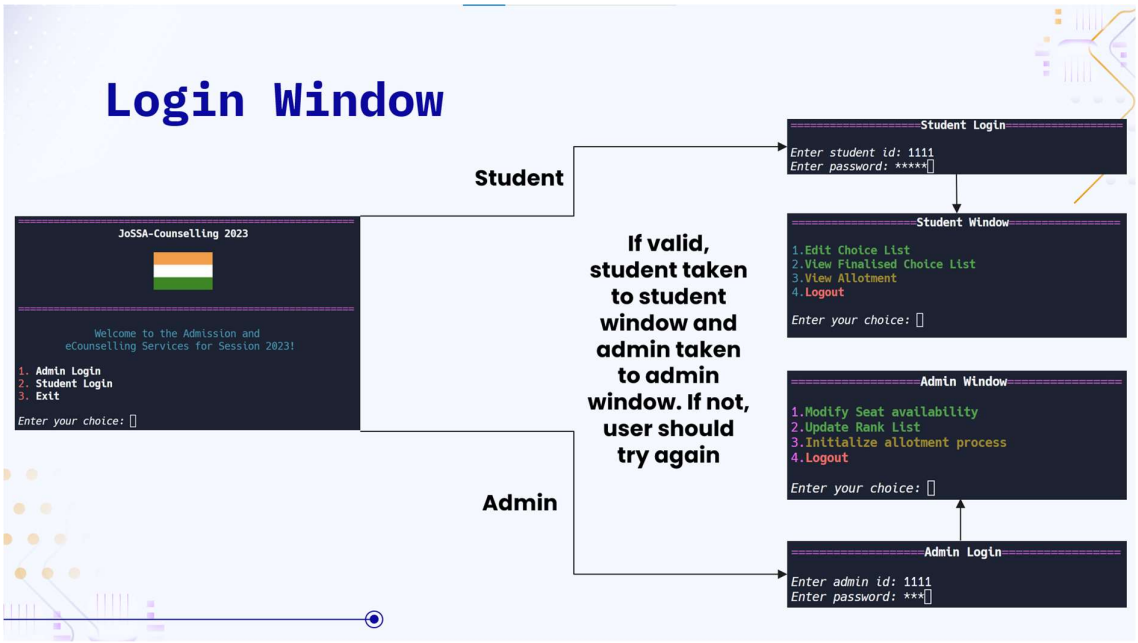


Figure 3: Post Password Verfication , Student and Admin window

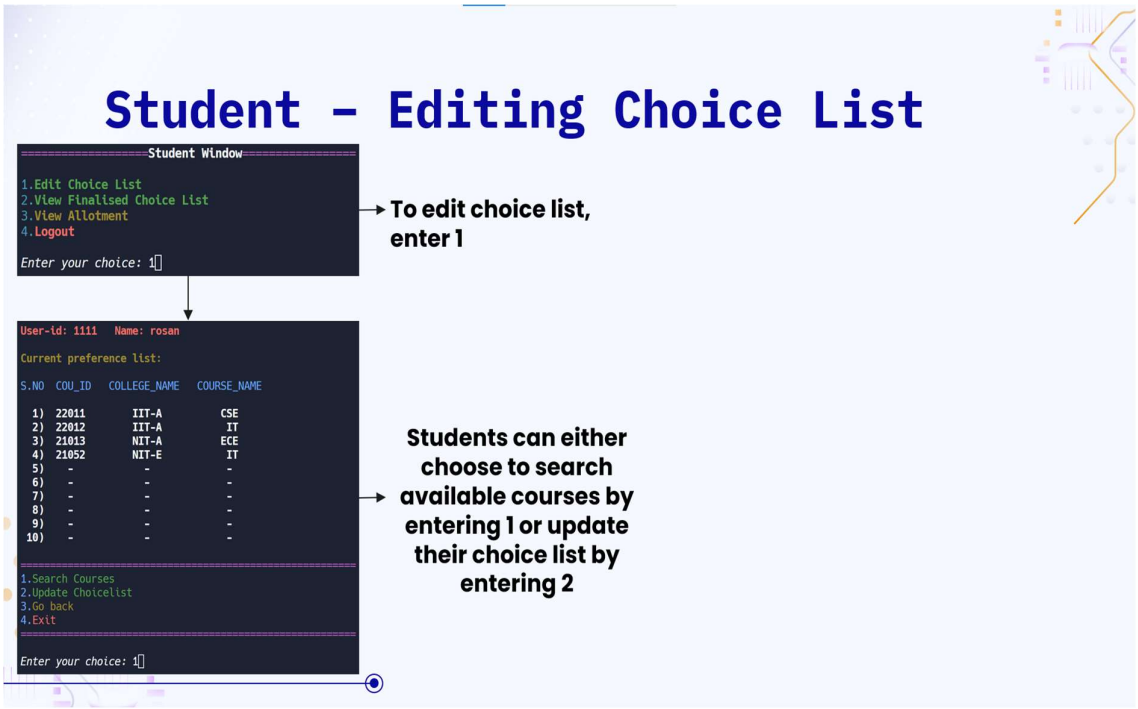


Figure 4: Edit Choice List Page

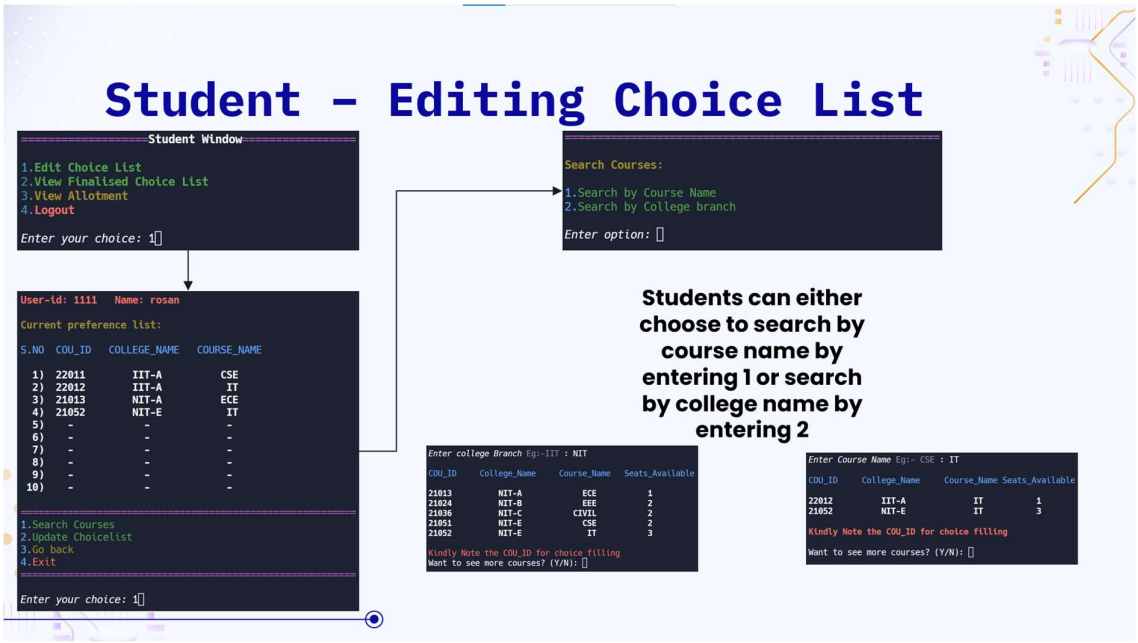


Figure 5: Search Courses by branch and by College name

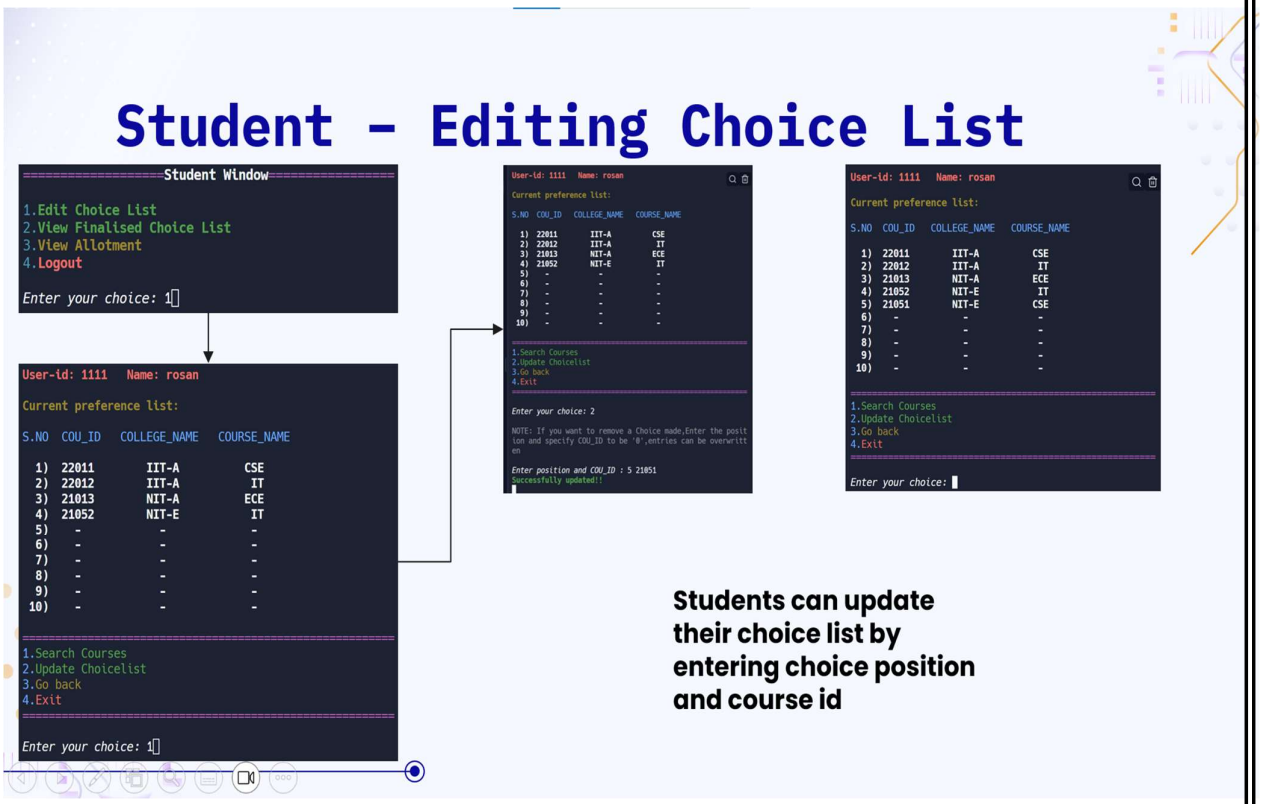


Figure 6: Upate Choice List

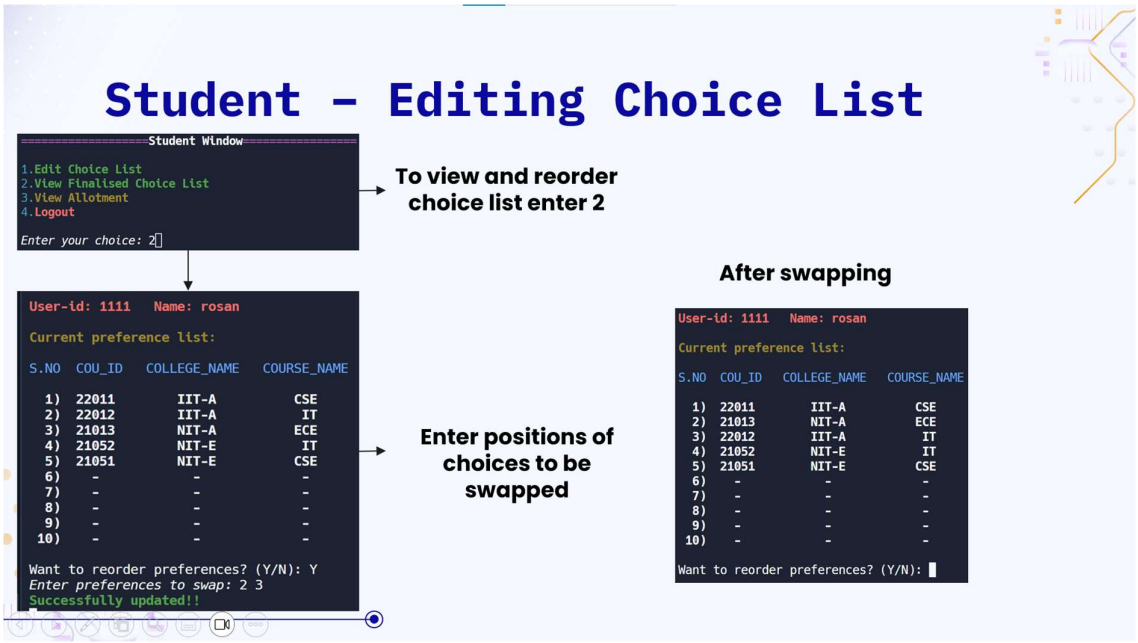


Figure 7: View Finalised Choice List and Re-order Choices

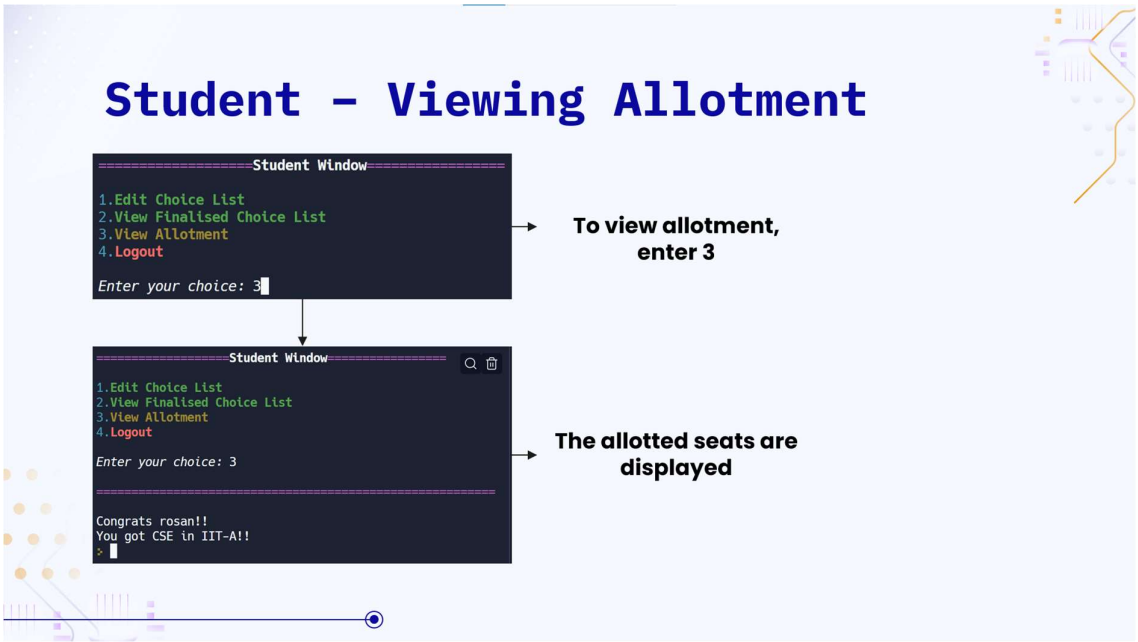


Figure 8: View Allotment

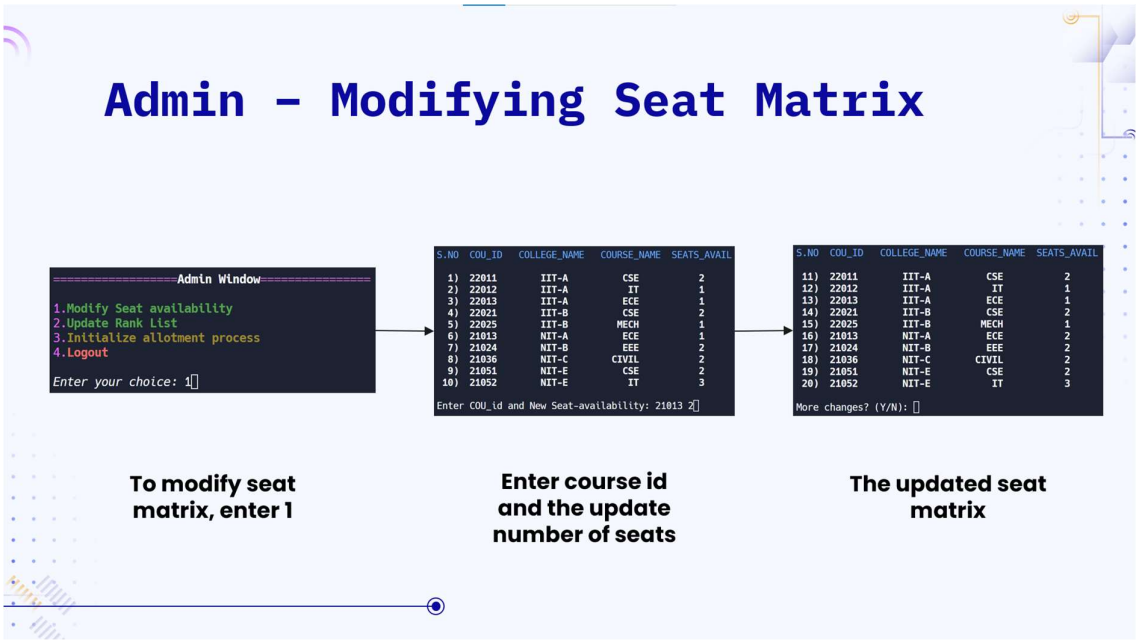


Figure 9: Modify Seat Availablity

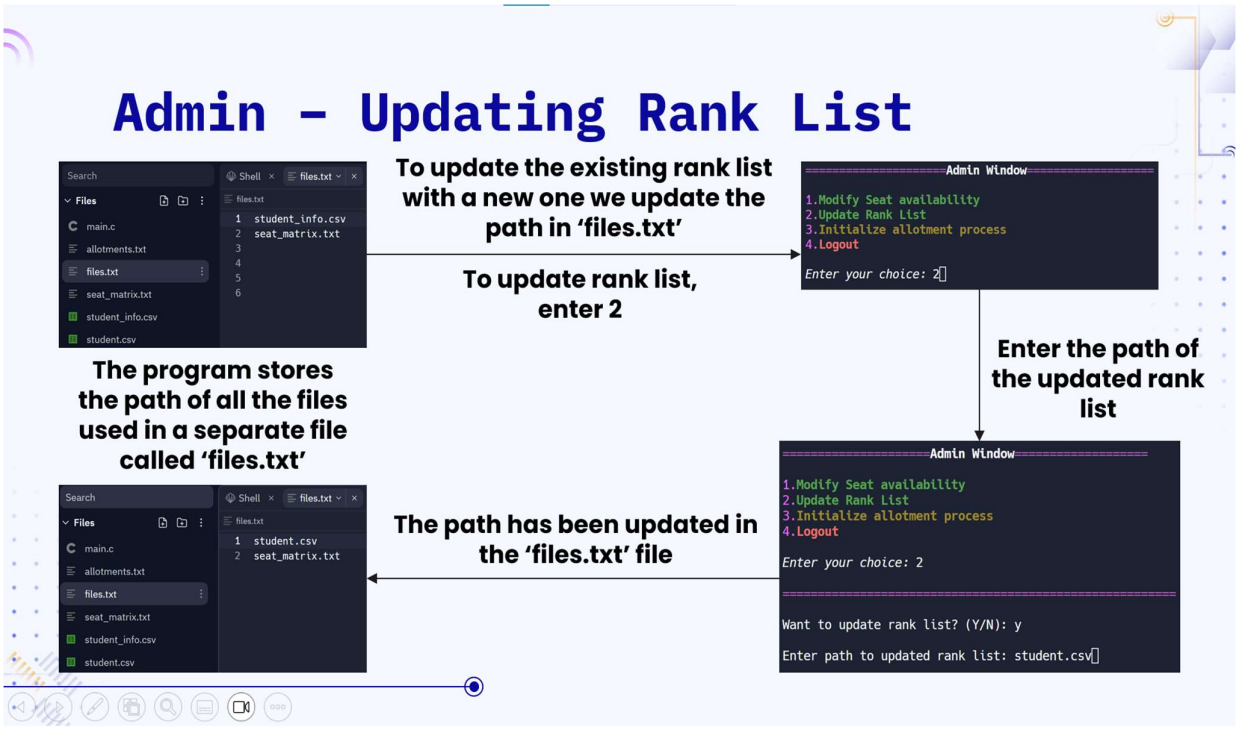


Figure 10: Update Rank List

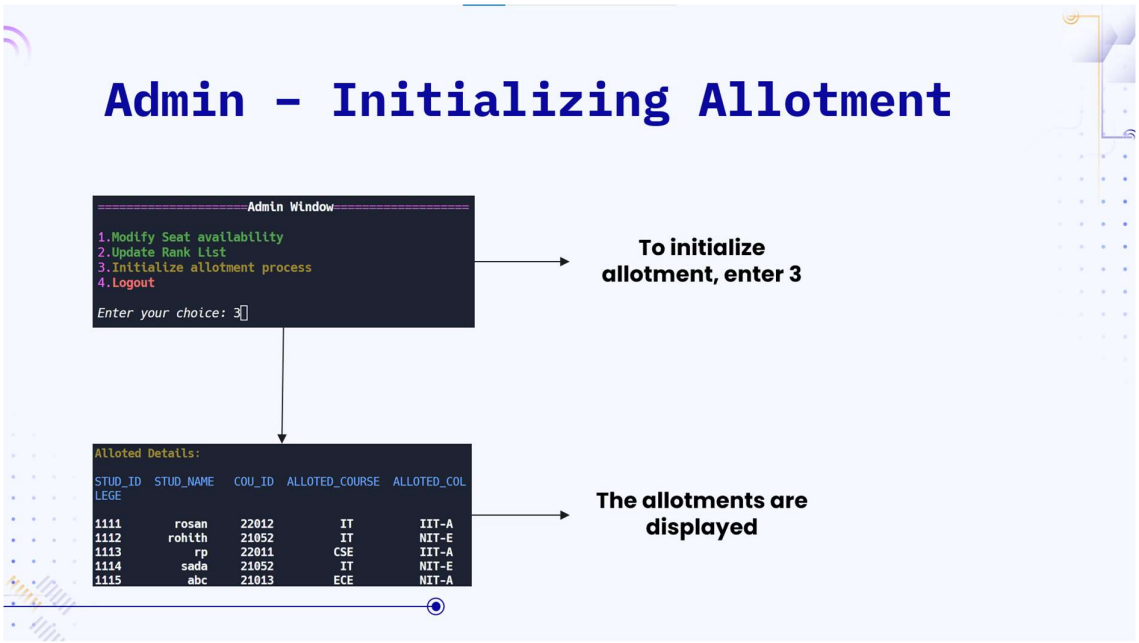


Figure 11: Initialize Allotment Process

Platform used for Code Development – Repl.it

1. Online Development Environment: Replit provides an online integrated development environment (IDE) that allows you to write, compile, and run your code directly in the browser. This eliminates the need for local installations and ensures that your project is accessible from any device with an internet connection.

2. Collaboration: Replit offers collaboration features that make it easy for multiple team members to work on the same project simultaneously. This is particularly beneficial if you are working on a group project or want to collaborate with others in real-time.

3. Easy Sharing and Deployment: Replit makes it simple to share your project with others by providing a unique URL that allows anyone to access and interact with your code. Additionally, you can easily deploy your project to the web using Replit's hosting capabilities, making it convenient to showcase your work to others.

4. Pre-installed Libraries and Packages: Replit comes with a wide range of pre-installed libraries and packages, which can save you time and effort in setting up your development environment. This is especially useful if your project requires specific libraries or dependencies.

5. Secure Storage of Sensitive Information: Replit provides a built-in "secrets" feature that allows you to securely store sensitive information, such as API keys, passwords, or other confidential data. This ensures that your sensitive information is protected and not exposed within your codebase or visible to others who may have access to your project.

Validation through Detailed Testcases

Candidates have Single – Same Preference, Excess Seats

Here all the candidates have only one preference to one particular course, here to 21013.

The course 21013 has an excess of seats, 10 available seats vs 5 candidates preferring.

Hence, everyone gets allotment at course 21013.

Input

testSeat.txt

```
22011 IIT-A CSE 10
22012 IIT-A IT 1
22013 IIT-A ECE 1
22021 IIT-B CSE 2
22025 IIT-B MECH 2
21013 NIT-A ECE 10
21024 NIT-B EEE 2
21036 NIT-C CIVIL 2
21051 NIT-E CSE 2
21052 NIT-E IT 1
```

testStudent.csv

```
1111,rosan,3,1,21013,0,0,0,0,0,0,0,0,0
1112,rohith,2,3,21013,0,0,0,0,0,0,0,0,0
1113,rp,4,2,21013,0,0,0,0,0,0,0,0,0
1114,sada,1,4,21013,0,0,0,0,0,0,0,0,0
1115,abc,5,0,21013,0,0,0,0,0,0,0,0,0
```

First Iteration	
1111	21013
1112	21013
1113	21013
1114	21013
1115	21013

Output

```
Alloted Details:

STUD_ID  STUD_NAME   COU_ID  ALLOTED_COURSE  ALLOTED_COLLEGE

1111        rosan    21013         ECE           NIT-A
1112       rohith    21013         ECE           NIT-A
1113           rp    21013         ECE           NIT-A
1114         sada    21013         ECE           NIT-A
1115          abc    21013         ECE           NIT-A
```

Candidates have Single – Same Preference, Limited Seats

Here all the candidates have only one preference to one particular course, here to 21013.

The course 21013 has a shortage of seats, 3 available seats vs 5 candidates preferring.

Hence, only candidates 1111, 1112 and 1114 gets allotment at course 21013.

This is because candidates 1111, 1112 and 1114 has a better rank than candidates 1113 and 1115 in Mains as they are all competing for a NIT course.

First Iteration	
1111	21013
1112	21013
1113	Unmatched
1114	21013
1115	Unmatched

Second Iteration	
1111	21013
1112	21013
1113	No Allotment
1114	21013
1115	No Allotment

Input

testSeat.txt

```
22011 IIT-A CSE 10
22012 IIT-A IT 1
22013 IIT-A ECE 1
22021 IIT-B CSE 2
22025 IIT-B MECH 2
21013 NIT-A ECE 3
21024 NIT-B EEE 2
21036 NIT-C CIVIL 2
21051 NIT-E CSE 2
21052 NIT-E IT 1
```

testStudent.csv

```
1111,rosan,3,1,21013,0,0,0,0,0,0,0,0,0
1112,rohith,2,3,21013,0,0,0,0,0,0,0,0,0
1113,rp,4,2,21013,0,0,0,0,0,0,0,0,0
1114,sada,1,4,21013,0,0,0,0,0,0,0,0,0
1115,abc,5,0,21013,0,0,0,0,0,0,0,0,0
```

Output

```
Alloted Details:

STUD_ID  STUD_NAME  COU_ID  ALLOTED_COURSE  ALLOTED_COLLEGE

1111        rosan    21013             ECE            NIT-A
1112       rohith    21013             ECE            NIT-A
1113           rp     None               -                -
1114         sada    21013             ECE            NIT-A
1115          abc     None               -                -
```

Candidates have Single – Different Preference, Excess Seats

Here all the candidates have only one preference to different courses, here to three courses 22011, 22012 and 21013.

All three course has an excess of seats, 22011 has 10 available seats vs 2 candidates (1112 and 1113) preferring, 22012 has 1 available seats vs 1 candidate (1114) preferring and 21013 has 3 available seats vs 2 candidates (1111 and 1115) preferring.

Hence, everyone gets allotment at their preferred course.

First Iteration	
1111	21013
1112	22011
1113	22011
1114	22012
1115	21013

Input

testSeat.txt

```
22011 IIT-A CSE 10
22012 IIT-A IT 1
22013 IIT-A ECE 1
22021 IIT-B CSE 2
22025 IIT-B MECH 2
21013 NIT-A ECE 3
21024 NIT-B EEE 2
21036 NIT-C CIVIL 2
21051 NIT-E CSE 2
21052 NIT-E IT 1
```

testStudent.csv

```
1111,rosan,3,1,21013,0,0,0,0,0,0,0,0,0
1112,rohith,2,3,22011,0,0,0,0,0,0,0,0,0
1113,rp,4,2,22011,0,0,0,0,0,0,0,0,0
1114,sada,1,4,22012,0,0,0,0,0,0,0,0,0
1115,abc,5,0,21013,0,0,0,0,0,0,0,0,0
```

Output

Alloted Details:

STUD_ID	STUD_NAME	COU_ID	ALLOTED_COURSE	ALLOTED_COLLEGE
1111	rosan	21013	ECE	NIT-A
1112	rohith	22011	CSE	IIT-A
1113	rp	22011	CSE	IIT-A
1114	sada	22012	IT	IIT-A
1115	abc	21013	ECE	NIT-A

Candidates have Single – Different Preference, Limited Seats

Here all the candidates have multiple preferences, here to three courses 22011, 22012 and 21013.

All three course has a shortage of seats, 22011 has 1 available seats vs 2 candidates (1112 and 1113) preferring, 22012 has 0 available seats vs 1 candidate (1114) preferring and 21013 has 1 available seats vs 2 candidates (1111 and 1115) preferring.

Hence, only candidates 1111 and 1113 get allotment at their preferred course 21013 and 22011 respectively.

This is because candidate 1111 has a better Mains rank than 1115 and they compete for the same NIT course. And 1112 has a better Advanced rank than 1112 and they compete for the same IIT course. 1114 didn't get an allotment as there was no seat available in their preferred course 22012.

First Iteration	
1111	21013
1112	Unmatched
1113	22011
1114	Unmatched
1115	Unmatched

Second Iteration	
1111	21013
1112	No Allotment
1113	No Allotment
1114	21013
1115	No Allotment

Input

testSeat.txt

```
22011 IIT-A CSE 1
22012 IIT-A IT 0
22013 IIT-A ECE 1
22021 IIT-B CSE 2
22025 IIT-B MECH 2
21013 NIT-A ECE 1
21024 NIT-B EEE 2
21036 NIT-C CIVIL 2
21051 NIT-E CSE 2
21052 NIT-E IT 1
```

testStudent.csv

```
1111,rosan,3,1,21013,0,0,0,0,0,0,0,0,0
1112,rohith,2,3,22011,0,0,0,0,0,0,0,0,0
1113,rp,4,2,22011,0,0,0,0,0,0,0,0,0
1114,sada,1,4,22012,0,0,0,0,0,0,0,0,0
1115,abc,5,0,21013,0,0,0,0,0,0,0,0,0
```

Output

Alloted Details:

STUD_ID	STUD_NAME	COU_ID	ALLOTED_COURSE	ALLOTED_COLLEGE
1111	rosan	21013	ECE	NIT-A
1112	rohith	None	-	-
1113	rp	22011	CSE	IIT-A
1114	sada	None	-	-
1115	abc	None	-	-

Candidates have Multiple Preferences, Limited Seat

Here all the candidates have multiple preferences, here to four courses 22011, 22012, 21013 and 21052.

Here during the first iteration, candidate 1111 gets allotted their 1st preference 21013. Then candidate 1112's 1st preference is checked. As it is 21013 and there is no seat available in it, 1112's Mains rank is compared with candidates already allotted to 21013. As it is candidate 1111, and they have a worser Mains rank than 1112, they are considered unmatched

Here during the first iteration, candidates 1111 and 1115 are unmatched. This is because 1111, 1115 and 1112 preferred for the same NIT course having only one seat and 1112 has a better Mains rank than 1111 and 1115. Others are allotted their first preference as seats are available.

In the second iteration, we are trying to match the unmatched candidates 1111 and 1115. 1111's 2nd preference is an IIT course which has no more seats and is taken by 1113. This causes 1111 to be matched with the course leaving 1113 unmatched, as 1111 has a better Advanced rank than 1113. 1115 gets allotted its 2nd preference as seats are available.

In the third iteration, as 1113 is unmatched its 3rd preference is checked. As the candidate has provided no choice, they have been not allotted a course.

The allotment process ends.

First Iteration		Second Iteration		Third Iteration	
1111	Unmatched	1111	22011	1111	22011
1112	21013	1112	21013	1112	21013
1113	22011	1113	Unmatched	1113	No Allotment
1114	22012	1114	22012	1114	22012
1115	Unmatched	1115	21052	1115	21052

Input

testSeat.txt

```
22011 IIT-A CSE 1
22012 IIT-A IT 5
22013 IIT-A ECE 1
22021 IIT-B CSE 2
22025 IIT-B MECH 2
21013 NIT-A ECE 1
21024 NIT-B EEE 2
21036 NIT-C CIVIL 2
21051 NIT-E CSE 2
21052 NIT-E IT 1
```

testStudent.csv

```
1111,rosan,3,1,21013,22011,0,0,0,0,0,0,0,0
1112,rohith,2,3,21013,22011,0,0,0,0,0,0,0,0
1113,rp,4,2,22011,21013,0,0,0,0,0,0,0,0
1114,sada,1,4,22012,21013,0,0,0,0,0,0,0,0
1115,abc,5,0,21013,21052,0,0,0,0,0,0,0,0
```

Output

```
Alloted Details:

STUD_ID  STUD_NAME   COU_ID  ALLOTED_COURSE  ALLOTED_COLLEGE

1111        rosan    22011          CSE            IIT-A
1112       rohith    21013          ECE            NIT-A
1113           rp     None            -              -
1114         sada    22012           IT            IIT-A
1115          abc    21052           IT            NIT-E
```

Final Test Case

A real word scenario where multiple candidates have multiple preferences, different merits and varied seat matrix.

Input

testStudent.csv

```
1000,Liam,49,0,21025,22021,22025,0,0,0,0,0,0,0,0,0
1001,Noah,11,10,21032,21025,21031,21026,21012,0,0,0,0,0,0,0
1002,Oliver,1,0,21032,21026,22025,21025,0,0,0,0,0,0,0,0
1003,James,10,26,22011,21011,21013,22012,21025,21026,22025,21032,21031,21024,22021,21012
1004,Elijah,30,32,21012,21026,21024,21013,22021,22025,21011,22011,21025,21031,22012,0
1005,William,12,0,22012,22025,22011,21012,21025,21031,21011,21026,21024,21032,22021,0
1006,Henry,7,3,21013,21031,22011,21012,0,0,0,0,0,0,0,0
1007,Lucas,3,0,21031,21024,22021,21032,22025,22012,21011,21013,21012,22011,0,0
1008,Benjamin,23,4,21011,21026,21012,21025,0,0,0,0,0,0,0,0
1009,Theodore,43,0,21013,22011,21011,21031,0,0,0,0,0,0,0,0
1010,Mateo,41,0,21026,21011,21013,22021,21032,21031,21025,22011,0,0,0,0
1011,Levi,15,27,21032,22021,21013,21012,22011,21031,0,0,0,0,0,0
1012,Sebastian,6,19,21024,0,0,0,0,0,0,0,0,0,0,0
1013,Daniel,17,40,21026,21025,22012,22025,22021,21024,0,0,0,0,0,0
1014,Jack,14,18,21011,21031,0,0,0,0,0,0,0,0,0,0
1015,Michael,18,12,22011,21025,0,0,0,0,0,0,0,0,0,0
1016,Alexander,45,14,21026,22011,21025,0,0,0,0,0,0,0,0,0
1017,Owen,21,0,22012,21011,21013,22011,21024,21032,22025,22021,21026,21031,21025,21012
1018,Asher,50,39,21026,21012,22021,22025,21025,21032,21011,0,0,0,0,0
1019,Samuel,5,0,21024,21032,21031,21026,22025,21013,0,0,0,0,0,0
1020,Ethan,9,21,21024,0,0,0,0,0,0,0,0,0,0,0
1021,Leo,37,38,21026,22012,21024,22011,21012,21013,22021,21011,21032,21025,0,0
1022,Jackson,47,16,22021,0,0,0,0,0,0,0,0,0,0,0
1023,Mason,33,33,21025,21012,22012,21031,22025,21013,0,0,0,0,0,0
1024,Ezra,20,34,22012,21011,0,0,0,0,0,0,0,0,0,0
1025,John,30,9,22012,21024,22021,21032,21025,21013,21012,22025,22011,0,0,0
1026,Hudson,31,5,22025,22012,21026,0,0,0,0,0,0,0,0,0
1027,Luca,36,30,21031,22012,22025,21012,21024,21011,21026,0,0,0,0,0
1028,Alden,29,23,21024,22011,22021,21025,21031,21026,21032,21012,21011,0,0,0
1029,Joseph,40,29,21025,21026,21013,21012,22011,21032,0,0,0,0,0,0
1030,David,46,20,22011,21012,21026,22012,21025,21031,0,0,0,0,0,0
1031,Jacob,35,35,21013,21031,21025,22011,22025,21012,22021,21032,21024,22012,0,0
1032,Logan,4,11,22011,21011,0,0,0,0,0,0,0,0,0,0
1033,Luke,16,0,21011,21032,21031,21013,21025,22011,21026,22025,21024,22012,21012,0
1034,Julian,13,17,21011,0,0,0,0,0,0,0,0,0,0,0
1035,Gabriel,22,22,22012,21011,21024,0,0,0,0,0,0,0,0,0
1036,Grayson,27,0,21013,22025,0,0,0,0,0,0,0,0,0,0
1037,Wyatt,48,1,21025,0,0,0,0,0,0,0,0,0,0,0
1038,Matthew,32,0,21024,21012,22012,21025,22021,0,0,0,0,0,0,0
1039,Maverick,2,15,21024,21025,22025,21011,22012,0,0,0,0,0,0,0
1040,Dylan,42,6,22021,22012,22025,21011,0,0,0,0,0,0,0,0
1041,Isaac,26,37,22025,21031,21026,21013,22011,21024,21011,0,0,0,0,0
1042,Elias,34,28,21011,21024,22021,21013,21025,22025,21012,21026,22011,21032,22012,21031
1043,Anthony,8,31,21026,21011,21012,0,0,0,0,0,0,0,0,0
1044,Thomas,44,7,21032,22011,22021,0,0,0,0,0,0,0,0,0
1045,Jayden,19,13,21013,21032,21011,22025,22012,21024,0,0,0,0,0,0
1046,Carter,28,36,21013,22012,21025,21012,21026,21031,21032,21011,0,0,0,0
1047,Santiago,25,2,21032,21013,22025,21026,0,0,0,0,0,0,0,0
1048,Ezekiel,24,25,21011,21024,21031,21012,21026,22012,21025,21013,0,0,0,0
1049,Charles,39,24,21032,21024,21031,22025,22011,21011,21013,22012,21026,0,0,0
```

testSeat.txt

```
1  22011 IIT-Madras CSE 2
2  22012 IIT-Madras IT 3
3  22021 IIT-Bombay CSE 1
4  22025 IIT-Bombay MECH 4
5  21011 NIT-Trichy CSE 2
6  21012 NIT-Trichy IT 3
7  21013 NIT-Trichy ECE 3
8  21024 NIT-Surat EEE 2
9  21025 NIT-Surat MECH 2
10 21026 NIT-Surat CIVIL 2
11 21031 NIT-Calicut CSE 3
12 21032 NIT-Calicut IT 3
```

Output

Alloted Details:

STUD_ID	STUD_NAME	COU_ID	ALLOTED_COURSE	ALLOTED_COLLEGE
1000	Liam	None	-	-
1001	Noah	21032	IT	NIT-Calicut
1002	Oliver	21032	IT	NIT-Calicut
1003	James	21011	CSE	NIT-Trichy
1004	Elijah	None	-	-
1005	William	None	-	-
1006	Henry	21013	ECE	NIT-Trichy
1007	Lucas	21031	CSE	NIT-Calicut
1008	Benjamin	21012	IT	NIT-Trichy
1009	Theodore	None	-	-
1010	Mateo	None	-	-
1011	Levi	21032	IT	NIT-Calicut
1012	Sebastian	None	-	-
1013	Daniel	21026	CIV	NIT-Surat
1014	Jack	21031	CSE	NIT-Calicut
1015	Michael	None	-	-
1016	Alexander	None	-	-
1017	Owen	None	-	-
1018	Asher	None	-	-
1019	Samuel	21024	EEE	NIT-Surat
1020	Ethan	None	-	-
1021	Leo	None	-	-
1022	Jackson	None	-	-
1023	Mason	None	-	-
1024	Ezra	None	-	-
1025	John	22012	IT	IIT-Madras
1026	Hudson	22025	MEC	IIT-Bombay
1027	Luca	22025	MEC	IIT-Bombay
1028	Alden	21025	MEC	NIT-Surat
1029	Joseph	None	-	-
1030	David	22012	IT	IIT-Madras
1031	Jacob	None	-	-
1032	Logan	22011	CSE	IIT-Madras
1033	Luke	21031	CSE	NIT-Calicut
1034	Julian	21011	CSE	NIT-Trichy
1035	Gabriel	22012	IT	IIT-Madras
1036	Grayson	None	-	-
1037	Wyatt	None	-	-
1038	Matthew	21012	IT	NIT-Trichy
1039	Maverick	21024	EEE	NIT-Surat
1040	Dylan	22021	CSE	IIT-Bombay
1041	Isaac	None	-	-
1042	Elias	22025	MEC	IIT-Bombay
1043	Anthony	21026	CIV	NIT-Surat
1044	Thomas	22011	CSE	IIT-Madras
1045	Jayden	21013	ECE	NIT-Trichy
1046	Carter	21025	MEC	NIT-Surat
1047	Santiago	21013	ECE	NIT-Trichy
1048	Ezekiel	21012	IT	NIT-Trichy
1049	Charles	22025	MEC	IIT-Bombay

Limitations of proposed solutions

While the proposed solution addresses the engineering counseling and admission process for institutes with different branches and seat availability, there are certain limitations to consider:

1. Scalability: The solution assumes a fixed ratio of 5 candidates per available seat. However, in practice, the **number of candidates may vary**, and the system may need to handle a larger number of applicants. **Ensuring scalability to accommodate a higher number of candidates and institutes is essential.**

2. Real-time Updates: The solution **does not account for real-time updates of seat availability or changes in candidate preferences.** If there are any changes during the counseling process, such as withdrawal of candidates or additional seat allocations, the system should be capable of handling and reflecting those changes.

3. Complex Allocation Scenarios: The proposed solution assumes a straightforward allocation process based on preferences and ranks. However, **real-world admission processes can involve more complex scenarios, such as reserved categories, quota systems, or inter-institute branch transfers.** The system may need to incorporate additional rules and algorithms to handle such scenarios effectively.

4. User Experience: The user interface design should be intuitive, user-friendly, and accessible across different devices. The system should also provide clear instructions and guidance to candidates during the registration and preference submission stages, **but we are restricting to a menu-driven terminal run program.**

5. Integration with Exam Systems: The proposed **solution assumes that the qualifying exam ranks are available and can be directly incorporated into the system**. However, integrating with the exam systems, such as JEE-Advanced and JEE-Main, to **fetch and update ranks automatically would enhance the efficiency and accuracy of the admission process.**

6. Security and Privacy: The solution **should prioritize the security and privacy of candidate data.** Implementing robust security measures, encryption techniques, access controls, and regular data backups are essential to protect sensitive information.

7. Maintenance and Support: Once deployed**, the system requires ongoing maintenance and support.** It should be **regularly updated to accommodate changes in admission policies, exam patterns, or other requirements.** Adequate technical support should be available to address any issues or queries from candidates or institutes.

8. Customization and Adaptability: The system may **lack flexibility to accommodate unique requirements and policies of different institutes, including variations in eligibility criteria, seat reservation policies, and branch-specific rules**. Customization options and adaptability to different institutional needs are crucial for effective implementation.

9. Transparency and Accountability: The system **may not provide a transparent and auditable process for decision-making and allocation.** Ensuring transparency, accountability, and the ability to track and trace the decision-making process are vital for maintaining trust in the admission system.

10. Resource Constraints: The system may **face resource constraints such as limited computing power, bandwidth, or storage capacity.** These constraints can impact the **performance and scalability** of the software solution.

Observations with respect to society, legal, ethical perspectives

SOCIETAL

1. Fairness Promotion: The software system can help address biases and inequalities by ensuring **a fair and transparent admission process** that offers equal opportunities to all candidates, regardless of their background or personal characteristics.

2. Increased Accessibility: By implementing an accessible software system, **individuals with disabilities can participate in the admission process on an equal footing,** promoting inclusivity and diversity in engineering institutes.

3. Public Trust: A transparent and accountable software system **fosters public trust and confidence** in the admission process, as stakeholders can understand and verify the fairness and integrity of the decisions made.

LEGAL

1. Data Privacy Compliance: The software system should **adhere to relevant data privacy laws**, such as obtaining informed consent for data collection and processing, protecting personal information, and ensuring secure storage and transmission of data.

2. Intellectual Property Protection: The system should respect intellectual property rights, **avoiding unauthorized use or reproduction of copyrighted materials** during the admission process.

3. Regulatory Compliance: The software system should comply with all applicable legal regulations related to admission processes, ensuring that it **operates within the boundaries of the law.**

ETHICAL

1. Bias Mitigation: The software system should be designed to minimize biases and discrimination, ensuring that candidates are **evaluated based on their merit and qualifications rather than irrelevant factors.**

2. Ethical Decision-Making: The system should incorporate ethical considerations into its algorithms and decision-making processes, **avoiding unethical practices that could harm individuals or groups** and ensuring the decisions made align with ethical standards.

3. Stakeholder Well-being: The software system should prioritize the well-being and interests of all stakeholders involved, including candidates, institutes, and society at large, while **ensuring that no harm or unfair advantage is imposed on any party.**

Learning outcomes

The development of a software system for the engineering counselling and admission process offers several learning outcomes for us:

- *Enhancing Technical Skills:* This project allows us to enhance our technical skills in software development, database management, and system design. We will have the opportunity to strengthen our programming abilities by working with relevant languages, frameworks, and tools required to build a robust and scalable software system.

- *Fostering Algorithmic Thinking:* Developing the seat allocation algorithm requires us to think algorithmically and employ problem-solving skills. We will gain valuable experience in designing and implementing efficient algorithms that consider various factors such as candidate preferences, ranks, and seat availability.

- *Mastering Data Management:* Managing a substantial amount of data, including candidate details, institute and branch information, seat matrix, and preferences, will sharpen our skills in data organization, storage, and retrieval. We will learn how to design and optimize databases to handle complex data structures effectively.

- *Improving User Experience Design:* Designing an intuitive and user-friendly interface for candidates to register, provide preferences, and view seat allocation results necessitates an understanding of user experience design principles. We will develop the ability to create interfaces that are easy to navigate, visually appealing, and meet the needs of the users.

- *Gaining Project Management Expertise*: Developing a software system involves project management skills, including requirement analysis, task planning, and coordination with stakeholders. We will gain experience in managing the development lifecycle, from requirements gathering to design, implementation, testing, and deployment.

- *Acquiring Domain Knowledge:* This project will provide us with in-depth knowledge of the engineering counselling and admission process. We will become familiar with the rules, guidelines, and procedures followed by institutes and governing bodies, enabling us to apply this knowledge in similar domains or future projects.

- *Enhancing Problem-Solving and Decision-Making Skills:* Throughout the project, we will encounter numerous challenges and decisions related to algorithm design, user requirements, and system performance. We will develop our problem-solving and decision-making skills as we analyse options, evaluate trade-offs, and make informed choices to overcome obstacles and deliver an effective solution.

Overall, this project will provide us with valuable learning experiences, combining technical skills, domain knowledge, and project management expertise. It will equip us with transferable skills applicable to other software development projects and enhance our employability in the software industry.

References

1. "The 'College Admissions Problem' with Constraints" by Abdulkadiroğlu and Sönmez (2003) discusses the Gale-Shapley algorithm and its guarantee of stability in college admissions.

2. "House Allocation with Existing Tenants" by Abdulkadiroğlu and Sönmez (1999) explores the strategy-proofness of the Gale-Shapley algorithm in the context of house allocation.

3. "On the Existence of Pareto-Optimal Stable Matchings" by Kojima and Manea (2010) discusses the Pareto efficiency of stable matchings. This paper provides theoretical support for the Gale-Shapley algorithm's ability to achieve Pareto efficiency.

4. "Admissions Reform at Chicago: A Case Study" by Fisman and Kamenica (2007) explores individual optimality in the context of college admissions. This study examines the application of the Gale-Shapley algorithm at the University of Chicago and its effectiveness in assigning students to their most preferred colleges.

5. "Algorithmic Game Theory" by Nisan, Roughgarden, Tardos, and Vazirani (2007) provides an overview of the practical implementation and scalability of the Gale-Shapley algorithm. This book discusses the algorithm's application in real-world scenarios, including college admissions and matching markets, and highlights its efficiency and practicality. It also discusses empirical studies and real-world examples that demonstrate the algorithm's effectiveness.