CS F213 Object-Oriented Programming 2022

Project - 5: Practice School Allotment System

2020B3A70851P Naman Madan
2021A7PS0661P Yash Pandey
2021A7PS0657P Priyank Shethia
2020B3A71425P Bhavishya Garg
2020A7PS1685P Rahul Ganesh

Anti-plagiarism Statement

We declare that the work done as a part of this assignment has been completely our own. We are conscious that the incorporation of code from other sources counts as plagiarism, and usage of such code is a violation of the policies of the course and the university. We have not referred to other students' work, nor have we let other groups use our own code for this project.

Contribution Table

Name	Contribution	Signature
Naman Madan	Documentation, USE case diagram, UML diagram	[illegible] Scanned with CamScanner
Yash Pandey	GUI Classes and methods, Implementation of Classes and logic of allotment and viewing results, AllotmentData and other classes along with integration into the program of GUI. Debugging and Testing. UML Sequence diagram, UML Use-Case Diagram	Yash Pandy
Priyank Shethia	GUI Classes and methods, Implementation of Main, Admin, AllotmentData and other classes along with integration into the program of GUI. Debugging and Testing	Priyank
Bhavishya Garg	Documentation, UML Class diagram,UML Sequence Diagram	Bhavishya
Rahul Ganesh	UML Class Diagram, UML Sequence Diagram. Debugging, testing. Implementation of classes, GUI and integration.	Rahul

VIDEOS

Drive Link: https://drive.google.com/drive/folders/1hizwJMRuWWqj8vQw1UJD6Em-lCttOU3-?usp=share_link

Design Pattern Used/Applicable

Singleton design pattern -
In the singleton pattern, a class that is edited by other classes and their methods is kept with a private constructor which creates a single object of the class which can be accessed by a method such as getObject() or getSingleInstance() of that class to access that singleton object and edit it in place.

This design pattern would be suitable for our project as the class allotmentData is accessed for all kinds of data, throughout the program files and if we could create a private singleton object inside that class and access it instead, we would not have to keep every field and method static.

Moreover, this would still solve the problem of all students and admin having common data as the same object is being edited everywhere, not causing a problem in multithreading.

This could be implemented with the following additional code in allotment data.

```
Class allotmentData{
Private alotmentData dataObject;

Public static allotmentData getObject(){
        Return dataObject;
}
```

This design pattern is used when we want to create a single instance of the class that can be used by classes. This is mostly used for multithreading and database implementation.

We have not used this pattern, but it is implementable in the *allotmentData* class. There could be just one instance of it, accessible only with the method. Currently, we have made the class static and access its methods and objects through allotmentData.methodName(), but this could be accomplished with allotmentData.getObject().methodName() with non-static methods.

In the future, if we get to work more on this project. Working with a singleton allotmentData class would be ideal and a key goal.

SOLID Principles

Throughout our code, we have tried to make the code readable and reduce redundancy along with keeping a centralized access to stations and students for editing and allotment in the allotmentData's allStudents and allStations ArrayLists, in all other storages of hashmaps and ArrayLists, students and stations are stored as their IDs, and they are changed/accessed in place in the allStudents and allStations ArrayLists. Moreover, code is kept separate with a file for each class and classes having methods only relevant to their objects, apart from the allotmentData class.

Single Responsibility Principle -
Different classes are used for different purposes.

The main method has less number of functions while the other classes have proper methods that denote a single yet important function. For example, The allotmentData class only deals with data and has functions like viewStation, getRound, studentData, viewUnalloted, inputStations etc. Other classes have also implemented this principle.

Open-Closed Principle - Classes are open for extension, but closed for modification
The defaultStudent(abstract class) and defaultStation class in the code implement this principle, where Student and Station classes are children of these classes. They can be extended to add new features to students but the defaultStation and defaultStudent classes need not be changed.

Newer type of students can be created by implementing the allottableStudentInterface and/or the defaultStudentInterface for students with different fields like internships or multiple type of preferences.

Liskov Substitution Principle - Subclasses are substitutable for their parent classes.
The defaultStation class is initialized once to create a defaultStation (i.e. a station for students who cannot be alloted to any available station). The Station class extends the defaultStation class.

As the allotment of a student can be defaultStation or a normal station, LSP is applicable when returning a station in allotmentData's getStation(String stationId) method.

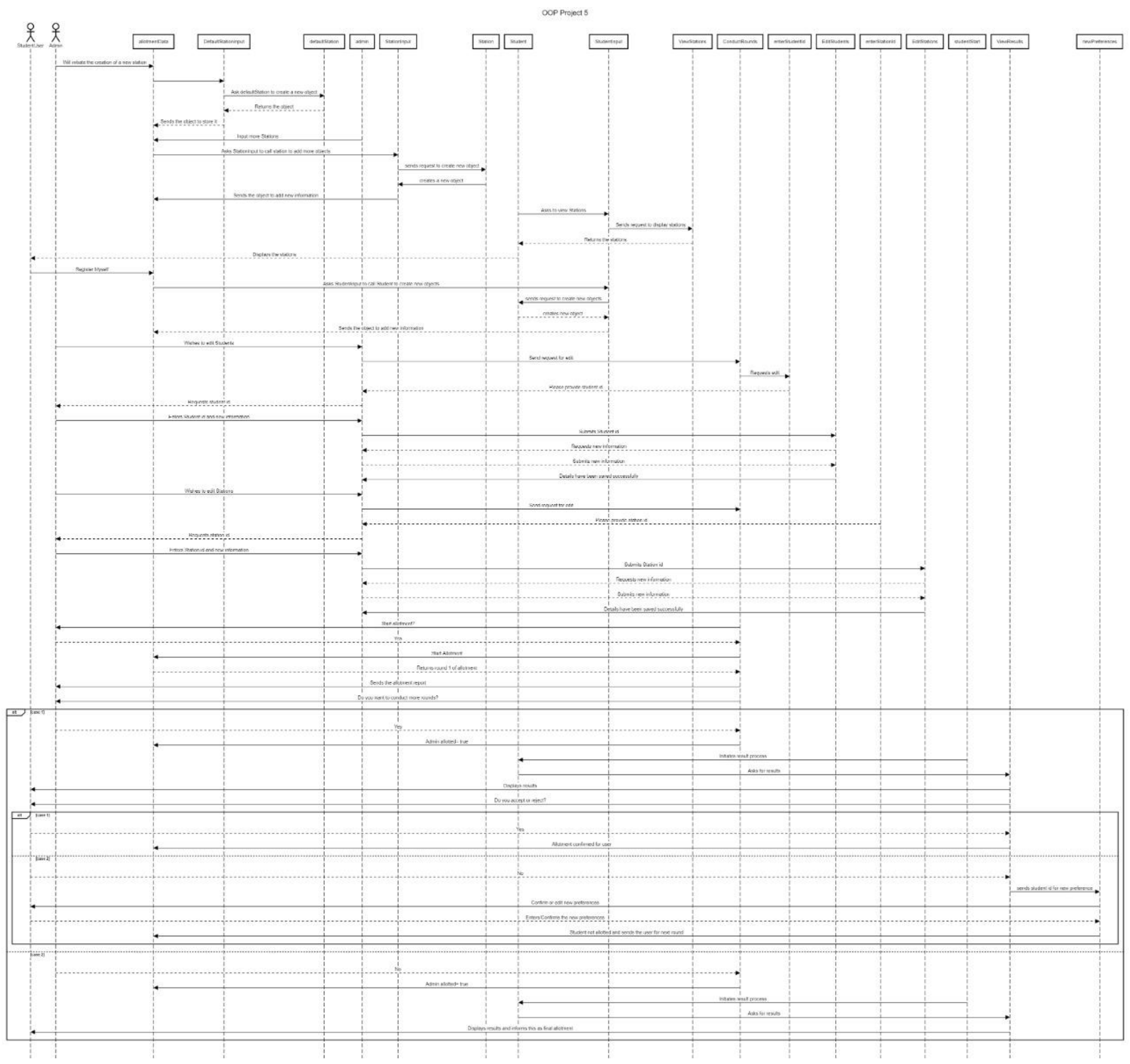
Interface Segregation Principle - Separate the interfaces.
The interfaces of defaultStudentInterfaceor and allottableStudentInterface are kept different as they have methods of different kind, with default interface having standard methods of a student and allottable interface having methods specific to allotment.
In the interfaces implemented by defaultStudent and defaultStation, the defaultStudentInterface and allotableStudentInterface are separated to distinguish between objects of the Student class and objects which have already been allotted a station. Similarly, the defaultStationInterface and preferenceStationInterface have also been separated for better clarity of their functionality.

Dependency Inversion Principle - Classes should be dependent on interfaces and abstract classes
The major classes in the code are dependent on interfaces and abstract classes, such as the Student Class which extends the defaultStudent abstract class and implements the defaultStudentInterface. The Station class too implements the preferenceStationInterface

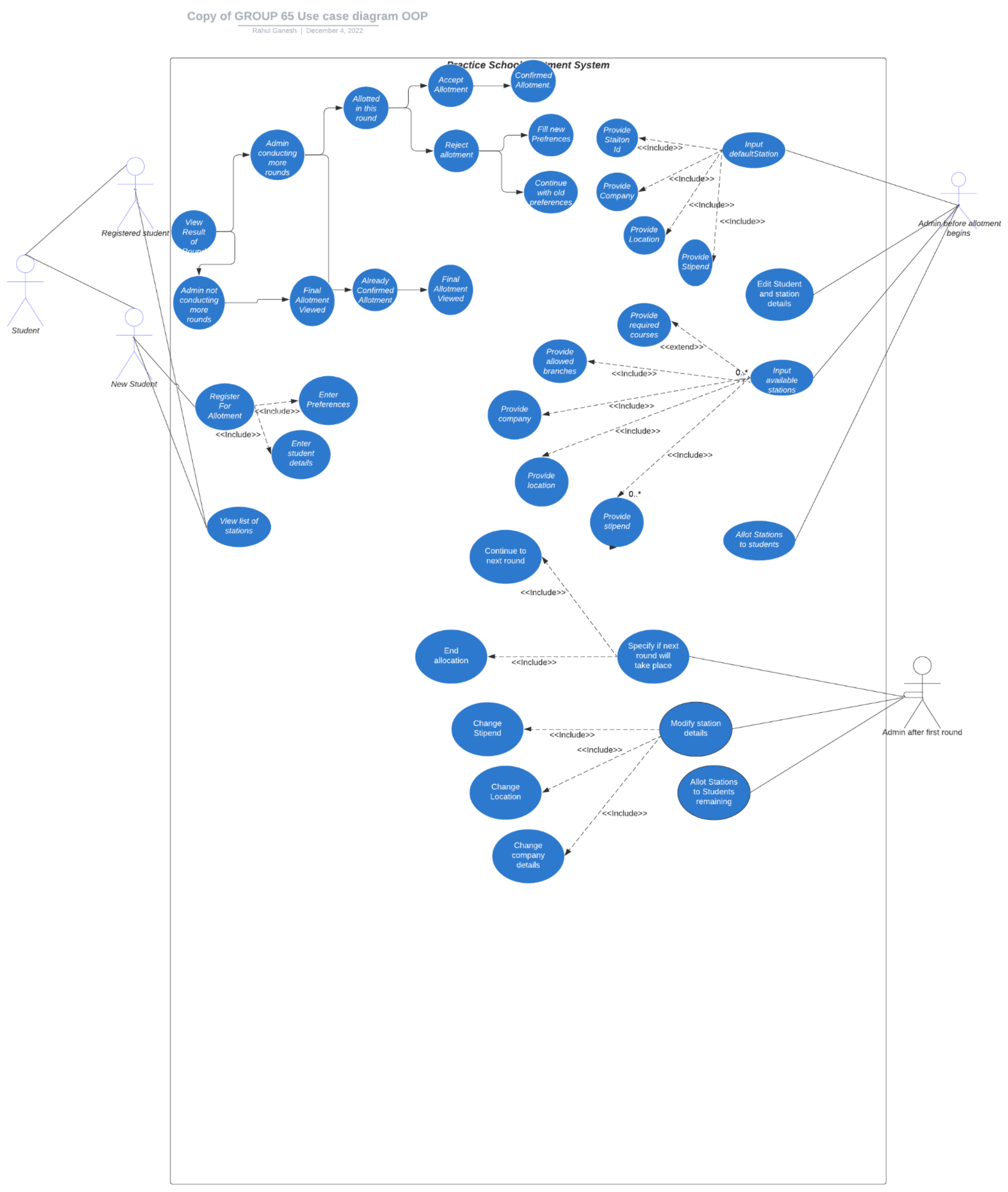
As all students and admin class have a common database, we have used static allotmentData class which is accessed in synchronized manner by students and admin, this could've been done by creating a single object of allotmentData and passing it to all threads

UML Diagrams

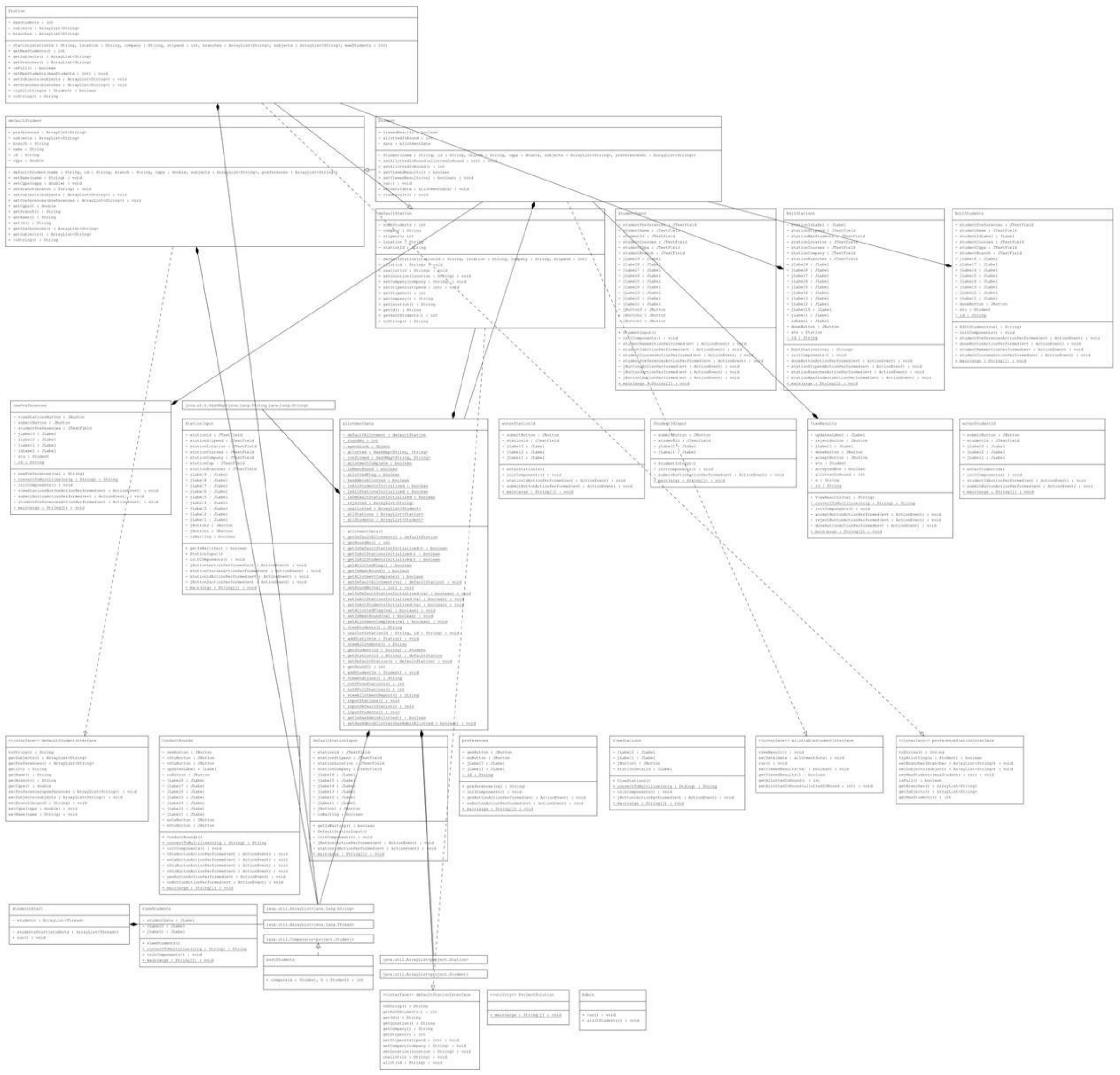
1)Sequence Diagram:


OOP Project 5
StudentUser
Admin
allotmentData
DefaultStationInput
defaultStation
admin
StationInput
Station
Student
StudentInput
ViewStations
ConductRounds
enterStudentId
EditStudents
enterStationId
EditStations
studentStart
ViewResults
newPreferences
Will initiate the creation of a new station
Ask defaultStation to create a new object
Returns the object
Sends the object to store it
Input more Stations
Asks StationInput to call station to add more objects
sends request to create new object
creates a new object
Sends the object to add new information
Asks to view Stations
Sends request to display stations
Returns the stations
Displays the stations
Register Myself
Asks StudentInput to call Student to create new objects
sends request to create new objects
creates new object
Sends the object to add new information
Wishes to edit Students
Send request for edit
Requests edit
Please provide student id
Requests student id
Enters Student id and new information
Submits Student id
Requests new information
Submits new information
Details have been saved successfully
Wishes to edit Stations
Send request for edit
Please provide station id
Requests station id
Enters Station id and new information
Submits Station id
Requests new information
Submits new information
Details have been saved successfully
Start allotment?
Yes
Start Allotment
Returns round 1 of allotment
Sends the allotment report
Do you want to conduct more rounds?
alt
[case 1]
Yes
Admin allotted= true
Initiates result process
Asks for results
Displays results
Do you accept or reject?
alt
[case 1]
Yes
Allotment confirmed for user
[case 2]
No
sends student id for new preference
Confirm or edit new preferences
Enters/Confirms the new preferences
Student not allotted and sends the user for next round
[case 2]
No
Admin allotted= true
Initiates result process
Asks for results
Displays results and informs this as final allotment

2)Use Case Diagram



3) UML Class Diagram


Station
defaultStudent
Student
DefaultStation
StudentInput
EditStation
EditStudents
newPreferences
java.util.HashMap<java.lang.String,java.lang.String>
StationInput
allotmentData
enterStationId
StudentIdInput
ViewResults
enterStudentId
<<interface>> defaultStudentInterface
ConductRounds
DefaultStationInput
preferences
ViewStations
<<interface>> allotableStudentInterface
<<interface>> preferenceStationInterface
studentsStart
viewStudents
java.util.ArrayList<java.lang.String>
java.util.ArrayList<java.lang.Thread>
java.util.Comparator<project.Student>
sortStudents
java.util.ArrayList<project.Station>
java.util.ArrayList<project.Student>
<<interface>> defaultStationInterface
<<utility>> ProjectSolution
Admin

SUMMARY OF WORKING

A. Firstly, allotmentData's static methods to input default and normal stations is called, which waits till the user presses on the submit button and is done. Then the static method to input students is called. These methods in turn call the GUI pages and wait until their input is done to continue through boolean flags

B. As the number of students is dependent on the input, we keep an ArrayList for student threads and initialize each thread into the array by using objects stored in allotmentData's allStudents ArrayList which stores students at time of input. Once the array is created, we pass this array of threads to an object of class studentsStart which waits until allottedFlag is set to true after admin goes through a round of allotment. When notified by admin thread, the studentsStart's run() method runs all student threads one by one, and as we are using GUI, until the student doesn't press on accept/reject (when Admin is conducting more rounds) or done (when no more rounds are being conducted), the current run method doesn't finish to call the next student's run().

1. Program starts by asking for default station input, the fields required are ID, Company, Stipend and Location. This station is the allotment given to a student who cannot be allotted to any of his preferences. It has no branch or course criteria along with no constraint on the number of students that can be allotted to it.

Constraints: stipend is a number, cannot be zero. ID and rest of the fields are strings Once fields are filled, press Submit to move on.

2. Next, enter the station details, where additional details of maximum capacity and courses, branches are required. Branches allowed and courses required have to be entered as space separated. Max number of students is the number of students for the station to be full, has to be a number. All fields are necessary except courses and branches which can be kept blank for no criteria. Keep pressing submit to add more stations and press done to finish inputting stations.

3. Student details have to be filled In next, All fields are necessary, courses done have to be space-separated. Preferences have to be entered as a space-separated id number of stations. A button to view all stations is available. Keep pressing submit to input next student, press submit once all students are inputted.

4. Now, before allotment, Admin gets to see a home page where he can edit/view all stations and edit/view all students. At this point, before starting allotment, admin can change any detail of student

such as branch, course, preferences and edit courses and branch criteria for station. This cannot be done after allotment begins. Admin presses yes to begin first round of allotment

5. Now, allotment takes place according to the following logic. First, we sort the array of unallotted students according to CGPA, two students with the same CGPA are sorted randomly. Now, we pick the student with the highest CGPA from the array. Next, we find the station which is the first preference of the student. First, we check if the student can be allotted this preference by calling the method tryAlloting(Student s). The requirements are that the station must have remaining slots, the branch of student must be allowed and all courses required must be present in the student's subjects. If a student can be allotted, we add his id related to the station's id in the allotted hashmap of allotmentData and break loops to move to the next student. If the student cannot be allotted, we try allotting him his next highest preference. If the student cannot be allotted any preference, he is allotted the default allotment.

6. Now, after the 1st round of allotments, the admin page opens again, he can no longer edit students or edit branch, course criteria or capacity of stations. However, he can still edit details like stipend, location, etc. and view both stations and students On his home page, he views all students and if they are confirmed allotments or unconfirmed. To continue, the admin must press yes or no, if he wants to conduct more rounds or not.

7. Next, the student pages open one by one, they can see their details and allotment. If the admin is conducting more rounds, he gets an option to accept or reject the allotment. If he accepts the allotment, his id is put with the station's id in allotmentData's confirmed hashmap. Next time the student views his results, he will view the same allotment along with the round he was allotted in. If the student rejects allotment, a new window opens where he can edit his preferences and press done to confirm his participation in the next round according to the updated preferences.

8. This multithreading continues with the Admin's homepage opening after each round of allotment and getting a choice to conduct more rounds, view/edit stations and view students along with latest allotment updates and student pages opening afterwards.

ERROR HANDLING AND UNIQUE FUNCTIONS IMPLEMENTED

1. All fields in student input have to be filled, if any are left empty or given wrong data value, we show a pop-up to correct the values and the record will not be accepted until all values are valid.

2. All fields in station input except courses have to be filled with some content, if any are left empty or given wrong data types, a pop-up to correct the values will be displayed and the record will not be accepted until all values are valid.

3. In student and station input, ID number already used by a different student and station cannot be used and a pop-up to use a unique ID will be shown.

4. Before allotment begins, the admin gets to view a home page with all 4 buttons functional of edit/view students and edit/view stations. This is the last point where admin can change student details or change allotment-sensitive criterias of stations like branches and courses. He gets a button to start allotment.

5. In editing stations and editing students, the admin first has to enter a ID, which if it does not exist, it shows an error to enter a valid ID number.

6. After each round, the admin gets to view his home page, but without edit students so that allotments are not affected and he can change only non-allotment-related details of station-like location.

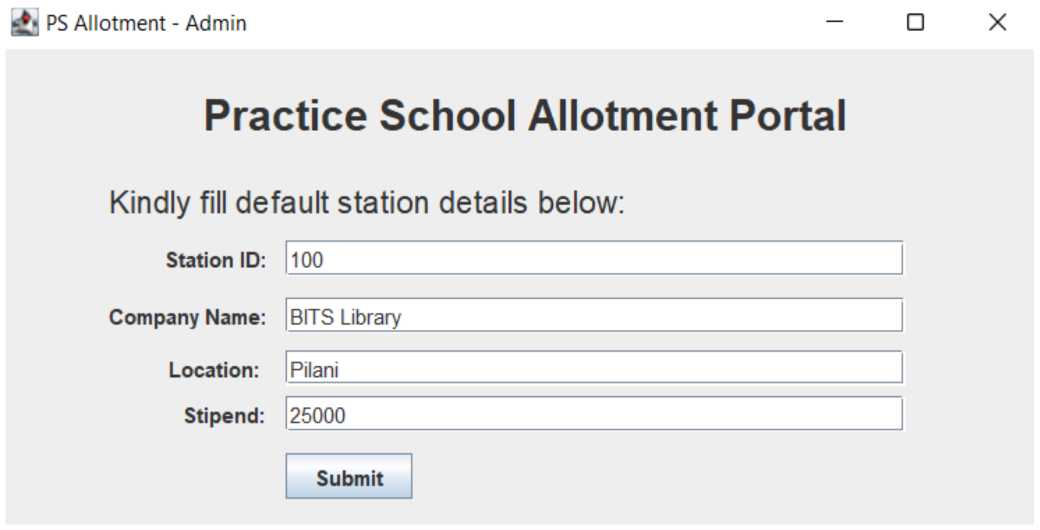
7. After each round, the admin gets a choice to continue more rounds. Students can only accept or reject allotments if the admin agreed to conduct another round, otherwise they can only press done as their allotment is final.

8. If a student rejects their allotment, we allow them to edit their preferences which shall be used for next rounds (unless edited again).

9. GUI has been implemented in the project, you can view the pages below.

GUI PAGES

1. Default Station Input page

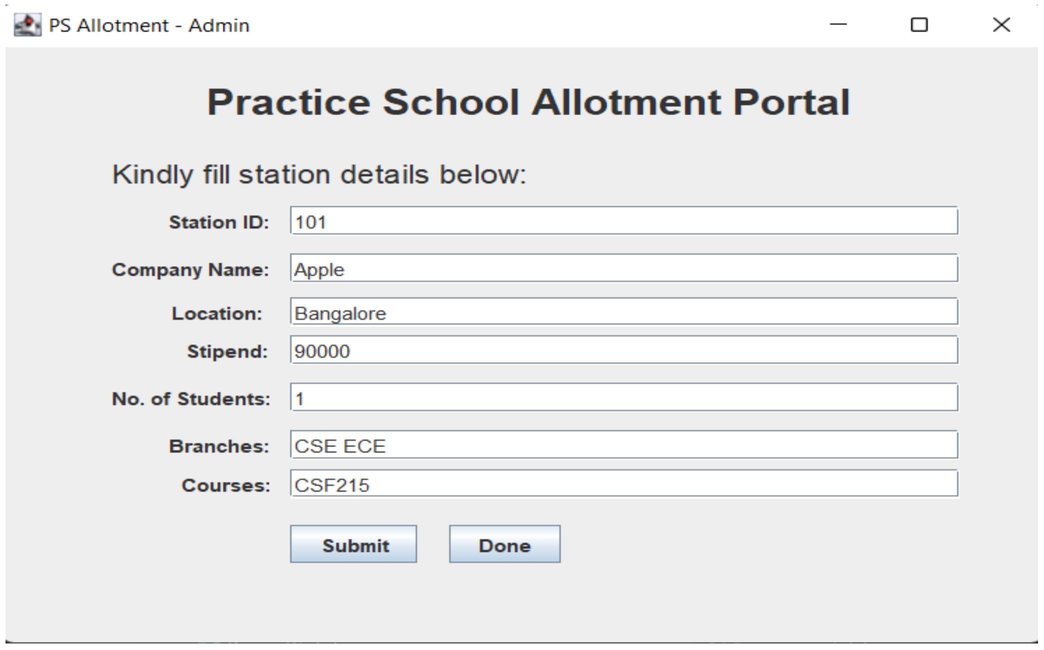


Practice School Allotment Portal

Kindly fill default station details below:

Station ID:	100
Company Name:	BITS Library
Location:	Pilani
Stipend:	25000

2. Stations Input Page

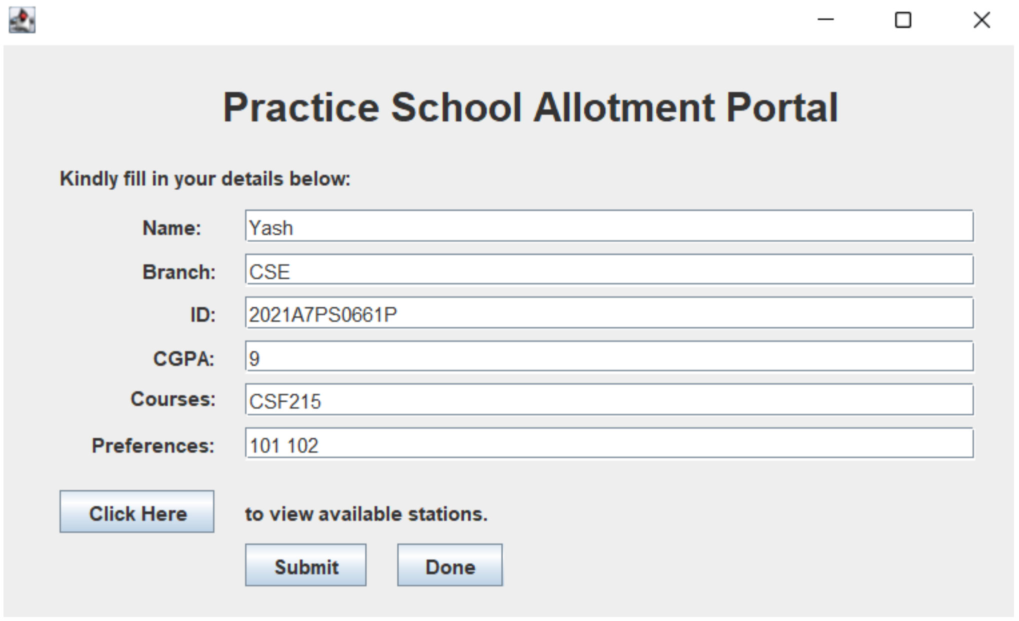


Practice School Allotment Portal

Kindly fill station details below:

Station ID:	101
Company Name:	Apple
Location:	Bangalore
Stipend:	90000
No. of Students:	1
Branches:	CSE ECE
Courses:	CSF215

3. Students Input Page

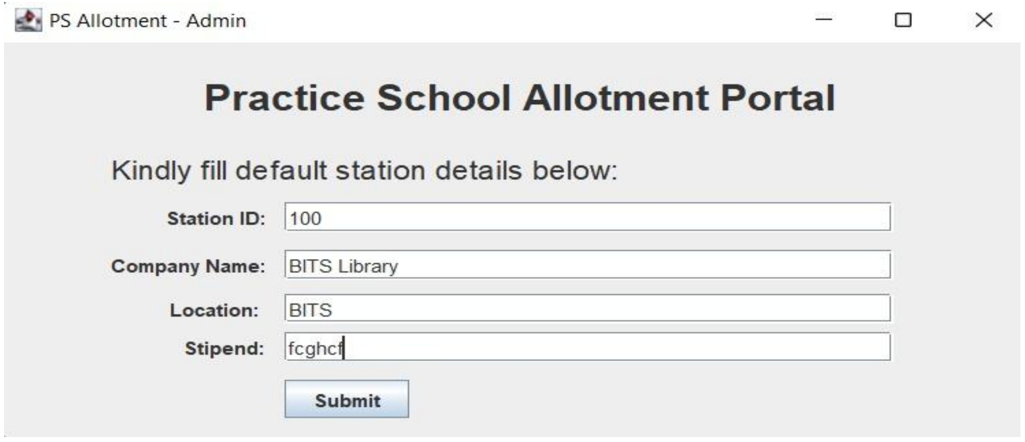


Practice School Allotment Portal

Kindly fill in your details below:

Name:	Yash
Branch:	CSE
ID:	2021A7PS0661P
CGPA:	9
Courses:	CSF215
Preferences:	101 102

4. Notification if wrong data type is added

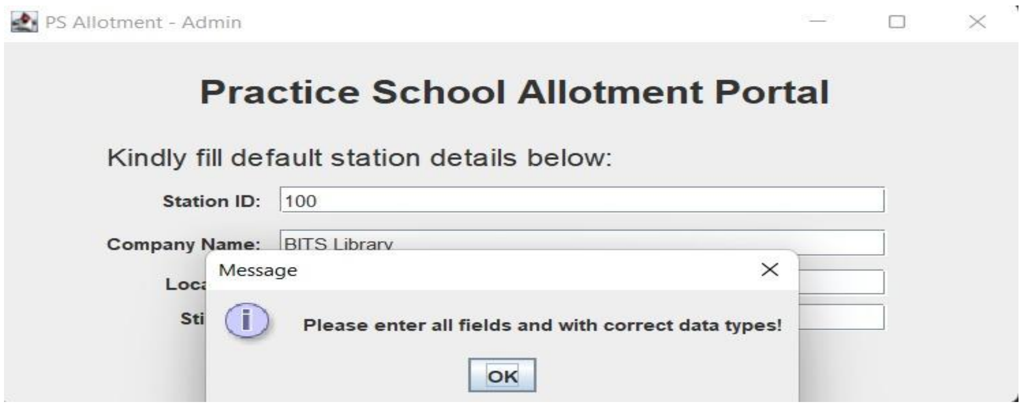


PS Allotment - Admin

Practice School Allotment Portal

Kindly fill default station details below:

Station ID:	100
Company Name:	BITS Library
Location:	BITS
Stipend:	fcghcf



PS Allotment - Admin

Practice School Allotment Portal

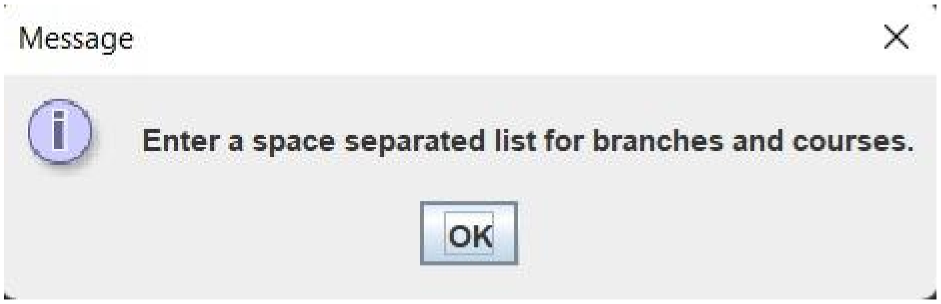
Kindly fill default station details below:

Station ID:	100
Company Name:	BITS Library

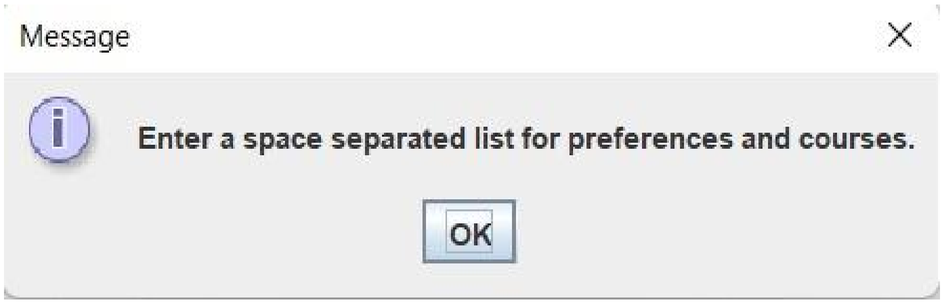
Message

Please enter all fields and with correct data types!

5. Notification if space separated list for branch and courses is not added

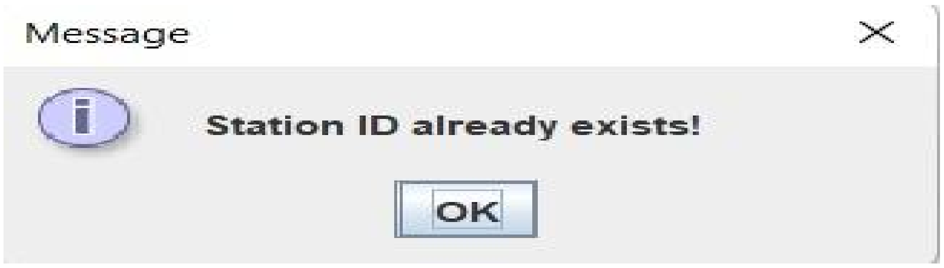

Message

Enter a space separated list for branches and courses.


Message

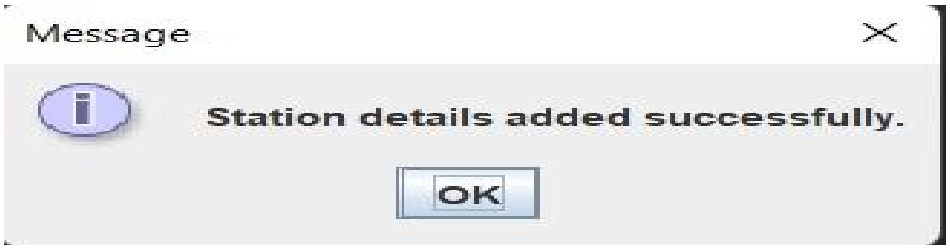
Enter a space separated list for preferences and courses.

6. Notification if same station id is added


Message

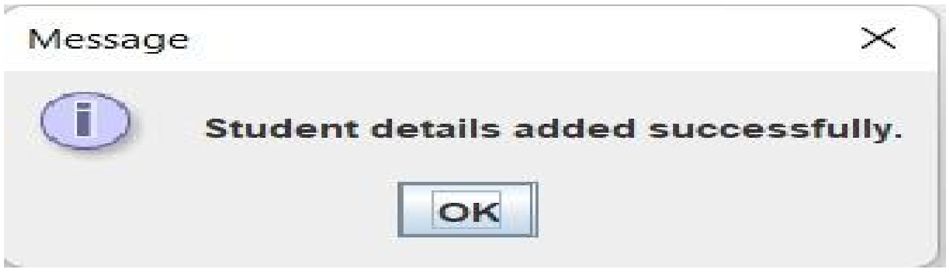
Station ID already exists!

7. Notification after submit button is clicked for station details


Message

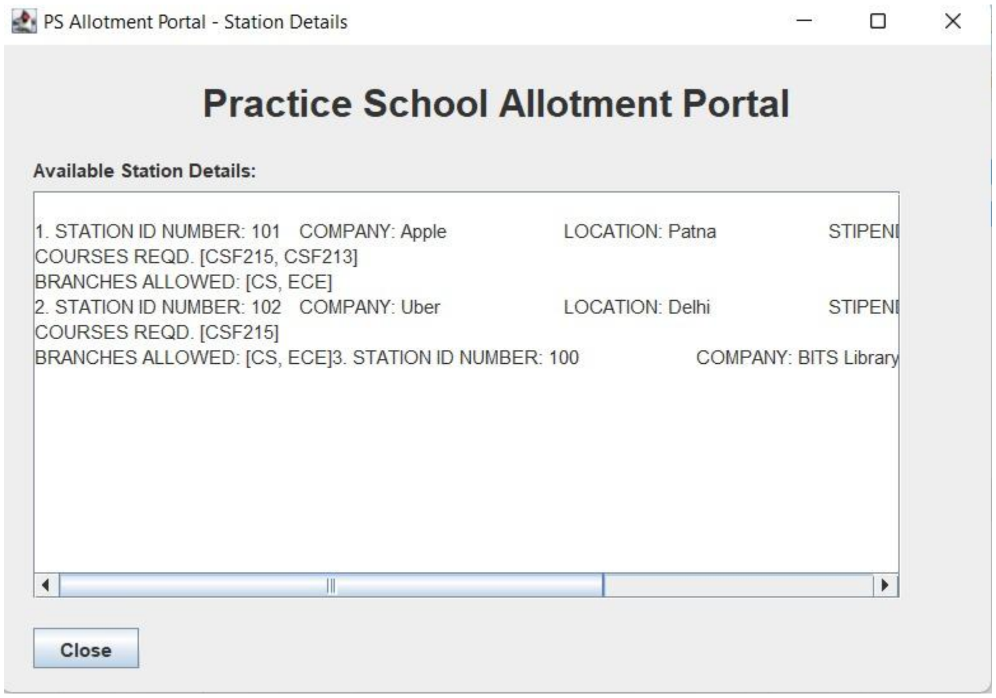
Station details added successfully.

8. Notification after submit button is clicked for student details


Message

Student details added successfully.

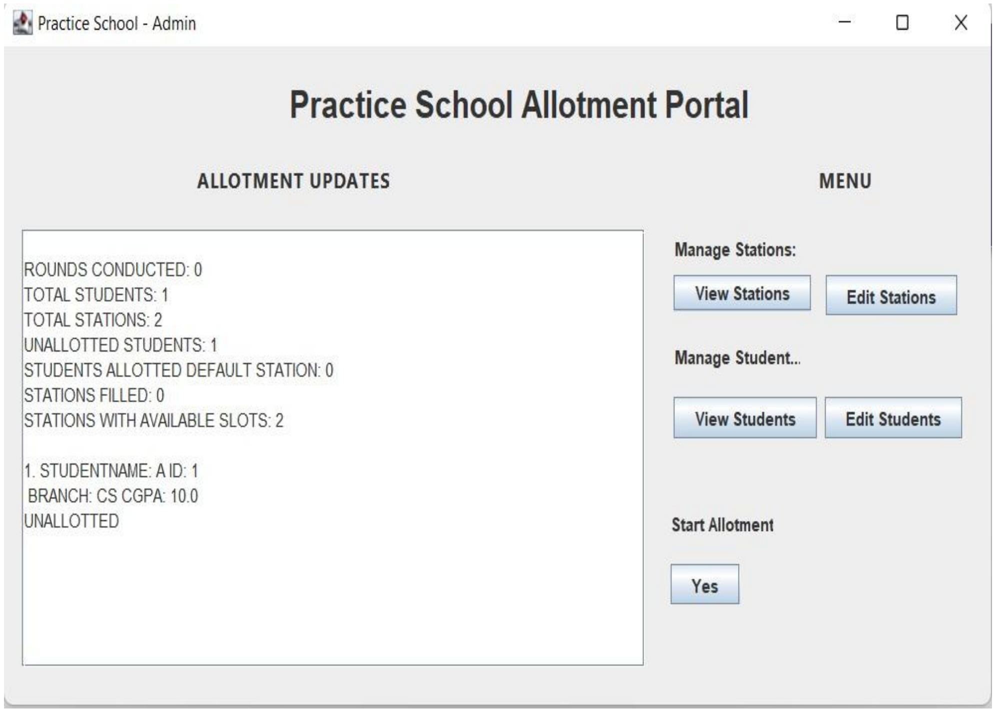
9. View Available Stations page


PS Allotment Portal - Station Details
Practice School Allotment Portal
Close

Available Station Details:

```
1. STATION ID NUMBER: 101    COMPANY: Apple                LOCATION: Patna              STIPENI
COURSES REQD. [CSF215, CSF213]
BRANCHES ALLOWED: [CS, ECE]
2. STATION ID NUMBER: 102    COMPANY: Uber                 LOCATION: Delhi              STIPENI
COURSES REQD. [CSF215]
BRANCHES ALLOWED: [CS, ECE]3. STATION ID NUMBER: 100                COMPANY: BITS Library
```

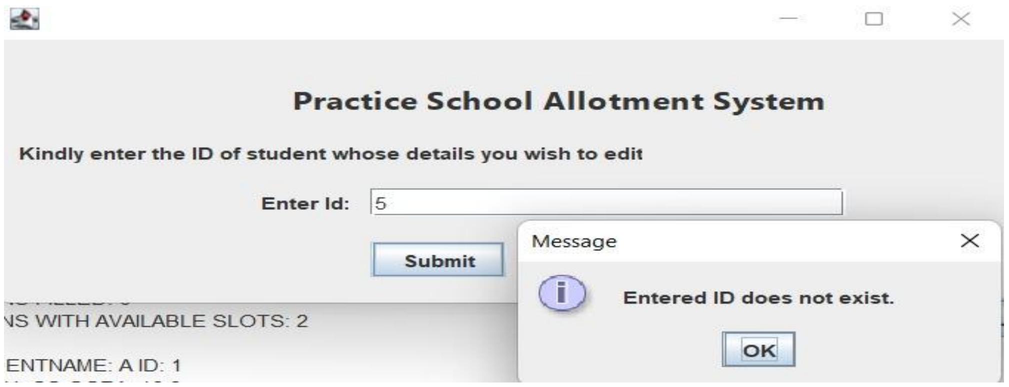
10. Allotment Update page



```
ROUNDS CONDUCTED: 0
TOTAL STUDENTS: 1
TOTAL STATIONS: 2
UNALLOTTED STUDENTS: 1
STUDENTS ALLOTTED DEFAULT STATION: 0
STATIONS FILLED: 0
STATIONS WITH AVAILABLE SLOTS: 2

1. STUDENTNAME: A ID: 1
 BRANCH: CS CGPA: 10.0
UNALLOTTED
```

11. Edit student first page along with Notification if student does not exist



Practice School Allotment System

Kindly enter the ID of student whose details you wish to edit

Enter Id: 5

Submit

NS WITH AVAILABLE SLOTS: 2

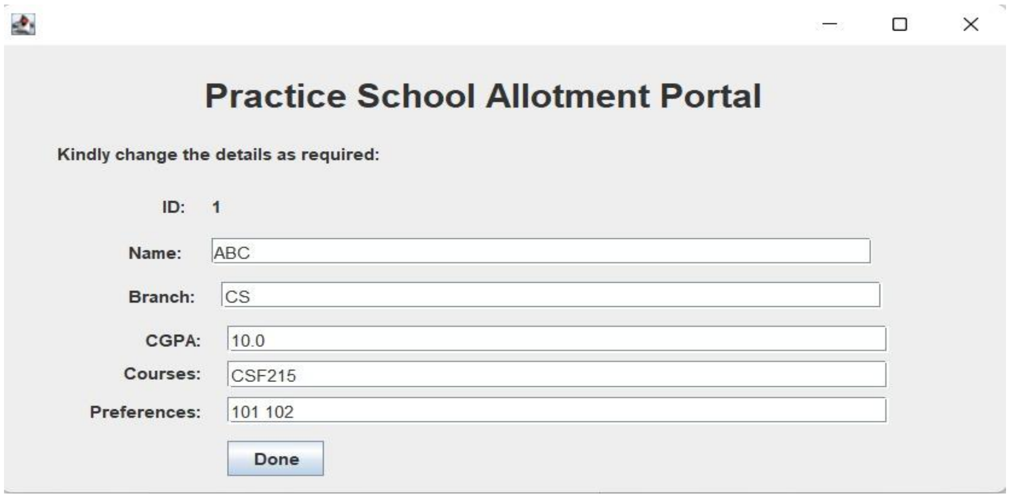
ENTNAME: A ID: 1

Message

Entered ID does not exist.

OK

12. Edit student second page



Practice School Allotment Portal

Kindly change the details as required:

ID: 1

Name: ABC

Branch: CS

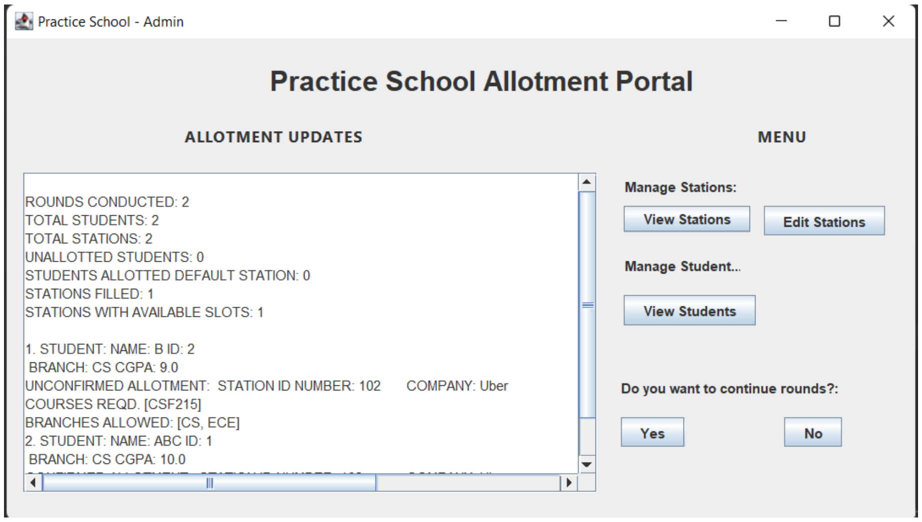
CGPA: 10.0

Courses: CSF215

Preferences: 101 102

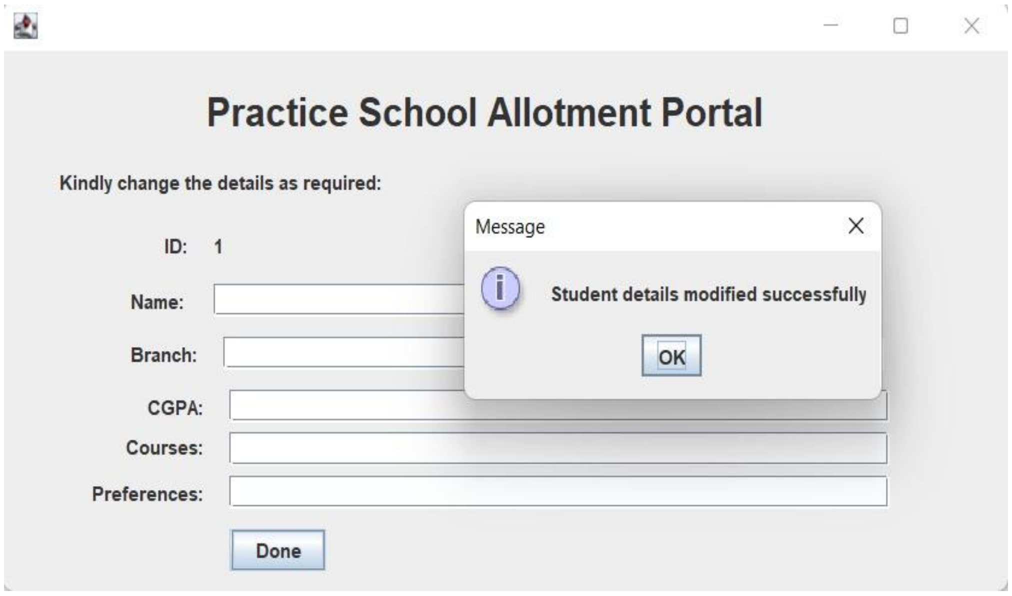
Done

13. Notification after student details are edited



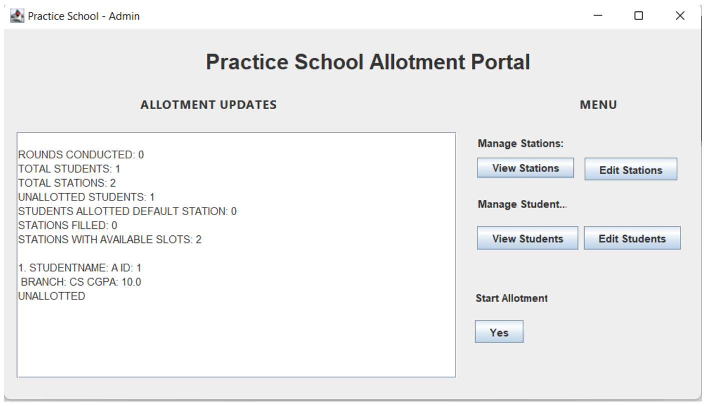
```
ROUNDS CONDUCTED: 2
TOTAL STUDENTS: 2
TOTAL STATIONS: 2
UNALLOTTED STUDENTS: 0
STUDENTS ALLOTTED DEFAULT STATION: 0
STATIONS FILLED: 1
STATIONS WITH AVAILABLE SLOTS: 1

1. STUDENT: NAME: B ID: 2
 BRANCH: CS CGPA: 9.0
UNCONFIRMED ALLOTMENT:  STATION ID NUMBER: 102      COMPANY: Uber
COURSES REQD. [CSF215]
BRANCHES ALLOWED: [CS, ECE]
2. STUDENT: NAME: ABC ID: 1
 BRANCH: CS CGPA: 10.0
```


Practice School Allotment Portal
Kindly change the details as required:
ID: 1
Name:
Branch:
CGPA:
Courses:
Preferences:
Done
Message
OK

Student details modified successfully

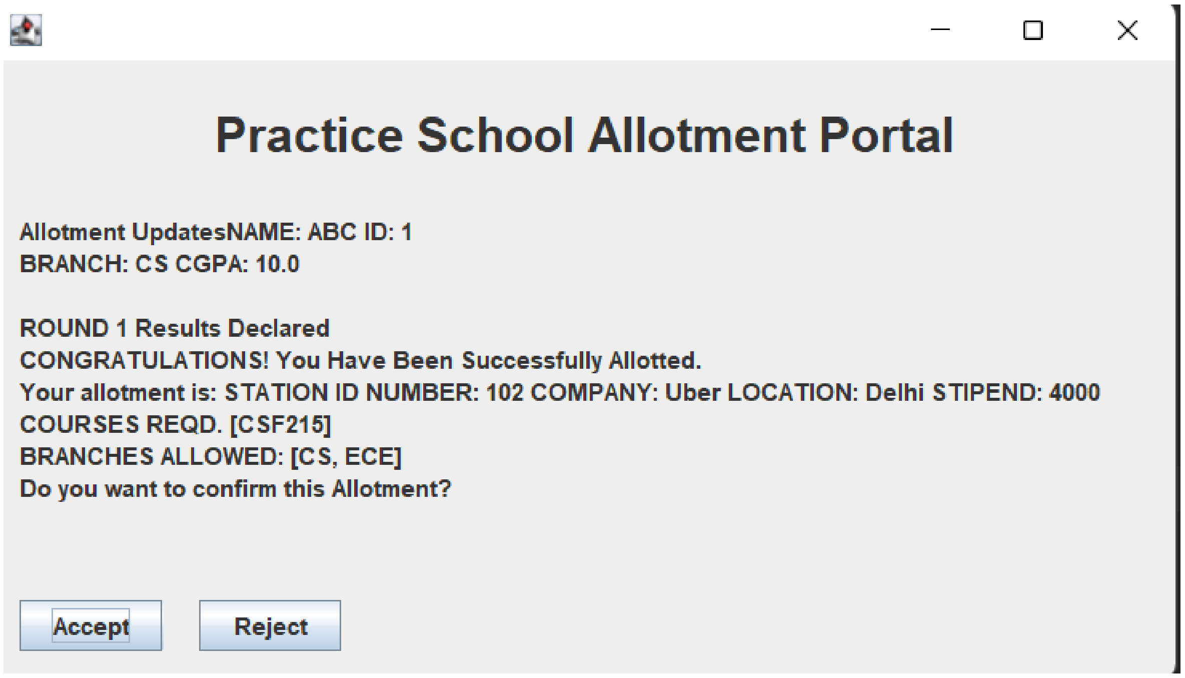
14. Admin clicks on yes to start allotment.



```
ROUNDS CONDUCTED: 0
TOTAL STUDENTS: 1
TOTAL STATIONS: 2
UNALLOTTED STUDENTS: 1
STUDENTS ALLOTTED DEFAULT STATION: 0
STATIONS FILLED: 0
STATIONS WITH AVAILABLE SLOTS: 2

1. STUDENTNAME: A ID: 1
 BRANCH: CS CGPA: 10.0
UNALLOTTED
```

15. Allotment Results after first Round. Student has the option to accept or reject the allotment.

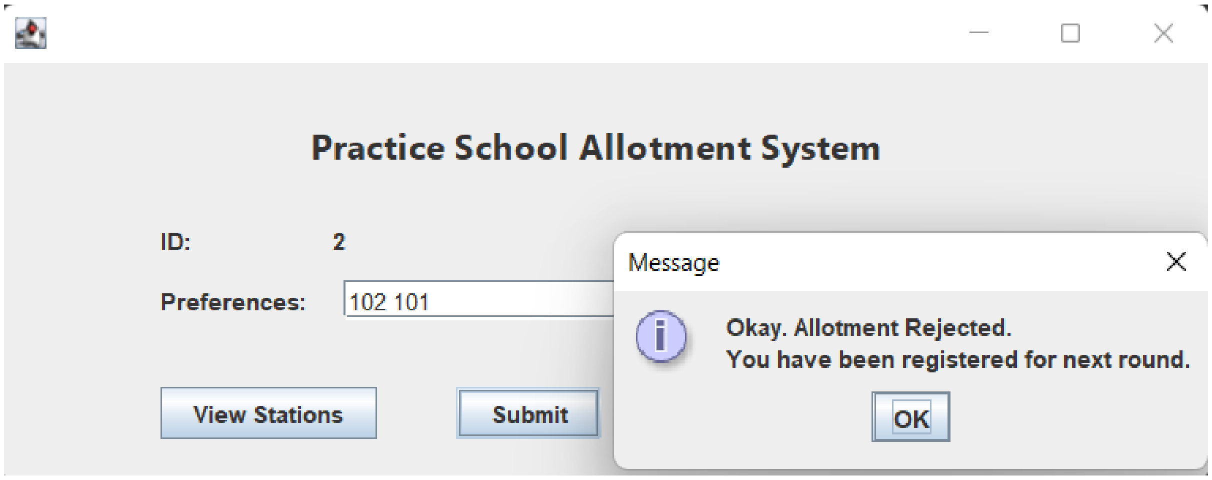


Practice School Allotment Portal

Allotment UpdatesNAME: ABC ID: 1
BRANCH: CS CGPA: 10.0

ROUND 1 Results Declared
CONGRATULATIONS! You Have Been Successfully Allotted.
Your allotment is: STATION ID NUMBER: 102 COMPANY: Uber LOCATION: Delhi STIPEND: 4000
COURSES REQD. [CSF215]
BRANCHES ALLOWED: [CS, ECE]
Do you want to confirm this Allotment?

16. Message after rejecting allotment

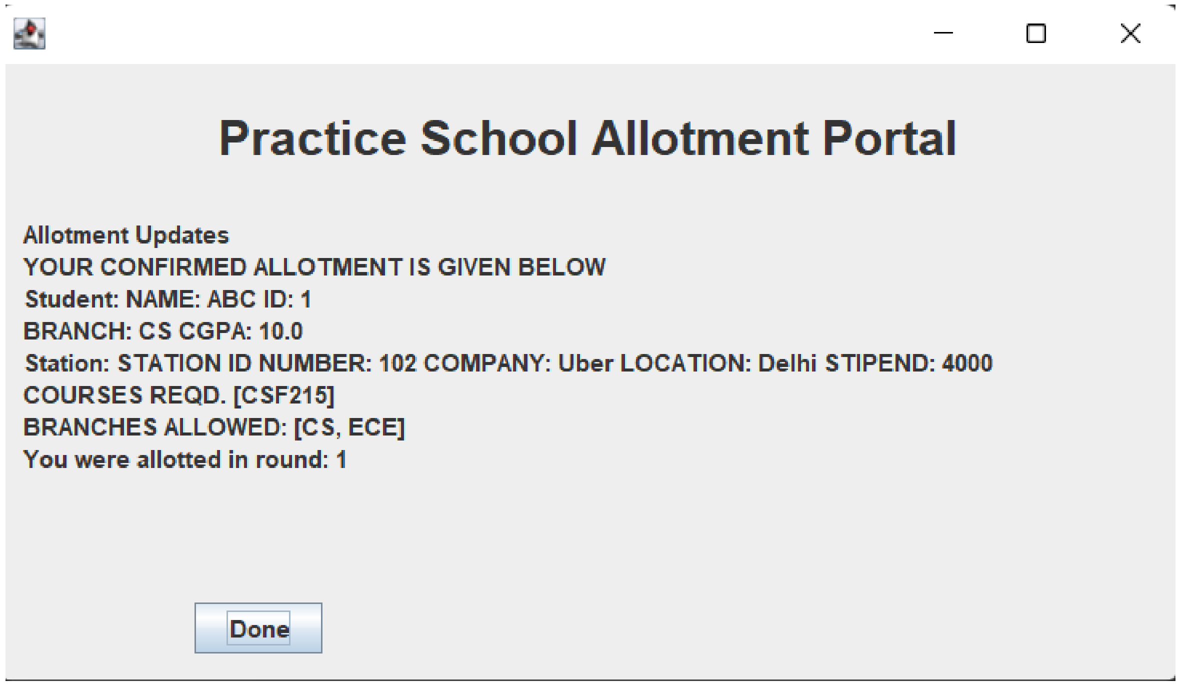


Practice School Allotment System

ID: 2
Preferences: 102 101

Message
Okay. Allotment Rejected.
You have been registered for next round.

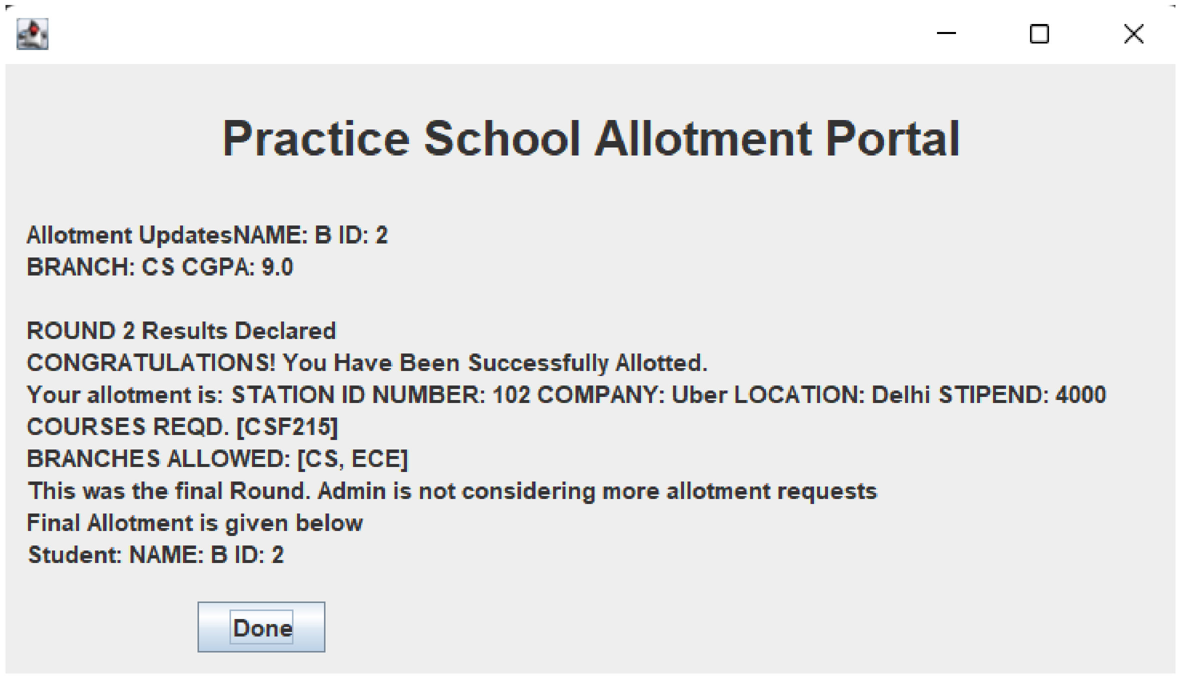
17. Message after Accepting allotment



Practice School Allotment Portal

```
Allotment Updates
YOUR CONFIRMED ALLOTMENT IS GIVEN BELOW
Student: NAME: ABC ID: 1
BRANCH: CS CGPA: 10.0
Station: STATION ID NUMBER: 102 COMPANY: Uber LOCATION: Delhi STIPEND: 4000
COURSES REQD. [CSF215]
BRANCHES ALLOWED: [CS, ECE]
You were allotted in round: 1
```

18. Message after allotment accepted in 2nd round

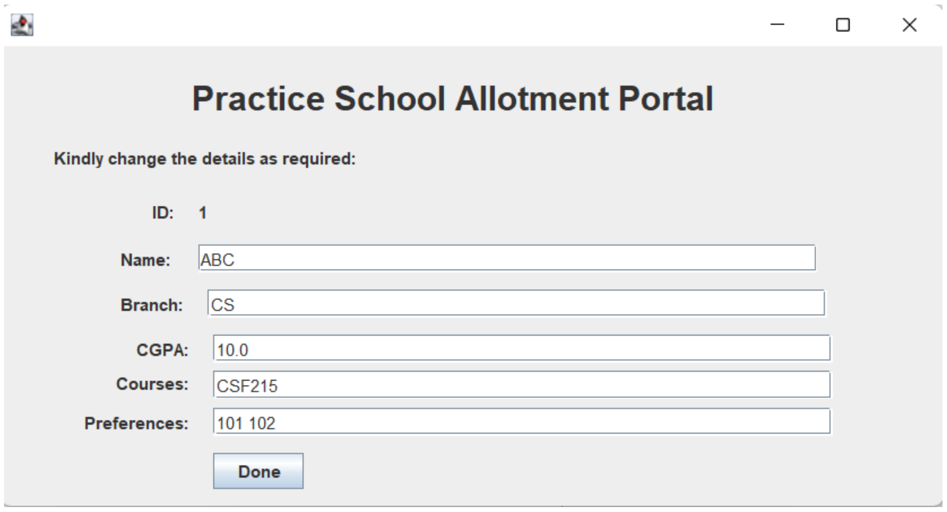


Practice School Allotment Portal

```
Allotment UpdatesNAME: B ID: 2
BRANCH: CS CGPA: 9.0

ROUND 2 Results Declared
CONGRATULATIONS! You Have Been Successfully Allotted.
Your allotment is: STATION ID NUMBER: 102 COMPANY: Uber LOCATION: Delhi STIPEND: 4000
COURSES REQD. [CSF215]
BRANCHES ALLOWED: [CS, ECE]
This was the final Round. Admin is not considering more allotment requests
Final Allotment is given below
Student: NAME: B ID: 2
```

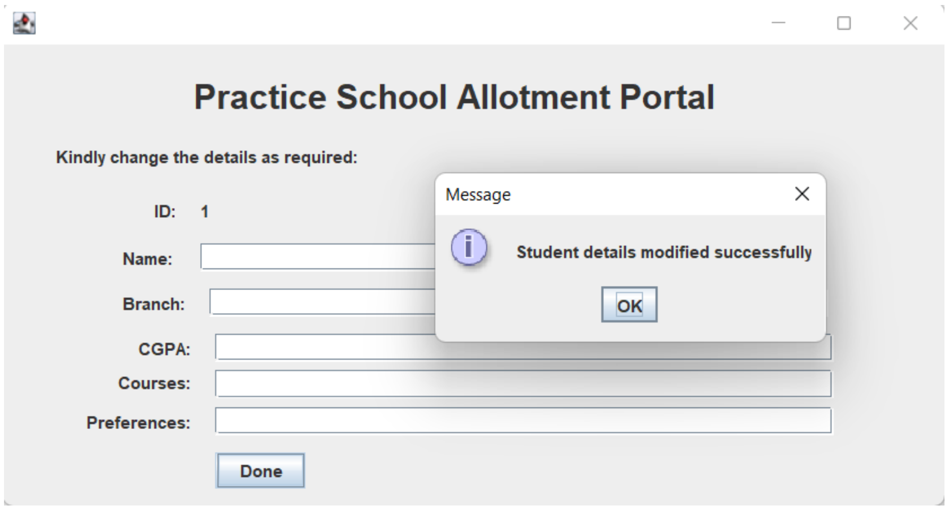
19. Modifying Student details



Practice School Allotment Portal

Kindly change the details as required:

ID:	1
Name:	ABC
Branch:	CS
CGPA:	10.0
Courses:	CSF215
Preferences:	101 102



Practice School Allotment Portal

Kindly change the details as required:

ID:	1
Name:	
Branch:	
CGPA:	
Courses:	
Preferences:	

Message

Student details modified successfully