OOAD PROJECT REPORT

PES UNIVERSITY

OBJECT ORIENTED ANALYSIS AND DESIGN WITH JAVA

COURSE CODE: UE20CS352

AUTOMATION OF PESU ELECTIVE PROCESS

SECTION : CSE- 6K

TEAM DETAILS:

SRN	NAME
PES1UG20CS616	TOUKHEER BAIG
PES1UG20CS644	HARSHA A C
PES1UG20CS647	HRISHIKESH BHAT P
PES1UG20CS656	KISHAN K S

Special Thanks to-

Guide: **BHARGAVI MOKASHI**

OOAD PROJECT REPORT

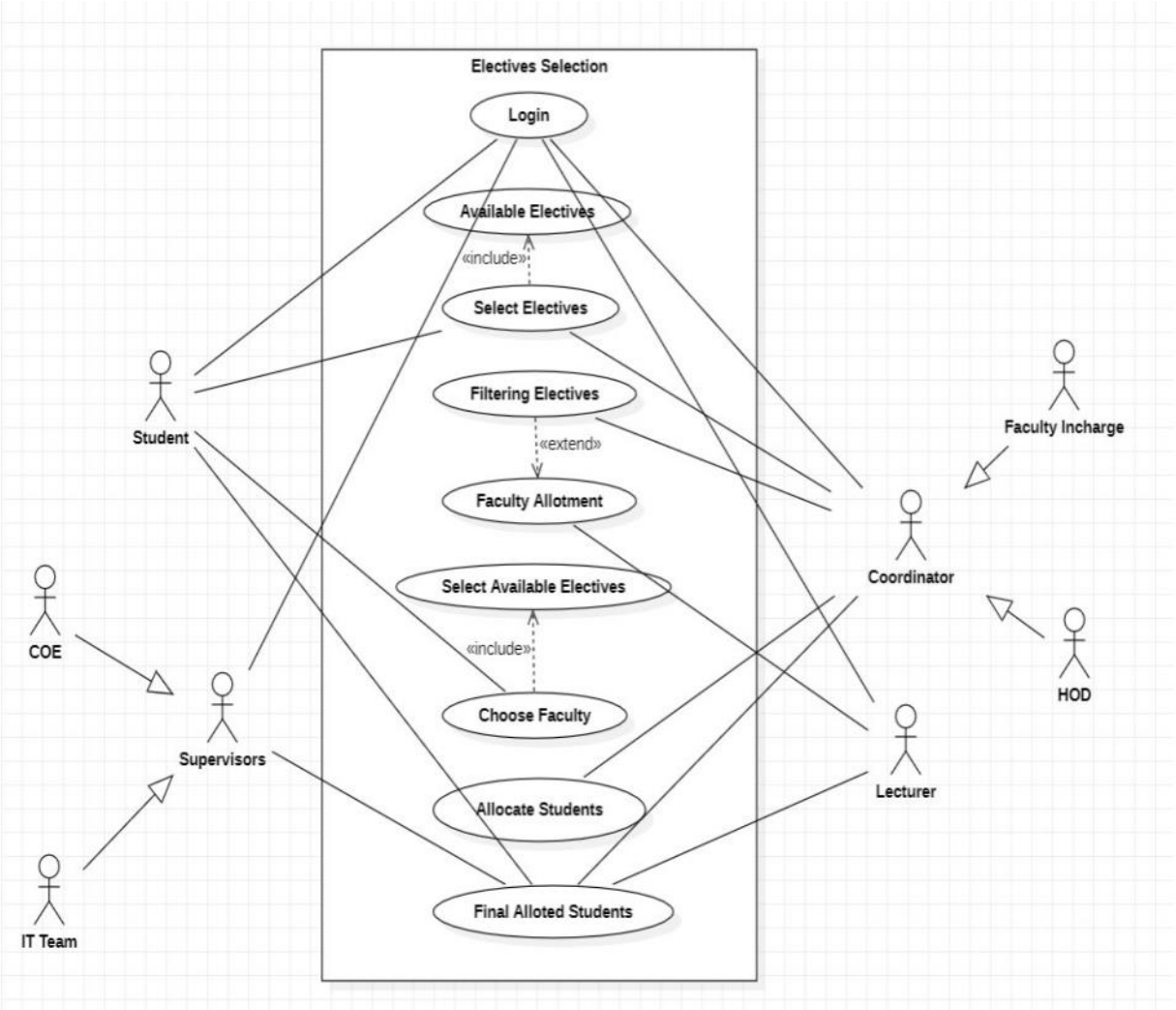
PROBLEM STATEMENT:

The current system at PES University faces challenges in managing elective courses and assigning faculty to students. The manual process of sharing Google forms for each iteration through mail leads to inconvenience and confusion for students. Once the student responses from forms are recorded, they have to be extracted into excel sheets and fed to the algorithm manually for allocation of students and faculty. Additionally, the coordinator and IT staff do not have a single platform to view the allocation database.

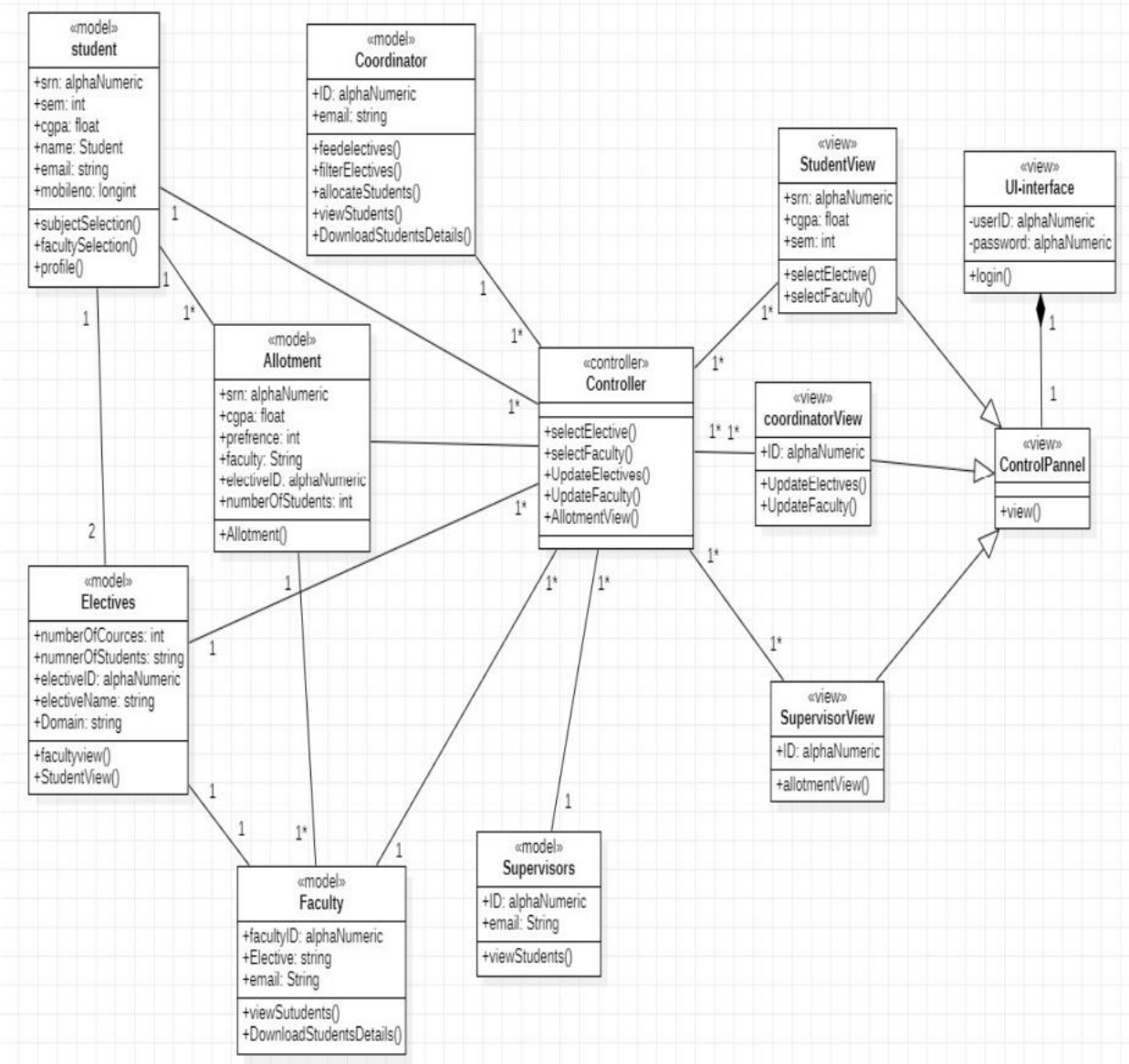
To address these issues, this project proposes an Elective Automation System that provides a single platform for students, coordinators, faculty and IT staff. The system automates the processes of allocation of students to electives and their query resolution.

The project will be developed as a modern web-based application using the Spring Boot framework and implementing the MVC architecture. The system will comprise four modules: Student, Faculty, Coordinator, and IT Staff, each with a distinct dashboard. The Student module will enable log in and select elective subjects for the first iteration, select elective subjects and faculty for respective subjects in the second and third iterations, finally view allocated electives-faculties and even raise queries. The Coordinator module will provide access to student databases and elective course information. Additionally the Coordinator module will allow coordinator to address queries raised by students. Functionalities provided by IT staff and Faculty are similar which is to view the allocated results. Upon completion of the submission process for each iteration (2 in total), an algorithm will allocate students based on their CGPA and faculty availability.

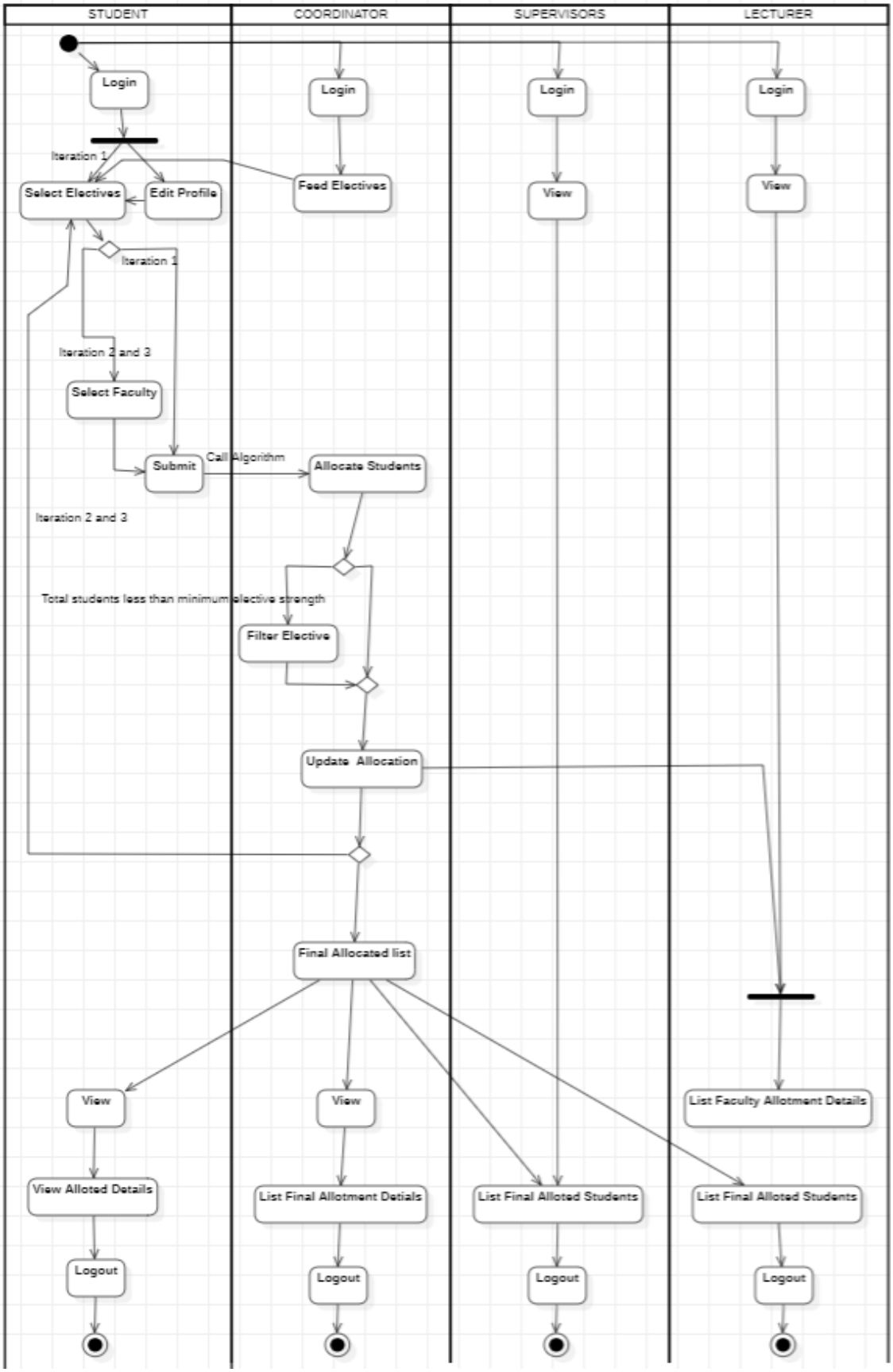
Use Case Diagram:



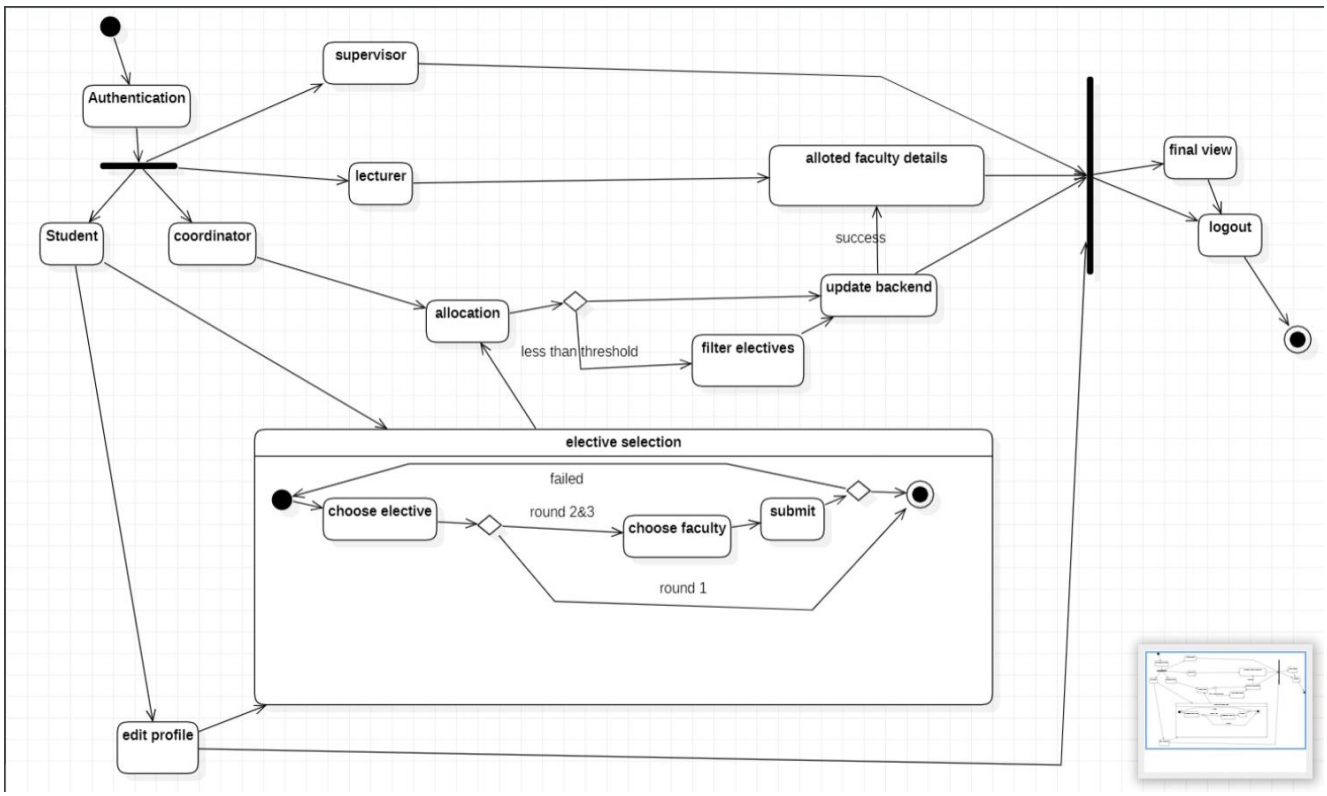
Class Diagram:



Activity Diagram:



State Diagram:



OOAD PROJECT REPORT

USE CASES:

- USE CASE 01: SIGNUP AND LOGIN
  - Actors: Student
  - Description: Users can sign up and login to their respective modules with their credentials. Signing up requires providing basic personal information such as name, email, and password. Login functionality will validate the user's credentials against the database and allow access to the module's dashboard.

- USE CASE 02: VIEW DATABASE
  - Actors: Student, Faculty, IT staff, Coordinator
  - Description: This use case allows all users to access the same database information for both the rounds, ensuring transparency. The presented information includes the student's SRN, name, semester, email, elective selected, and the faculty assigned to that elective. This provides an overview of the student's allocation details for the respective elective.

- USE CASE 03: SELECT ELECTIVES AND FACULTIES
  - Actors: Student
  - Description: Students can select their preferred elective courses and faculty members for respective courses in each iteration. The web application will validate the student's eligibility for the elective course based on their CGPA and faculty member's availability. Once the student submits their choices, the system will allocate them to a high priority faculty member if available.

- USE CASE 04: QUERY RESOLUTION
  - Actors: Student, Coordinator
  - Description: Students can raise queries related to the elective course allocation process or any other technical issue, which will be stored in the database. The Coordinator can view and answer these queries, which will be communicated back to the student. The query resolution process will ensure that the students' concerns are addressed promptly, and the IT staff can maintain a record of the queries and their resolutions for future reference.

OOAD PROJECT REPORT

CLASS MODELS:

- MODEL 01: USER
  The User model represents a user of the system and contains attributes such as username, password, and userType (role). It is used for authentication and authorization purposes.

- MODEL 02: STUDENT (Iteration 01)
  The Student model represents a student in the first iteration of the elective selection process. It contains attributes such as student SRN, name,semester, email, phone number, section, CGPA, query made if any and Elective 1 and 2 selected by the student.

- MODEL 03: STUDENT (Iteration 02 and 03)

  This Student model is an updated version of the previous iteration that represents a student in the second and third rounds of the elective selection process. In addition to the attributes featured in the first iteration, this model includes information on the specific faculties chosen for each elective.

- MODEL 04: QUERY MODEL
  The Query model represents a query raised by a student. It contains attributes such as query ID, student SRN, semester, email and query description. This model is used to track and manage student queries by the faculty.

- MODEL 05: ALLOTMENT RESULT
  The Allotment Result model enables logged in user to view the allotment results for round 1 and 2. All the types of users will view the same database. This is to maintain transparency and clarity among students, faculty, Coordinator and IT staff.

OOAD PROJECT REPORT

ARCHITECTURE PATTERN:

MVC (Model View Controller):

This pattern separates the application logic into three interconnected components: the model (data and business logic), the view (user interface), and the controller (handles user input and updates the model and view).

- Model: The model will represent the data and business logic of the application. This will include classes such as Student, Query and User (for login) etc. The model classes will be responsible for retrieving and manipulating data, as well as performing any necessary calculations or algorithms.
- View: The view will be responsible for rendering the data from the model and presenting it to the user in a user-friendly way. This will include the web pages and interfaces that the user interacts with, such as login screens, dashboards, and queries.
- Controller: The controller will act as an intermediary between the model and the view. In the project's context, single controller is employed for all the user categories. It will receive requests from the user via the view, and use the model to perform the necessary operations to fulfill those requests. This will include handling user inputs, validating user data, and invoking the appropriate methods on the model to update or retrieve data. The controller will then return the results back to the view to be displayed to the user.

DESIGN PRINCIPLES :

- Single Responsibility Principle (SRP): This principle suggests that each module or class should have only one responsibility or reason to change. In the context of this project, each module (Student, Query) have their own service methods which is only responsible for saving the changes into the database.

- Dependency Inversion Principle (DIP): This principle suggests that high-level modules should not depend on low-level modules, but rather on abstractions. In the context of this project, the Student, Faculty, Coordinator, and IT Staff modules depend on abstractions, such as interfaces, rather than concrete implementations. The implementations are done in JpaRepositories interfaces, which are implemented by various modules to perform the database operations.

- Interface Segregation Principle (ISP): This principle suggests that a class should not be forced to implement interfaces that it does not use. In the context of this project, the modules only implements the interfaces that are relevant to their functionalities. This is possible since each model implements different JpaRepository interfaces.

OOAD PROJECT REPORT

DESIGN PATTERNS:

- SINGELTON PATTERN: This pattern ensures that only one instance of a class is created and provides a global point of access to it. In the Elective Automation System, a singleton pattern is used to ensure that only one instance of the student, faculty, Coordinator and IT staff is created.

- FACTORY PATTERN: This pattern provides a way to create objects without exposing the creation logic to the client. A factory interface provides methods to create objects, while a factory implementation provides the actual implementation of these methods. Spring boot uses this pattern by default. In the Elective Automation System, a factory pattern is used to create instances of the different modules (Student, Faculty, Coordinator, and IT staff) during login.

- REPOSITORY PATTERN: This pattern separates the application's data access logic from the rest of the application. A repository interface provides methods to access and manipulate data, while a repository implementation provides the actual implementation of these methods. The repository pattern is used to handle database operations in the Elective Automation System. We have also extended this interface to include our customised SQL query for few operations.

- CHAIN OF RESPONSIBILITY: Chain of Responsibility is a behavioral design pattern where a series of handlers are chained together, and each handler has the opportunity to handle a request or pass it to the next handler in the chain. In the context of student allotment, the chain can be used to check if the 1st preference faculty has reached the student limit (60 students), and if so, pass the request to the handler responsible for checking the 2nd preference faculty. Kindly note that this has not been completely implemented in the project. However we are interested in implementing it in the future.

GitHub link to the Code base: https://github.com/Hrishi34/OOAD_Project

OOAD PROJECT REPORT

Individual contributions of the team members:

a. HRISHIKESH BHAT P
Created the data models for the project (for Student, User, AllotmentResult, and Query). This involved establishing the basic structure/foundation for the project. The process required some modifications to the original diagrams (Class/State/Activity diagrams), particularly regarding the methods and attributes of the models, in response to challenges that would be encountered during implementation.

b. TOUKHEER BAIG:
Implemented a controller that manages the control and logic flow or routing between different HTML pages. This includes fetching models from the database and utilizing the JpaRepository interface for performing database operations. It required a good understanding of the schema of various models in the project.

c. KISHAN K S

Was responsible for the front-end development tasks. This involved designing and creating all the HTML pages, applying CSS styles, and planning the presentation of information on various pages and dashboards. Handled the communication between the views and controllers, ensuring the proper implementation of GET and POST methods for data exchange.

d. HARSHA A C
Had two key responsibilities. Firstly, developed the algorithm for student allocation for rounds, taking into consideration factors such as student CGPA, faculty preferences, and faculty availability. This algorithm was implemented within the service package. Secondly, was in charge of integrating and organizing all the team members' work to create the final functional web application.

Screenshots:

1. MVC implementation.

```
public String it_dash(Model model) {
     List<postallot> students = postallotRepository.findAll();
        model.addAttribute("students", students);
        System.out.println(students);
    return "it_dash.html";

}
```

This class is written in Controller. In the first line, controller fetches students list through model, sends it to the view (it_dash.html) which is presented to client.

2. Design Principles Implementation:

   a. Single Responsibility Principle (SRP):

```
@Service
public class postallotService {
    postallotRepository postallotRepository;

    public void save(postallot postallot) {
        // TODO Auto-generated method stub
        System.out.println("save method "+postallot);
        postallotRepository.save(postallot);

    }

}
```

Only save method is implemented in this class.

   b. Interface Segregation Principle (ISP): Separate interface for each models

queryRepository.java

```
package com.pesu.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface queryRepository extends JpaRepository<query, Integer>{

    @Modifying
    @Query("UPDATE query q SET q.answer = ?2 WHERE q.id = ?1")
    int updateAnswerById(Integer id, String answer);

}
```

UserRepository.java

```
package com.pesu.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, String>{

    @Query("SELECT p FROM User p WHERE p.userId = :userId AND p.password = :password AND p.use
    User findByUserIdAndPasswordAndUserType(String userId, String password, String userType);

}
```

Here queryRepository and UserRepository have their own repository implementations.

c. Dependency Inversion Principle (DIP):

```
165    @Autowired
166    StudentRepository studentRepository;
167    @RequestMapping(value = "/addStudent")
168    public String addStudent(student student) {
169        System.out.println(student);
170        studentRepository.save(student);
171        return "redirect:/stu_dash";
172    }
173
```

Here the studentRepository implements JpaRepository which performs low level database operations such as save.

3. Design Patterns Implementation:

a. Factory Pattern: Login class in controller

```
@PostMapping(value = "/login")
public String login(@RequestParam String username, @RequestParam String password, @RequestParam String login_type) {
       System.out.println(login_type);
       System.out.println(username);
       System.out.println(password);
        if (login_type.equals("Student")) {
            User user = repository.findByUserIdAndPasswordAndUserType(username, password,login_type);
            System.out.println(user);
            if (user != null) {
                return "redirect:/stu_dash";
            }

        }
        else if (login_type.equals("Coordinator")) {
            User user = repository.findByUserIdAndPasswordAndUserType(username, password,login_type);
            System.out.println(user);
            if (user != null) {
                return "redirect:/co_dash";
            }
        }else if (login_type.equals("it-team")) {
            User user = repository.findByUserIdAndPasswordAndUserType(username, password,login_type);
            System.out.println(user);
            if (user != null) {

                return "redirect:/it_dash";
            }
        }
        return "redirect:/";
    }
```

b. Singleton Pattern :

```
43          for (studentreg2 student : students) {
44              postallot postallot = new postallot();
45              if (student.getE1() != null) {
46                  String key1 = student.getE1() + " - e1_f1_1";
47                  String key2 = student.getE1() + " - e1_f2_1";
48                  System.out.println("1"+key1+" "+key2);
49                  if (allocatedStudentsByElectiveAndFaculty.get(key1).size() < 60) {
50  //                      student.setE1(student.getE1());
51  //                      student.setE1f1_1("e1_f1_1");
52                      System.out.println("1"+student);
```

Here postallot object is created only once in the entire application.

c. Repository Pattern: JpaRepository interface implementation

```
1  package com.pesu.demo.repository;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
10
11 public interface queryRepository extends JpaRepository<query, Integer>{
12
13     @Modifying
14     @Query("UPDATE query q SET q.answer = ?2 WHERE q.id = ?1")
15     int updateAnswerById(Integer id, String answer);
16
17 }
18
```

4. Adding User and Student to the database:

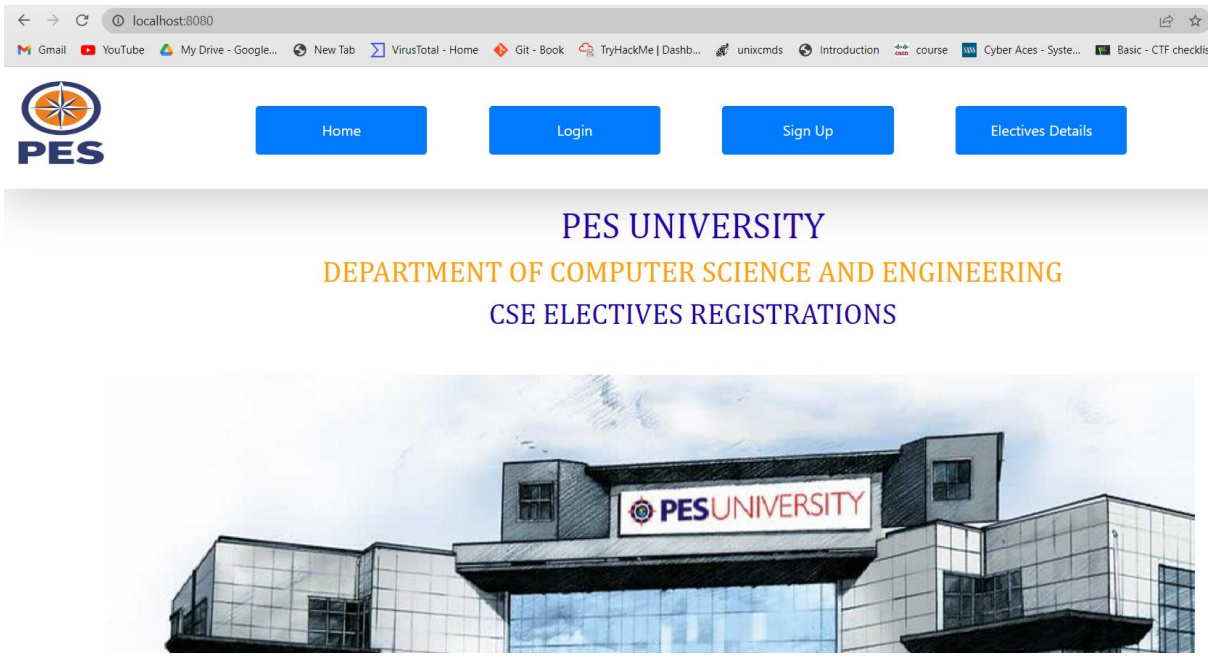
```
145     @Autowired
146     UserRepository repository;
147     @RequestMapping(value = "/addUser")
148     public String addUser(User user) {
149         System.out.println(user);
150         repository.save(user);
151         return "redirect:/login";
152     }
153     @Autowired
154     StudentRepository studentRepository;
155     @RequestMapping(value = "/addStudent")
156     public String addStudent(student student) {
157         System.out.println(student);
158         studentRepository.save(student);
159         return "redirect:/stu_dash";
160     }
```

If student was to signup, s/he would be added to both User and Student tables.

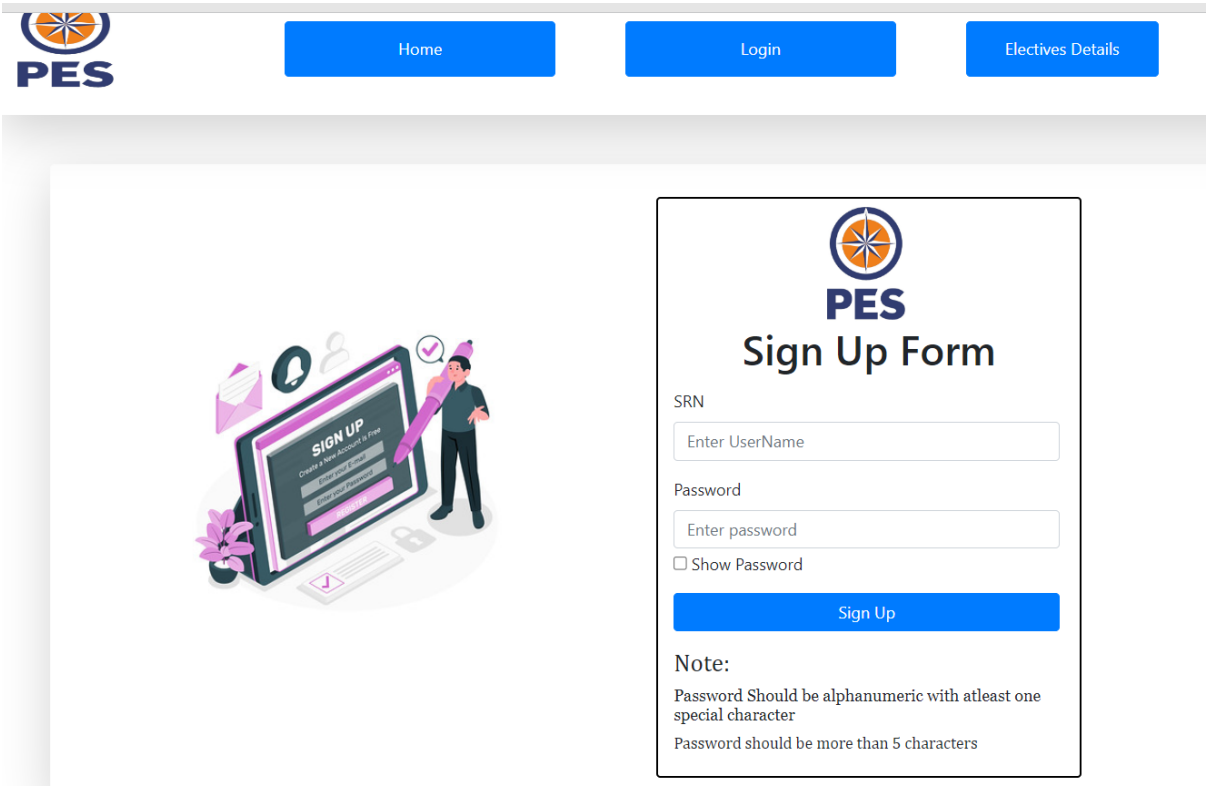
5. Allocation algorithm implementation.

```
        allocatedStudentsByElectiveAndFaculty.put("e21 - e2_f2_2", new ArrayList<>());
        for (studentreg2 student : students) {
            postallot postallot = new postallot();
            if (student.getE1() != null) {
                String key1 = student.getE1() + " - e1_f1_1";
                String key2 = student.getE1() + " - e1_f2_1";
                System.out.println("1"+key1+" "+key2);
                if (allocatedStudentsByElectiveAndFaculty.get(key1).size() < 60) {
//                    student.setE1(student.getE1());
//                    student.setE1f1_1("e1_f1_1");
                    System.out.println("1"+student);
//                    studentreg2Repository.save(student);
                    allocatedStudentsByElectiveAndFaculty.get(key1).add(student);
//                    postallot postallot = new postallot();
                    postallot.setS_name(student.getSname());
                    postallot.setSrn(student.getSrn());
                    postallot.setE1(student.getE1());
                    postallot.setE1f1(student.getE1f1_1());
                    System.out.println("1"+postallot);
//                    postallotRepository.save(postallot);
                    System.out.println("Allocated " + student.getSname() + " to " + key1);
                } else if (allocatedStudentsByElectiveAndFaculty.get(key2).size() < 60) {
//                    student.setE1(student.getE1());
//                    student.setE1f2_1("e1_f2_1");
                    System.out.println("2"+student);
//                    studentreg2Repository.save(student);
                    allocatedStudentsByElectiveAndFaculty.get(key2).add(student);
//                    postallot postallot = new postallot();
                    postallot.setS_name(student.getSname());
                    postallot.setSrn(student.getSrn());
                    postallot.setE1(student.getE1());
                    postallot.setE1f1(student.getE1f2_1());
                    System.out.println("2"+postallot);
//                    postallotRepository.save(postallot);
                    System.out.println("Allocated " + student.getSname() + " to " + key2);
                } else {
                    System.out.println("Unable to allocate " + student.getSname() + " to " + student.getE1());
                }
```

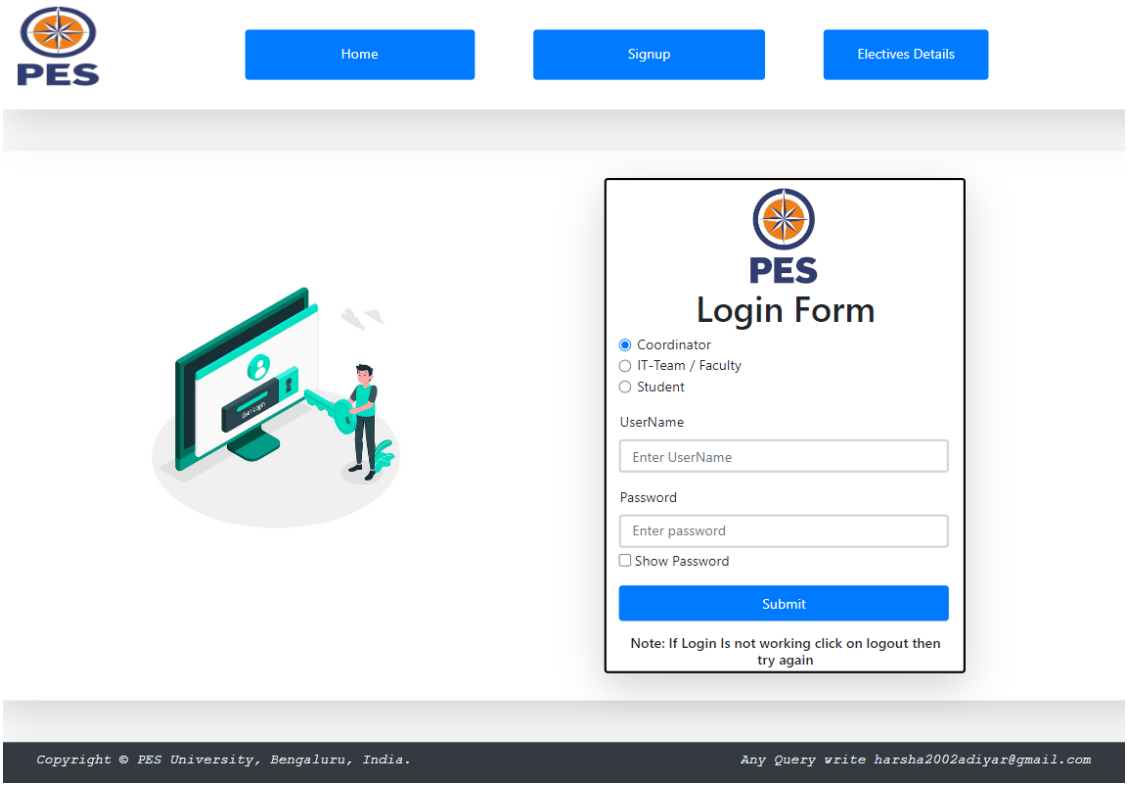
6. Front Page



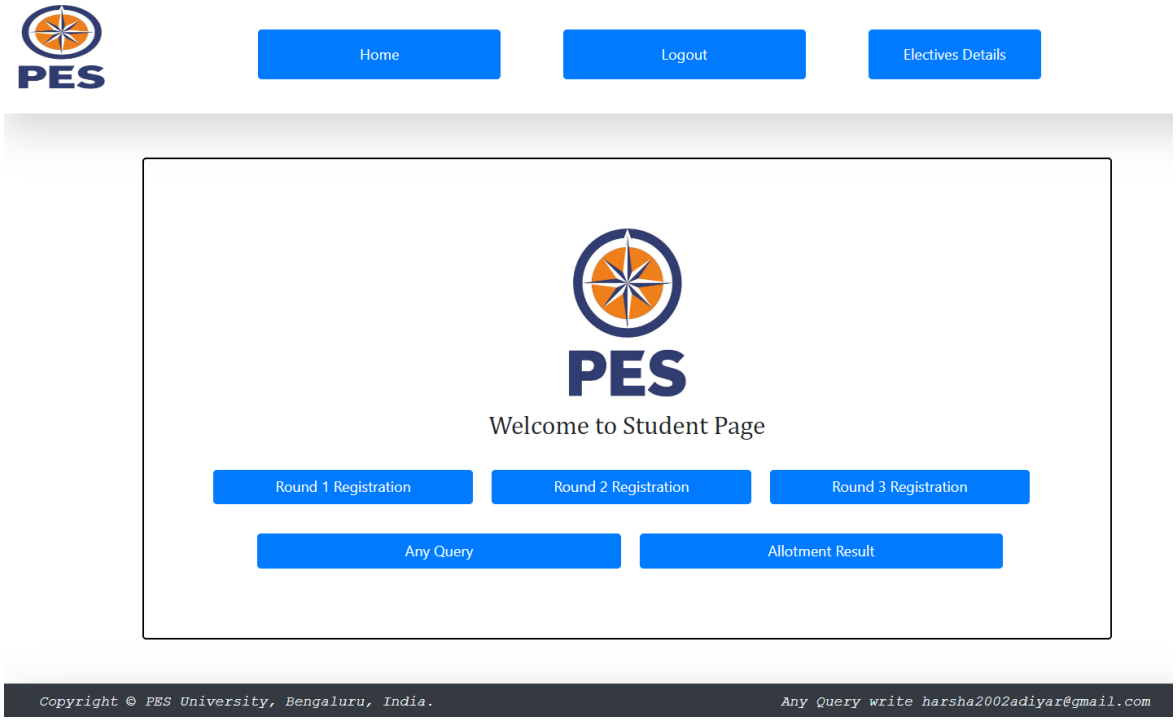
7. Sign Up page.



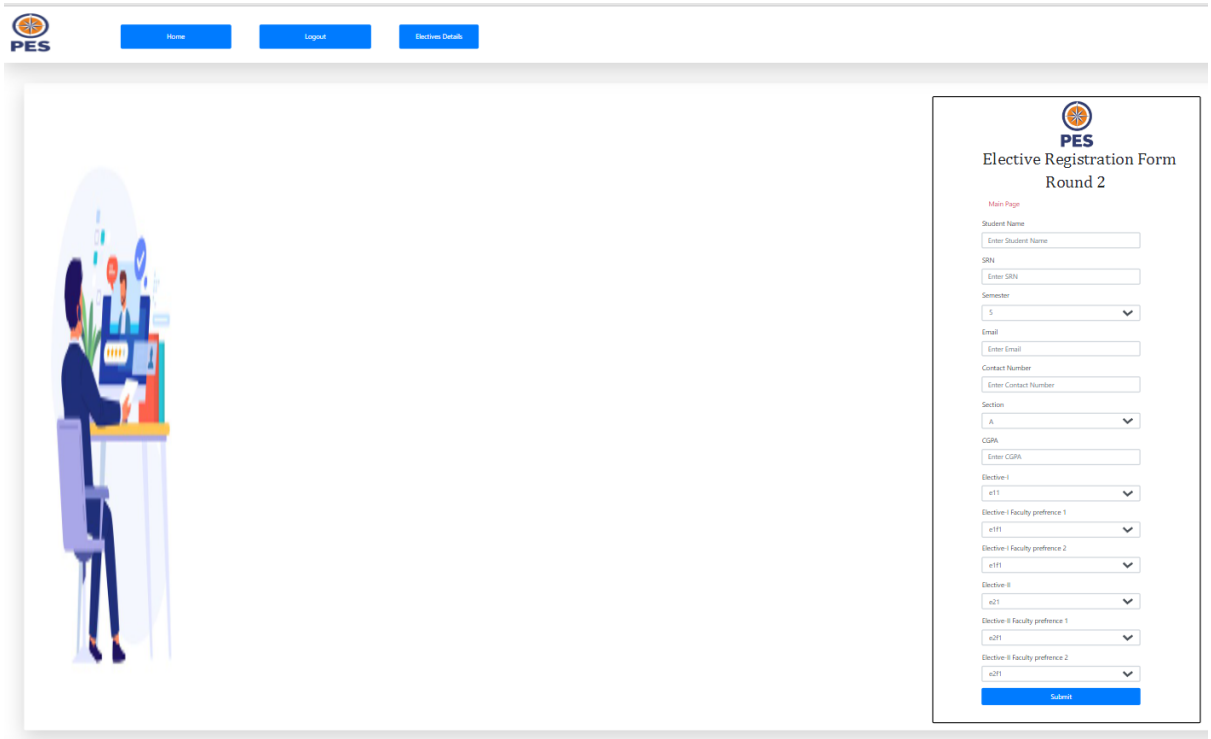
8. User Login



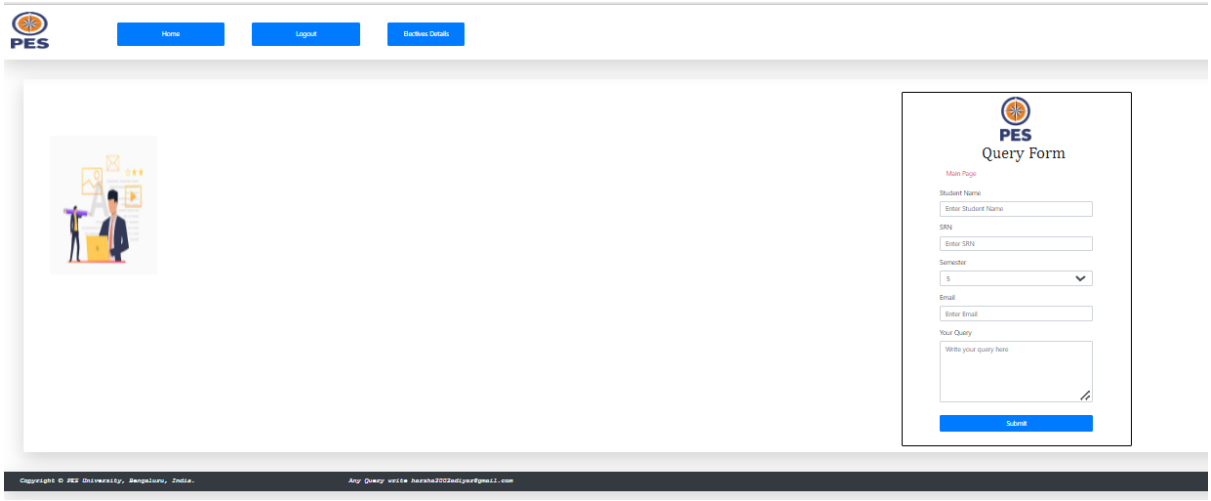
9. Student Dashboard



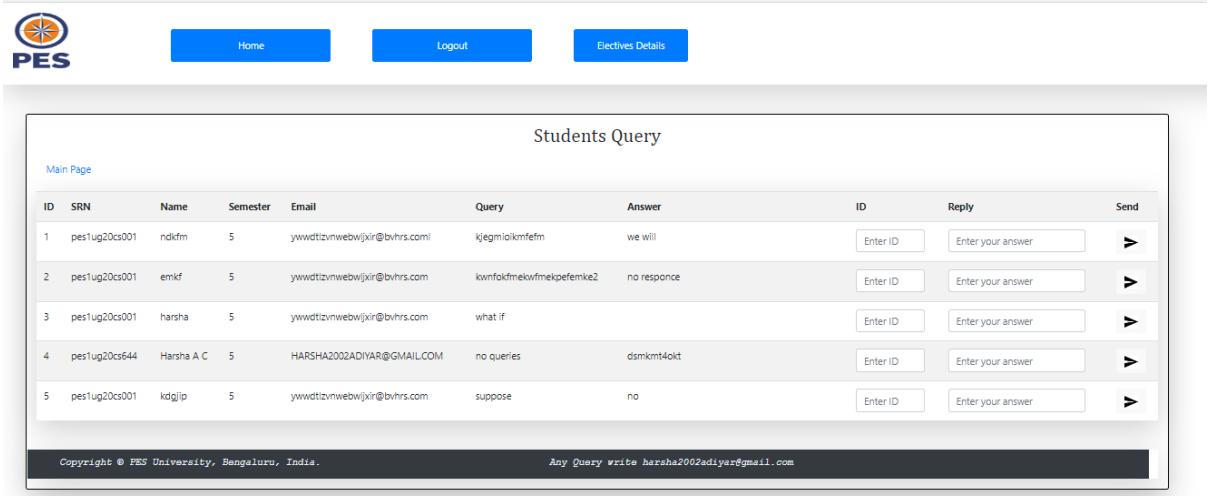
10. Elective Enrollment form



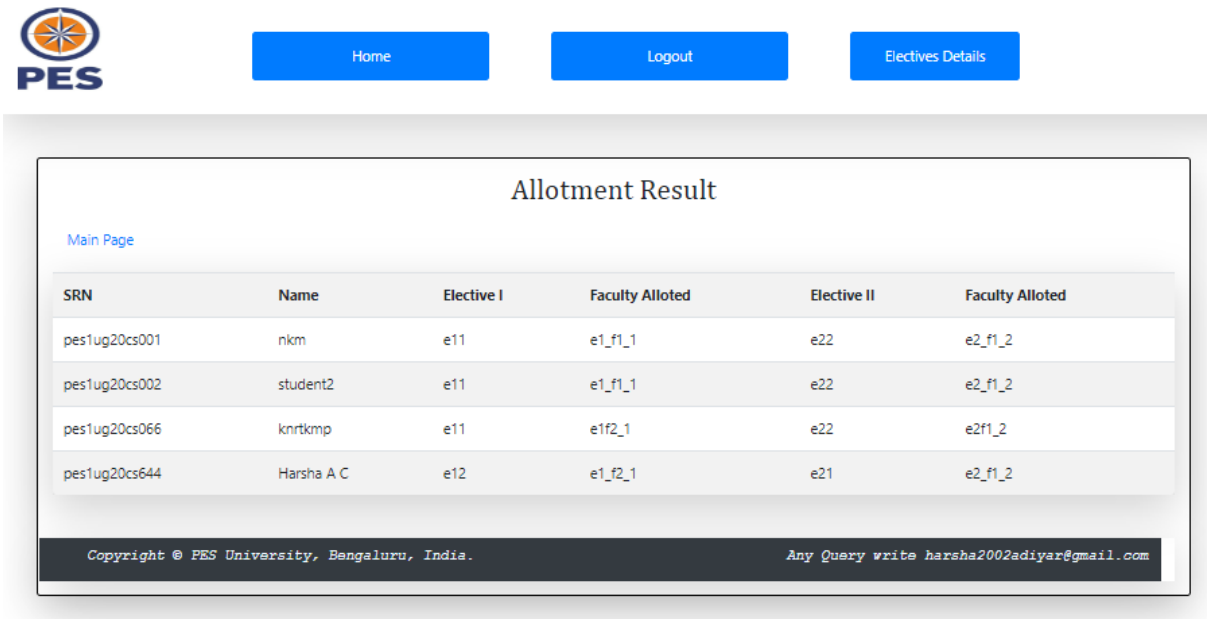
11. Student query form



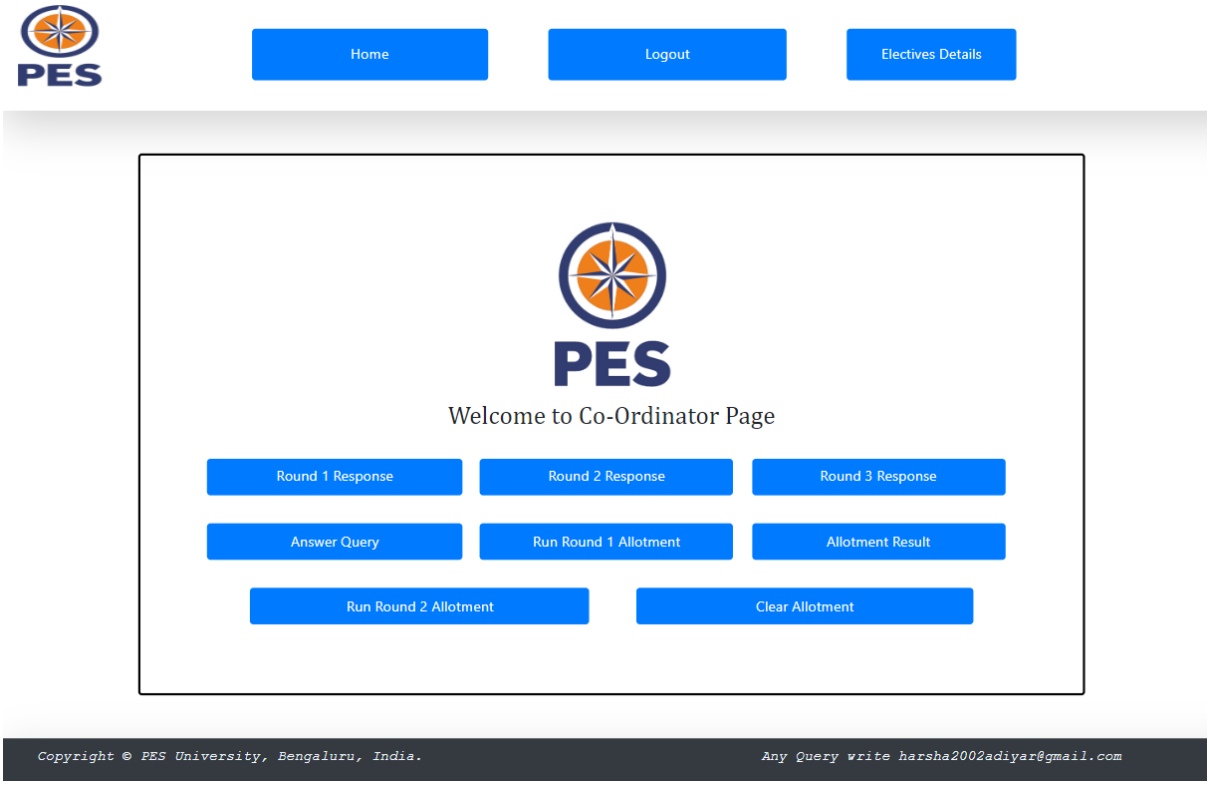
12. Student query



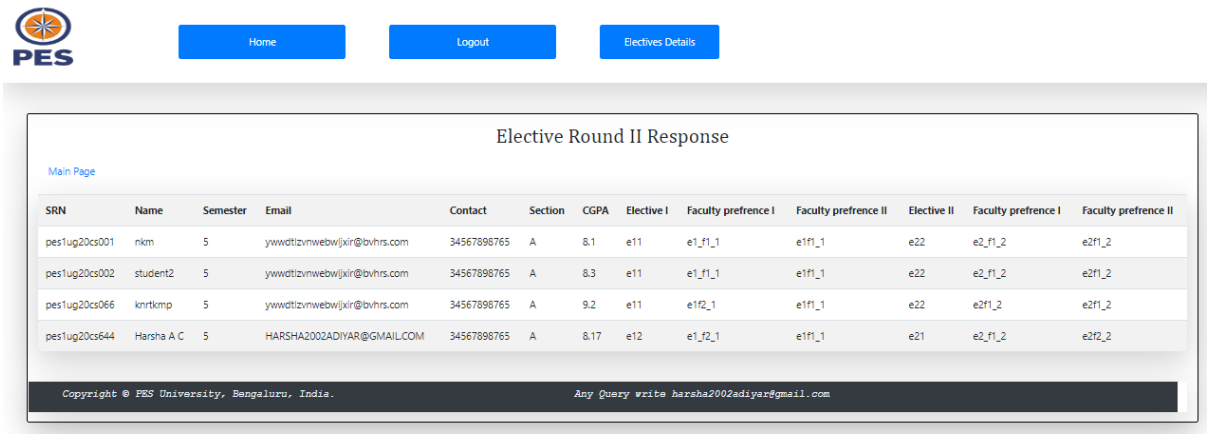
13. Final Allotment



14. Coordinator dashboard



15. Round 2/3 response



OOAD PROJECT REPORT