1. INTRODUCTION

1.1 Basic Definition:

A **Student Management System (SMS)** in Java is a project designed to manage student-related data for educational institutions. It typically includes features for adding, viewing, updating, and deleting student records. The system can be expanded to include functionalities like course management, academic performance tracking, and report generation. It often involves using Java for backend logic and a database (like MySQL) for data storage.

1.2 Motivation:

The motivation behind developing a Student Management System (SMS) in Java includes automating administrative tasks, ensuring efficient data management, providing a user-friendly interface for staff, offering customization for institutional needs, enhancing developers' technical skills, and implementing robust security measures to protect sensitive student data. This project aims to streamline operations, improve accuracy, and facilitate organized handling of student information in educational institutions.

1.3 Problem Statement:

Educational institutions often face challenges in managing student data efficiently and accurately. Manual handling of student records, attendance, grades, and other administrative tasks can lead to errors, inefficiencies, and time-consuming processes. Additionally, the lack of a centralized system makes it difficult to access and update information quickly, affecting the overall effectiveness of administrative functions.

1.4 Objective:

To design and develop a Student Management System (SMS) in Java that provides a centralized platform for efficiently managing student-related data. The system should automate tasks such as record keeping, attendance tracking, grade management, and report generation, ensuring data accuracy and easy access. It should also offer a user-friendly interface and robust security measures to protect sensitive information, thus enhancing the operational efficiency of educational institutions.

.

2. PROPOSED METHODOLOGY

2.1 System specifications:

The Student Management System (SMS) in Java is designed to efficiently manage student data, automating administrative tasks such as record-keeping, attendance tracking, and grade management. The system ensures easy data access, provides a user-friendly interface, allows for customization, enhances security measures to protect data, and serves as a practical learning experience for developers. It includes various hardware and software requirements, such as an Intel i3 processor, Java Development Kit, MySQL, and JavaFX, among others. The system supports key functionalities like user management, course management, attendance tracking, grade input, and report generation.

2.2 Proposed work:

The proposed work for the Student Management System (SMS) in Java involves designing a system architecture that is scalable, maintainable, and secure. This includes defining the database schema, user interface design, and overall system flow. The core modules to be developed include user management for adding and managing student records, course management for handling course details and student enrollments, attendance tracking, grade management, and report generation for student performance and attendance.

Additionally, the project involves integrating a robust database, such as MySQL, to securely store and manage student data. The user interface will be developed using JavaFX or JSP/Servlets, ensuring it is intuitive and user-friendly. Security measures such as SSL/TLS for secure communication and role-based access control will be implemented to protect sensitive information. Thorough testing will be conducted to ensure the system's functionality and efficiency, followed by deployment on suitable platforms with necessary documentation and user support.

This comprehensive approach aims to create an efficient Student Management System that enhances the administrative capabilities of educational institutions and ensures accurate and streamlined management of student data.

3. IMPLEMENTATION STEPS AND OUTPUT

1. Defining the Student Class

This *Student* class represents individual students and their attributes. Each student has an *id*, a *name*, and a *grade*.
The class also contains:

- Constructor to initialize a student's details.
- Getter and Setter Methods to access and modify student attributes.
- toString() Method to represent the student's information in a string format.

Here is the code for the *Student* class:

```
public class Student
{
   private int id;
   private String name;
   private String grade;

   public Student(int id, String name, String grade)
  {
      this.id = id;
      this.name = name;
      this.grade = grade;
   }

   // Getters and setters
   public int getId()
  {
      return id;
   }

   public void setId(int id)
   {
      this.id = id;
   }

   public String getName()
  {
      return name;
   }

   public void setName(String name)
  {
      this.name = name;
   }

   public String getGrade()
```

```java
    {
        return grade;
    }

    public void setGrade(String grade)
    {
        this.grade = grade;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Grade: " + grade;
    }
}
```

2. Implementing the Student Management System

With our Student class in place, we can now implement our CRUD and search operations:

```java
import java.util.ArrayList;
import java.util.List;

public class StudentManagementSystem
{

    private List<Student> students = new ArrayList<>();

    // Create (Add) a student
    public void addStudent(Student student)
    {
        students.add(student);
    }

    // Read (View) all students
    public void viewStudents()
    {
        students.forEach(System.out::println);
    }

    // Update a student's details
    public void updateStudent(int id, String newName, String newGrade)
    {
        for (Student s : students)
        {
            if (s.getId() == id)
            {
                s.setName(newName);
                s.setGrade(newGrade);
                break;
            }
        }
    }
    // Delete a student by ID
    public void deleteStudent(int id)
   {
        students.removeIf(s -> s.getId() == id);
    }

    // Search student by name
    public Student searchStudentByName(String name)
    {
        for (Student s : students)
        {
            if (s.getName().equalsIgnoreCase(name))
            {
                return s;
            }
        }
        return null;
    }
}
```

This class manages a list of students and provides various operations:

- *addStudent(Student student)*: Adds a new student to the list.
- *viewStudents()*: Displays all the students.
- *updateStudent(int id, String newName, String newGrade)*: Finds a student by ID and updates their name and grade.
- *deleteStudent(int id)*: Removes a student based on their ID.
- *searchStudentByName(String name)*: Searches for a student by their name and returns the first match.

3. Main Execution

Let's see our system in action:

```
import java.util.Scanner;

public class Main
{
    public static void main(String[] args)
   {
        StudentManagementSystem sms = new StudentManagementSystem();
        Scanner scanner = new Scanner(System.in);
        boolean exit = false;

        while (!exit)
        {
            System.out.println("Menu:");
            System.out.println("1. Add Student");
            System.out.println("2. View Students");
            System.out.println("3. Update Student");
            System.out.println("4. Search Student by Name");
            System.out.println("5. Delete Student");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            switch (choice)
            {
                case 1:
                    System.out.print("Enter Student ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine(); // consume newline

                    System.out.print("Enter Name: ");
                    String name = scanner.nextLine();

                    System.out.print("Enter Grade: ");
                    String grade = scanner.nextLine();

                    sms.addStudent(new Student(id, name, grade));
                    System.out.println("Added student: ID: " + id + ", Name: " + name + ", Grade: " + grade);
                    break;
                case 2:
                    System.out.println("Students List:");
                    sms.viewStudents();
                    break;
                case 3:
                    System.out.print("Enter Student ID to update: ");
                    int updateId = scanner.nextInt();
                    scanner.nextLine(); // consume newline

                    System.out.print("Enter New Name: ");
                    String newName = scanner.nextLine();

                    System.out.print("Enter New Grade: ");
                    String newGrade = scanner.nextLine();

                    sms.updateStudent(updateId, newName, newGrade);
                    System.out.println("Updated student: ID: " + updateId + ", Name: " + newName + ",
Grade: " + newGrade);
                    break;
                case 4:
                    System.out.print("Enter Name to search: ");
                    String searchName = scanner.nextLine();
                    Student foundStudent = sms.searchStudentByName(searchName);
                    System.out.println(foundStudent != null ? foundStudent : "Student not found.");
                    break;
                case 5:
                    System.out.print("Enter Student ID to delete: ");
                    int deleteId = scanner.nextInt();
                    sms.deleteStudent(deleteId);
```

```
                System.out.println("Deleted student with ID: " + deleteId);
                break;
            case 6:
                exit = true;
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
        System.out.println();
    }

    scanner.close();
  }
}
```

All three classes—Main, Student, and StudentManagementSystem—are interrelated and work together to form a functional Student Management System. Here's how they are connected:

1. **Student Class**:
   - Represents individual student data with fields like id, name, and grade.
   - Provides getters, setters, and a toString method for easy data manipulation and display.
2. **StudentManagementSystem Class**:
   - Manages a collection of Student objects.
   - Contains methods to add, view, update, delete, and search for students.
   - Interacts with Student objects to perform these operations.
3. **Main Class**:
   - Acts as the user interface through the console.
   - Creates an instance of StudentManagementSystem to manage students.
   - Takes user input to call methods in StudentManagementSystem for various operations.
   - Utilizes the Student class to create and manipulate student records.

Example Flow

- **Adding a Student**:
  - User chooses to add a student in the Main class.
  - Main class collects input (ID, name, grade) from the user.
  - Main class creates a Student object using the collected data.
  - Main class calls addStudent method of StudentManagementSystem to add the new student.
- **Viewing Students**:
  - User chooses to view all students in the Main class.
  - Main class calls viewStudents method of StudentManagementSystem.
  - StudentManagementSystem retrieves and prints all Student objects.
- **Updating a Student**:
  - User chooses to update a student in the Main class.
  - Main class collects new data (ID, new name, new grade) from the user.
  - Main class calls updateStudent method of StudentManagementSystem to update the specific student.
- **Searching a Student**:
  - User chooses to search for a student by name in the Main class.
  - Main class collects the name to search.
  - Main class calls searchStudentByName method of StudentManagementSystem.
  - StudentManagementSystem searches and returns the matching Student object.

- **Deleting a Student**:
    - User chooses to delete a student by ID in the Main class.
    - Main class collects the ID of the student to delete.
    - Main class calls deleteStudent method of StudentManagementSystem.

This interconnected design ensures that each class has a specific role and collaborates effectively to provide a comprehensive student management solution.

Explanation of the Student Management System in Java

1. Student Class
The Student class represents individual students in the system.
- **Attributes**: The class has three private fields: id, name, and grade.
- **Constructor**: Initializes these fields when a new Student object is created.
- **Getters and Setters**: Methods to access and modify the fields.
- **toString() Method**: Provides a string representation of a Student object, useful for displaying student details.

2. StudentManagementSystem Class
The StudentManagementSystem class manages a list of Student objects and provides methods to manipulate the student data.
- **Attributes**: The class has a private list of Student objects.
- **addStudent() Method**: Adds a new student to the list.
- **viewStudents() Method**: Prints all students in the list using their toString() method.
- **updateStudent() Method**: Updates the name and grade of a student based on their ID.
- **deleteStudent() Method**: Removes a student from the list based on their ID.
- **searchStudentByName() Method**: Searches for a student by their name and returns the matching student object, or null if not found.

3. Main Class
The Main class acts as the user interface, using a console-based menu to interact with the user and perform operations on the StudentManagementSystem.
- **Attributes**: Creates an instance of StudentManagementSystem and a Scanner for user input.
- **Menu Loop**: Displays a menu to the user repeatedly until the user chooses to exit.
- **Switch Statement**: Executes the corresponding operations based on the user's menu choice:
    - **Case 1**: Adds a new student by collecting ID, name, and grade from the user.
    - **Case 2**: Displays all students.
    - **Case 3**: Updates a student's details by collecting new data from the user.
    - **Case 4**: Searches for a student by name.
    - **Case 5**: Deletes a student by ID.
    - **Case 6**: Exits the program.

Program Flow:
1. The user selects an operation from the menu.
2. The Main class collects necessary data and calls the relevant method in StudentManagementSystem.
3. StudentManagementSystem performs the operation on the list of Student objects.
4. The results are displayed to the user

OUTPUTS:

Adding 10 Students

```
Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 1
Enter Name: Alice
Enter Grade: A
Added student: ID: 1, Name: Alice, Grade: A

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 2
Enter Name: Bob
Enter Grade: B
Added student: ID: 2, Name: Bob, Grade: B

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 3
Enter Name: Charlie
Enter Grade: C
Added student: ID: 3, Name: Charlie, Grade: C

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
```

```
Enter Student ID: 4
Enter Name: David
Enter Grade: D
Added student: ID: 4, Name: David, Grade: D

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 5
Enter Name: Eve
Enter Grade: E
Added student: ID: 5, Name: Eve, Grade: E

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 6
Enter Name: Frank
Enter Grade: F
Added student: ID: 6, Name: Frank, Grade: F

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 7
Enter Name: Grace
Enter Grade: G
Added student: ID: 7, Name: Grace, Grade: G

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
```

```
Choose an option: 1
Enter Student ID: 8
Enter Name: Harry
Enter Grade: H
Added student: ID: 8, Name: Harry, Grade: H

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 9
Enter Name: Ivy
Enter Grade: I
Added student: ID: 9, Name: Ivy, Grade: I

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 1
Enter Student ID: 10
Enter Name: Jack
Enter Grade: J
Added student: ID: 10, Name: Jack, Grade: J
```

Displaying All Students

```
Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 2
Students List:
ID: 1, Name: Alice, Grade: A
ID: 2, Name: Bob, Grade: B
ID: 3, Name: Charlie, Grade: C
ID: 4, Name: David, Grade: D
ID: 5, Name: Eve, Grade: E
ID: 6, Name: Frank, Grade: F
ID: 7, Name: Grace, Grade: G
ID: 8, Name: Harry, Grade: H
ID: 9, Name: Ivy, Grade: I
```

```
ID: 10, Name: Jack, Grade: J
```

Deleting a Student

```
Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 5
Enter Student ID to delete: 5
Deleted student with ID: 5
```

Displaying All Students

```
Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 2
Students List:
ID: 1, Name: Alice, Grade: A
ID: 2, Name: Bob, Grade: B
ID: 3, Name: Charlie, Grade: C
ID: 4, Name: David, Grade: D
ID: 6, Name: Frank, Grade: F
ID: 7, Name: Grace, Grade: G
ID: 8, Name: Harry, Grade: H
ID: 9, Name: Ivy, Grade: I
ID: 10, Name: Jack, Grade: J
```

Displaying a Single Student

```
Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 4
Enter Name to search: Alice
ID: 1, Name: Alice, Grade: A

Menu:
1. Add Student
2. View Students
3. Update Student
```

```
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 4
Enter Name to search: Eve
Student not found.
```

Updating a Student and Displaying All Students

```
Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 3
Enter Student ID to update: 2
Enter New Name: Robert
Enter New Grade: B+
Updated student: ID: 2, Name: Robert, Grade: B+

Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 2
Students List:
ID: 1, Name: Alice, Grade: A
ID: 2, Name: Robert, Grade: B+
ID: 3, Name: Charlie, Grade: C
ID: 4, Name: David, Grade: D
ID: 6, Name: Frank, Grade: F
ID: 7, Name: Grace, Grade: G
ID: 8, Name: Harry, Grade: H
ID: 9, Name: Ivy, Grade: I
ID: 10, Name: Jack, Grade: J
```

Exiting

```
Menu:
1. Add Student
2. View Students
3. Update Student
4. Search Student by Name
5. Delete Student
6. Exit
Choose an option: 6
```

4. CONCLUSION

The Student Management System (SMS) project in Java provides an efficient, centralized solution for managing student data. The system automates administrative tasks such as adding, viewing, updating, and deleting student records, reducing manual effort and minimizing errors. By utilizing Java, JDBC for database connectivity, and JavaFX for the user interface, the project ensures that student data is easily accessible, accurately maintained, and secure.

Furthermore, the system offers a user-friendly interface that makes it intuitive for users to interact with the application. This enhances operational efficiency in educational institutions by streamlining the process of managing student information. The scalable nature of the system allows for future enhancements and additional features, ensuring that it can grow and adapt to changing needs.

Overall, the Student Management System not only improves data management but also serves as a valuable educational tool for developers, providing practical experience with key Java concepts, database management, and software development practices. This project successfully addresses the common challenges faced by educational institutions in handling student data, making it a practical and robust solution.