B.Tech Computer Science and Engineering (Data Science)

-311 Program Semester – II

Academic Year 2023-24

Object Oriented Programming and Design

Mini Project

COLLEGE ADMISSION SYSTEM

Developed by

L006 Parth M. Dhadke

L020 Divyanshu Patil

L012 Siya Jain

Faculty In charge

Dr. Dhirendra Mishra

Dr. Priteegandha Naik

SVKM's NMIMS

Mukesh Patel School of Technology Management and Engineering

Vile Parle West Mumbai-56

Contents

1. ABSTRACT 3
2. INTRODUCTION 4
3. SCOPE 5
4. SOCIETAL NEED FOR THIS PROJECT 6
5. ORGANIZATION OF THIS PROJECT 7
6. LIST OF INPUTS & OUTPUTS 8
7. RESEARCH QUESTION 9
8. LITERATURE REVIEW 9
9. PROBLEM STATEMENT 10
10. LIST OF CLASSES AND MEMBER FUNCTIONS 11
11. BLOCK DIAGRAM 14
12. CLASS DIAGRAM 15
13. USE-CASE DIAGRAM 17
14. ACTIVITY DIAGRAMS 18
15. SEQUENCE DIAGRAM 24
16. OBJECT ORIENTED PROGRAMMING IMPLEMENTATION 25
17. ALGORITHM 27
18. FLOWCHART 27
19. CODE 31
20. OUTPUT 31
21. LIST OF ERRORS RESOLVED 50
22. SCOPE OF IMPROVEMENT 51
23. Bibliography 52

1. ABSTRACT

Our project creates an efficient instance of a College Admission System to streamline the process of admission in a college for students and make managing data for the administration easy. The system is created in an efficient and user-friendly way to ease the workload of the administrative offices of a college. It encompasses features that are crucial towards operating and writing of the code, such as file handling. In addition, the program also provides a very secure pathway for the student to use. The program uses database management systems to ensure data security. The code of uses the basic functionalities of C++ namely file management, inheritance, data hiding and encapsulation.

2. INTRODUCTION

A college admission system is a system created to ease the process of admission in colleges. It creates an organized system applied applications data which further helps to access a specific record from a huge chunk of data. It provides security to each one of its users by a username and password window which prevents anyone from accessing data of another person.

This process aims to streamline the process of admission for both the user and the manager of the system. This system helps to apply in any university by sitting anywhere around the world.

3. SCOPE

The scope of this project is:

1)Application Management: This includes managing the submission of applications from prospective students, which may involve online application portals, paper applications, or both. It also involves verifying the completeness and accuracy of application materials.

2)Admissions Criteria: Defining the criteria for admission, which may include academic performance (such as GPA or standardized test scores), extracurricular activities, letters of recommendation, essays or personal statements, and other factors deemed relevant by the institution.

3)Selection Process: Evaluating applicants based on the established criteria and making decisions about whom to admit. This process may involve individual review by admissions officers, committee review, or a combination of both.

4. SOCIETAL NEED FOR THIS PROJECT

Societal Need for a college admission system can arise from several factors:

Equal Opportunity: A structured admission system ensures that all individuals have an equal opportunity to access higher education regardless of their background, socioeconomic status, or other factors.

Resource Allocation: By implementing an admission system, institutions can efficiently allocate resources such as faculty, facilities, and financial aid to admit students based on their needs and qualifications.

3.Easy accessibility: By implementing these system students can apply in any college for admission by sitting anywhere across the world.

5. ORGANIZATION OF THIS PROJECT

When a student opens our system, he will be first asked to create an account, after that he will have to login onto the system. If anyone already has made an account, they will have to directly login into the system. After that they will have to input all the basic details required.

After that they will have a schedule to choose a slot for entrance exam and after taking the exam the students will have to fill in the marks obtained in their exam. After entering marks, the students will have to select a course of their choice. The system will check the eligibility of the student according to the marks obtained in the entrance exam.

6. LIST OF INPUTS & OUTPUTS

Input Variables:

1.choice (integer): User's choice to sign up (1) or log in (0).

2.name (string): User's name during sign-up.

3.age (integer): User's age during sign-up.

4.username (string): User's username during sign-up and log-in.

5.password (string): User's password during sign-up and log-in.

6.confirmUsername (string): Username entered during log-in verification.

7.confirmPassword (string): Password entered during log-in verification.

8.slot (integer): Slot number for selecting exam date.

9.marksScored (integer): Marks scored in the entrance exam.

Output Variables:

1.courseSelected (string): Course selected by the user.

2.examDate (string): Exam date selected by the user.

3.flag (integer): Flag to indicate successful authentication (1) or unsuccessful authentication (0).

4.suggestedCourses (vector<string>): List of suggested courses if the user does not qualify for the selected course.

5.newChoice (integer): User's choice of a new course from the list of suggested courses.

7. RESEARCH QUESTION

How to develop a simple, efficient “college admission system” which provides a user to input data, select his choice of course using concepts such as file handling, inheritance in Object Oriented Programming?

8. LITERATURE REVIEW

We reviewed the case study based on a College Admission System conducted by Kiwaunka Malik. The case study gave us an overview of admission systems. This helped us to gain knowledge and understanding of the system. Also gave us insights on deciding the input and outputs of the system.

[1]

We also reviewed a book by E. Balguruswamy which helped strengthen our knowledge of the fundamental concepts of OOPD and successfully implement the concepts we learned in our program as well.

[2]

9. PROBLEM STATEMENT

Several colleges lack a basic college admission system which leads to lack of efficiency and loss of data. Similarly, the students also don't have a platform to apply in a college or they cannot apply online to a college from anywhere around the world.

The college admission system includes creating accounts for security, organized storage of data. Along with it must have user friendliness and efficiency using applications of object oriented such as encapsulation, abstraction.

Proposed solution:

1) User authentication system:

This system helps to enhance security by asking for username and password to access the system. Due to this only the person who wants to access his/her account can have access to it.

2) Record Keeping:

This system helps to maintain a systematic form of data reducing the loss of data due to the high number of applications.

3) Efficient Process:

This system is user-friendly, accessible from anywhere around the world so this will improve efficiency to the next level.

10. LIST OF CLASSES AND MEMBER FUNCTIONS

1. Admission record class

Data Members:

Name(string):student name

Coursename(string):to enter course name

Marks(int):enter marks

Functions:

Getadmissiondetais():allows user to enter name,his choice of course,marks obtained

Udpaterecord():to store the data entered by user in databse

Accessrecord():allow admin to access the record stored

2. Student Class (inherits from admission records)

Data Members:

name (string): Student's name.

password (string): Student's password.

username (string): Student's username.

age (int): Student's age.

Functions:

signup(): Allows the student to sign up by entering their name, age, username, and password.

authentication(): Allows the student to authenticate by entering their username and password.

3.AdmissionProcess Class (Inherits from Student):

Data Members:

choice (int): User's choice of course.

marksScored (int): Marks scored in the entrance exam.

courseSelected (string): Course selected by the student.

examDate (string): Exam date selected by the student.

availableDates (vector<string>): Vector containing available exam dates.

Functions:

courseSelection(): Allows the student to select a course from the available options.

selectExamSlot(): Allows the student to select an exam slot from the available dates.

simulateExam(): Simulates an exam and records the marks scored by the student.

paymentProcess(): Processes the payment and enrolls the student in the selected course or suggests alternative courses based on marks.

initiateAdmission(): Initiates the admission process by calling the above functions in sequence.

Overall:

main() Function:

Manages the flow of the program, allowing the user to sign up or log in and then initiates the admission process for the student.

11. BLOCK DIAGRAM

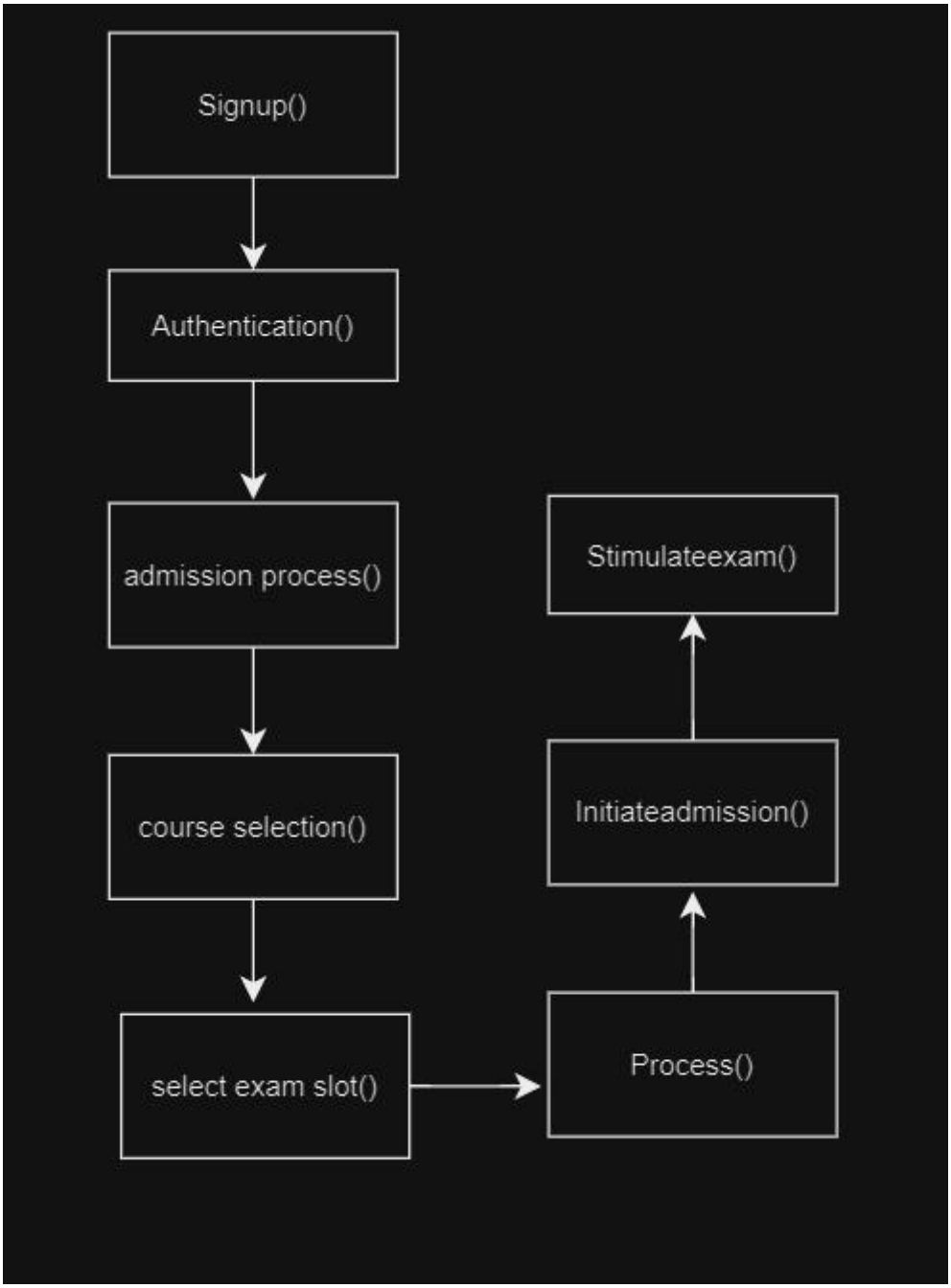


Figure 11.1 Block Diagram of Functionalities

The above diagram explains the relationship between block functionalities. It covers all the functions used within the project.

12. CLASS DIAGRAM

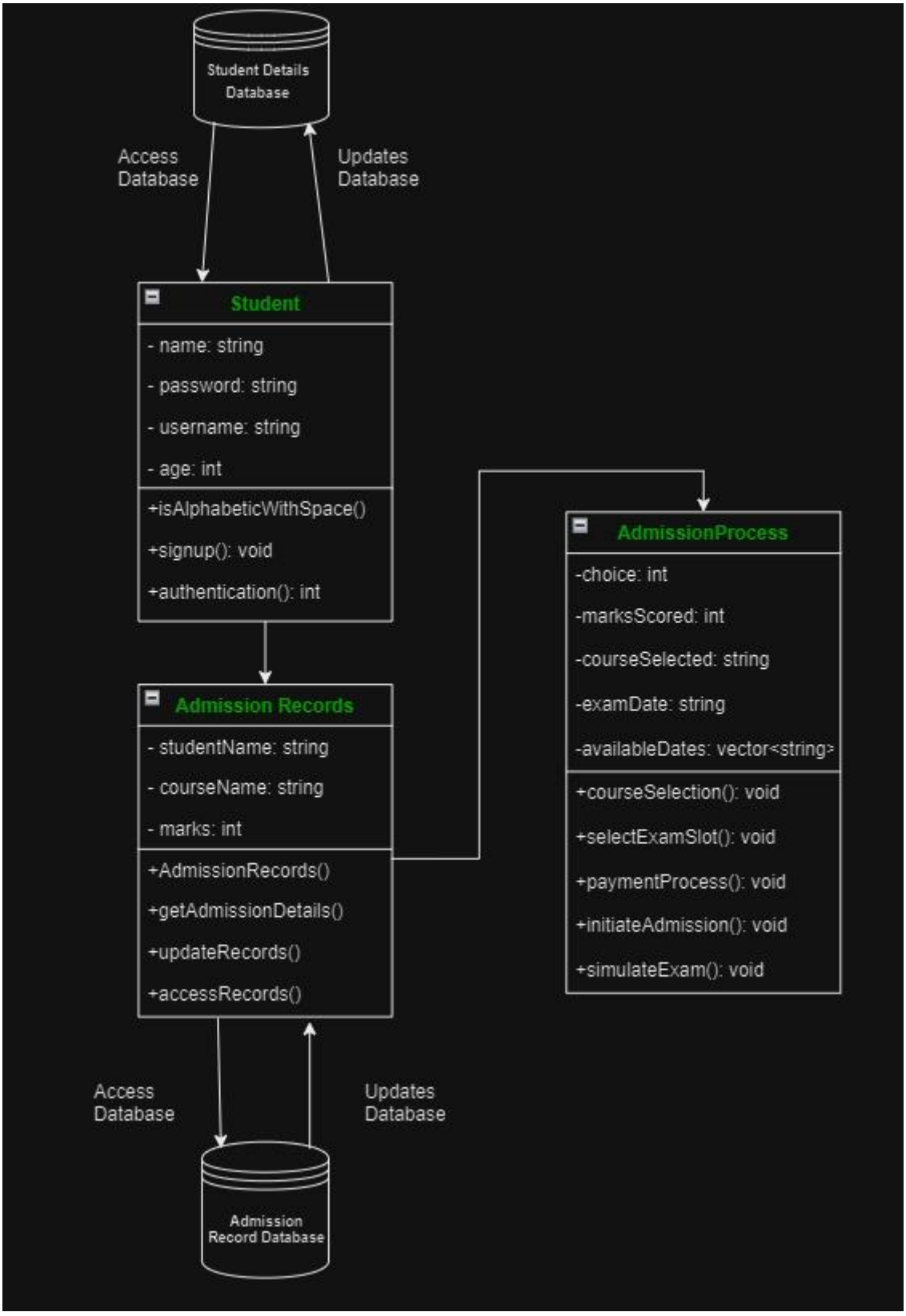


Figure 12 .1 Class Diagram

This diagram shows the list of class and its data members and member functions. This diagram shows the relationship between the classes used in the project. The class admission record is used to store the data entered by the user. The member function of the class student is used to take input by the students about their basic details. The first member function used is the constructor used to initialize the variables. The second function is used for the user to create an account and the third function is used to check whether the user is an existing customer or not. In the class Admission Process the variables are used to provide choices to the user. Moving on to the member functions, the first function is constructor used to initialize variables. The second function provides the option to the user to select choices of courses. The third function provides slots to the user to select options to schedule the entrance exam. The last three functions complete the admission process and store the records in the database.

13. USE-CASE DIAGRAM

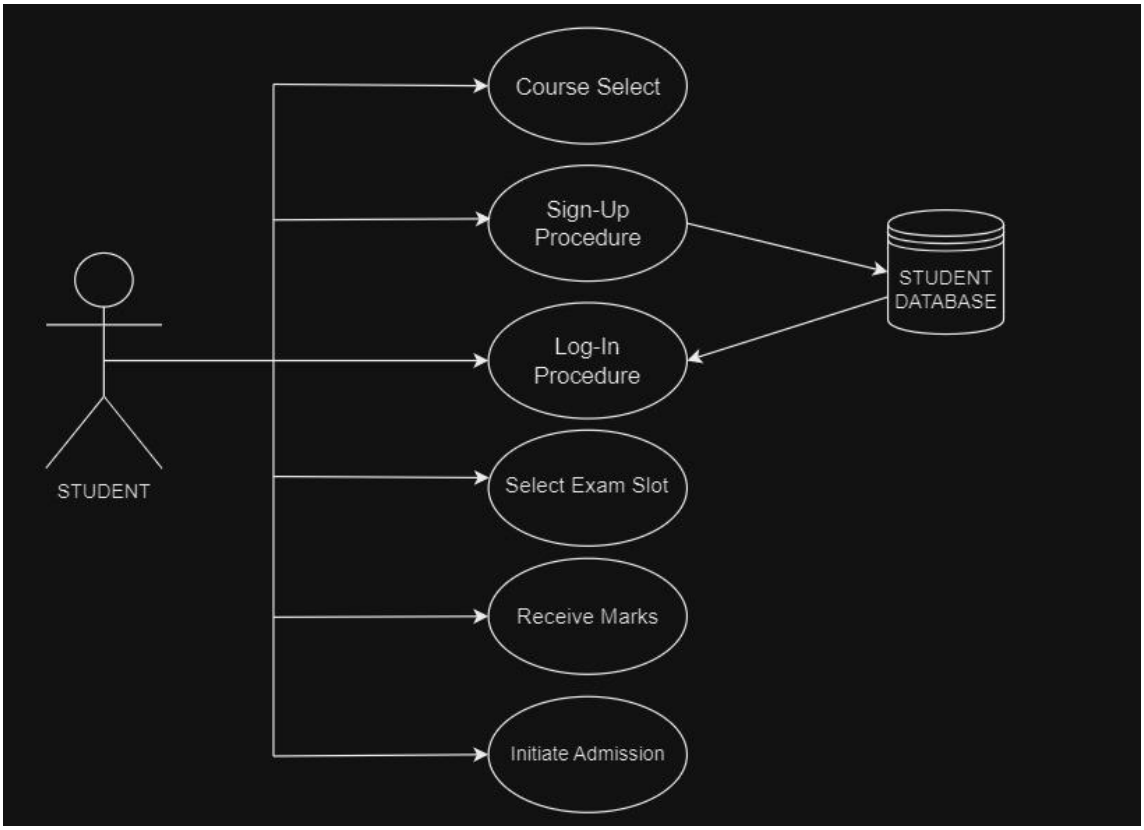


Figure 13.1 Use-Case Diagram

The above diagram explains the functions that can be accessed by the user. The first functionality helps the user to select courses available in the system. The second functionality allows users to create an account to access the system. The third functionality allows user to login in the system. The fourth one provides choices schedule entrance exam. The fifth one provides window to enter the marks obtain. The last one checks the eligibility if the user is eligible for the selected course or not.

14. ACTIVITY DIAGRAMS

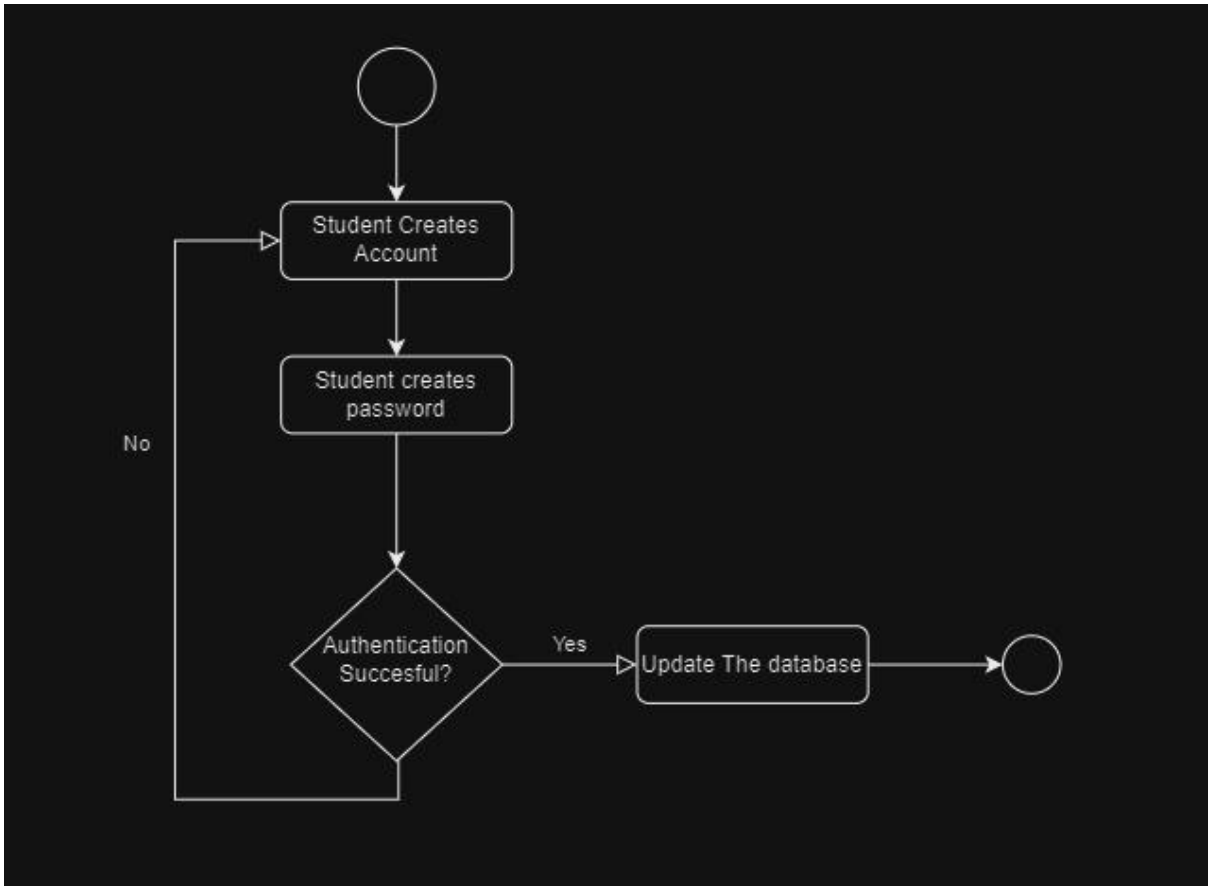


Fig 14.1 Activity Diagram

The above diagram shows that the user must create an account to access the system. It checks whether the student has an account or requires creating an account to access the system.

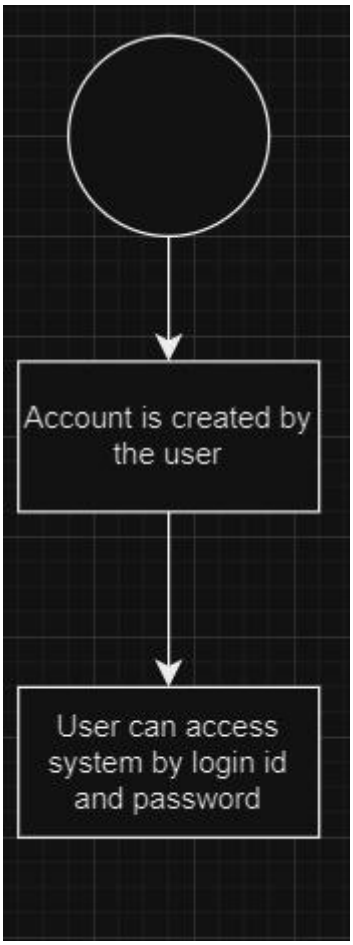


Fig 14.2 Activity Diagram

The above diagram represents the user can access the system by using his login id and Password which he created earlier.

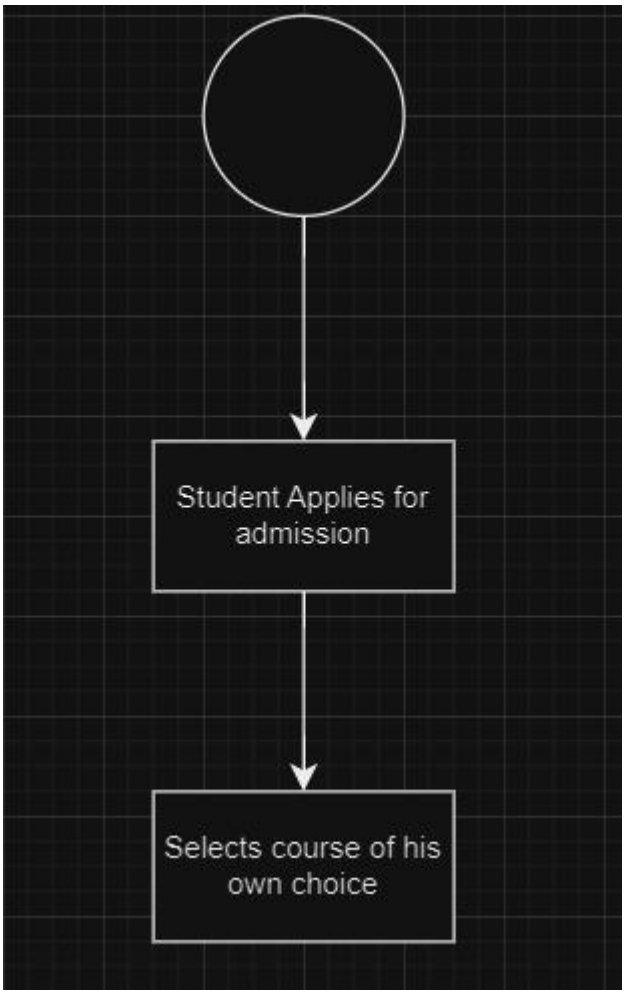


Fig 14.3 Activity Diagram

The above diagram shows how the system will work. The above diagram shows how the system allows user to select his type of course which is provided by the system.

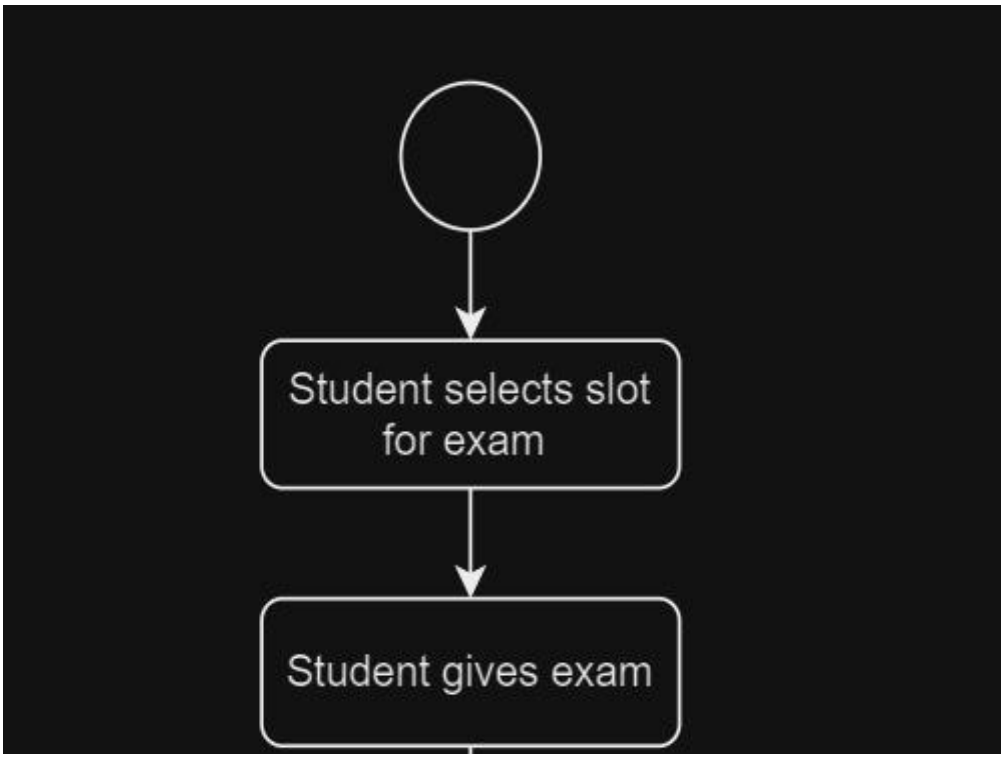


Fig 14.4 Activity Diagram

The above diagram shows how the user will select the slots available for his entrance exam. These slots are predefined in the system and provided to the user as a choice.

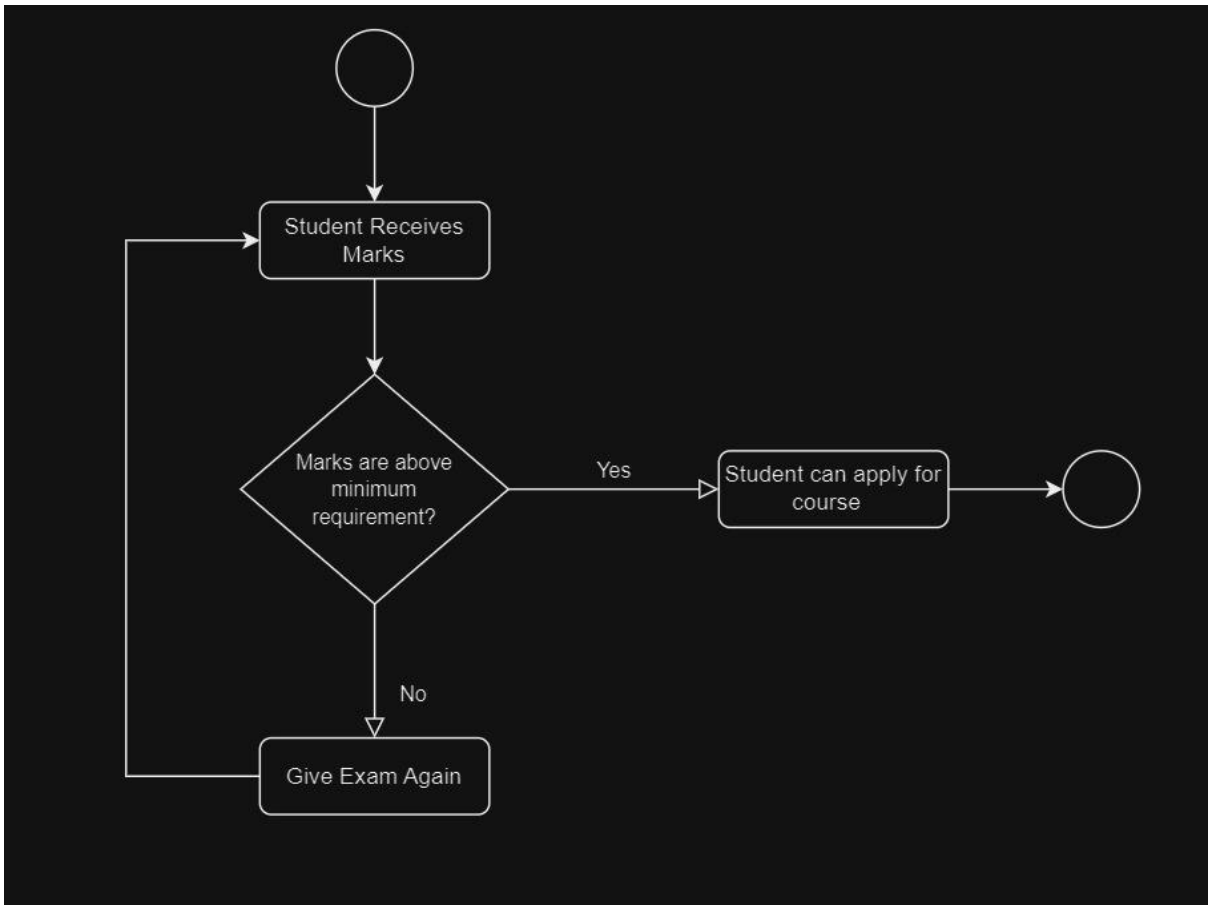


Fig 14.5 Activity Diagram

The above diagram shows that the student is provided with a window to enter his marks and then the system checks whether the student is applicable for his selected choice of course or not.

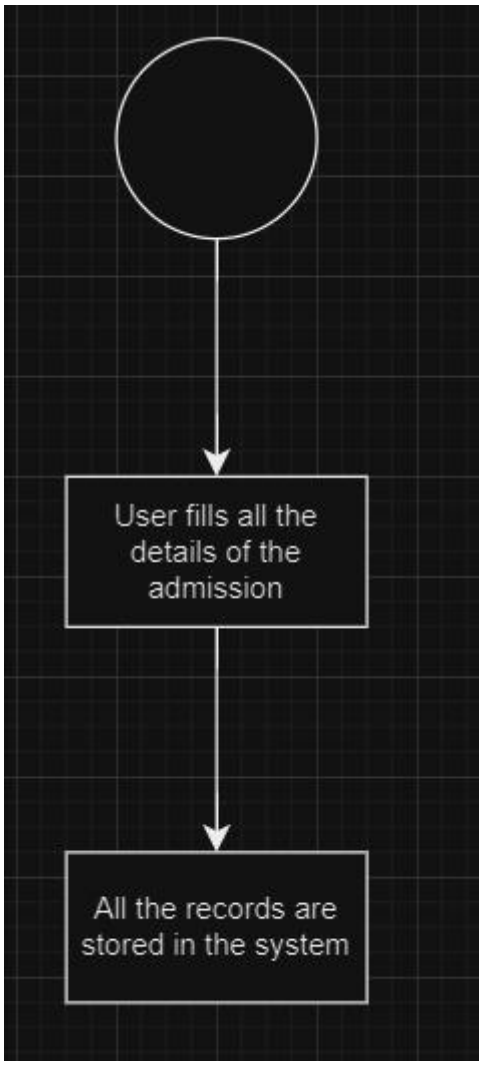


Fig 14.6 Activity Diagram

The above diagram explains how all the details to be stored in the database of college.

15. SEQUENCE DIAGRAM

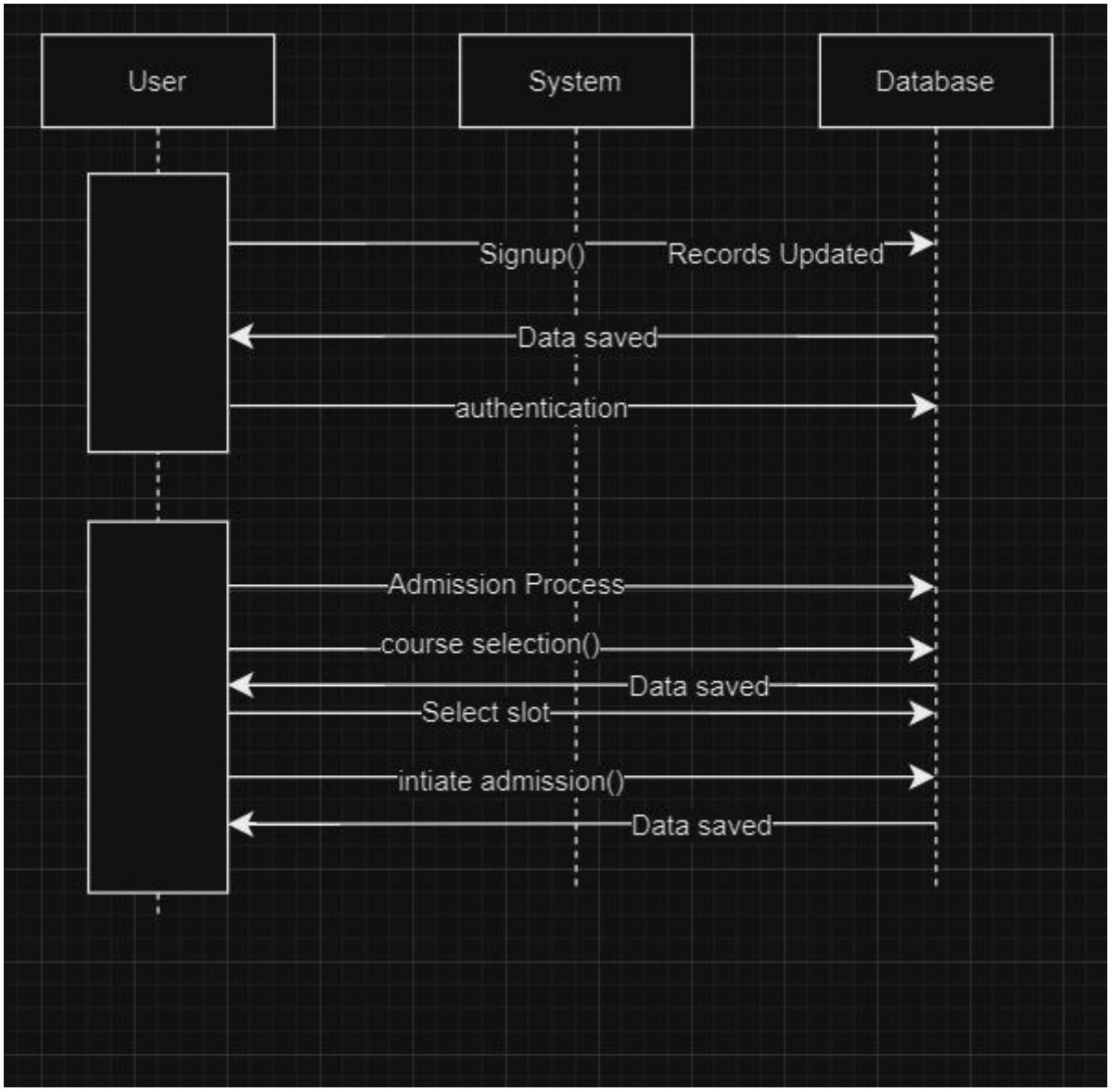


Fig 15.1 Sequence Diagram

This diagram represents how the system will work according to each functionality which is executed which is defined in the project

16. OBJECT ORIENTED PROGRAMMING IMPLEMENTATION

Classes and Objects:

Classes are blueprints for creating objects. They define the properties (attributes) and behaviors (methods) that objects of the class will have.

Objects are instances of classes. They represent real-world entities and can interact with each other.

Implementation:

Defined two classes, Student and AdmissionProcess, which are used to create objects representing students and their admission processes, respectively.

Encapsulation:

Encapsulation is the bundling of data (attributes) and the methods that operate on that data into a single unit (a class).

It hides the internal state of an object from the outside world and only exposes a controlled interface for interacting with the object.

Implementation:

Member variables like name, password, username, age, choice, marksScored, courseSelected, and examDate are declared as protected in the classes, encapsulating their state, and hiding them from direct access outside the class. Access to these variables is controlled through member functions.

Inheritance:

A mechanism where a new class (subclass or derived class) is created based on an existing class (superclass or base class).

The subclass inherits the attributes and methods of the superclass, allowing for code reuse and the creation of a hierarchical class structure.

Implementation:

The AdmissionProcess class inherits from the Student class, which means it can access the protected members of the Student class. Inheritance is used to create a specialized class (AdmissionProcess) based on a more general class (Student).

Abstraction:

The concept of hiding complex implementation details and showing only the essential features of an object.

It allows the programmer to focus on what an object does rather than how it does it, making the code more readable and maintainable.

Implementation:

Used abstraction to hide the implementation details of the classes from the outside world. For example, users interact with the Student and AdmissionProcess classes through public member functions without needing to know how those functions are implemented internally.

17. ALGORITHM

1. Start: Display a welcome message for the College Admission System.

2. Input: Ask the user if they want to sign up or log in.

3. Sign-Up:

Prompt the user to enter their name, age, username, and password.

Validate the inputs (name should be alphabetic, age should be between 18 and 19, username and password should not contain negative values).

Store the username and password in the StudentDetails.txt file.

Display a success message and proceed to the login step.

4. Log-In:

Prompt the user to enter their username and password.

Check the entered username and password against the entries in the StudentDetails.txt file.

If the details are correct, display a verification message and proceed to the next step. If not, ask the user to try again.

5. Course Selection:

Display a list of available courses.

Prompt the user to select a course by entering a number between 1 and 5.

Store the selected course in a variable.

Exam Slot Selection:

Display a list of available exam dates.

Prompt the user to select an exam date by entering a number between 1 and 5.

Store the selected exam date in a variable.

6.Exam Simulation:

Prompt the user to input their exam marks.

Validate the input (marks should be between 0 and 100).

Store the entered marks in a variable.

7. Payment Process:

Calculate the minimum passing marks based on the selected course.

Check if the user's marks are greater than or equal to the passing marks.

If yes, display a success message and enroll the student in the selected course. If not, display a message suggesting alternative courses based on the user's marks.

8. Repeat Exam:

If the user chooses to take the exam again, reset the marks and repeat the exam simulation and payment process.

End: Display a final message and exit the program.

18. FLOWCHART

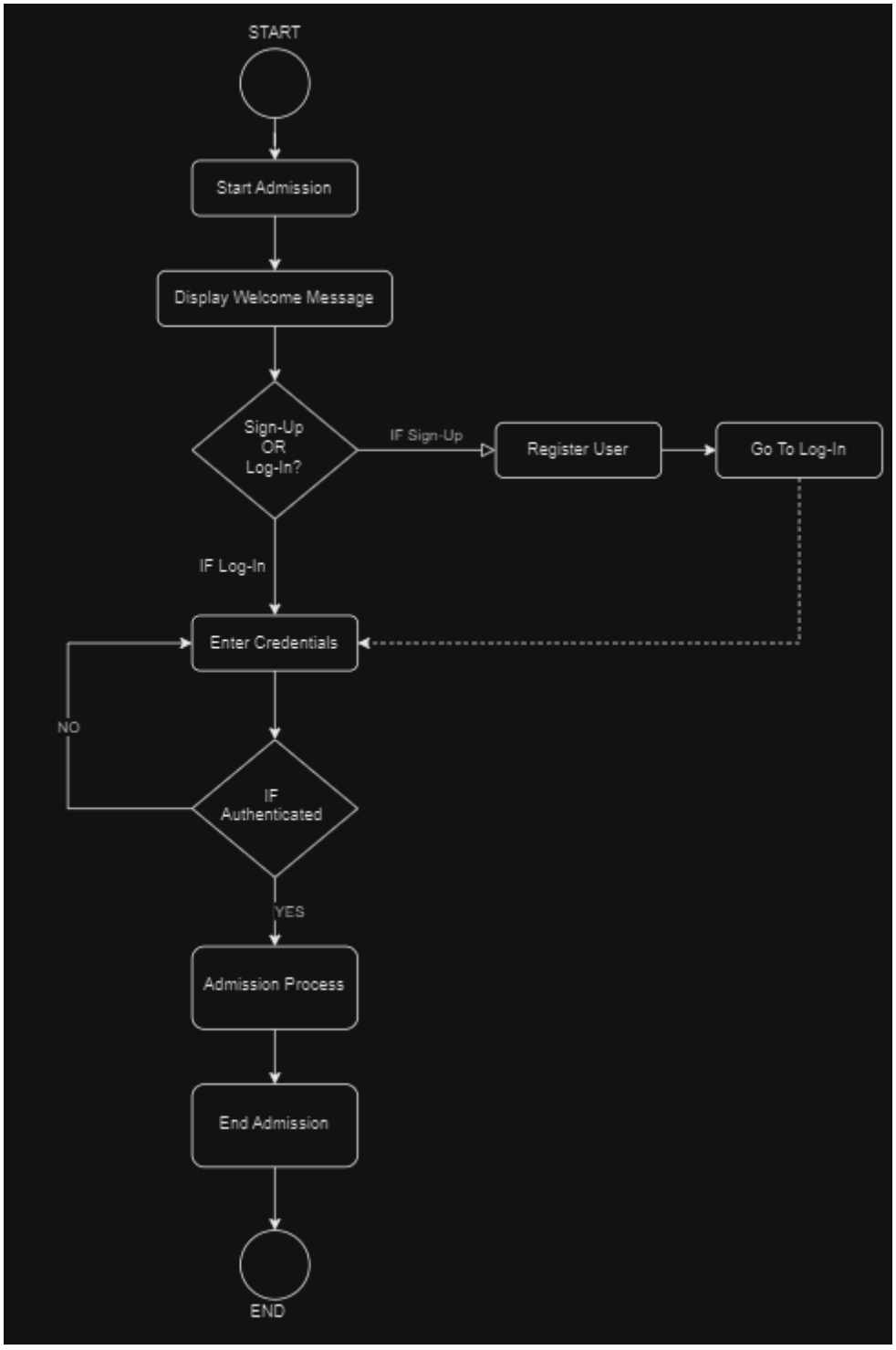


Fig 17.1 Flowchart

The above diagram is the flowchart which is a diagrammatic representation of the algorithm. It shows the process which are carried out in the project

19. CODE

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <limits>
#include <cctype>
#include <vector>
using namespace std;

// Function to check if a string contains only alphabetic characters
bool isAlphabetic(const string& str)
{
   for (char c : str)
   {
      if (!isalpha(c))
      {
         return false;
      }
   }
   return true;
}

// Class to manage admission records
class AdmissionRecords
{
   string studentName, courseName;
```

```
    int marks;


public:
    AdmissionRecords() : marks(0) {}


    // Function to set admission details
    void getAdmissionDetails(const string& studentname, const string& coursename, int
marksScored)
    {
        studentName = studentname;
        courseName = coursename;
        marks = marksScored;
    }


    // Function to update admission records in a file
    void updateRecords()
    {
        ofstream ofile("AdmissionRecords.txt", ios::app);
        if (ofile.is_open())
        {
            ofile << studentName << "\n" << courseName << "\n" << marks << endl;
            cout << "Records Updated\n";
            ofile.close();
        }
        else
        {
            cout << "Unable to open file for writing.\n";
```

```
    }
}

// Function to access and display admission records from a file
void accessRecords(const string& name)
{
    ifstream ifile("AdmissionRecords.txt");
    if (!ifile.is_open())
    {
        cout << "Unable to open file for reading.\n";
        return;
    }
    string line;
    while (getline(ifile, studentName))
    {
        getline(ifile, courseName);
        ifile >> marks;
        if (studentName == name)
        {
            cout << "Records of student: " << studentName << endl;
            cout << "\tCourse Name: " << courseName << endl;
            cout << "\tMarks Scored: " << marks << endl;
            cout << endl;
        }
        ifile.ignore(5, '\n');
    }
    ifile.close();
```

```
    }
};

// Class to manage student operations
class Student : public AdmissionRecords
{
protected:
    string name, password, username;
    int age;

public:
    Student() : age(0) {}

    // Function to check if a string contains only alphabetic characters and spaces
    bool isAlphabeticWithSpace(const string& str)
    {
        for (char c : str)
        {
            if (!isalpha(c) && c != ' ')
            {
                return false;
            }
        }
        return true;
    }

    // Function to register a new student
```

```
void signup()
{
    cout << "Enter your Name: ";
    cin.ignore();
    getline(cin, name);
    while (!isAlphabeticWithSpace(name))
    {
        cout << "Invalid input. Please enter a valid name (alphabetic characters only): ";
        getline(cin, name);
    }
    bool validAge = false;
    while (!validAge)
    {
        cout << "Enter Age: ";
        cin >> age;
        if (cin.fail() || age < 18 || age > 19)
        {
            cout << "Invalid input. Please enter a valid age (18 or 19): ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        else
        {
            validAge = true;
        }
    }
    while (true)
```

```
{
    cout << "Enter Username: ";
    cin >> username;
    if (username.find('-') != string::npos)
    {
        cout << "Invalid input. Username cannot contain negative values.\n";
    }
    else
    {
        break;
    }
}
while (true)
{
    cout << "Enter Password: ";
    cin >> password;
    if (password.find('-') != string::npos)
    {
        cout << "Invalid input. Password cannot contain negative values.\n";
    }
    else
    {
        break;
    }
}
ofstream ofile("StudentDetails.txt", ios::app);
if (ofile.is_open())
```

```
    {
        ofile << username << " " << password << endl;
        cout << "User Registered\n";
        cout << "LOGIN\n";
        ofile.close();
    }
    else
    {
        cout << "Unable to open file for writing.\n";
    }
}

// Function to authenticate a student
int authentication()
{
    string confirmUsername, confirmPassword;
    int flag = 0;
    while (true)
    {
        string username, password;
        cout << "Enter username: ";
        cin >> username;
        cout << "Enter password: ";
        cin >> password;
        ifstream ifile("StudentDetails.txt");
        if (ifile.is_open())
        {
```

```
            while (ifile >> confirmUsername >> confirmPassword)
            {
               if (confirmUsername == username && confirmPassword == password)
               {
                  flag = 1;
                  break;
               }
            }
            ifile.close();
         }
         else
         {
            cout << "Unable to open file for reading.\n";
            return 0;
         }
         if (flag == 1)
         {
            cout << "Details Verified\n";
            return 1;
         }
         else
         {
            cout << "Incorrect details. Please try again.\n";
         }
      }
   }
};
```

```
// Class to manage the admission process
class AdmissionProcess : public Student
{
    int choice, marksScored = 0;
    string courseSelected, examDate;
    vector<string> availableDates = {"April 5", "April 10", "April 15", "April 20", "April 25"};

public:
    AdmissionProcess() : choice(0) {}

    // Function to select the course
    void courseSelection() {
        bool validChoice = false;
        while (!validChoice) {
            cout << "Select a Course:\n";
            cout << "1. Computer Science\n";
            cout << "2. Computer Science (Data Science)\n";
            cout << "3. Mechanical\n";
            cout << "4. Civil Engineering\n";
            cout << "5. Electrical Engineering\n";
            cin >> choice;
            if (choice >= 1 && choice <= 5)
            {
                validChoice = true;
            }
```

```
    else
    {
        cout << "Invalid Choice. Please enter a number between 1 and 5.\n";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
}

switch (choice) {
    case 1:
        courseSelected = "Computer Science";
        break;
    case 2:
        courseSelected = "Computer Science (Data Science)";
        break;
    case 3:
        courseSelected = "Mechanical";
        break;
    case 4:
        courseSelected = "Civil Engineering";
        break;
    case 5:
        courseSelected = "Electrical Engineering";
        break;
    default:
        cout << "Invalid Choice\n";
        break;
```

```
        }
    }

// Function to select the exam slot
void selectExamSlot() {
    cout << "Select an Exam Date:\n";
    for (int i = 0; i < 5; ++i) {
        cout << i + 1 << ". " << availableDates[i] << endl;
    }

    int slot;
    while (true) {
        cout << "Enter the number corresponding to your chosen date: ";
        cin >> slot;
        if (cin.fail() || slot < 1 || slot > 5) {
            cout << "Invalid input. Please enter a number between 1 and 5.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        } else {
            break;
        }
    }
    cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore newline character
    examDate = availableDates[slot - 1];
    cout << "Your exam slot is on: " << examDate << endl;
    cout << "Please proceed with the payment of 1000 INR.\n";
}
```

```
// Function to simulate the exam
void simulateExam() {
    bool validInput = false;
    while (!validInput) {
        cout << "Exam Occurs\n";
        cout << "Input the marks of the exam to check eligibility: ";
        if (!(cin >> marksScored) || marksScored < 0 || marksScored > 100) {
            cout << "Invalid input. Please enter a numeric value between 0 and 100.\n";
            cin.clear();
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Clear input buffer
        } else {
            validInput = true;
        }
    }
    cout << "Your marks scored in the entrance exam: " << marksScored << endl;
}

// Function to process the payment and finalize admission
void paymentProcess() {
    int passMarks;
    vector<string> suggestedCourses;

    // Setting pass marks based on the selected course
    if (courseSelected == "Computer Science")
    {
        passMarks = 95;
```

```
}
else if (courseSelected == "Computer Science (Data Science)")
{
    passMarks = 90;
}
else if (courseSelected == "Mechanical")
{
    passMarks = 85;
}
else if (courseSelected == "Electrical Engineering")
{
    passMarks = 80;
}
else if (courseSelected == "Civil Engineering")
{
    passMarks = 75;
}
else
{
    cout << "Invalid course selection.\n";
    return;
}

// Checking eligibility based on marks scored
if (marksScored >= passMarks)
{
```

```
        cout << "Congratulations! You are enrolled for the " << courseSelected << " course." << endl;
    }
    else
    {
        cout << "Sorry, you did not score enough marks to be eligible for the " << courseSelected << " course." << endl;
        cout << "You can consider taking one of the following courses based on your marks:" << endl;

        // Reset suggestedCourses
        suggestedCourses.clear();

        // Suggesting alternative courses based on marks scored
        if (marksScored > 95)
        {
            suggestedCourses.push_back("Computer Science");
        }
        if (marksScored > 90)
        {
            suggestedCourses.push_back("Computer Science (Data Science)");
        }
        if (marksScored > 85)
        {
            suggestedCourses.push_back("Mechanical");
        }
        if (marksScored > 80)
        {
```

```
    suggestedCourses.push_back("Civil Engineering");
}
if (marksScored > 75)
{
    suggestedCourses.push_back("Electrical Engineering");
}
if (marksScored < 95)
{
    suggestedCourses.push_back("Give Exam Again");
}

// Displaying suggested courses
for (size_t i = 0; i < suggestedCourses.size(); ++i)
{
    cout << i + 1 << ". " << suggestedCourses[i] << endl;
}

int newChoice;
while (true) {
    cout << "Enter the number corresponding to the course you want to select: ";
    cin >> newChoice;
    if (cin.fail() || newChoice < 1 || newChoice > static_cast<int>(suggestedCourses.size()))
    {
        cout << "Invalid choice. Please enter a valid number.\n";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
```

```
            else
            {
                break;
            }
        }

        // Resetting marks and re-running exam simulation and payment process if the student chooses to give the exam again
        if (newChoice == suggestedCourses.size()) {
            marksScored = 0; // Reset marks
            cout << "Exam marks reset. Please proceed to take the exam again.\n";
            simulateExam();   // Re-run exam simulation
            paymentProcess(); // Re-run payment process
        } else {
            courseSelected = suggestedCourses[newChoice - 1];
            cout << "You have selected: " << courseSelected << endl;
            cout << "Please proceed with the payment of 1000 INR.\n";
            cout << "SUCCESSFUL ADMISSION\n";
        }
    }
}

// Function to initiate the admission process
void initiateAdmission()
{
    courseSelection();
    selectExamSlot();
```

```
    simulateExam();
    paymentProcess();
}
};

int main()
{
    AdmissionProcess A;

    int choice, flag = 0;
    cout << "Welcome to College Admission System\n";

    cout << "Enter 1 to Sign-Up or Enter 0 to Log-In\n";
    while (true)
    {
        cin >> choice;
        if (cin.fail())
        {
            cout << "Invalid input. Please enter 1 for yes or 0 for no.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        else if (choice == 1 || choice == 0)
        {
            break;
        }
        else
```

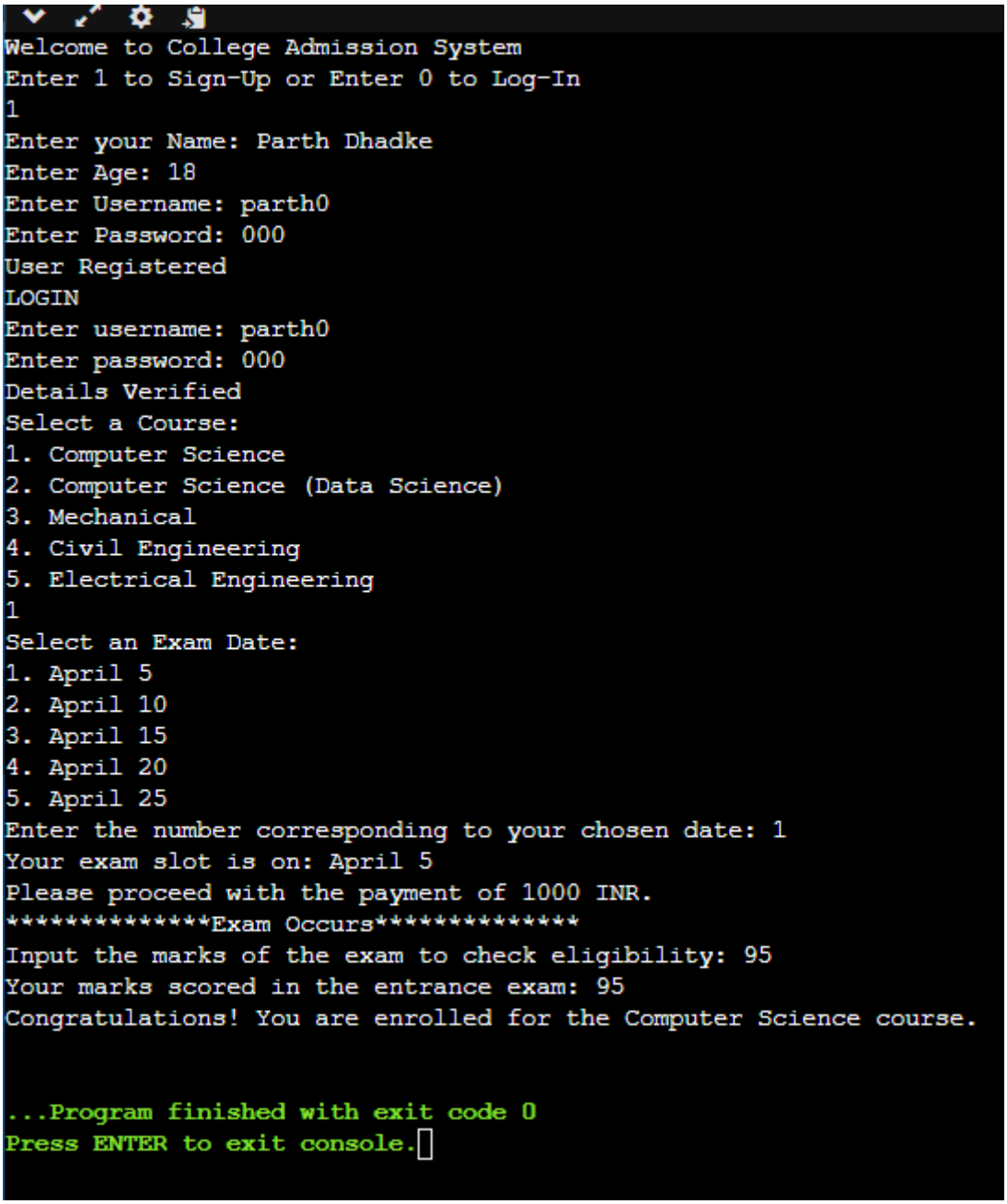
```
        {
            cout << "Invalid input. Please enter 1 for yes or 0 for no.\n";
        }
    }

    if (choice == 1)
    {
        A.signup();
    }

    flag = A.authentication();

    if (flag == 1)
    {
        A.initiateAdmission();
    }
    else
    {
        cout << "Exiting program.\n";
        exit(0);
    }
    return 0;
}
```

20. OUTPUT


Welcome to College Admission System
Enter 1 to Sign-Up or Enter 0 to Log-In
1
Enter your Name: Parth Dhadke
Enter Age: 18
Enter Username: parth0
Enter Password: 000
User Registered
LOGIN
Enter username: parth0
Enter password: 000
Details Verified
Select a Course:
1. Computer Science
2. Computer Science (Data Science)
3. Mechanical
4. Civil Engineering
5. Electrical Engineering
1
Select an Exam Date:
1. April 5
2. April 10
3. April 15
4. April 20
5. April 25
Enter the number corresponding to your chosen date: 1
Your exam slot is on: April 5
Please proceed with the payment of 1000 INR.
**********Exam Occurs**********
Input the marks of the exam to check eligibility: 95
Your marks scored in the entrance exam: 95
Congratulations! You are enrolled for the Computer Science course.
...Program finished with exit code 0
Press ENTER to exit console.

21. LIST OF ERRORS RESOLVED

The errors obtained were;

1. If the choice was not valid the program was still running:- Example:-

If there was a choice given to the user and if the user enter a choice which was not given in the option window the system would stop working. To resolve this we added an if statement to limit the choices between the given range of numbers

2. The system was crashed if there was a character entered in username:-

In the string data type when we entered the special character, then the system would go in infinite loop. To fix this we added multiple statements analysis each to enter choice.

3. There was no string file included:- We didn't include string files so the string variables weren't declared.

4. No semicolon, bracket ending:- While programming we forgot to put semicolon or finish the bracket so there we errors in the program.

5. Username, Password error:- If we entered the wrong user name and password then also the program was running further therefore we added function to check the username and password from the database.

22. SCOPE OF IMPROVEMENT

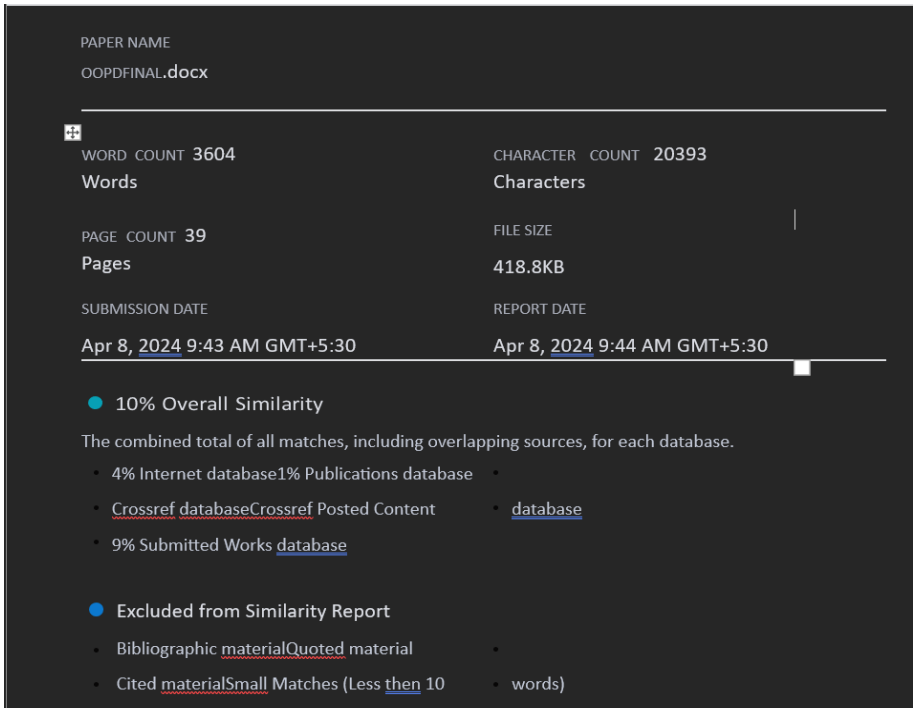
1. Error Handling: The code can be written in a way that the errors pertaining to the file opening and appending can be handled as well.
2. Modularity: The code can be written in a more modular way by breaking the three classes into smaller separate functions.
3. Memory Management: The code can be written with the help of pointers to manage memory dynamically.
4. User Interface: Having an attractive user interface will make the operation of the code by a user much easier.
5. Addition Of Payment Module: Due to high complexity, could not create a viable payment solution for my code.

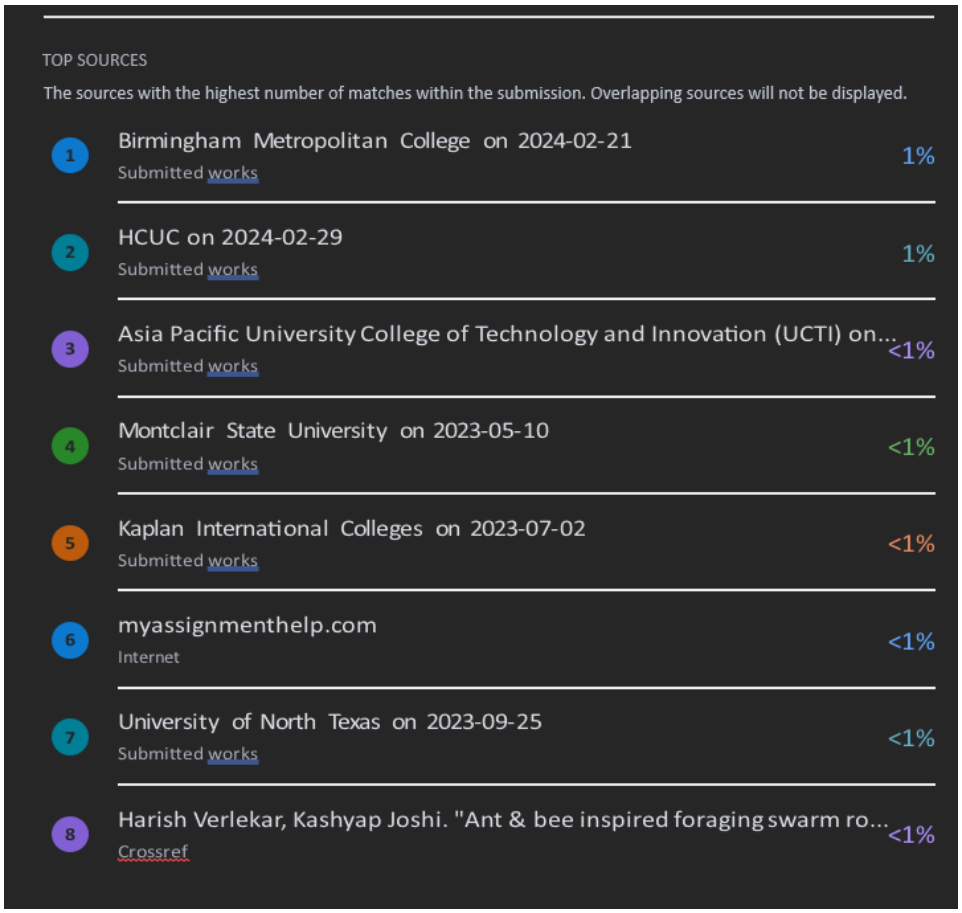
23. Bibliography

[1] E. Balguruswamy, Balguruswamy 4th Edition, McGraw Hill Companies, 2006.

[2] K. Malik, Monday November 2023. [Online]. Available: https://www.researchgate.net/publication/375962943_ONLINE_STUDENT_ADMISSION_REGISTRATION_MANAGEMENT_SYSTEM_A_CASE_STUDY_OF_METROPOLITAN_INTERNATIONAL_UNIVERSITY.

24. PLAGIARISM CHECK





TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Birmingham Metropolitan College on 2024-02-21 Submitted works	1%
2	HCUC on 2024-02-29 Submitted works	1%
3	Asia Pacific University College of Technology and Innovation (UCTI) on... Submitted works	<1%
4	Montclair State University on 2023-05-10 Submitted works	<1%
5	Kaplan International Colleges on 2023-07-02 Submitted works	<1%
6	myassignmenthelp.com Internet	<1%
7	University of North Texas on 2023-09-25 Submitted works	<1%
8	Harish Verlekar, Kashyap Joshi. "Ant & bee inspired foraging swarm ro... Crossref	<1%