OOADJ Project Report
Capstone Project Management System

Team Members -

Ananya Rao	PES1UG20CS046
Ananya Shashishekar	PES1UG20CS047
Aditi Thamankar	PES1UG20CS016
Abhay K Iyengar	PES1UG20CS004

Synopsis -

The Capstone management project is a web-based application developed using Springboot that allows students, teachers, and admins to manage the details of capstone projects. The application has separate login portals for each user type, and each user has specific roles and functionalities.

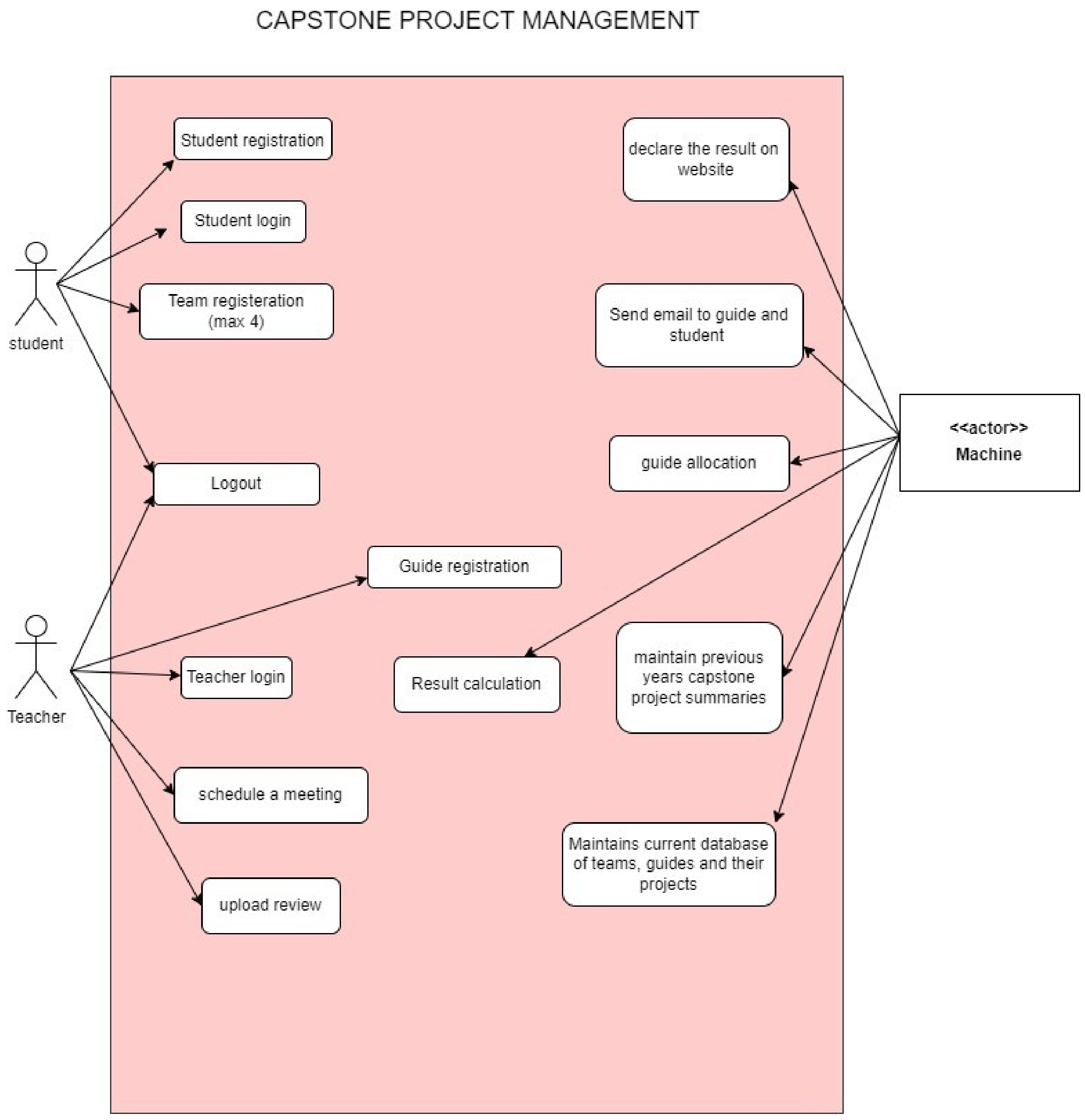
Students can register to a team of four members and choose a domain and topic for their capstone project. Teachers can register as guides and get assigned to a team. They can also schedule weekly reviews for each team they are guiding. Admins can view consolidated details of the students, teams, and meetings scheduled, and make changes if necessary.

This project aims to simplify the process of managing capstone projects by providing a centralized platform for students, teachers, and admins to communicate and collaborate efficiently. It helps in improving the overall management of the project and ensures that all the stakeholders are on the same page.

The Capstone management project also includes various features that enhance the user experience,such as an interactive dashboard that provides a visual representation of the project status. The application is user-friendly, secure, and scalable, making it suitable for use in both small and large-scale projects. By using this project, educational institutions can streamline their capstone project management process, reduce errors and delays, and ensure timely completion of projects.

Use Case Diagram -

The diagram shows three users Student, Teacher and Machine interacting with the application. The Student can register, submit weekly report and login. The teacher can schedule meetings with the team and can upload reviews. The machine assigns the teacher to the team, maintains database.


CAPSTONE PROJECT MANAGEMENT
Student registration
Student login
Team registeration (max 4)
student
Logout
Guide registration
Teacher
Teacher login
schedule a meeting
upload review
Result calculation
declare the result on website
Send email to guide and student
guide allocation
<<actor>> Machine
maintain previous years capstone project summaries
Maintains current database of teams, guides and their projects

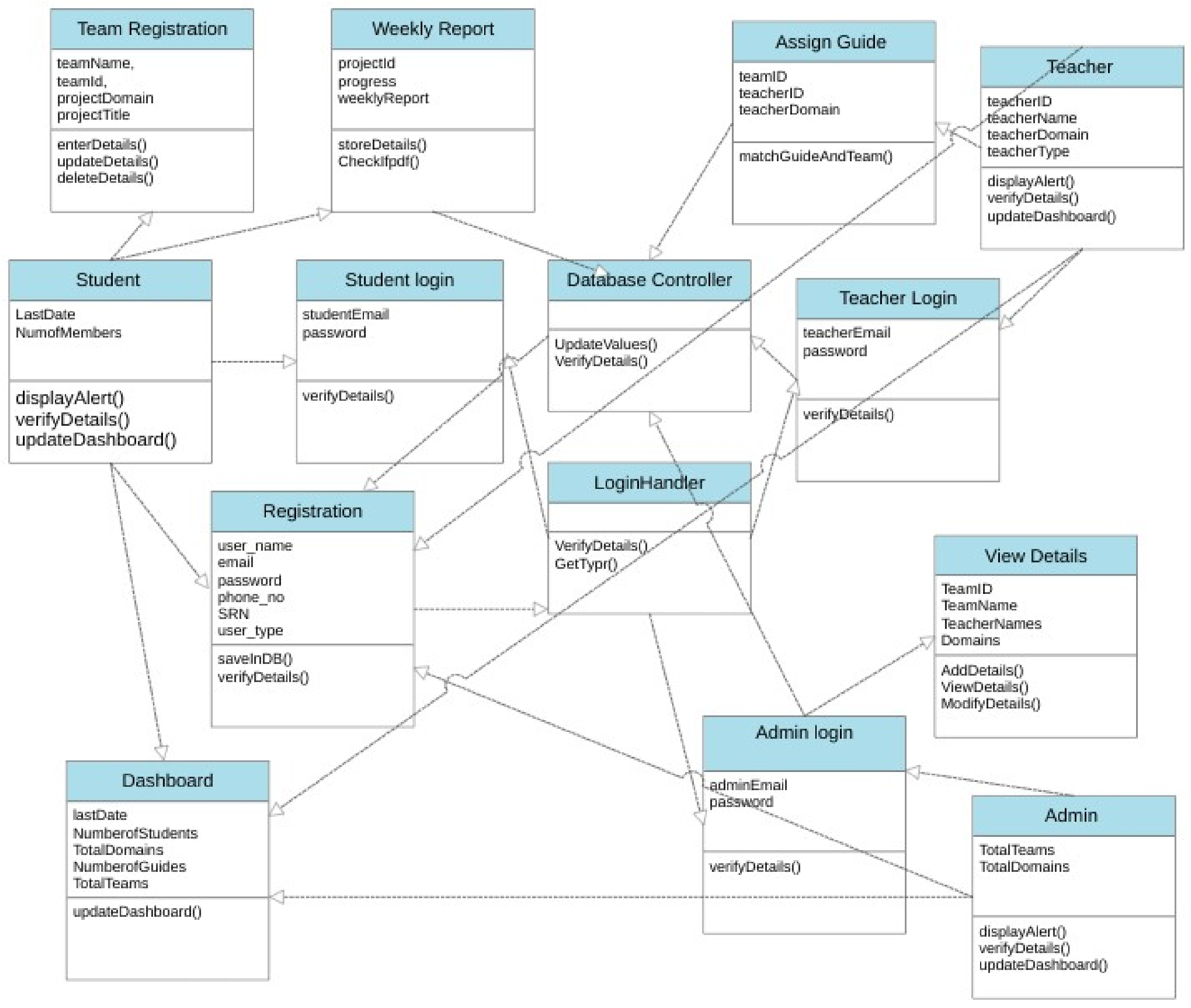
Class Diagram

- The registration class handles registeration for all types of users. The loginHandler class manages the classes StudentLogin, AdminLogin and TeacherLogin which manage the login of each type of user respectively.

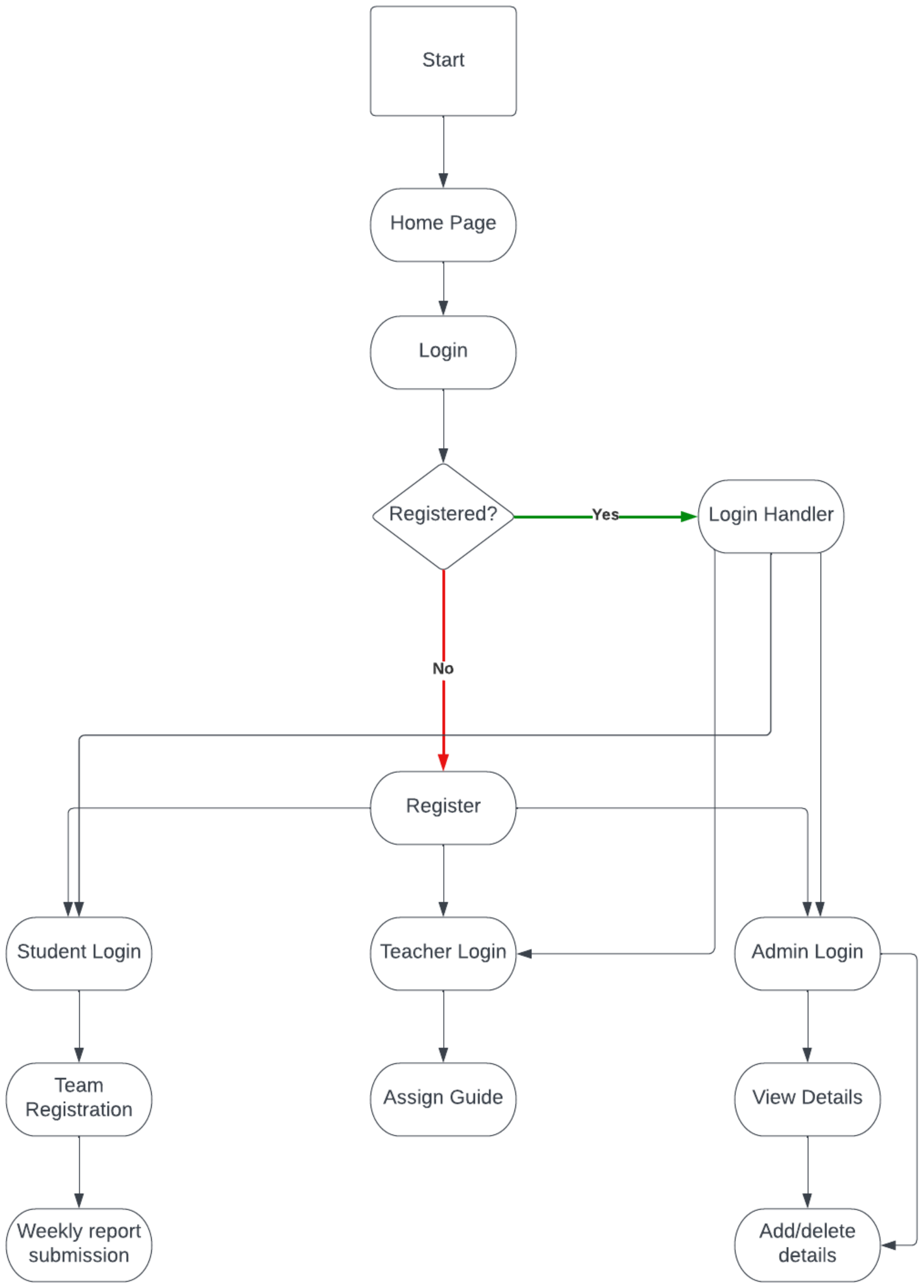
- The Student class manages the classes StudentLogin, TeamRegistration, and WeeklyReport. The Teacher classes manages the classes TeacherLogin, AssignGuide. The Admin class manages the classes AdminLogin and ViewDetails.

- The Database Controller class is linked to all classes and manages writing to and database and querying data.

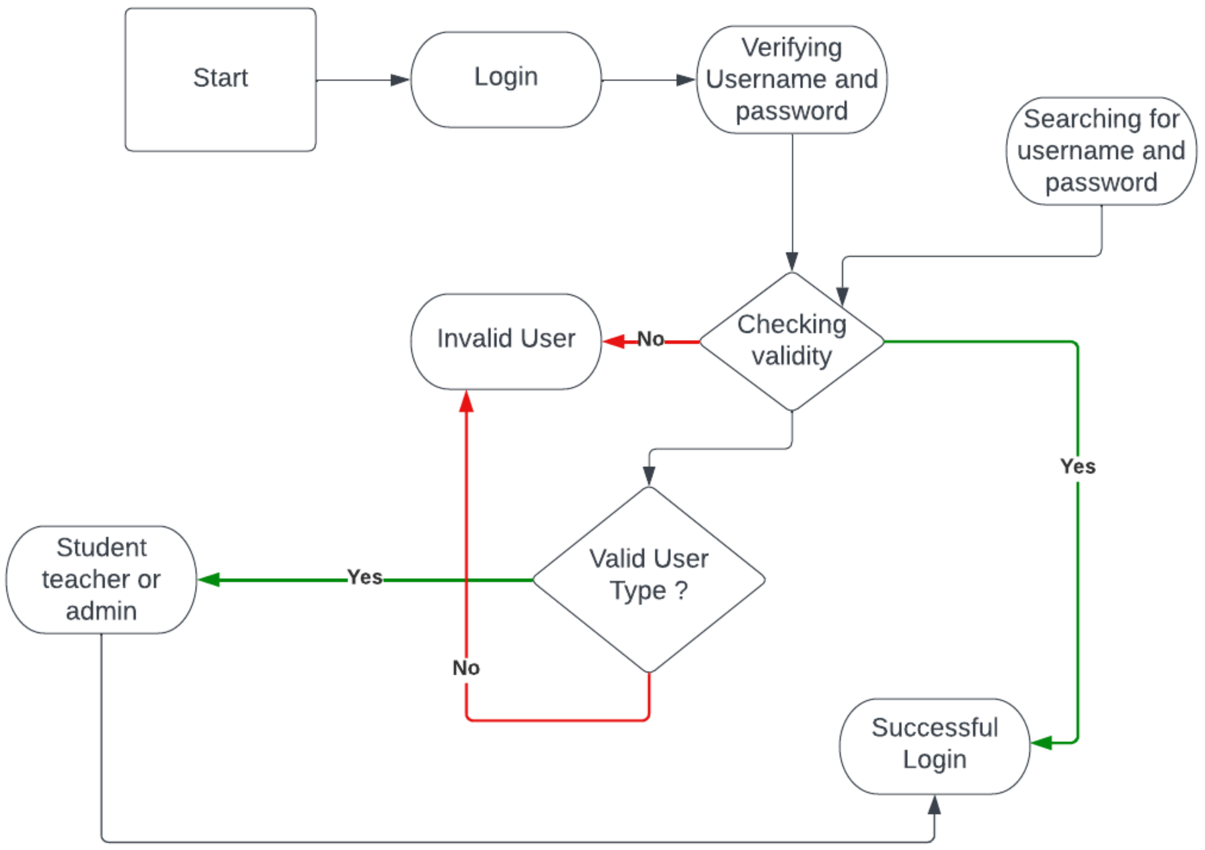
- The Dashboard is responsible for displaying quick details on the homepage for each type of User



Activity Diagram



State Diagram



Design Pattern – Factory Pattern

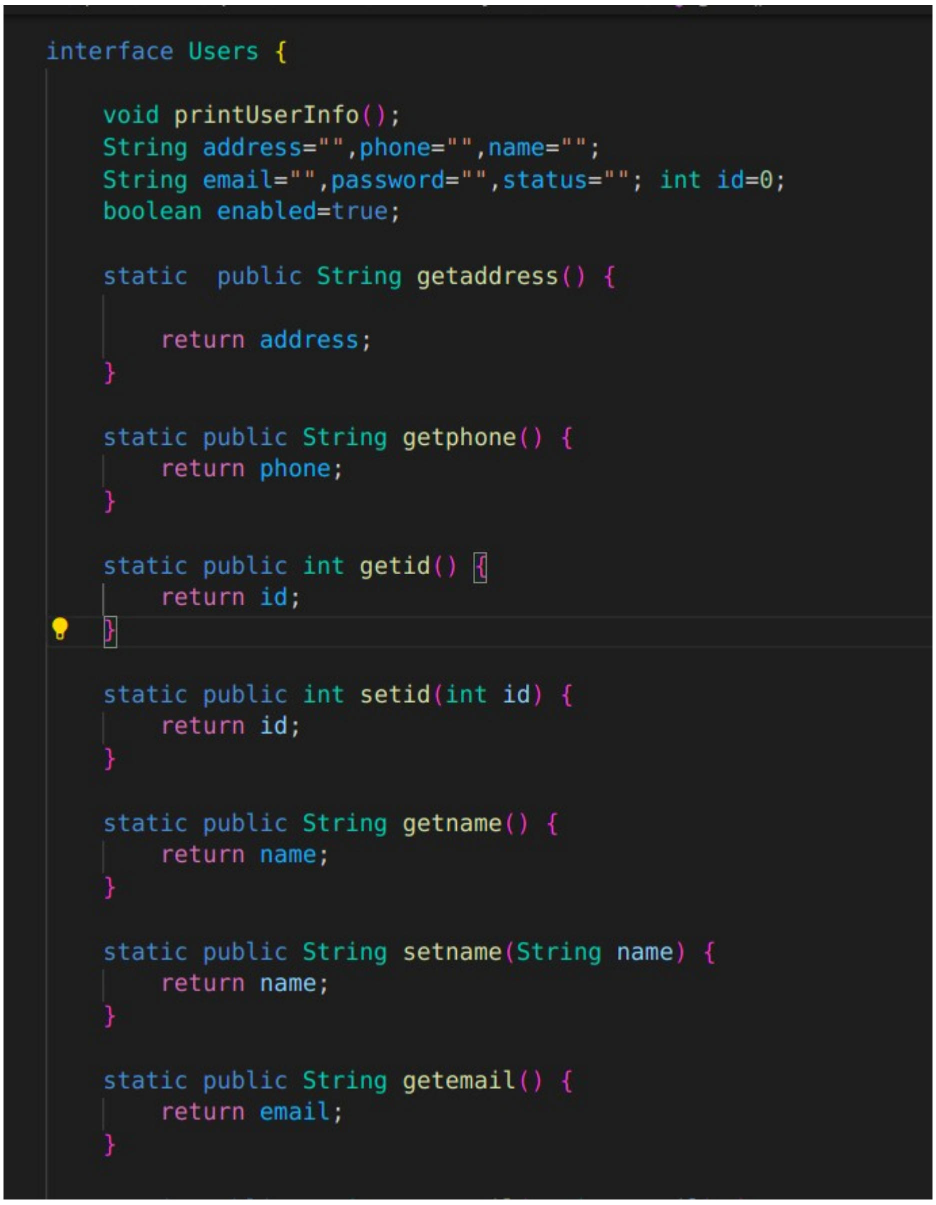
The Factory Pattern is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. The Factory Pattern is implemented through the use of a factory method. This method is responsible for creating objects, and it can be defined in an abstract class or interface. The factory method is typically abstract, and it is implemented by concrete subclasses of the abstract class or interface.

In our code, we have used the Factory Pattern to create objects of different user types, such as student, teacher, and admin. The Users interface defines the common behavior and properties of all user types, and each user type implements this interface in its own way.

The UserFactory class is responsible for creating instances of user types. It has a static method createUser() that takes a String parameter indicating the type of user to create. Based on this parameter, the UserFactory creates a new instance of the appropriate user type and returns it as an object of the Users interface.

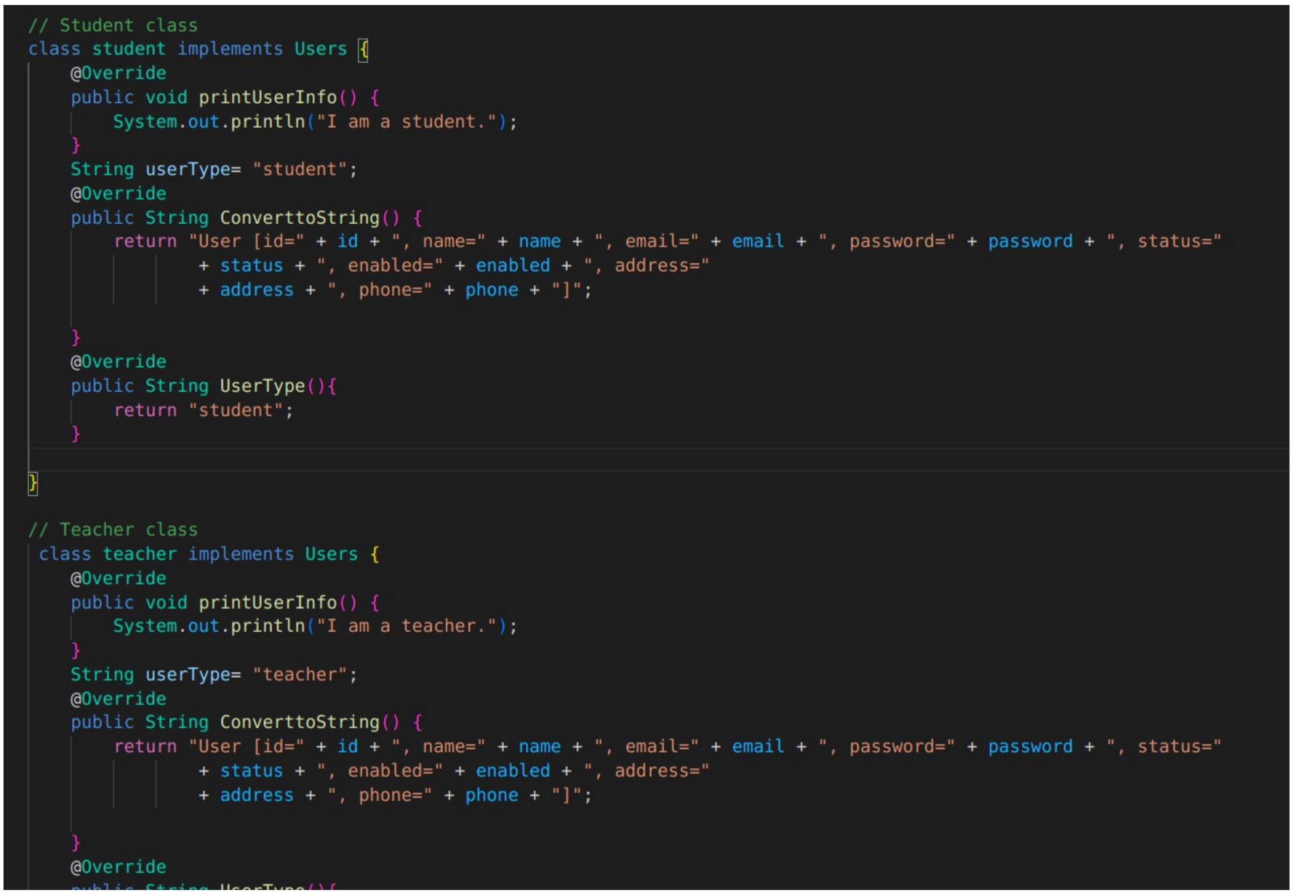
By using the Factory Pattern in this way, the code avoids creating objects directly with the new keyword, and instead delegates object creation to a separate factory class. This makes it easier to add new user types in the future without modifying the existing code. Additionally, it helps to decouple the client code from the specific implementations of the user types, making the code more flexible and maintainable.

Interface Users-

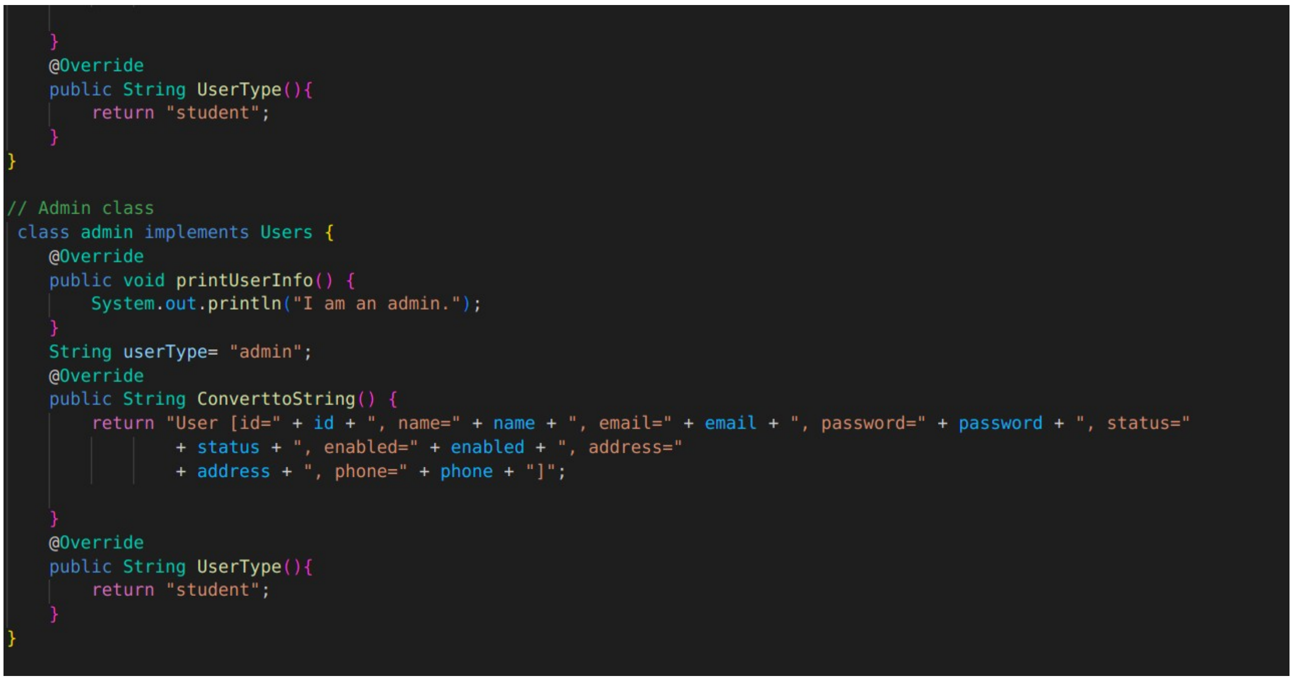


```
interface Users {

    void printUserInfo();
    String address="",phone="",name="";
    String email="",password="",status=""; int id=0;
    boolean enabled=true;

    static  public String getaddress() {

        return address;
    }

    static public String getphone() {
        return phone;
    }

    static public int getid() {
        return id;
    }

    static public int setid(int id) {
        return id;
    }

    static public String getname() {
        return name;
    }

    static public String setname(String name) {
        return name;
    }

    static public String getemail() {
        return email;
    }
```

Student , Teacher and Admin classes implementing Users

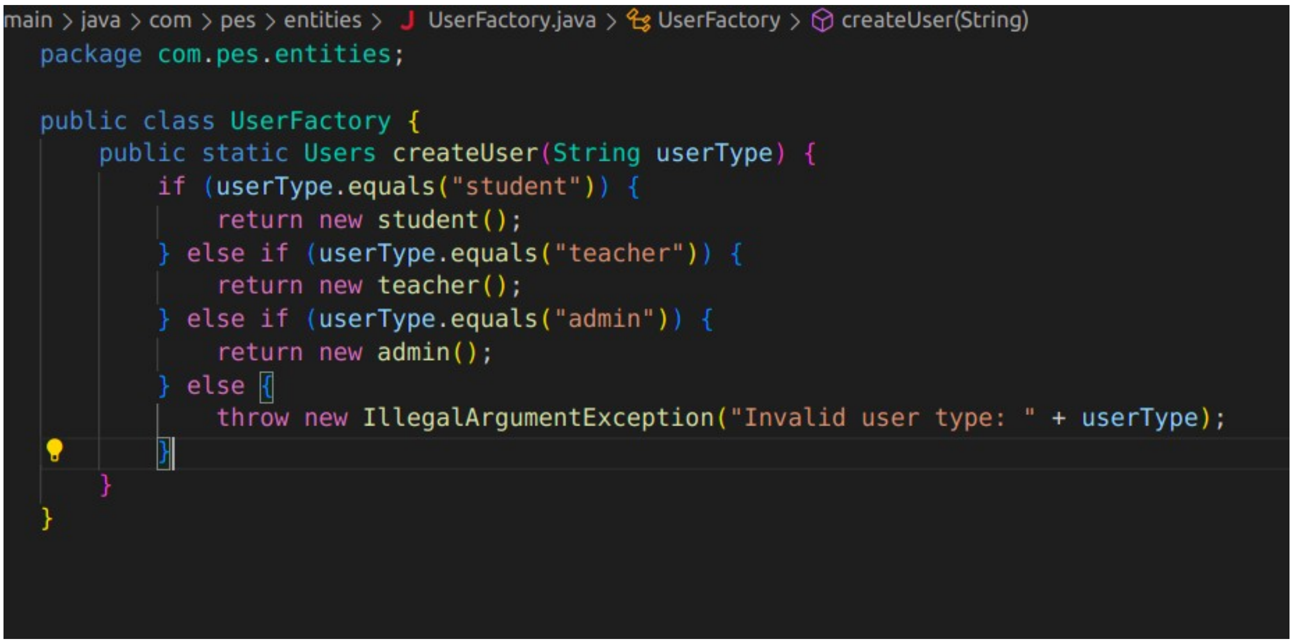


```java
// Student class
class student implements Users {
    @Override
    public void printUserInfo() {
        System.out.println("I am a student.");
    }
    String userType= "student";
    @Override
    public String ConverttoString() {
        return "User [id=" + id + ", name=" + name + ", email=" + email + ", password=" + password + ", status="
                + status + ", enabled=" + enabled + ", address="
                + address + ", phone=" + phone + "]";

    }
    @Override
    public String UserType(){
        return "student";
    }

}

// Teacher class
 class teacher implements Users {
    @Override
    public void printUserInfo() {
        System.out.println("I am a teacher.");
    }
    String userType= "teacher";
    @Override
    public String ConverttoString() {
        return "User [id=" + id + ", name=" + name + ", email=" + email + ", password=" + password + ", status="
                + status + ", enabled=" + enabled + ", address="
                + address + ", phone=" + phone + "]";

    }
    @Override
    public String UserType(){
```



```java
    }
    @Override
    public String UserType(){
        return "student";
    }
}

// Admin class
 class admin implements Users {
    @Override
    public void printUserInfo() {
        System.out.println("I am an admin.");
    }
    String userType= "admin";
    @Override
    public String ConverttoString() {
        return "User [id=" + id + ", name=" + name + ", email=" + email + ", password=" + password + ", status="
                + status + ", enabled=" + enabled + ", address="
                + address + ", phone=" + phone + "]";

    }
    @Override
    public String UserType(){
        return "student";
    }
}
```

Factory Class -



main > java > com > pes > entities > UserFactory.java > UserFactory > createUser(String)

```java
package com.pes.entities;

public class UserFactory {
    public static Users createUser(String userType) {
        if (userType.equals("student")) {
            return new student();
        } else if (userType.equals("teacher")) {
            return new teacher();
        } else if (userType.equals("admin")) {
            return new admin();
        } else {
            throw new IllegalArgumentException("Invalid user type: " + userType);
        }
    }
}
```

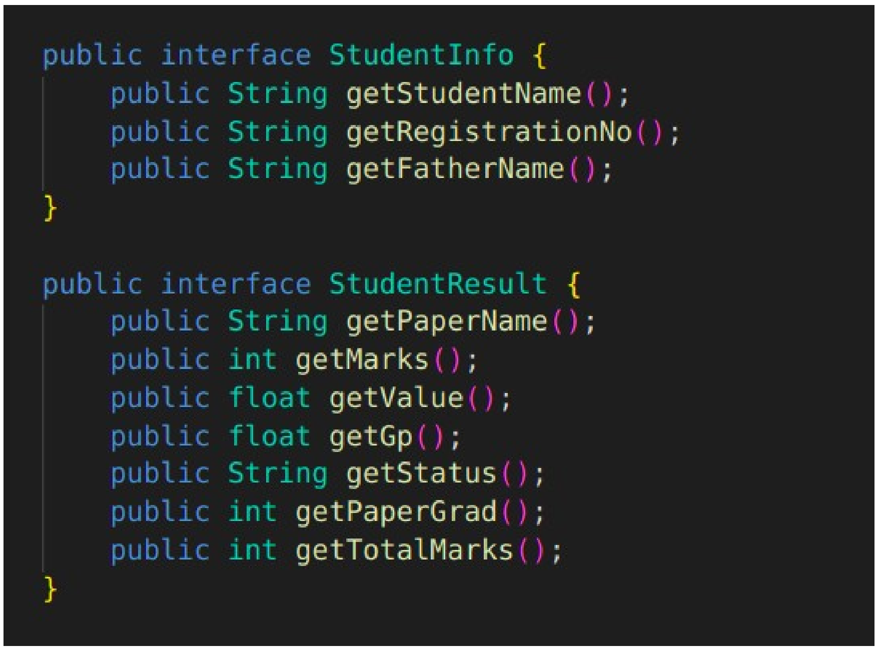
Design Principles -

Interface Segregation Principle

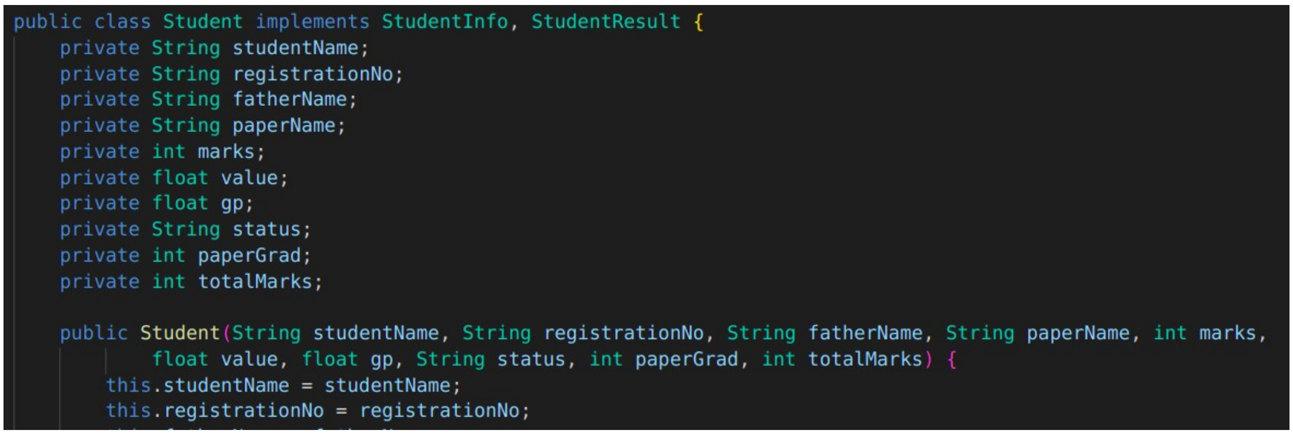
(OOP) that states that clients should not be forced to depend on methods they do not use. It is one of the five SOLID principles of software design that helps developers create more maintainable, flexible, and scalable software systems.

The ISP suggests that interfaces should be designed to be cohesive, meaning that they should contain a set of methods that are closely related to each other and likely to be used together by clients. It also suggests that interfaces should be kept as small as possible, and not contain any unnecessary or irrelevant methods.

In our code, we have divided the interface StudentDetails() into two separate interfaces: StudentInfo and StudentResult that contain the methods related to their respective responsibilities. The Student class can then implement only the necessary interfaces. In this way, the Student class will not implement the methods which it will not be using.



```java
public interface StudentInfo {
    public String getStudentName();
    public String getRegistrationNo();
    public String getFatherName();
}

public interface StudentResult {
    public String getPaperName();
    public int getMarks();
    public float getValue();
    public float getGp();
    public String getStatus();
    public int getPaperGrad();
    public int getTotalMarks();
}
```



```java
public class Student implements StudentInfo, StudentResult {
    private String studentName;
    private String registrationNo;
    private String fatherName;
    private String paperName;
    private int marks;
    private float value;
    private float gp;
    private String status;
    private int paperGrad;
    private int totalMarks;

    public Student(String studentName, String registrationNo, String fatherName, String paperName, int marks,
            float value, float gp, String status, int paperGrad, int totalMarks) {
        this.studentName = studentName;
        this.registrationNo = registrationNo;
```

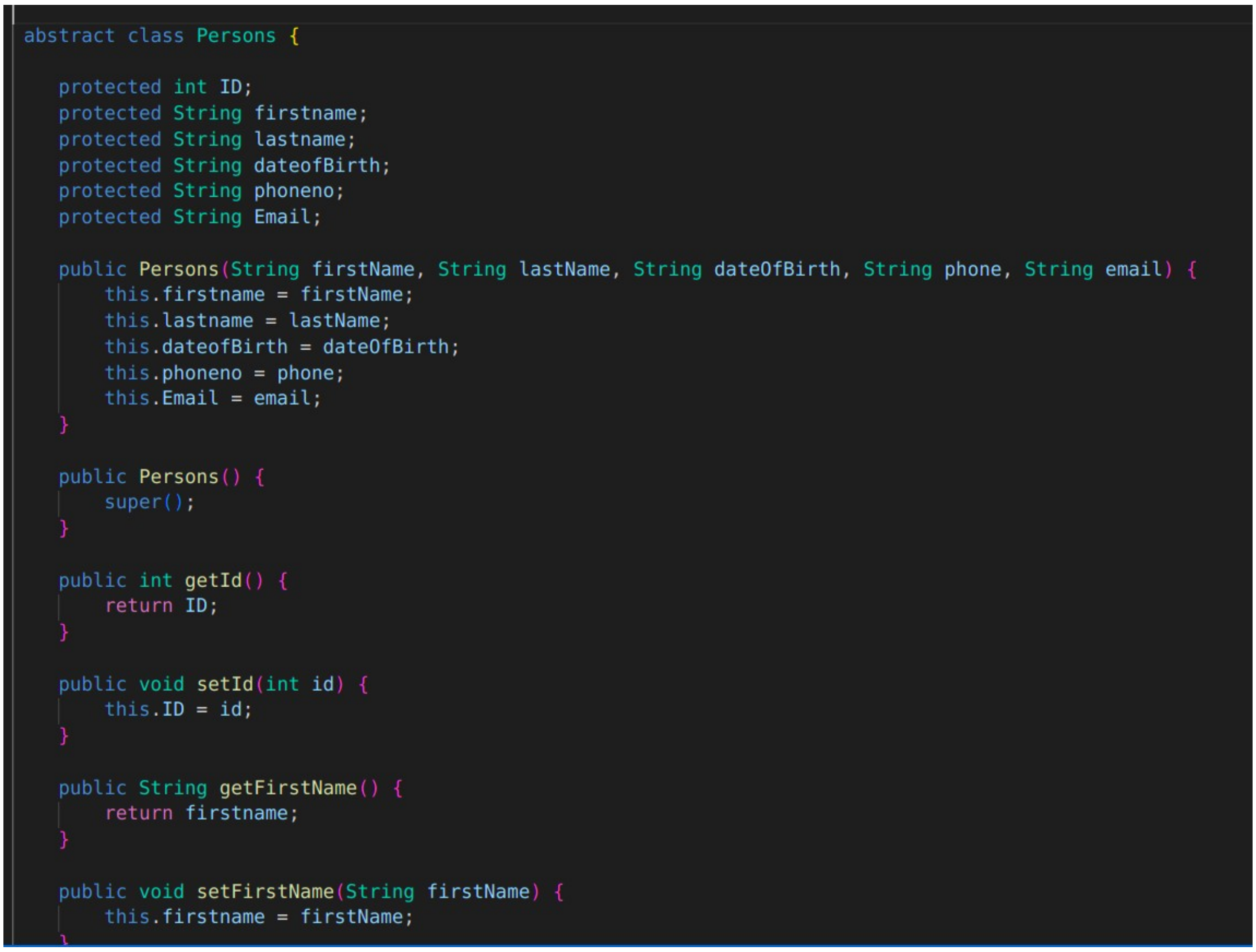
Open-Closed Principle

The Open-Closed Principle (OCP) is a principle of Object-Oriented Programming (OOP) that states that software entities should be open for extension but closed for modification. The OCP suggests that software entities, such as classes, modules, and functions, should be designed in a way that allows them to be extended without modifying their existing code.

To implement the Open-Closed principle in our code, we should make the Student class closed for modification but open for extension.
This means that we should avoid modifying the class directly when new requirements come up. Instead, we should create new classes that inherit from the Student class and add the new functionality there.

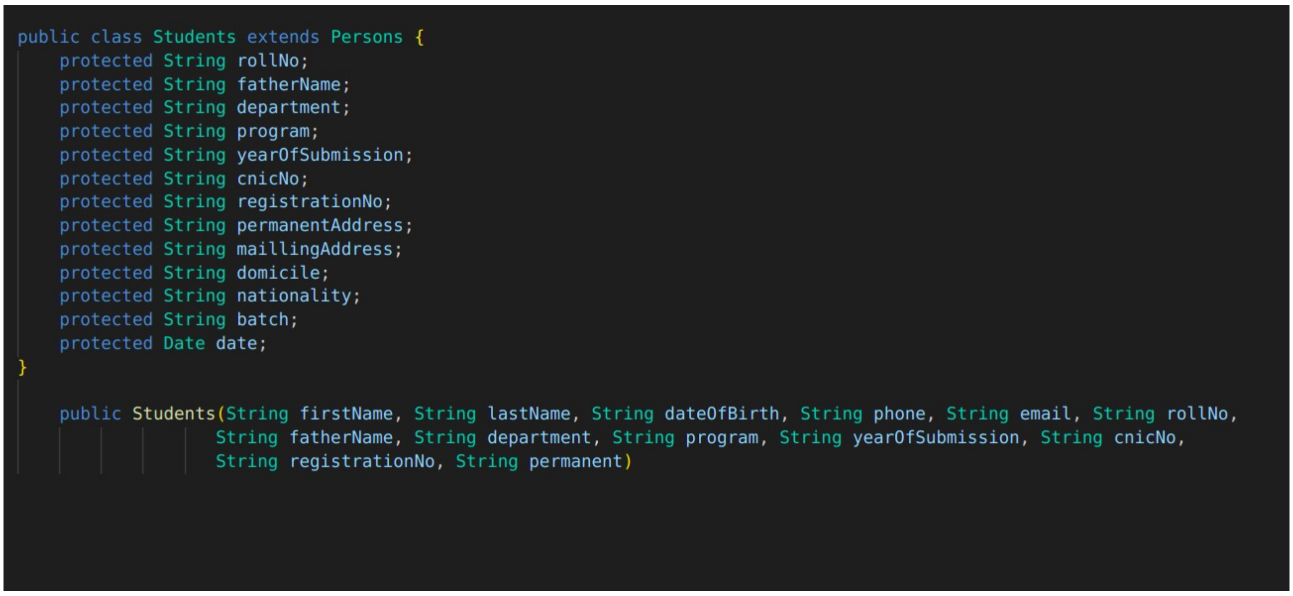
We can achieve this by creating an abstract class called "Person" that represents a generic person and has common attributes such as name, date of birth, and contact information. The Student class can then inherit from the Person class and add attributes that are specific to a student, such as roll number and registration number.

Abstract Class Person -



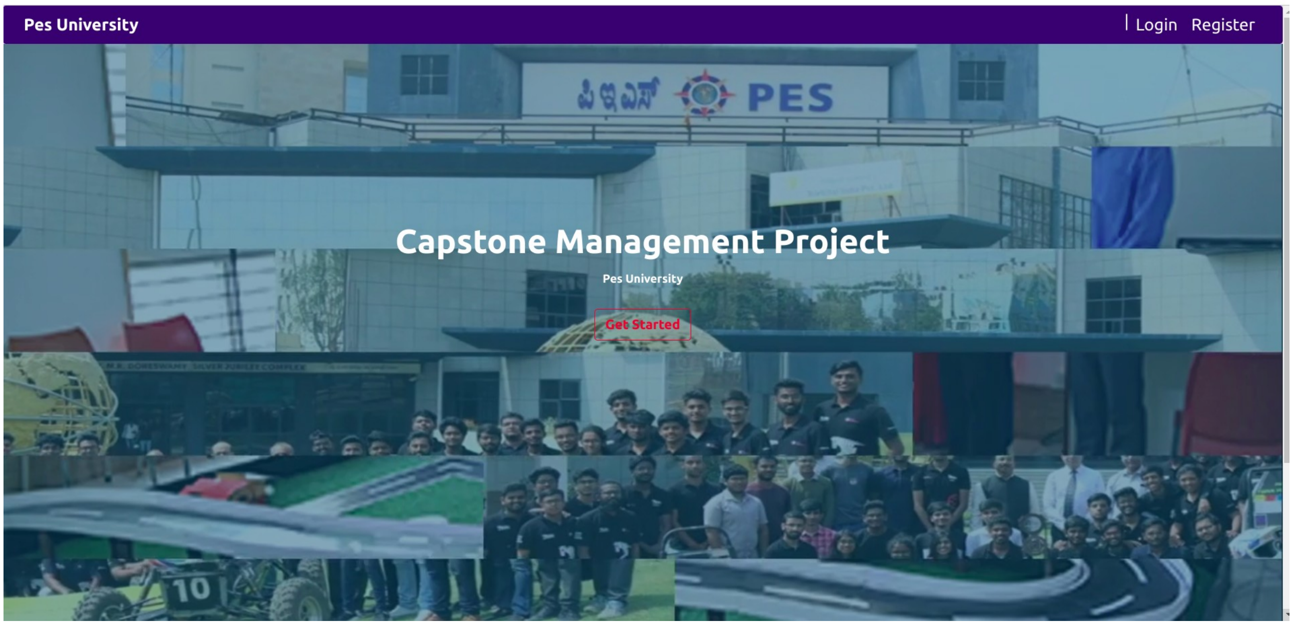
```java
abstract class Persons {

   protected int ID;
   protected String firstname;
   protected String lastname;
   protected String dateofBirth;
   protected String phoneno;
   protected String Email;

   public Persons(String firstName, String lastName, String dateOfBirth, String phone, String email) {
       this.firstname = firstName;
       this.lastname = lastName;
       this.dateofBirth = dateOfBirth;
       this.phoneno = phone;
       this.Email = email;
   }

   public Persons() {
       super();
   }

   public int getId() {
       return ID;
   }

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

   public String getFirstName() {
       return firstname;
   }

   public void setFirstName(String firstName) {
       this.firstname = firstName;
```

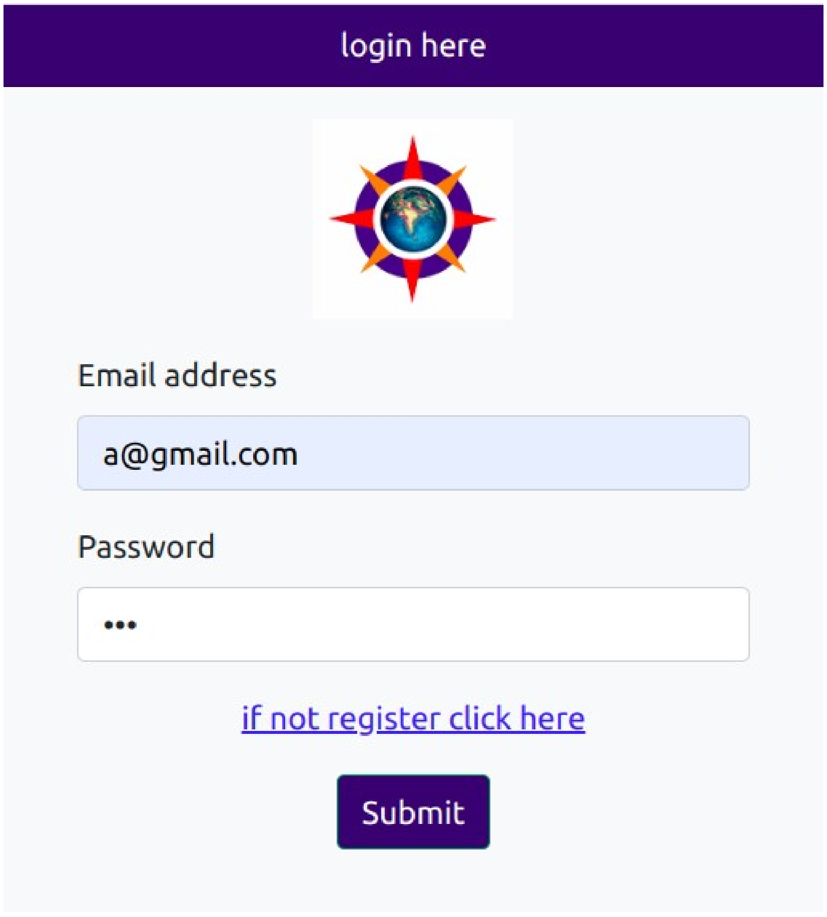
Student class extends Persons



```java
public class Students extends Persons {
    protected String rollNo;
    protected String fatherName;
    protected String department;
    protected String program;
    protected String yearOfSubmission;
    protected String cnicNo;
    protected String registrationNo;
    protected String permanentAddress;
    protected String maillingAddress;
    protected String domicile;
    protected String nationality;
    protected String batch;
    protected Date date;
}

    public Students(String firstName, String lastName, String dateOfBirth, String phone, String email, String rollNo,
                    String fatherName, String department, String program, String yearOfSubmission, String cnicNo,
                    String registrationNo, String permanent)
```

Website Screenshots




login here

Email address

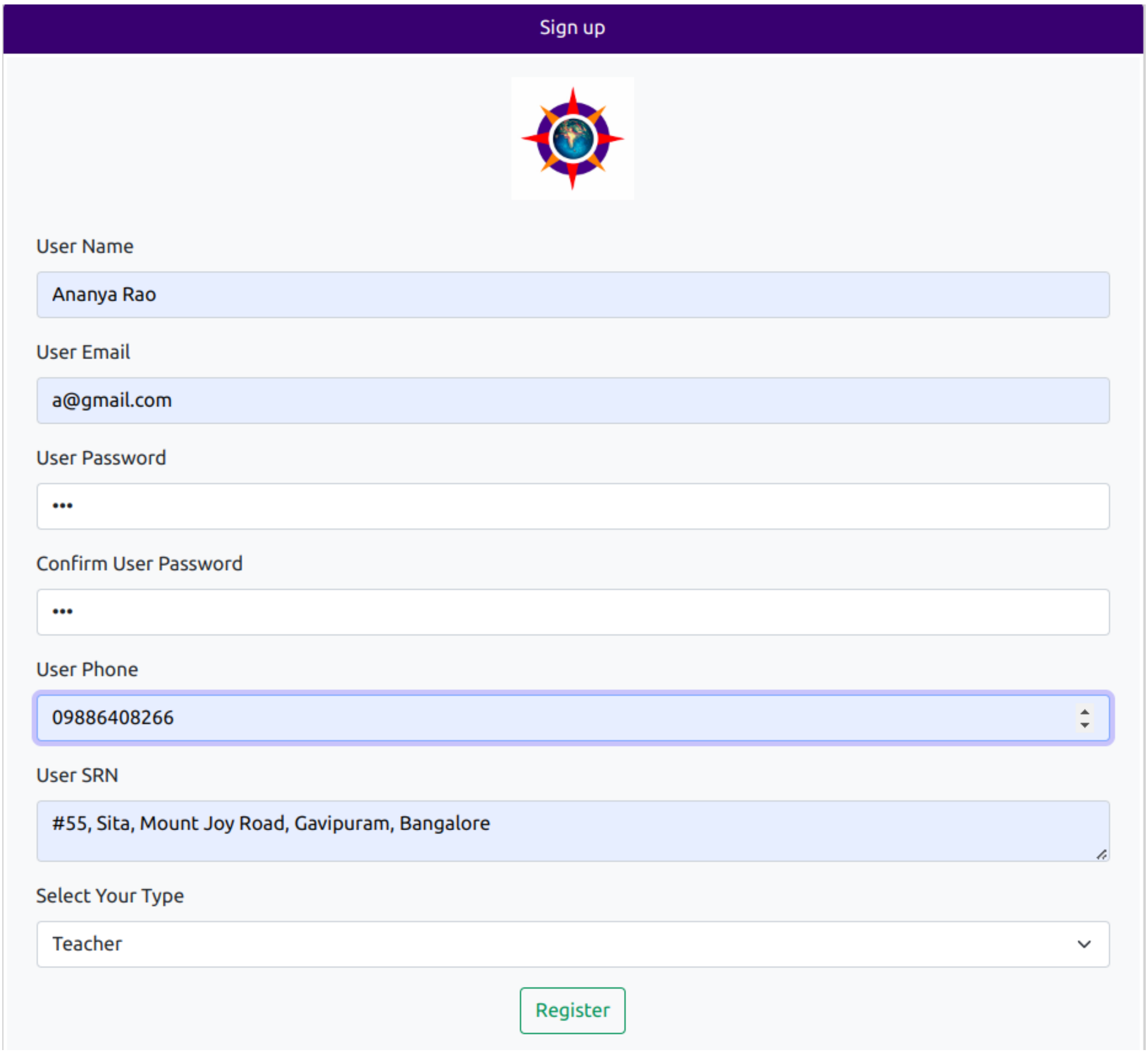
a@gmail.com

Password

•••

if not register click here

Submit



Sign up

User Name

Ananya Rao

User Email

a@gmail.com

User Password

•••

Confirm User Password

•••

User Phone

09886408266

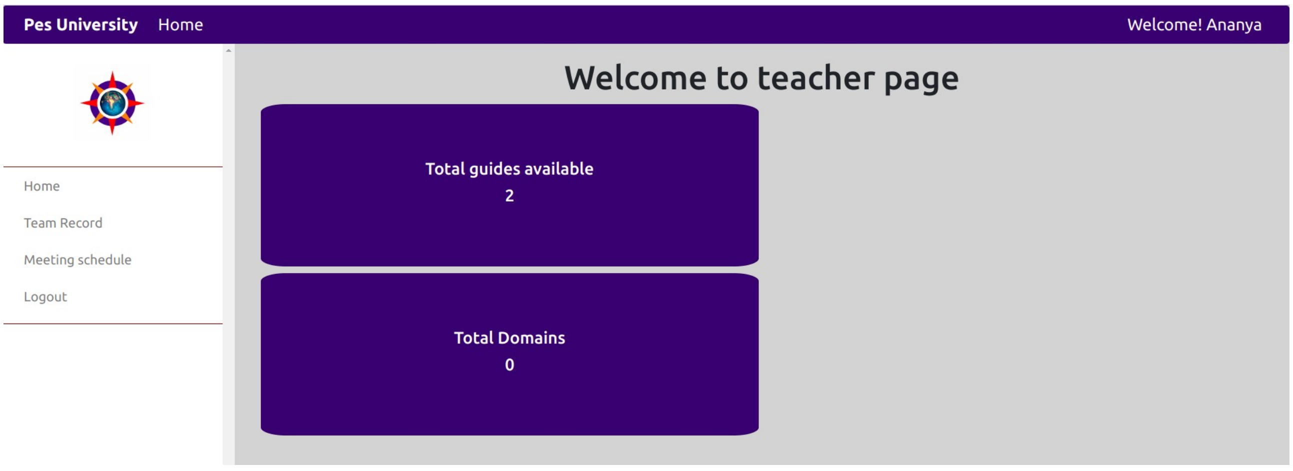
User SRN

#55, Sita, Mount Joy Road, Gavipuram, Bangalore

Select Your Type

Teacher

Register



Pes University Home Welcome! Ananya

Home

Team Record

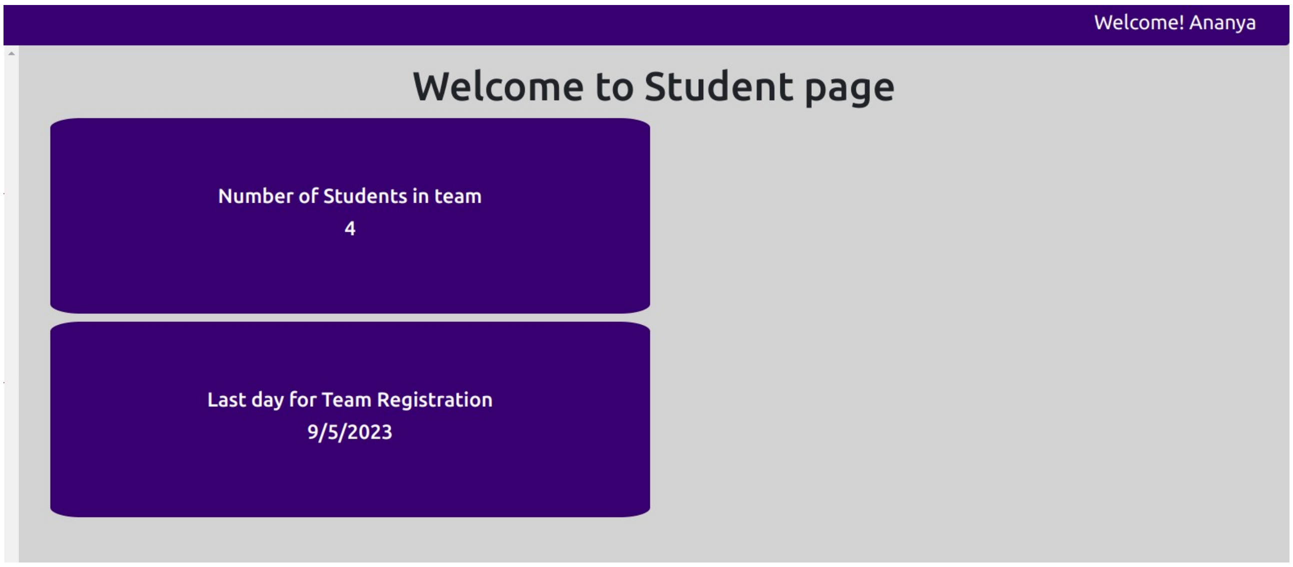
Meeting schedule

Logout

Welcome to teacher page

Total guides available
2

Total Domains
0

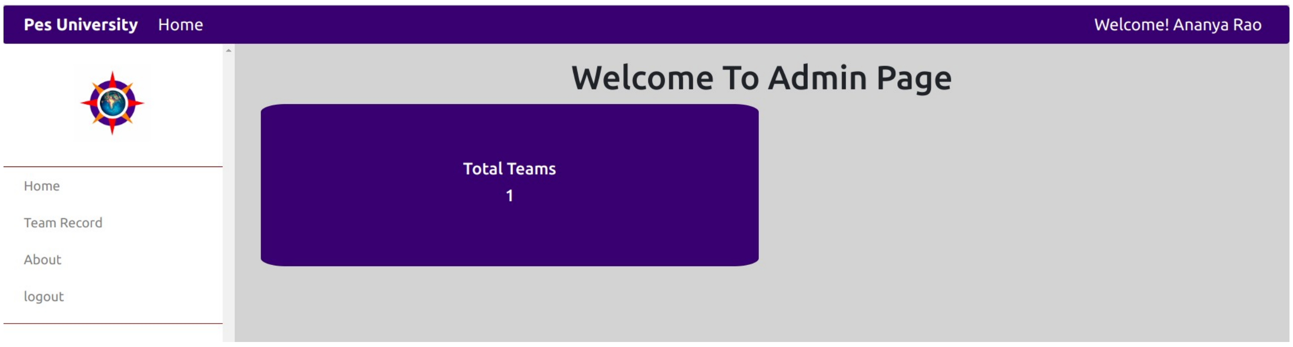


Welcome! Ananya

Welcome to Student page

Number of Students in team
4

Last day for Team Registration
9/5/2023



Pes University Home Welcome! Ananya Rao

Home
Team Record
About
logout

Welcome To Admin Page

Total Teams
1