Database Management Project

Final Report

Group 01

Name :	ID:
Udipta Mazumder	1710138
Tamima Sigma	1630436
Hasan Ahmed	1610266
Misbah Ahmed	1510998
Md. Mahmud Hassan Rabby	1731609
Md Rakib Hossain	1731400
Rakibul Hasan Rajib	1921834

Contents

CHAPTER 1 - INTRODUCTION:

The Independent University, Bangladesh (IUB) has robust and versatile schools - notably consisting of following:

- Business & Entrepreneurship
- Engineering, Technology & Sciences
- Environment and Life Sciences
- Liberal Arts & Social Sciences
- Pharmacy and Public Health.

The university has been an active participant in the growth of the education sector in Bangladesh and produced capable and knowledgeable scholars contributing both here and abroad. [1]

IUB has achieved this through working closely with relevant government education institutions and organizations such as the University Grants Commission (UGC), Ministry of Education, and other necessary institutes for each of the schools, regularly updating its curriculums and putting in a system to monitor student performance based on a quantified approach between course curriculum and standards set by UGC and the Bangladesh government and constantly tracking student performance for every semester – mainly, using Outcome-Based Education (OBE) for monitoring performance and setting university curriculum. [1]

The focus of this report is to study the current student performance monitoring system that IUB uses, do the required analysis of its processes, and propose a new and better improved system that reduces error, makes analysis of data and report generation easier by all vested quarters and produce/show valuable information needed for IUB and its collaborators in making necessary improvements in academia to produce better scholars. The first part focuses on the details of the organization in question and the project that we have undertaken for it. The second part focuses on the existing system and its shortcomings and an introduction of the proposed system that we plan to replace the existing system with. The third and fourth will be heavily technical and focus on how we plan to bring the proposed system into being.

During our research into the existing system for student performance monitoring we have found

many areas where valuable changes could be made to make each process of monitoring student performance faster, make communication between necessary stakeholders easier, take away chances for errors and data duplication, and most importantly make it easier for all stakeholders to easily surf through large datasets to get meaningful information to their requirement.

As we go through this report, we will dig deeper into how the current student performance monitoring system operates, the business processes involved, where there are concerns and issues related to data management, and how we can make a better system to address these issues for fixing and improvement.

A. BACKGROUND OF THE ORGANIZATION- IUB:

Independent University, Bangladesh (IUB), established in 1993, is one of the oldest private universities in Bangladesh, currently has more than an estimation of 7,048 undergraduate and graduate students and over 10,455 alumni. This student population is mostly predicted to grow at 10% annually. [2]

IUB, over-time, has shown remarkable outcomes in producing graduates with marketable skills only because of staying disciplined and up to date with the on-going curriculum and progress system. Dedicating attention towards IUB's Departments, and more specifically focusing the Department of Computer Science and Electrical science into a well-funded research hub running several research projects. IUB is also committed to curve potential graduates of international standard who are mainly equipped to provide new leadership to the national economy through skilled employment, entrepreneurship and/or applied research. This is successful due to the overwhelming support of the Bangladesh Government and the UGC for IUB to be able to create state-of-the-art lab facilities in their department. It is because of IUB's approach to academics as an "Application Oriented Learning" philosophy that "not only teaches students the fundamental principles of learning, situation -handling, and have better overall perception by providing them with hands-on training sessions." [3]

Continuously growing it's lab facilities and flourishing on its curriculum according to current market economic demands, the SECS and the Department of Computer Science and Engineering at IUB has constantly worked with IEB, UGC and the Ministry of Education to track their students'

overall performance under specific periods by quantifying specific courses and its relating assessments into measurable trackers to gain valuable insights for improvement of students over the years as a student in a certain department.

These processes and criteria credentials courses are ultimately set by IEB along with relevant government potentials to set the bar for up-coming graduating engineers from top universities in Bangladesh. These set of standards come in the form of Program Educational Objectives (PEO) and Program Learning Outcomes (PLO) [1] for specific departments in an Accreditation Manual which are mapped to specific courses by relevant Course Instructors and Co-Ordinator's. This allows the Department of CSE at IUB, SECS, IEB and all other relevant stakeholders to have a calculating assessment of the current state-of-affairs and the performance of each student under each course for every semester. This will also allow users to track performance of faculties, courses, departments and schools and provides valuable insight for making necessary improvements.

B. BACKGROUND OF THE PROJECT SPMS 2.0:

Measuring the output of students, faculties, departments, and their respective courses in order to measure their productivity in regard to the outcome relevance of the course activities. Basically, to provide a range of tools and data intended to help universities and education authorities such as IEB, UGC, as well as other stakeholders to evaluate the performance of students and inform strategies for improvements. Developing a national framework for Outcome-Based Education while at the same time leaving considerable freedom to universities in implementing local approaches.

C. OBJECTIVE OF THE PROJECT SPMS 2.0:

The SPMS 2.0 system monitors and summarizes the performances of the stakeholders - students, faculties, schools, and departments through the database of the assessments. For evaluation purposes the system would be able to store individual assessment marks (midterm, quizzes, assignment, projects, presentations and so on). As well as the marks of those assessments with respect to their Course Outcomes (CO) and Program Learning Outcomes (PLO) accordingly in the database of the system to observe the outcome and performance of the student's faculties, schools,

and departments.

The students being the primary stakeholder, would be able to statistically directly monitor the overall performance to their satisfaction of certain course objectives. Hence based on their performances and faculty evaluation the higher stakeholders (Head of department and Admin) can understand and manage the degree in comparison to which different course outcomes targets and their achievements are being understood by the student, department, school, and university body as a whole. SPSMS 2.0 also monitors the impact of policies against overall administrative goals and targets by the system. The system's main target is to monitor the whole university activities through the database and produce analytics for the Head of Department, Faculty, School, Students, and their Courses in a given period of time (yearly and semester wise).

D. SCOPE OF THE PROJECT:

We did a complete analysis of the existing system and found out places in the business processes which can cause severe lapses in time and communication, which we will discuss in the next chapter.

Our solution is to create a Web application, called SPMS 2.0 (Student Performance Monitoring System 2.0), using a Relational Database Management System (RDMS) to store, edit, add, and update necessary data for monitoring student performance and producing and storing related OBE data, reports, and documents.

We produced potential users for the web based SPMS 2.0 system and speculated how they would be using the system and the necessary information and data they would need access to. Since the problems can arise from many points of all business processes, we will make custom user interfaces and login capabilities for all stakeholders who will also be the users of this system.

Since we use a (RDBMS) for data storage, retrieving necessary files, tabular data, page layouts and reports becomes incredibly easy and allows us to interact with the necessary data to occur real-time. We also create interfaces for all users to easily access these data and use them to generate and download reports.

We build an interface for faculties to be able to collaborate with each other on developing course

outlines, course reports, marksheets, assessments, mapping assessments to CO's and PLOs for PLO achievements, and record assessments of students throughout the semester for all their courses.

Students, the IUB leadership team and government agencies can also access the systems for drawing conclusions. Data will also be protected, and each stakeholder will be shown only that data, which is relevant to them, respectively.

CHAPTER 2 - REQUIREMENT ANALYSIS:

The Requirement Analysis is the means of using industry tools, methods, and standards, to research and visualize the current system and the processes that go into the business operation of a certain organization. “Requirements Analysis is the process of determining what the database is to be used for. It involves interviews with user groups and other stakeholders to identify what functionality they require from the database, what kinds of data they wish to process and the most frequently performed operations.” [4]

By doing this we can see each stakeholder and how they interact with each other. We use simple notations and symbols to give anyone the idea of how a business process works and dissect it accordingly.

As we will see, this process of analyzing lets us find out apparent and not so apparent problems with an existing system of monitoring student performance that is manual and depends on involving third party actors and stakeholders causing errors in the system.

A. RICH PICTURE – EXISTING BUSINESS SYSTEM:

A Rich Picture is a way to explore, acknowledge and define a business process and express it through diagrams to create a preliminary mental model. A rich picture helps to open discussion and come to a broad, shared understanding of a situation. [5]The finished rich picture could be of value to other stakeholders of the problems in an existing system, but also allows them to capture many different facets of the situation. Rich pictures concentrate on both the structure and the processes of a given situation. [6]

The Rich Picture Analysis also takes in to account the following:

- Structures
- Processes
- Climate
- People
- Issues expressed by people.
- Conflict

As we can see, this rich picture was prepared keeping exactly those things in mind.

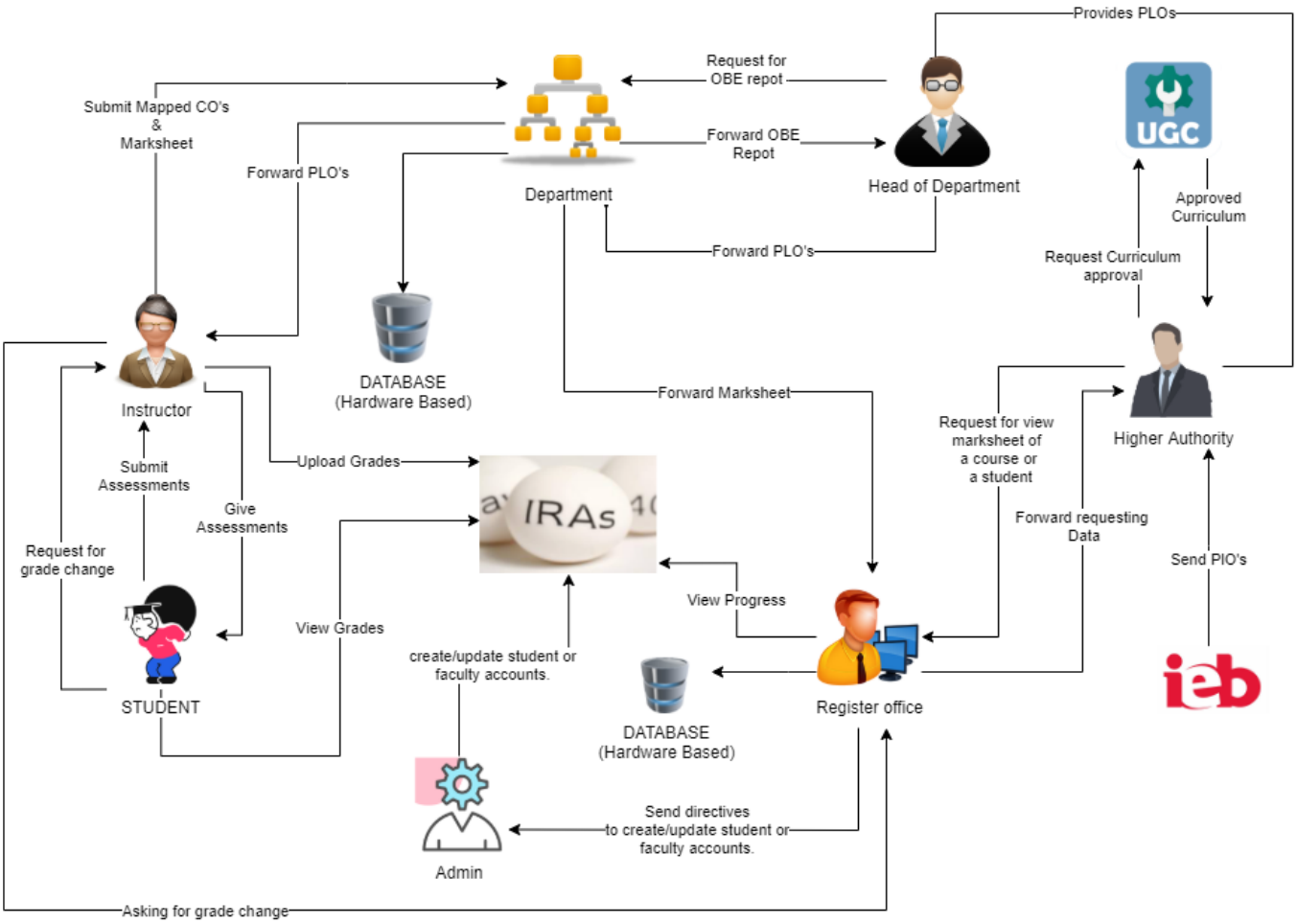


Figure 1.0: Rich Picture of Existing System to Monitor SPMS.

The Rich Picture Analysis shows us that we have the following types of stakeholders:

1. IEB/UGC/Ministry of Education
2. VC/Board of Trustees
3. Head of Department/Dean of School
4. Department (working under Head of Department/Dean of School)
5. Faculty/Course Coordinators
6. Registrar's Office
7. Admin (working under Registrar's Office)
8. Students

We can also identify three separate storage systems or facilities, namely:

1. The Department Storage
2. The Registrar's Office Storage
3. IRAS

From this Rich Picture we have drawn out 7 process that are key to monitoring student performance and improving curriculum. The processes are as follows:

1. Map Course Outcomes (COs) to Program Learning Outcomes (PLOs).
2. Record Student Assessment Data.
3. View Assessment Reports over a given time-period for inspection and analysis of student performance trend.
4. Produce OBE Marksheet & Course Assessment Report.
5. Create student/faculty account and enter/customize necessary data.
6. View Records OBE Marksheets, Course
7. Request for review and change of grades.

B. SIX ELEMENTS ANALYSIS - EXISTING BUSINESS SYSTEM:

The Six Elements Analysis provides a detailed description of the role of each element in each process. It is clear from the table below that Human entities dominate all key functions of this system (especially in the most critical two processes- mapping course outcomes and viewing document related to them.) For example, the current system is heavily dependent on manually processed and handled hardcopy databases. Thus, there is a significantly long chain of waiting between interdependent procedures before the Human elements can fulfill their end of the bargain in the process.

Process	System Roles					
	Human	**Non-Comp Hardware**	**Computing Hardware**	**Software**	**Database**	**Network & communication**
Map Course Outcomes (COs) to Program Learning Outcomes (PLOs)	**IEB/UGC/ Ministry of Education:** 1. Send Accreditation Manual with PLOs defined to Heads of Department/Dean of School. **Head of Department / Dean of School:** 1. Receive Accreditation Manual from IEB. 2. Send the Accreditation manual to Department Staff. 3. Direct Department Staff to tell Course Instructors and Coordinators to design Course Outline and Course Assessment Reports. **Department:** 1. Send Course Instructors the Accreditation Manual with Defined PLOs.	**Pen and paper:** 1. Is used for noting down intermediate Brainstorming ideas. **Board and marker:** 1. Is used for noting down intermediate Brainstorming ideas.	**Computer:** 1. Course Coordinators use computers to make softcopies of Course Outcomes (COs) of the specific courses they are experts in. **Printer:** 1. To print out hardcopies of Course Outcomes (COs).	**MS Word:** 1. Course Coordinators use MS Word to make a detailed course outline and Course Assessment Reports with Course Outcomes (COs) mapping to Program Learning Outcomes (PLOs). **Excel Sheet:** 1. Excel Sheet is used by Course. Coordinators to map specific questions in the Midterm, Final exams, and Project work to specific Course outcomes (COs).		**Internet & Email:** 1. Use the internet and emails to communicate with UGC/IEB or other stakeholders to discuss important topics related to mapping Course Outcomes to Program Learning Outcomes. **Others:** 1. Use phones or physical means with stakeholders to discuss important topics related to mapping Course. Outcomes to Program Learning Outcomes.

	Course Instructor: 1. List course content. 2. List CO's. 3. Map Course Content to Course Outcomes (COs). 4. Map COs to PLOs. 5. Map COs to specific questions of Mid-term, Final Exams questions and Project Work. 6. Starting to design course assessment report using course outline, Course Content and CO's.					
Record Student Assessment Data	**Faculty/ Course Coordinator:** 1. Assign project work and Assignments. 2. Take quizzes and exams throughout the semester. 3. Record assessment data of students throughout the semester of each student for every assessment (quizzes, assignments,	**Pen & Paper:** 1. Use pen & paper to record assessment data and marks obtained on physical paper in tabular Format (hardcopies).	**Computer:** 1. Creating softcopies of records of all assessment data for specific courses are done on Computers.	**Excel Sheet:** 1. Record necessary assessment data and final grades on Excel Sheets. **IRAS:** 1. Upload students' final grades to IRAS for viewing by students or the Registrar's office.	**Department Storage:** 1. Records of students' assessment data and final grades may be saved in the department office and registrar's office for future reference. **IRAS Database server:** 1. IRAS uses a	**Internet:** 1. The Internet is used to communicate with IRAS to Store final grades of students.

	project, exams) on softcopies and hardcopies. 4. Record marks for each specific question in the midterms and final exams. 5. Calculate total marks of quizzes, assignments, and midterm and final exams and assign final grades to each student of specific courses. 6. Convert finals and midterms marks. 7. Bring all the marks of every student for a course into a Marksheet. 8. Grade the student. 9. Upload students' final grades on IRAS. 10. Send the Marksheet to the Department. 11. Send the Marksheet to the Registrar's Office.				database server to store and maintain student grades' information.	

Produce OBE Marksheet & Course Assessment Report	**Faculty:** 1. Calculate total marks received for each CO by calculating the marks received for questions and/or other Assessments mapped to CO's. 2. Calculate total percentages received for each COs on the OBE Marksheet. 3. Declare if a student has achieved a specific CO (if CO percentage is greater than or equal to 40). 4. Declare if a student has received a PLO for a related CO. 5. Make a table giving the verdict and analysis of how many students were able to receive a certain CO and PLO and other documents containing necessary information and data. 6. Design Course Assessment Report	**Pen and Paper** 1. OBE marksheet Stored in hardcopy. Additional markings may be made to further separate between students.	**Computer/ Phone:** 1. Uses computers to make softcopies of the OBE Marksheet and Course Assessment Reports. **Printer:** 1. Print hardcopies of final versions of the OBE Marksheets and Course Assessment Reports.	**Coded Excel sheet:** 1.Faculty/Course Coordinator uses automated excel sheets to calculate the student's success/ failure in Achieving PLOs. **MS Word:** 1. Used to make Course Assessment Report softcopies.	**Department Storage:** 1. Records of students' assessment data and final grades will be saved in the department for future reference. **Registrar's Office Storage:** 1. OBE Marksheets, Course Assessment Reports and other documents submitted by the department is stored for future reference.	**Internet/Mail:** **1.** An Online platform (such as Google Sheets) may be used for processing the OBE assessment data spreadsheet.

	using Course Outline, Course Content and Course Outcomes. 7. Send the final version of the OBE Marksheet to the Dept. Office. **Department Office:** 1. Send the OBE marksheet, Course Assessment Report and others to the Registrar's Office. 2. Store the OBE Marksheet and Course Assessment Report in the department. **Registrar's Office:** 1. Stores the OBE Marksheet and Course Assessment Reports and other documents and reports in the Registrar's Office.					

View grades and download Transcripts	**Students**: 1. Log into IRAS. 2. Search semester wise result for intended semester. 3. See grades for specific semesters. 4. Download transcript through browser into hard disk. **Registrar's Office:** 1. Access IRAS. 2. View students' grades if and when it is necessary. 3. Download their transcripts.	**Pen and Paper** 1. Tabulated transcripts may be printed onto paper. Hardcopy is used as the primary source of truth during applications and other paperwork.	**Computer/ Phone:** 1. Used for accessing IRAS. **Printer:** 1. Used to print the tabulated transcript. Prints tabulated transcripts.	**IRAS:** 1. Store's letter grades of each completed course. 2. Provides the online user interface for viewing grades and transcripts.	**Registrar's Office Storage:** 1. Student information is kept in admin in hardcopies for future reference. **IRAS Database Server:** 1. A Database Management Service is used to store, maintain, edit and receive student grades information in IRAS. **Web Server:** 1. User interface and website pages are served using a remote web server.	**Internet/ Email** 1. The Internet is used to communicate with IRAS to store final grades of students. 2. Softcopies may be mailed.
Create student/faculty account & enter/customize necessary data	**Admin**: 1. New students' information is collected from registration processes. 2. New faculty information is received from HR.	**Pen and Paper:** 1. May be used for writing/ copying student/ faculty's vital login information for account creation.	**Computer:** 1. Used for accessing & adding/editing data to IRAS.	**IRAS:** 1. User interface is provided to interact with student/faculty data.	**Registrar's Office Storage:** 1. Student/ Faculty information is kept in admin in hardcopies for future reference.	**Internet:** 1. The internet is needed to interact with IRAS to store account information on a remote database server. 2. User interface and website

	3. Creates an account for students and faculties. 4. Customize some account details when necessary, for students or faculty.				**IRAS Database Server:** 1. A Database Management Service is used to store, maintain, edit and receive student/faculty information in IRAS. **Web Server:** 1. User interface and website pages are served using a remote web server.	pages are served using internet access.
View Records OBE Marksheets, Course Assessment Reports over a time period for inspection and analysis of student performance trend	**IEB/ UGC:** 1. Inform the university head of a deadline within which OBE Marksheets, Course Assessment Reports and other documents are needed for quality inspection to make necessary improvements to degree programs. 2. Inform the university head if govt.	**Pen and Paper:** 1. May be used for noting/marking down key points of the report. 2. Hardcopies of reports may be used.	**Computer:** 1. Used to display OBE Marksheet and Course Assessment Report's softcopies. 2. Send OBE and Course Assessment Reports to other computers.		**Department Records** 1. Retrieval of OBE marksheets and Course Assessment reports when needed. 2. Stores records on stakeholders interpretation of student performance trends.	**The internet:** 1. OBE marksheets and course assessment reports may be mailed online. 2. Online platforms such as Google Docs/Sheets display reports of softcopies.

	official will visit the campus. 3. Visit university and relevant depts to receive the necessary documents and reports. **Head of Dept/Dean of School:** 1. Request to view records of OBE Marksheets, Assessment Reports to analyze students' performance trends. 2. Direct Department Staff to gather necessary documents, OBE Marksheets, Assessment report for a given time-period specified by govt. officials. 3. Receive the necessary documents gathered by the dept. 4. Evaluate the need to change/ improve the department's educational resources					

	based on students' performance trends. **VC/Board of Trustees:** 1. Request to view records of OBE Marksheets, Assessment Reports to analyze students' performance trends. **Departmental Staff:** 1. Gather necessary OBE Marksheets, Assessment Reports & other documents. 2. Provide all the necessary documents to govt. officials.					
Request for review and change of grades.	**Students:** 1. Request for grade change and review to faculty. **Faculty/ Course Coordinator:** 1. Check exam papers and other assessments upon request. 2. If change needs to be made, send a grade change request	**Pen and Paper:** 1. May be used to note down key points or marks on the students' answer sheets.	**Computer/ Phone:** 1. Used for communicating with the faculty.	**IRAS**: 1. Used by the admin for changing the grade.	**IRAS server:** 1. Update student grade data. **Department Storage:** 1. Update student grade data. **Registrar's Office Storage:** 1. Update student grade data.	**Internet:** 1. Email is primarily used for communication. **Phone:** 1. May be used for communication.

	of a specific student to admin if not, end the process. **Admin:** 1. Receive a request to change the grade of a specific student. 2. Change grade of student based on Faculty request.					

C. PROCESS MODEL – EXISTING BUSINESS SYSTEM:

Business Process Model and Notation (BPMN) is a graphical representation for specifying business processes in a business process model. [7] We use business process model diagrams to dissect each of the business processes mentioned in the previous section.

Each diagram separates the stakeholders involved in the processes, the exchanges among them and the decisions each of them need to make.

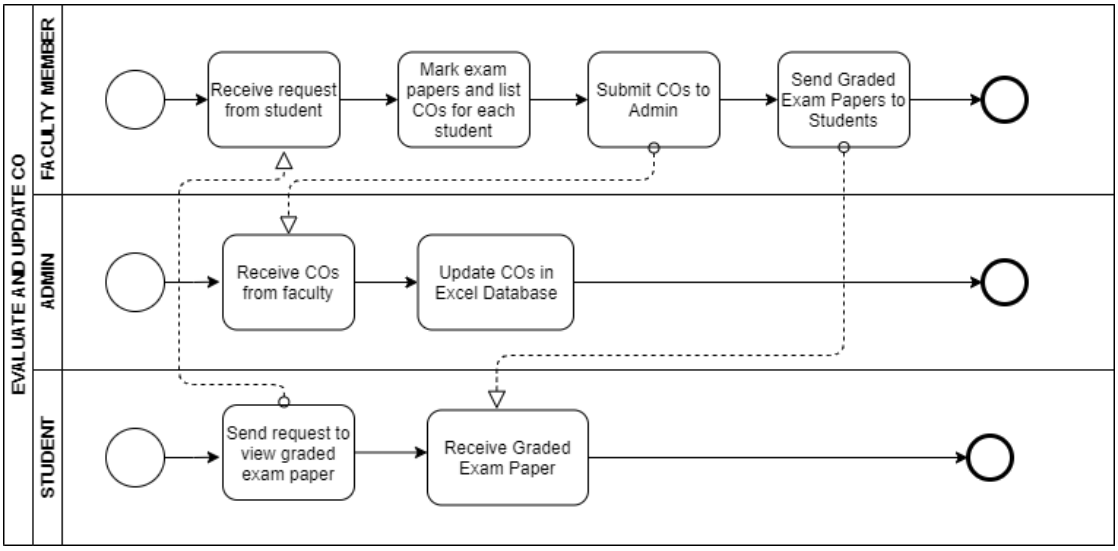


Figure 1.1: Evaluate and update CO.

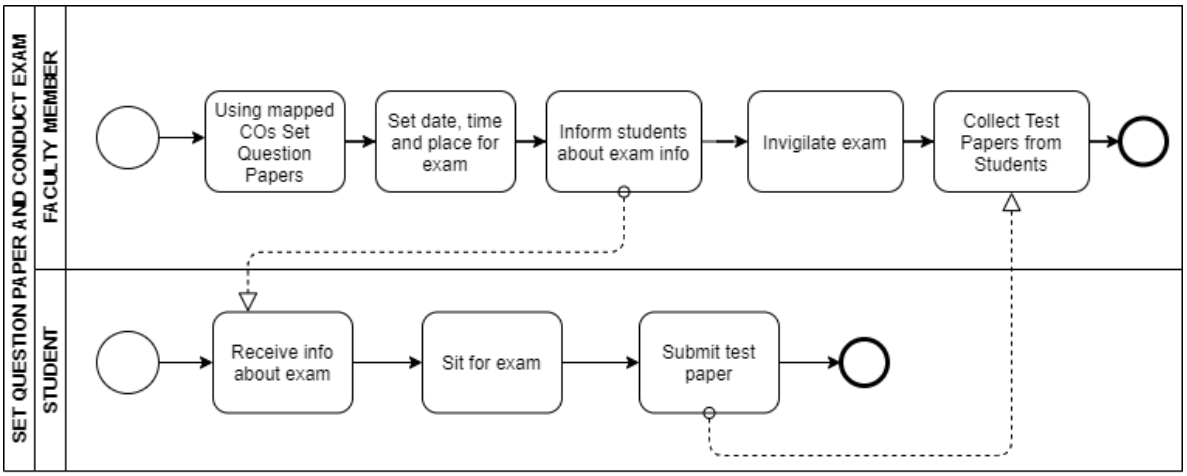


Figure 1.2: Set question paper and conclude exam.

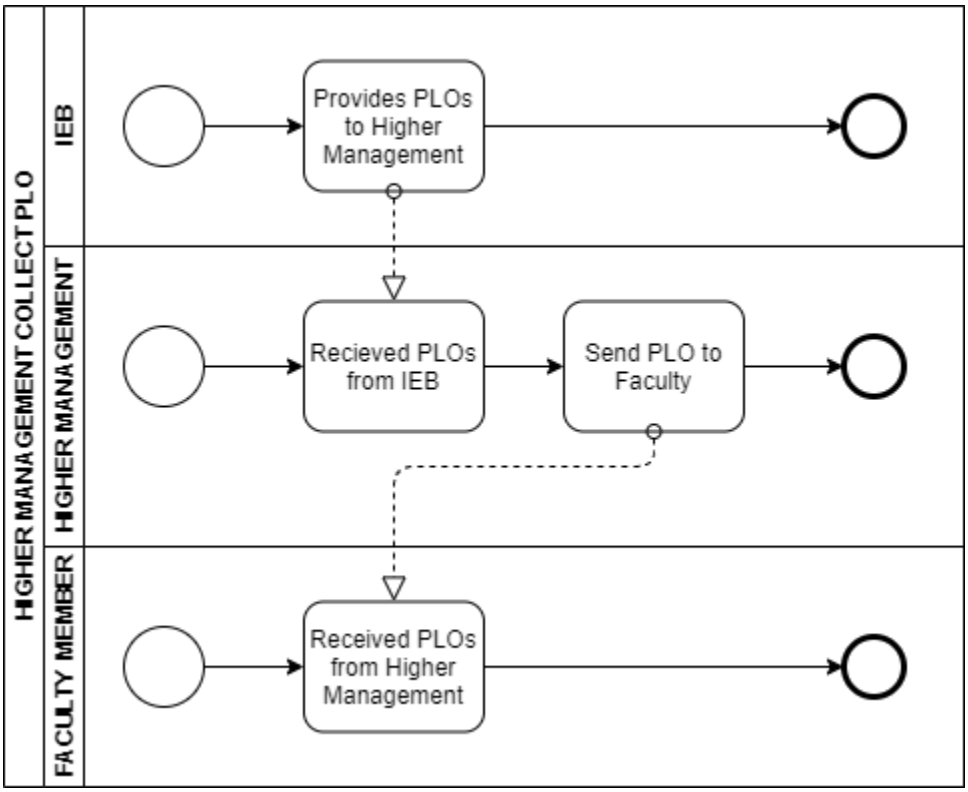


Figure 1.3: Higher management collect PLO.

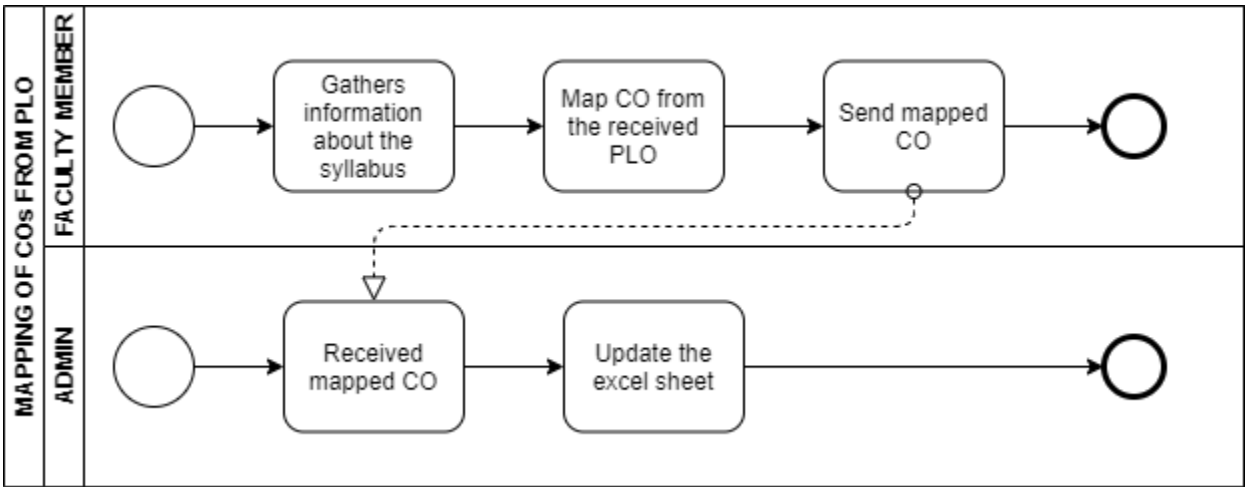


Figure 1.4: Mapping of CO from PLO

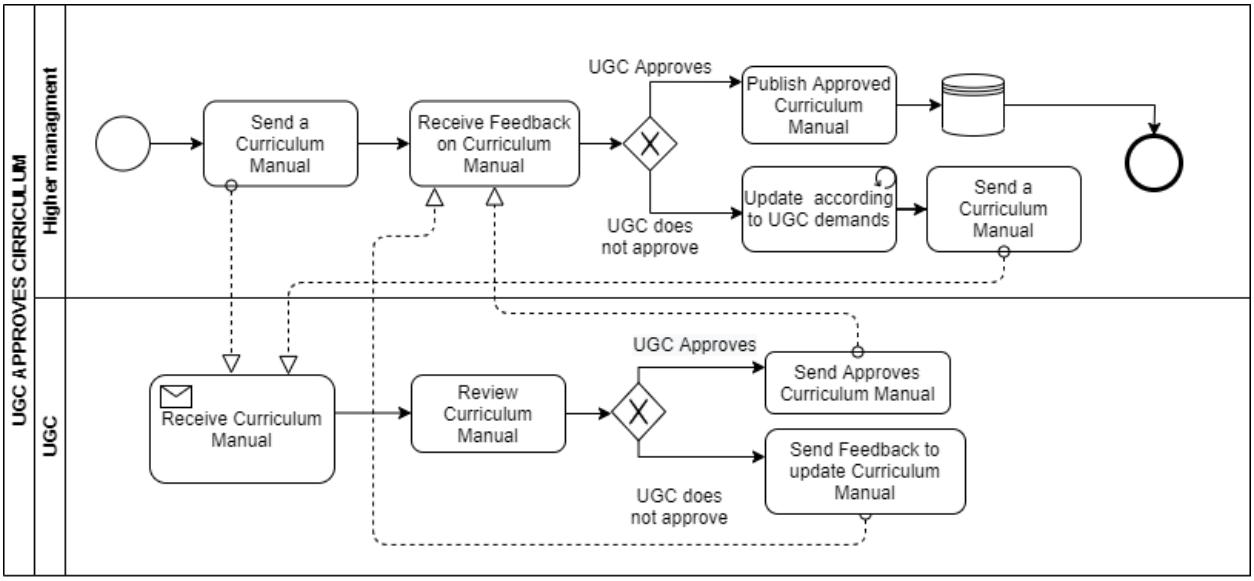


Figure 1.5: UGC approves curriculum.

D. PROBLEM ANALYSIS – EXISTING BUSINESS SYSTEM:

Based on the existing systems' Six Elements Analysis, the shortcomings in each process were identified. There is a repeating pattern in the far-right column of this table. It appears that the facilitation of a private online platform will improve the system in many ways.

Process Name	Stakeholders	Concerns(Problems)	Analysis (Reason of the Problems)	Proposed Solution
Student Enrollment	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	School-wise, department-wise, and program-wise comparison of students have enrolled in each department with respect to a given period of time/semesters.	Student enrolled stats is recorded School, department and program-wise but was never compared with respect to time period/semester.	We want to keep the in the count of students enrolled along with a visual comparison of the student stats as per school-wise, department-wise, and program-wise and semester-wise.
Student performance based on CGPA	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	School-wise, department-wise, and program-wise student performance trends based on CGPA with respect to a given period of time/semesters.	Students and other mentioned stakeholders have been able to only observe the CGPA status that gets updated every semester individually.	Our system should be allowing the users to statistically analyze the CGPA progress of the students not only on individually but also based on schools, department, and program with respect to a given period of time/semesters.
Course-wise student performance based on GPA.	1. Student 2. Department Head 3. Registrar's office 4. Faculty	Course-wise (for a selection of courses) student performance trend based on GPA with respect to a given period of	The GPA of the students were used as verdicts only and never visualize into course-wise	Through the software application the Stakeholders would be able to select the course

	5. Dean 6. VC	time/semesters.	student's performance based of their GPA.	and view performance trend depending on the GPA with respect to a given period of time/semesters.
Selective Number of Instructor-wise student performance based on the GPA of the students	1. Department Head 2. Registrar's office 3. Faculty 4. Dean 5. VC	Instructor-wise (for a selection of instructors) student performance trend based on the GPA of the students in that courses taught by each of the instructors so far with respect to a given period of time/semesters.	Higher Authorities have been unable to observe the statistics of their selective faculties performances all together based on the GPA of the students.	The SPMS2.0 system would allow to record the GPA of the students taught by the selective number of faculties. Storing and converting the data to appropriate graphical forums and measure performance of the instructors with respect to a given period of time/semesters. with respect to a given period of time/semesters.
VC-wise, dean-wise, or department head-wise student performance	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	VC-wise, dean-wise, or head-wise student performance trend based on the GPA of the students under the school/program corresponding to the leadership team.	Higher authority (VC/Dean and Department Head) was unable to view VC, Dean or Department Head-wise student's performance under school/program.	The system would Will be able to visualize the performance of the students based on VC, Dean and Department-head.
Instructor-wise student performance	1. Department Head 2. Registrar's	Instructor-wise student performance trend for a chosen	Higher authorities were not able to monitor Instructor	The SPMS2.0 system would allow the

based on the GPA of the students	office 3. Faculty 4. Dean 5. VC	course with respect to a given period of time/semesters.	performance for a selected number of faculty based on the GPA of the students they have taught.	stakeholders to record the GPA of the students taught by the selective faculty. Storing and converting the data to appropriate graphical forms and measure performance of the instructors with respect to a given period of time/semesters.
Total PLO percentage achieved and attempted by the student along with the departmental average	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	PLO total percentage score for each PLO calculated from the scores achieved in each CO associated with the corresponding PLO among all the courses the student has done so far, along with the departmental average performance for comparison. Also, for each PLO, what percentage of it was achieved from each of the courses associated with the corresponding PLO, and what percentage was achieved via each of all the COs associated with the corresponding PLO. All of this for a chosen school, program, or department.	The PLO and corresponding CO for all the courses the student has done so far is never compared cumulatively along the departmental average performance.	The system will provide the total of all PLO percentage corresponding to CO and calculate the score for all the courses a student has done for a chosen school, program, or department.

PLO achievement	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	PLO achievement of a student for each of the courses taken so far.	Students are unable to monitor progress of their PLO achieved for respective courses as it only available to the faculties and has access to rest of the higher authorities.	Record and tabulate the number of PLO's achieved by the student for individual course taken and completed so far.
Comparison of PLO-achieved percentage versus PLO-attempted	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	Comparison of PLO-achieved percentage versus PLO-attempted percentage.	Students are unable to compare progress of their PLO achieved vs PLO they should be aiming for with respect to courses they have done as it only available to the faculties and is analyzed manually and can be extremely time consuming.	The system would allow the students and rest of the stakeholders to monitor automatically using relational data model using proper SQL operations- their PLO achieved vs attempted comparisons individually.
Expected PLO-achievement versus actual score (For course's, student's, department's, program's, or school's)	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	Comparison of a course's, student's, department's, program's, or school's expected PLO-achievement versus actual with respect to a given period of time/semesters.	The existing system allows to calculate manually and does not provides adequate information for comparisons of PLO. The verdict is filled up in an Excel sheet and is time consuming for the stakeholders to reach to respective faculties or	SPM software would allow the stakeholders to monitor automatically (login into the system) their PLO achieved vs attempted comparisons for course's, program's, departments, and school with respect to a given period of time/semesters.

			department head for OBE mark sheet.	
CO-PLO achievement summary	1. Student 2. Department Head 3. Registrar's office 4. Faculty 5. Dean 6. VC	Summary of CO-PLO achievement stats for a chosen course, program, department, school.	The existing system by far was abled the higher authorities only to track CO and PLO achieved for a course manually only.	SPM in a table will provide PLO-CO achievement stats to the stakeholders to choose for course wise, program, department, and school wise.

E. RICH PICTURE - PROPOSED SYSTEM:

The Course Outcomes (COs) and Program Learning Outcomes (PLOs) will be visible in a new system, an online platform called SPMS, where it will have its own database that host the data of all the courses, faculties, as well as updated tables every semester to keep track of which courses have been assigned to which faculties in a given semester. We are making the new system (to track student performance, but also to track faculties teaching a specific course or the performance of students in a course over a period) and why it is hard to track these trends and data right now. Briefly, we can see that the SPMS relational database (a non-human) quite literally plays a significant role in the student performance monitoring system. Also, this entity holds the greatest number of interconnections between all other processes.

We will use different user interfaces designed for specific user needs based on the concerns and problems we found in the problem analysis. The Head of the Department/Dean of School, Course Instructor/Coordinator/Faculty, Admin, Student, IEB/UGC/Ministry of Education, VC/Board of Trustees, Department Staff, all these stakeholders mentioned will have access to view the report of a student.

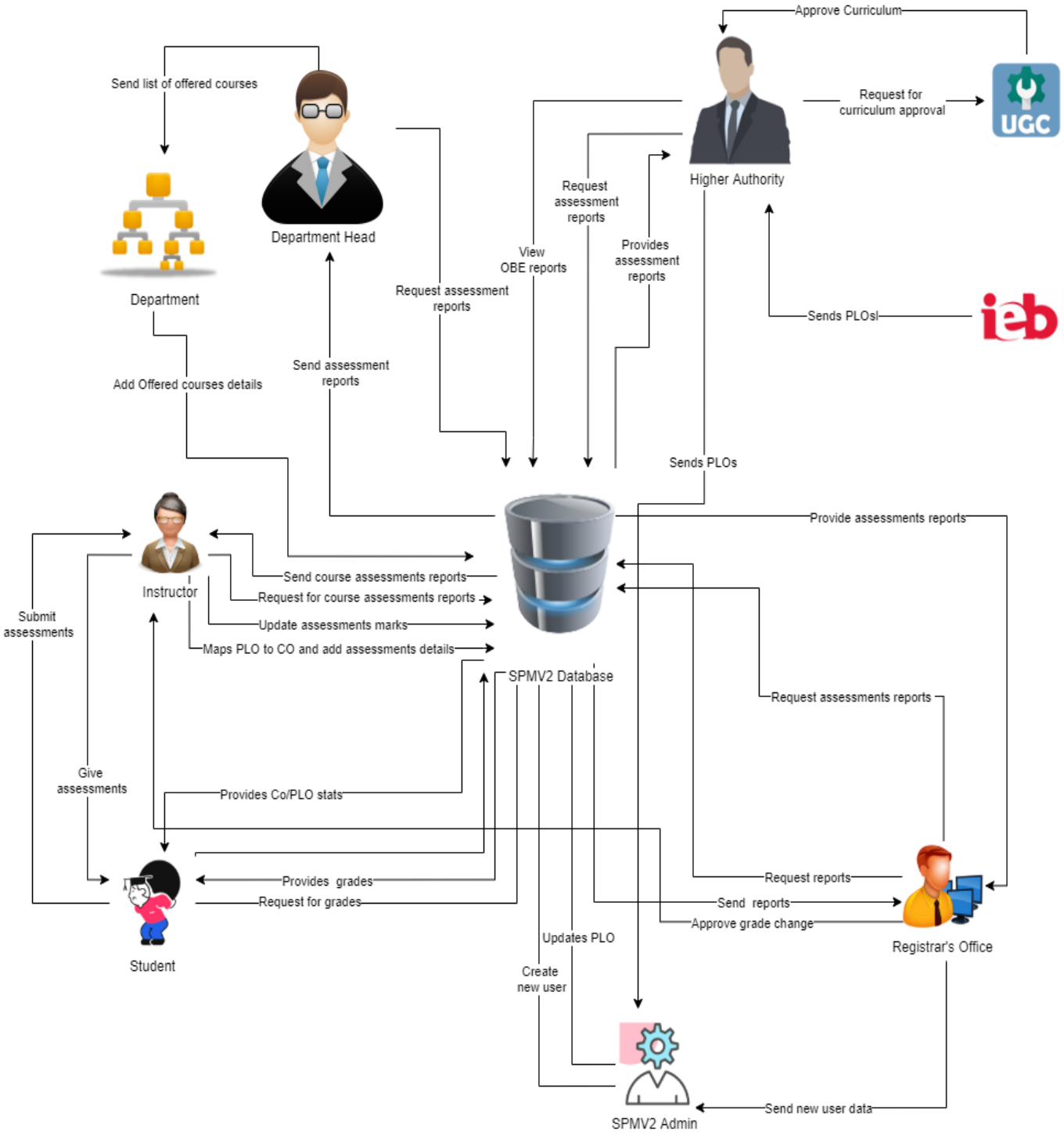


Figure 1.6: Rich Picture of Proposed System to Monitor Student Performance.

F. SIX ELEMENTS ANALYSIS – PROPOSED SYSTEM :

The six elements analysis of the proposed system is a continuation of an analysis process where each analysis is based on the one that comes before it. Based on the rich picture, the role of each element in the new system is further understood in the table below.

Process	System Roles					
	Human	**Non-Computing Hardware**	**Computing Hardware**	**Software**	**Database**	**Network and Communication**
Student Enrollment	**Student** a) Goes to the website b) Clicks on the form option. c) Fills the form with required information **Registrar Office** a) Checks and verifies student enrollment information from the forms from the website or hardcopy forms b)Registrar Office's Admin logs into the system using adminID	**Paper and Stationary** a) Used to collect information in forms from Students.	**Computer/ Laptop** a) SPMS2.0.0 admin will use Computers to access and update data. b) Users will use the computer to view the data. **Database Server** a) Used by SPMS2.0 Developers to collect data and maintain the software. **Networking Devices (Router, Switch, Bridge, Hub):** a) Used to access SPMS 2.0	**Operating Software** Used by Registrar Office and SPMS2.0.0 **Student** Uses to fill the form when filling the form from the website. **SPMS2.0** The software for which the admin will create accounts.	**Register Office Database** Used By the registrar office to collect the student information in a excel file to send it to SPMS2.0. **SPMS2.0** Information is stored in the Database for New user Account or any other updates. **Excel** Student account data may be stored in excel file for later usage in SPMS2.0.	**Internet** a) It is used to access and store data to SPMS2.0. b) Used to collect the student form from the student to registrar office. c) Used by the Registrar Office to send all the student information to SPMS2.0 Admin.

	and password. c)Sends verified student informatio n as an attachment to SPMS2.0 Admin/Tea m. **SPMS2.0 Admin:** a)SPMS2.0 Admin logs into the system using SPMS2.0 userID and password. b) Receives the student enrollment informatio n in the attached files. c)Admin updates the student enrollment informatio n in SPMS2.0 Database. d)Notifies respected Stakeholde rs **Departme nt Head** a)Logs into the system					

	using their UserID and password. b)Inputs the desired time-period for number of students enrolled. **Higher Authority (VC/Dean)** a)Logs into the system using their UserID and password. b)Inputs the desired time-period and compare School/Department for the number of students enrolled accordingly **Faculty** a)logs into the system using FacultyID and password b)Inputs the ID of the section the faculty is taking to view the					

	students enrolled.					
Student Performance Based on CGPA	**Student** a)Logs into the System using StudentID and password. b) Inputs the desired time-period to view self CGPA progress. **Department Head:** a)Logs into the System using UserID and password. b) Inputs the desired time-period and School, Department or Program c)View statistically analyzed CGPA trend of students or any Individual Student. **Registrar's office:**		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer** a)Used to print out the report if need be. **Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.	**Operating Software** Used by the user to run SPMS2.0 **SPMS2.0** a)The software will generate a performance trend.	**SPMS2.0 Database** a)Use the database to obtain the performance.	**Internet** a)It is used to login into and access the SPMS2.0

	a)Logs into the System using userID and password. b) Inputs the desired time-period and School, Department or Program to view statistically analyzed CGPA trend of students. **Faculty** a)Logs into the System using FacultyID and password. b) Inputs the desired time-period and Program to view statistically analyzed CGPA trend of students or any Individual Student those who attended the faculty's section. **Higher Authority**					

	(Dean/VC) a)Logs into the system using their UserID and password. b)Inputs the desired time-period, School and Department c)View statistically analyzed CGPA trend of students.					
Course-wise student performance based on GPA	**Student** a)Logs into the System using StudentID and password. b) Inputs the course c)View self GPA for the course. **Department Head** a)Logs into the System using UserID and password. b) Inputs the desired time-period courseID c)View		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer** a)Used to print out the report if need be. **Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.	**SPMS2.0** a)The software will generate a performance trend based of GPA	**SPMS2.0 Database** a)The performance will be stored and updated here	**Internet** a)It is used to login into and access the SPMS2.0

	statistically analyzed GPA trend of students **Registrar's office** a)Logs into the System using adminID and password. b) Inputs the desired time-period and coursed c) view statistically analyzed GPA trend of students. **Faculty** a)Logs into the System using FacultyID and password. b) Inputs the desired time-period CourseID under the faculty c)view statistically analyzed GPA trend of students who were in that					

	faculty's section. **Higher Authority (Dean/VC)** a)Logs into the system using their UserID and password. b)Inputs the desired time-period and CourseID c)View statistically analyzed GPA trend of students for that specific course.					
Selective Number of Instructor-wise student performance based on the GPA	**Department Head** a)Logs into the System using UserID and password. b) Inputs the desired time-period courseID c)View statistically analyzed GPA trend of students for a selective number of Instructors.		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer** a)Used to print out the report if need be. **Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.	**SPMS2.0** a)The software will generate a performance trend for a selective instructor wise.	**SPMS2.0 Database** a)The performance will be stored and updated here.	**Internet** a)It is used to login into and access the SPMS2.0

	Registrar's office a)Logs into the System using AdminID and password. b) Inputs the desired time-period courseID c)View statistically analyzed GPA trend of students for a selective number of Instructors. **Faculty** a)Logs into the System using FacultyID and password. b) Inputs the desired time-period & courseID c)View statistically analyzed GPA trend of students for a selective number of Instructors. **Higher**					

	Authority (Dean/VC) a)Logs into the System using UserID and password. b) Inputs the desired time-period courseID c)View statistically analyzed GPA trend of students for a selective number of Instructors.					
VC-wise, dean-wise, or departmen t head-wise student performa nce	**Department Head** a)Logs into the System using UserID and password. b)Select Input from from VC/Dean/De partment Head c)View the student performance trend as per choice. **Registrar's office** a)Logs into the System		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer** a)Used to print out the report if need be. **Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.	**SPMS2.0** a)The software will generate a performanc e trend.	**SPMS2.0 Database** a)The performance will be stored here.	**Internet** a)It is used to login into and access the SPMS2.0

	using UserID and password. b)Select Input from from VC/Dean/Department Head c)View the student performance trend as per choice. **Dean/ VC** a)Logs into the System using UserID and password. b)Select Input from from VC/Dean/Department Head c)View the student performance trend as per choice.					
Instructor-wise student performance based on the GPA of the students	**Department Head** a)Logs into the System using DepartmentID and password.		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer**	**SPMS2.0** a)The software will generate a performance trend.	**SPMS2.0 Database** a)The performance will be stored and updated in the database.	**Internet** a)It is used to login into and access the SPMS2.0

	b)Inputs a particular instructor Name/ID c)View the student performance trend of selected instructor. **Registrar's office** a)Logs into the System using UserID and password. b)Inputs a particular instructor c)View the student performance trend of selected instructor. **Faculty** a)Logs into the System using UserID and password. b)Input their Name/ID. c)View the student performance trend. **Dean**		a)Used to print out the report if need be. **Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.			

	a)Logs into the System using UserID and password. b)Inputs a particular instructor c)View the student performance trend of selected instructor. **VC** a)Logs into the System using UserID and password. b)Inputs a particular instructor c)View the student performance trend of selected instructor.					
Total PLO percentage achieved and attempted by the student along with the departmental average	**Student** a)Logs into the system using Student ID and Password b)Inputs the time period c)Views their comparison of attempted vs achieved		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer** a)Used to print out the report if need be.	**SPMS2.0** a)The software will generate a comparison of attempted vs achieved PLO as well as the departmental average.	**SPMS2.0 Database** a)The performance will be stored here.	**Internet** a)It is used to login into and access the SPMS2.0

	PLO percentage along with the departmental average. **Department Head** a)Logs into the system using User ID and Password b)Inputs the time period c)Views the comparison of students attempted PLO vs achieved PLO percentage along with the departmental average. **Registrar's office** a)Logs into the system using User ID and Password b)Inputs the time period c)Views the comparison of students attempted		**Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.	**Operating system** Used by the SPMS2.0		

	PLO vs achieved PLO percentage along with the departmental average. **Faculty** a)Logs into the system using User ID and Password b)Inputs the time period c)Views the comparison of students attempted PLO vs achieved PLO percentage along with the departmental average. **Dean** a)Logs into the system using User ID and Password b)Inputs the time period c)Views the comparison of students attempted					

	PLO vs achieved PLO percentage along with the departmental average. **VC** a)Logs into the system using User ID and Password b)Inputs the time period c)Views the comparison of students attempted PLO vs achieved PLO percentage along with the departmental average.					
PLO achievement	**Student** a)Logs into the System using studentID and password.		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer**	**SPMS2.0** a)The software will generate PLO achievement.	**SPMS2.0 Database** a) The performance will be stored and updated here.	**Internet** a)It is used to login into and access the SPMS2.0

	b) Selects PLO achievement c) view PLO achievement **Department Head** a)Logs into the System using userID and password. b) Selects PLO achievement c) view PLO achievement **Registrar's office** a)Logs into the System using userID and password. b) Selects PLO achievement. c) view PLO achievement. **Faculty** a)Logs into the System using facultyID and password. b) Selects PLO achievement		a)Used to print out the report if need be. **Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.			

	c) view PLO achievement **Dean** a)Logs into the System using userID and password. b) Selects PLO achievement c) view PLO achievement **VC** a)Logs into the System using userID and password. b) Selects PLO achievement c) view PLO achievement					
Expected PLO-achievement versus actual score (For course's, student's, department's, program's, or school's)	**Student** a)Logs into the System using studentID and password. b) Selects PLO achievement comparison c) view PLO achievement comparison		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer** a)Used to print out the report if need be. **Networking Devices (Router, Switch,**	**SPMS2.0** a)The software will generate the expected vs achieved PLO.	**SPMS2.0 Database** a)The performance will be stored and updated in the database.	**Internet** a)It is used to login into and access the SPMS2.0

	Department Head a)Logs into the System using userID and password. b) Selects PLO achievement comparison c) view PLO achievement comparison **Registrar's office** a)Logs into the System using userID and password. b) Selects PLO achievement comparison c) view PLO achievement comparison **Faculty** a)Logs into the System using facultyID and password. b) Selects PLO achievement comparison c) view PLO achievement		**Bridge, Hub):** a)Used to access the Internet.			

	comparison **Dean** a)Logs into the System using userID and password. b) Selects PLO achievement comparison c) view PLO achievement comparison **VC** a)Logs into the System using userID and password. b) Selects PLO achievement comparison c) view PLO achievement comparison					
CO-PLO achievement summary	**Student** a)Logs into the System using studentID and password. b) Selects CO-PLO achievement summary c) view CO-PLO		**Computer/ Laptop** a)User will need a computer to access SPMS2.0 **Printer** a)Used to print out the report if need be**.**	**SPMS2.0** a)The software will generate the summary of CO-PLO achievement	**SPMS2.0 Database** a)The Summary will be stored and updated in the database.	**Internet** a)It is used to login into and access the SPMS2.0

	achievement summary. **Department Head** a)Logs into the System using userID and password. b) Selects CO-PLO achievement summary c) view CO-PLO achievement summary. **Registrar's office** a)Logs into the System using userID and password. b) Selects CO-PLO achievement summary c) view CO-PLO achievement summary **Faculty** a)Logs into the System using facultyID and password. b) Selects		**Networking Devices (Router, Switch, Bridge, Hub):** a)Used to access the Internet.			

	CO-PLO achievement summary c) view CO-PLO achievement summary. **Dean** a)Logs into the System using userID and password. b) Selects CO-PLO achievement summary c) view CO-PLO achievement summary **VC** a)Logs into the System using userID and password. b) Selects CO-PLO achievement summary. c) view CO-PLO achievement summary.					

G. PROCESS MODEL - PROPOSED SYSTEM:

After understanding the role of each element in each process, the Business process model and notation provides an unambiguous dictation of the exact sequence of steps that will follow to fulfill each process. Every module of this diagram will serve as a high-level starting point for deriving the implementation details in the later chapter.

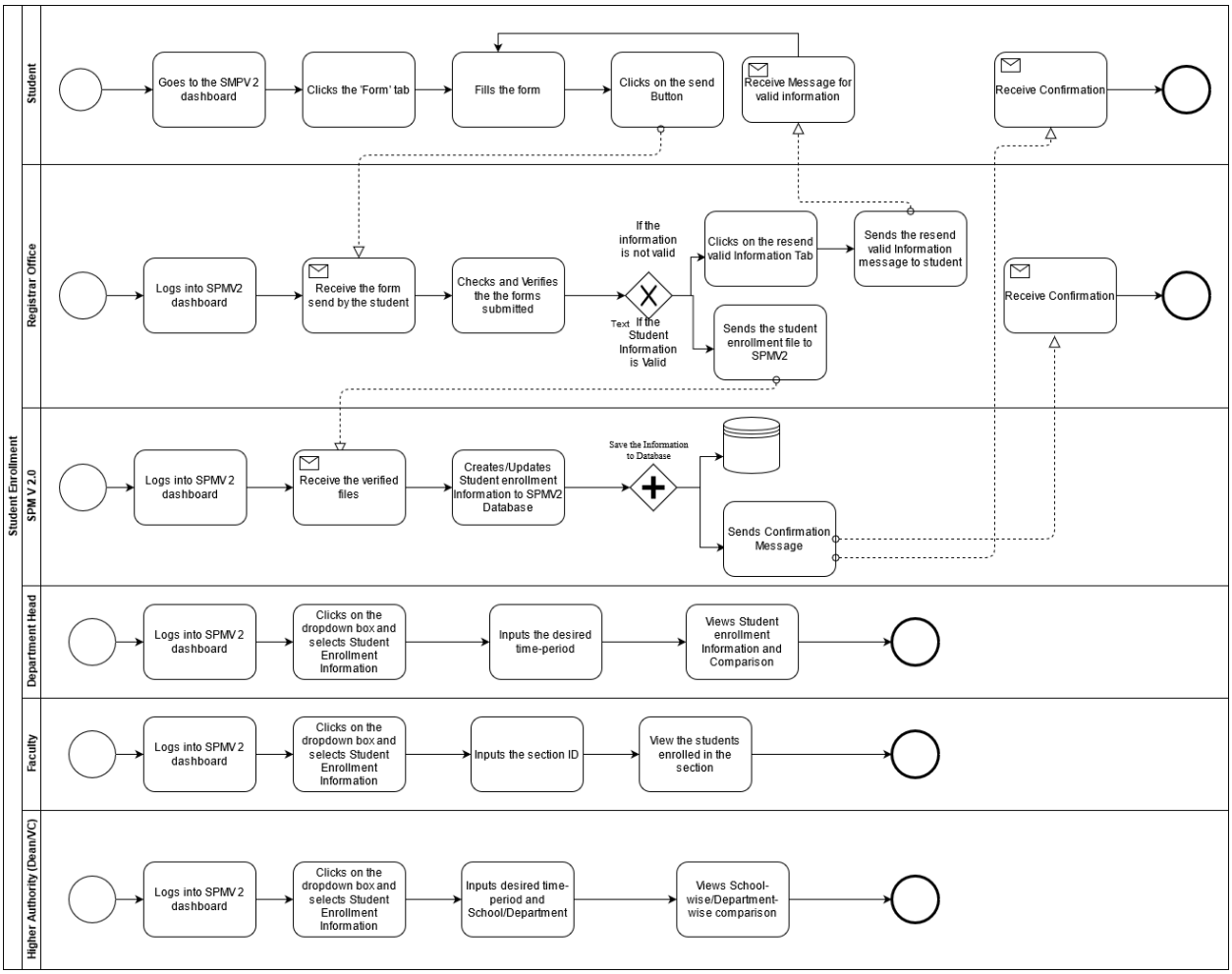


Figure 1.7: Student Enrolment

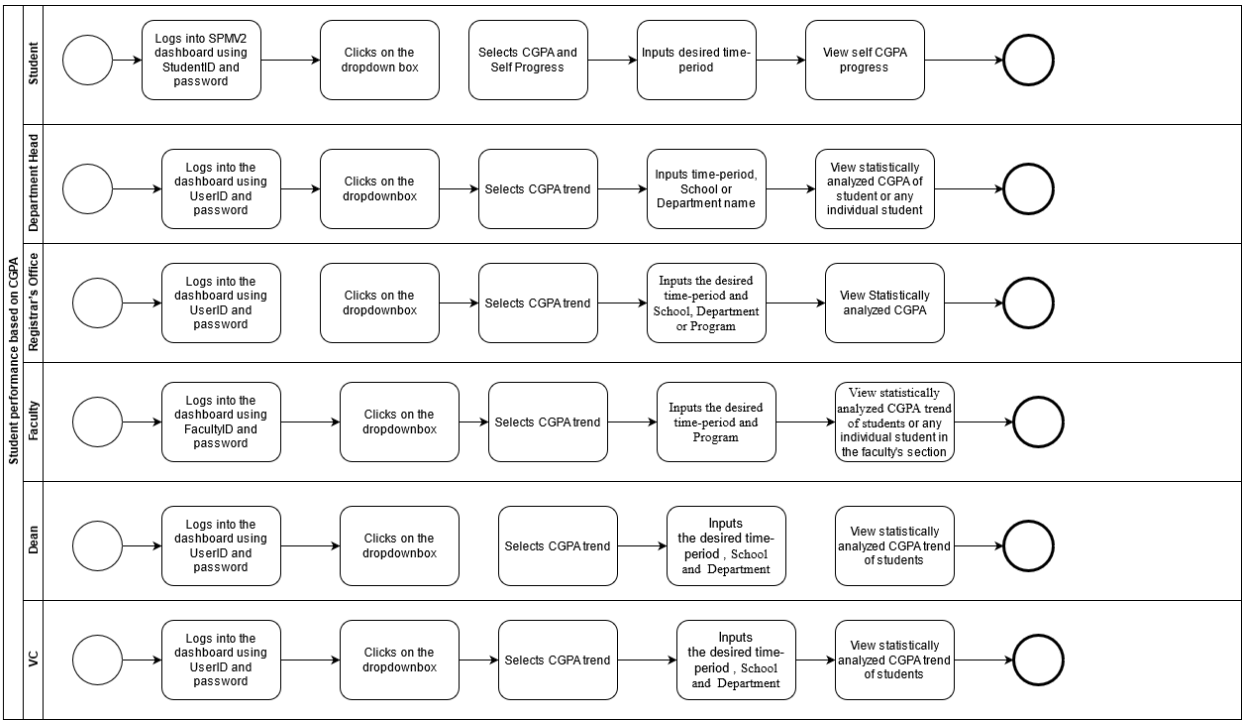


Figure 1.8: Student Performance based on CGPA

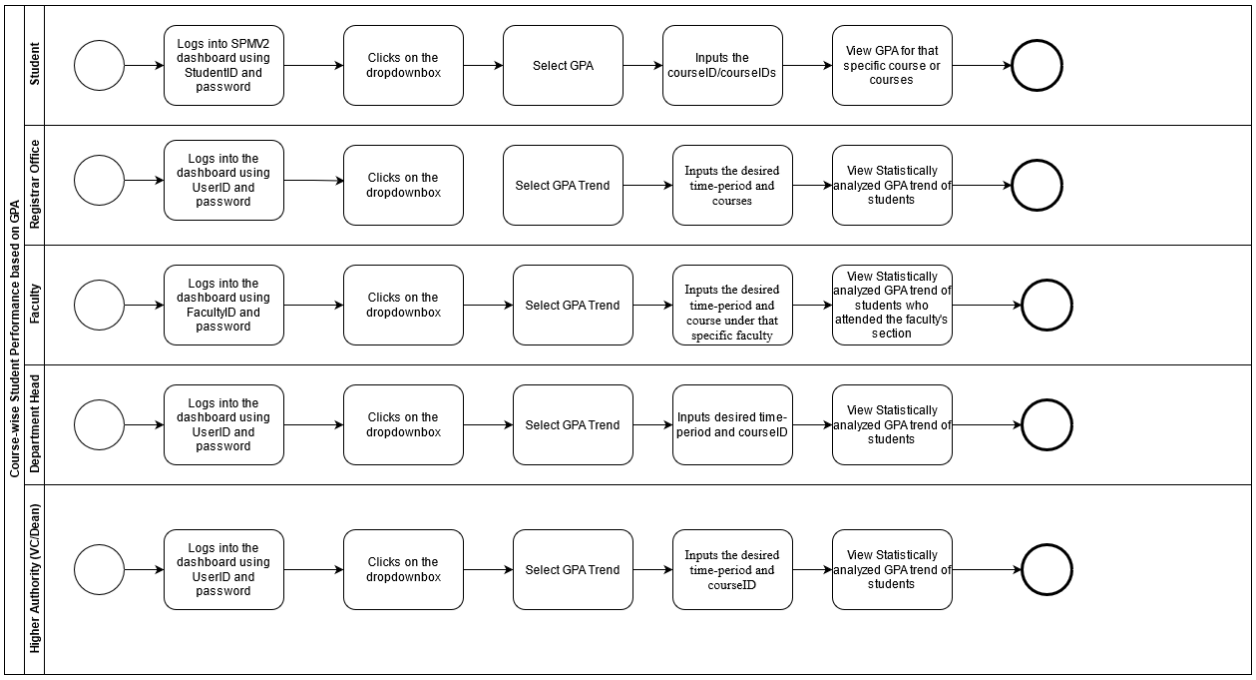


Figure 1.9: Course-wise Student Performance based on GPA

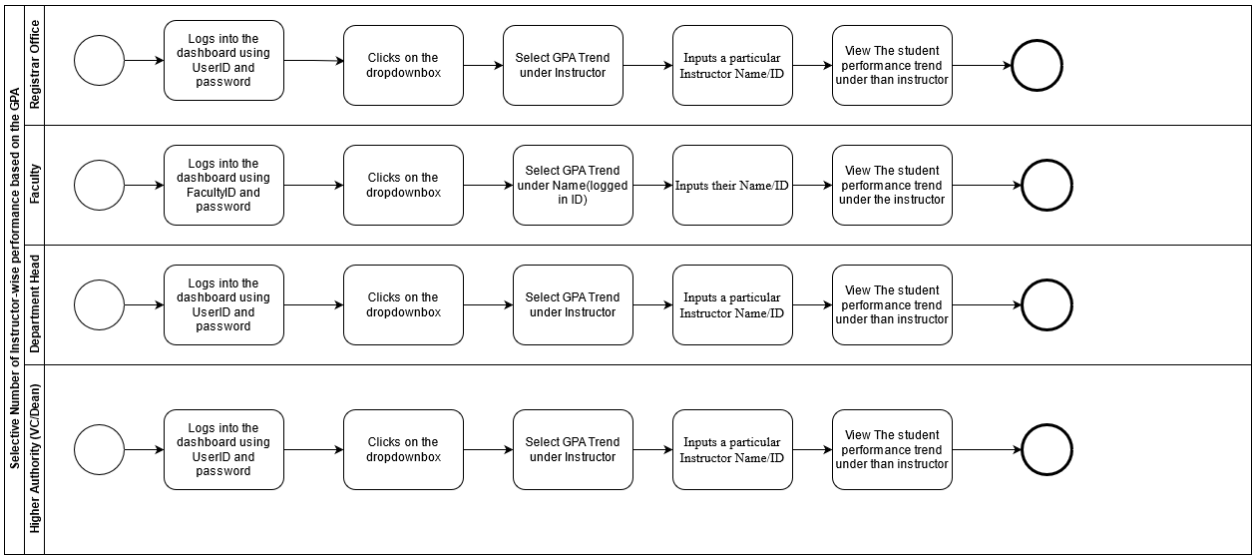


Figure 2.0: Selective Number of Instructor-wise performance based on the GPA

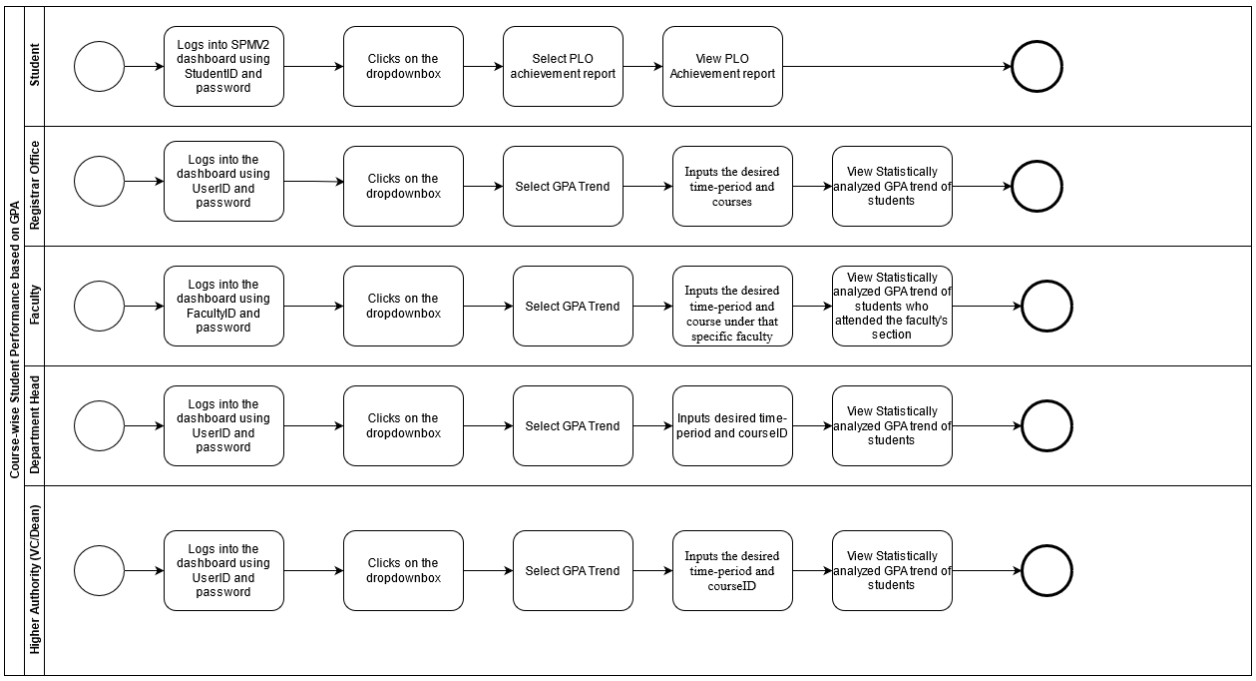


Figure 2.1: Course-wise Student Performance based on GPA

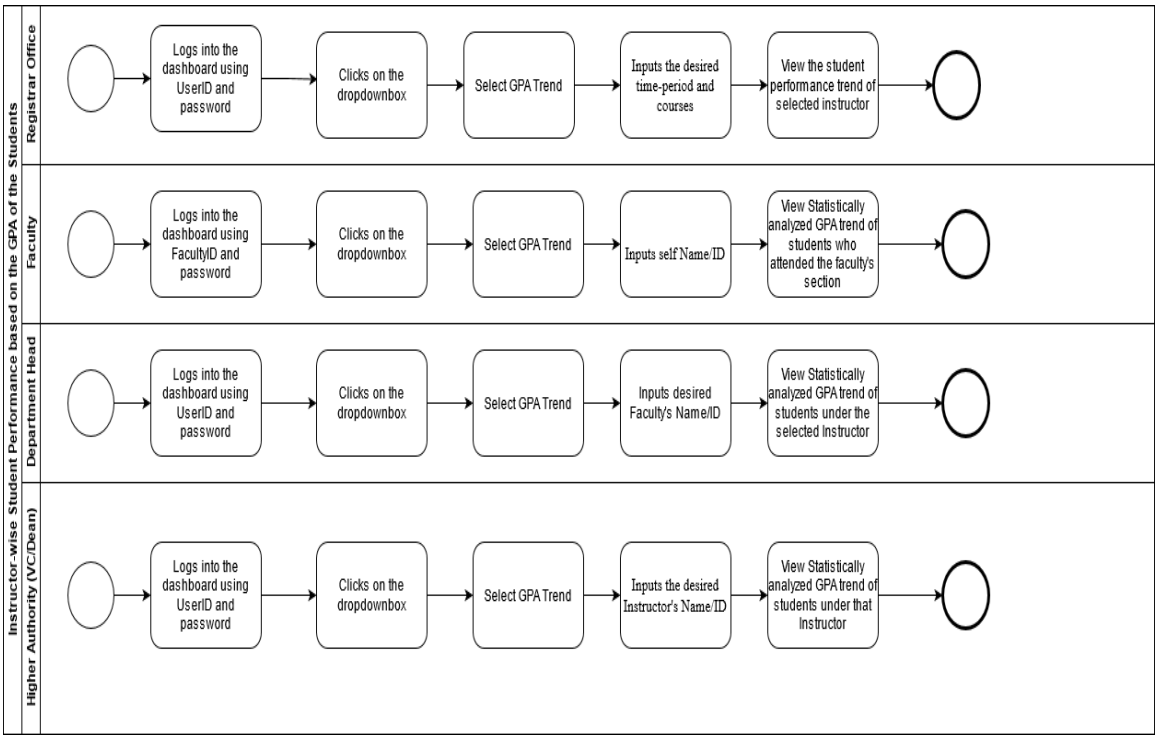


Figure 2.2: Instructor-wise Student Performance based on the GPA of the Students

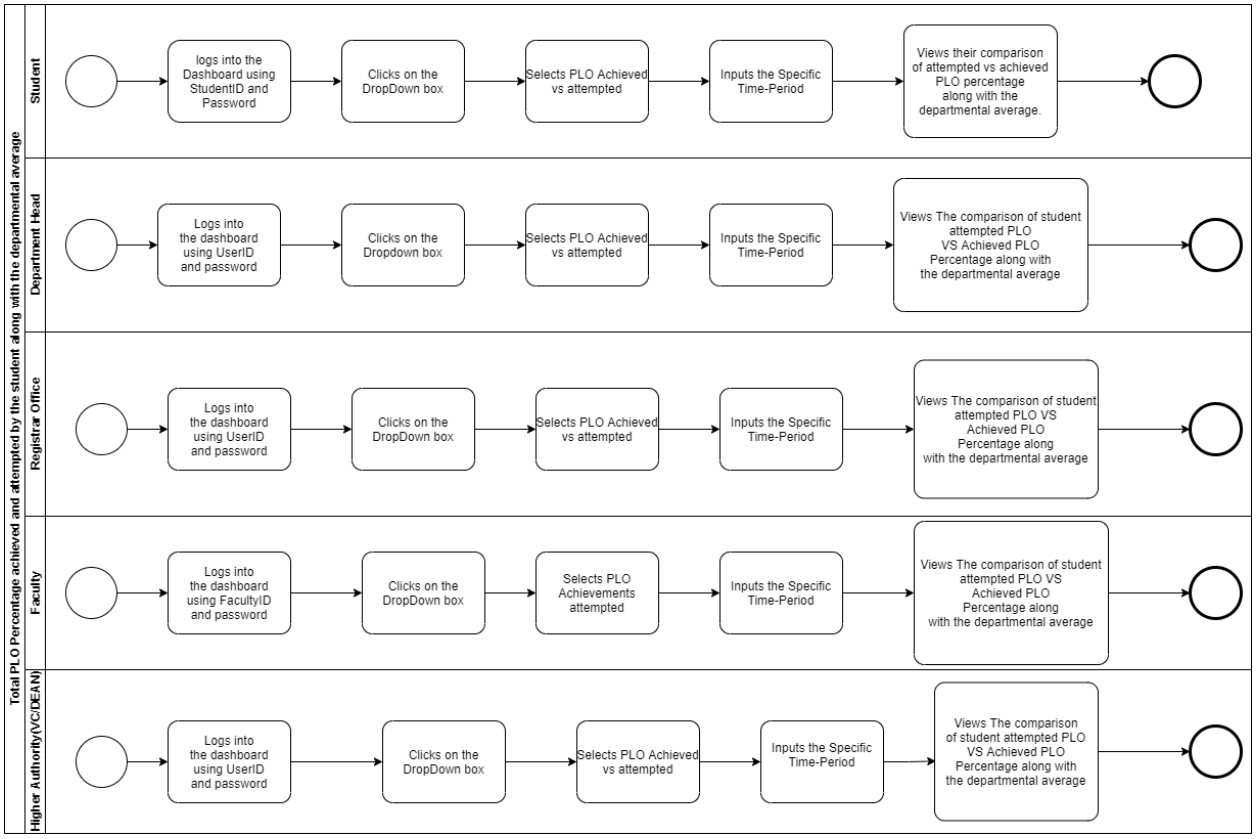


Figure 2.3: Total PLO Percentage achieved and attempted by the student along with the departmental average

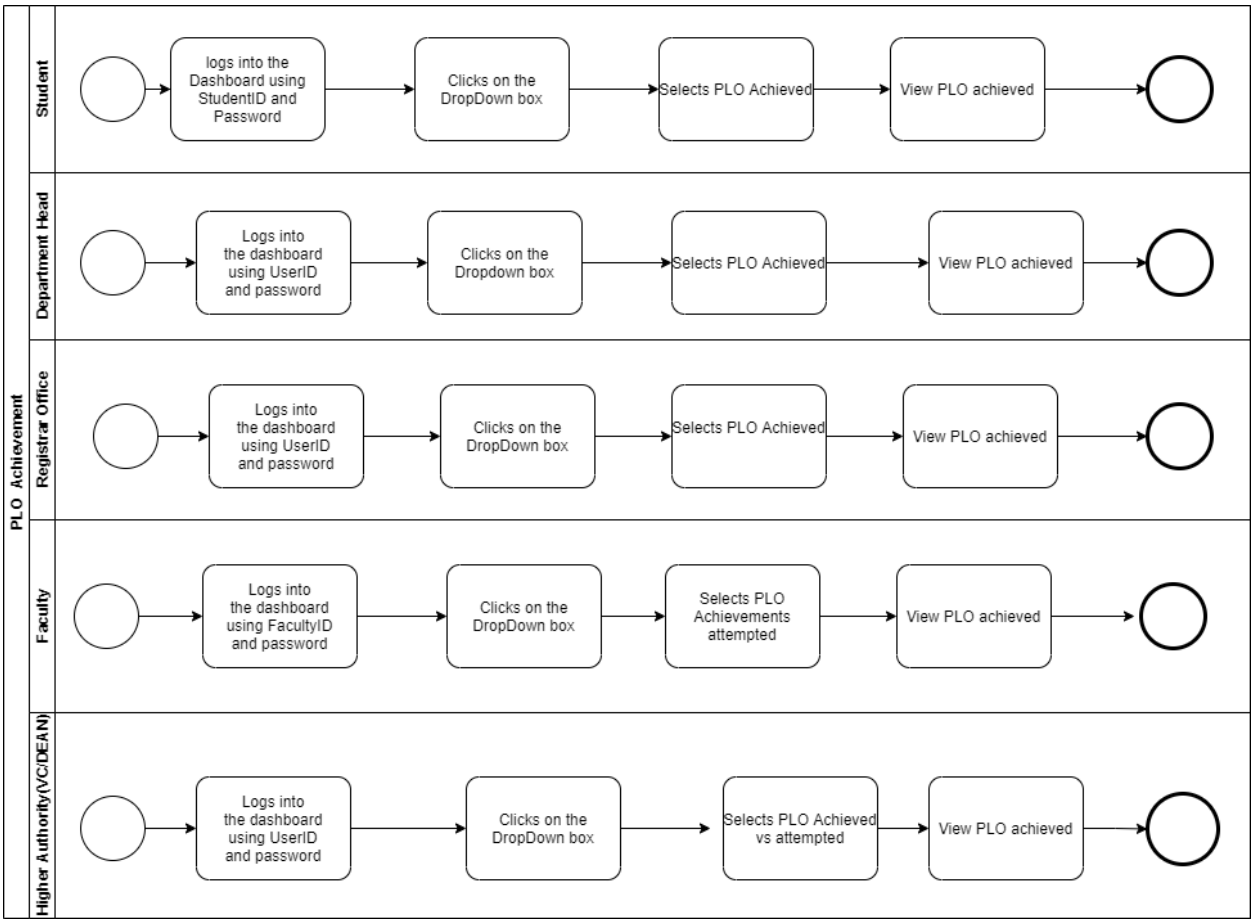


Figure 2.4: PLO Achievement (Process)

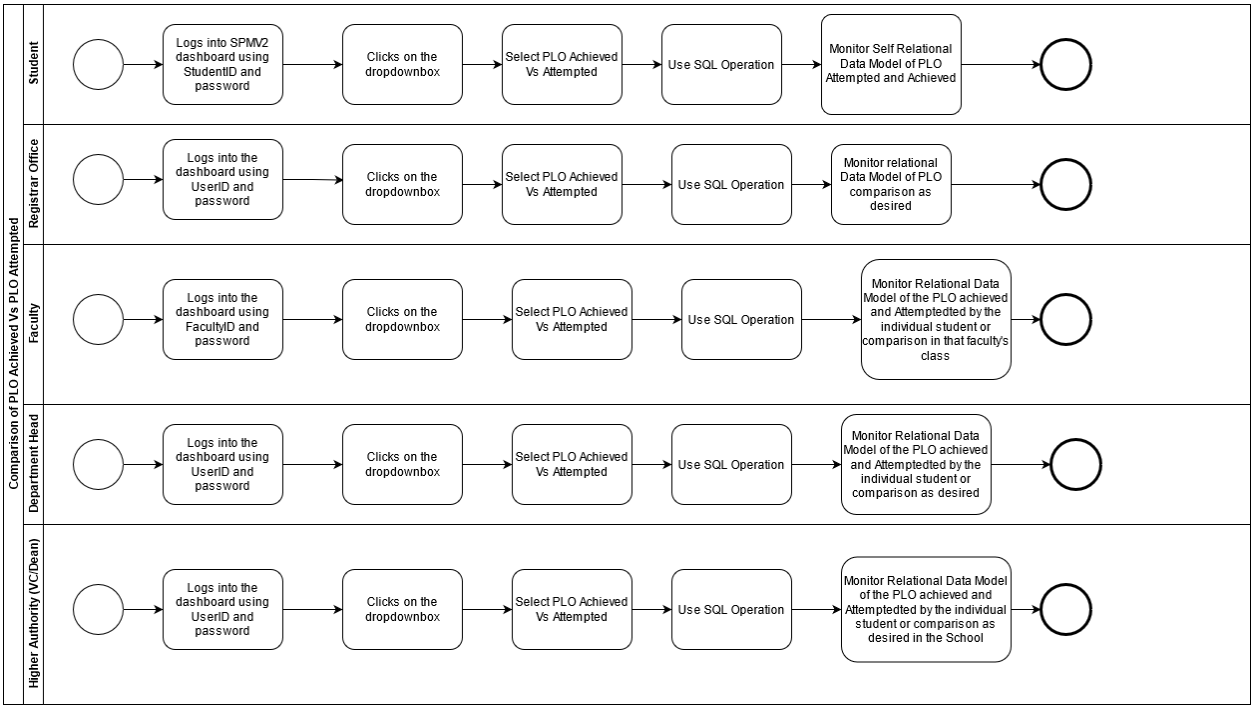


Figure 2.5: Comparison of PLO Achieved vs Attempted (Process)

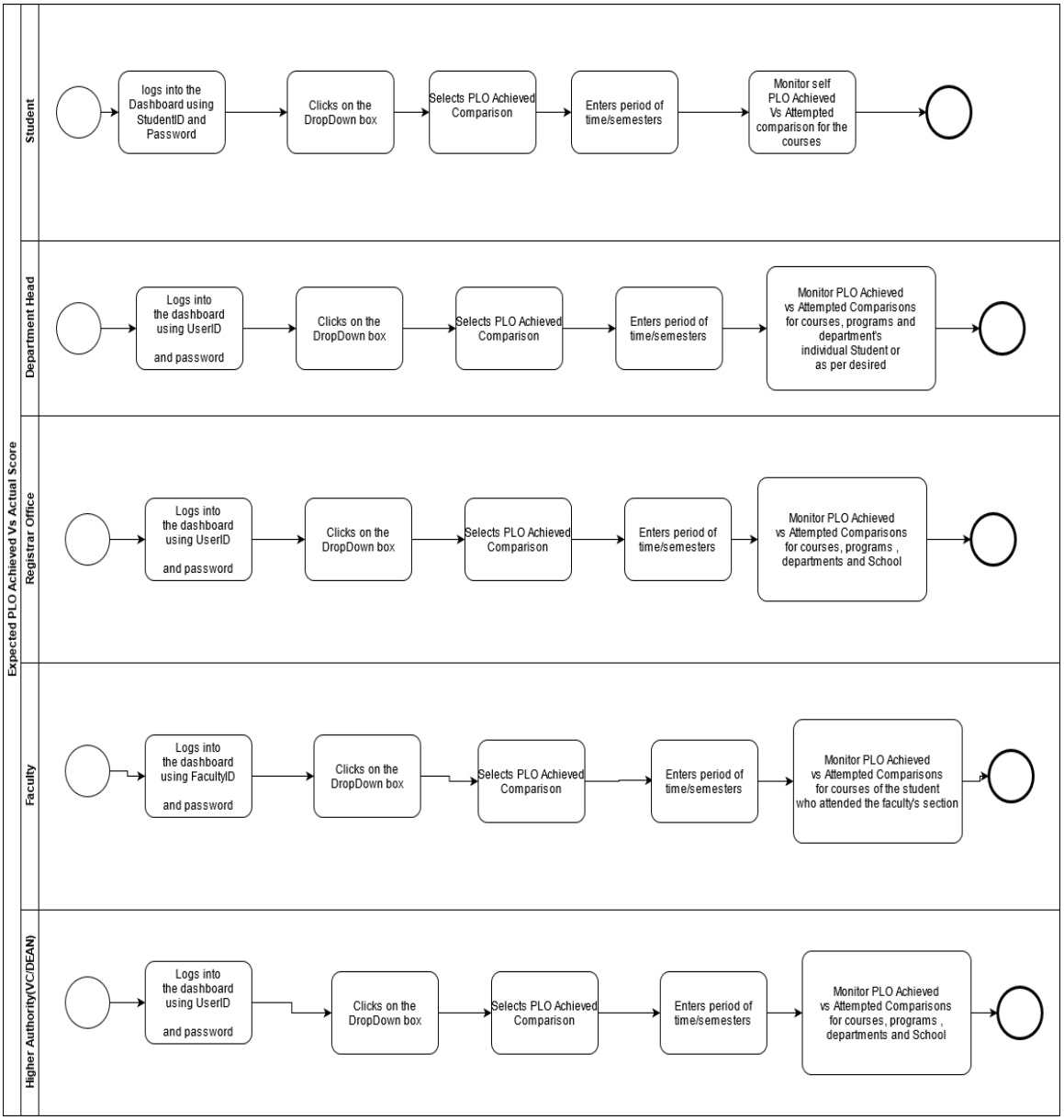


Figure 2.6: Expected PLO Achieved Vs Actual Score (Process)

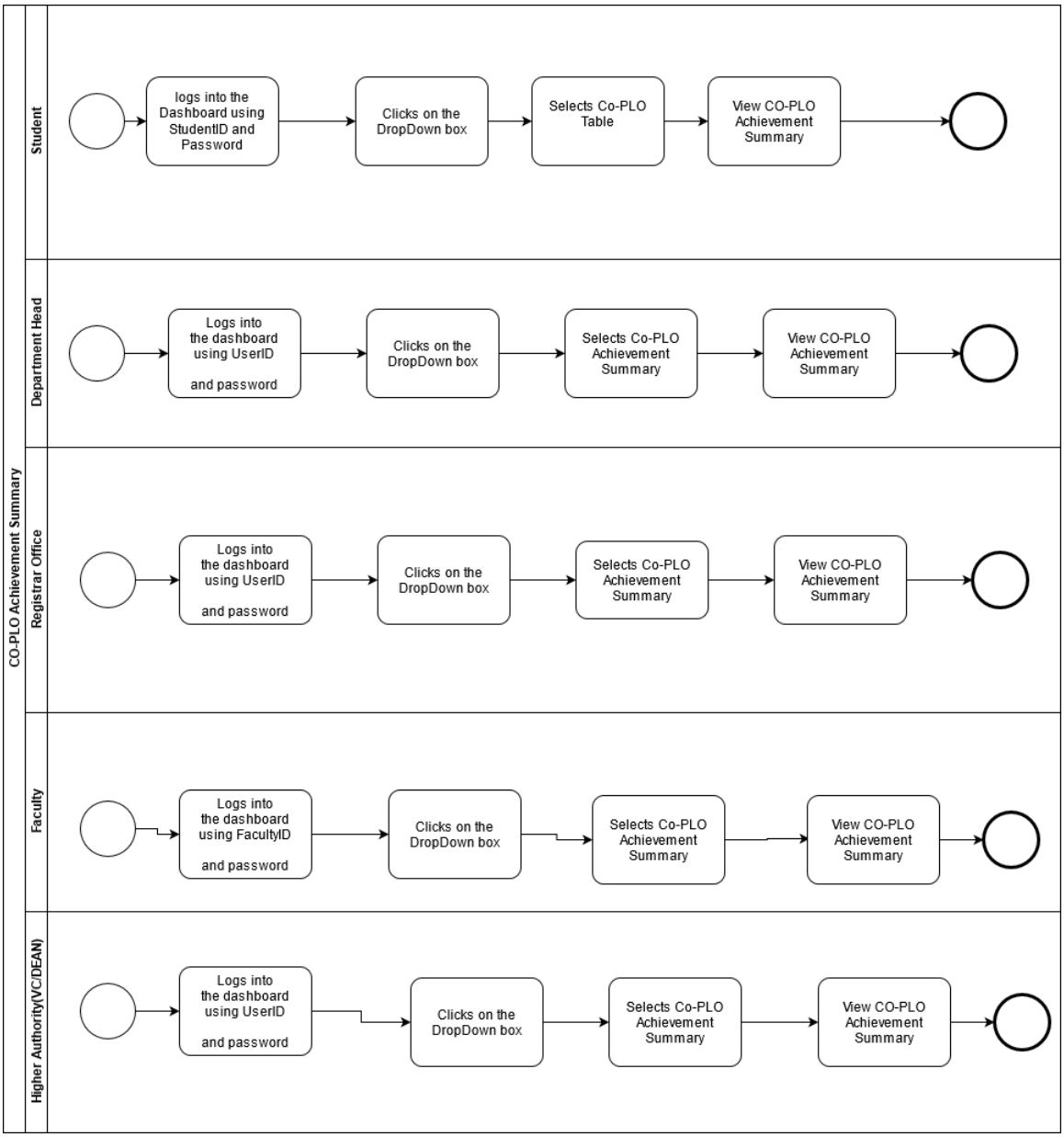


Figure 2.7: CO-PLO Summary (Process)

CHAPTER 3 - LOGICAL SYSTEM DESIGN:

In this chapter, we will be doing the processes of creating a data model of our proposed system for the data to be stored in a database. This data model is a conceptual representation of Data objects, the associations between different data objects, and the rules. Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data. Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data. We will be designing our proposed system for a better representation of all the data.

A. BUSINESS RULE [SPMS 2.0]:

Business rules describe the operations, definitions and constraints that govern the data model. As opposed to the ERD, they are made using regular English sentences so that a non-technical stakeholder can decipher information about the data model without notation knowledge.
The business rules that govern our data model are as follows:

1. A student must have one department. A STUDENT has StudentID, FirstName, LastName, DateofBirth, Gender, Email, Phone, Address, EnrollmentDate. A department must have many students.

2. Student may perform many registrations. A REGISTRATION includes RegistrationID, Semester, Year, Section Id, StutendID. A registration must be performed by at least one student.

3. A section mandatorily have many registrations. A registration has at least one section. A section includes SectionID, SectionNum, CourseId, FacultyID, Semester, Year.

4. A registration may belong to many EVALUATIONS. An evaluation mandatorily belongs to one registration. An evaluation contains EvaluationID, ObtainedMarks, AssessmentID, RegistrationID.

5. An evaluation must have one assessment. An Assessment must have many evaluations. Assessments contains AssesmentsID, AssessmentName, TotalMarks, SectionID, COID. An assessment must contain one section. A section contains one or many assessments.

6. An assessment must map with one CO's. A CO's maps with one or many assessments. A CO's includes COID, CourseID, PLOID. A CO must contain one Course. A Course contain one or many CO's. A course may have many prerequisites. A course must affiliate one mark distribution. A mark distribution may affiliate many courses. A Mark Distribution includes DistID, A, A-, B+, B, B-, C+, C, C-, D+, D, ThresoldMarks.

7. A CO's must map with one PLO's. A PLO's must map with one or many CO's. PLO includes PLOID, PLONum, Details, ProgramID.

8. A PLO must contain one program. A program contains one or many PLO's. A program has ProgramID, ProgramName, DepartmentID. A program must contain one or many courses. A Course must contain one course.

9. A program must belong to one department. A department must belong to one or many programs. A department contain DepartmentID, DepartmentName, SchoolID.

10. A department must contain one school. A School must contain one or many departments. A school includes SchoolID, SchoolName.

11. An employee has four sub-type(Dean, Department Head, Faculty, VC). An employee includes EmployeeID, FirstName, LastName, DateofBirth, Gender, Email, Phone, Address, EmployeeType.

12. A school must run by one or many Dean. A dean must run one school. A Dean has SchoolID, StartDate, EndDate.

13. A Department must manage one or many Department head. A department head must manage one department. A department head includes DepartmentID, StartDate, EndDate.

14. A Faculty must have one Department. A department must have one or many Faculties. A Faculty includes DepartmentID, Rank, JoinDate. A faculty may teach many sections. A section must be taught by one faculty.

B. ENTITY RELATIONSHIP DIAGRAM:

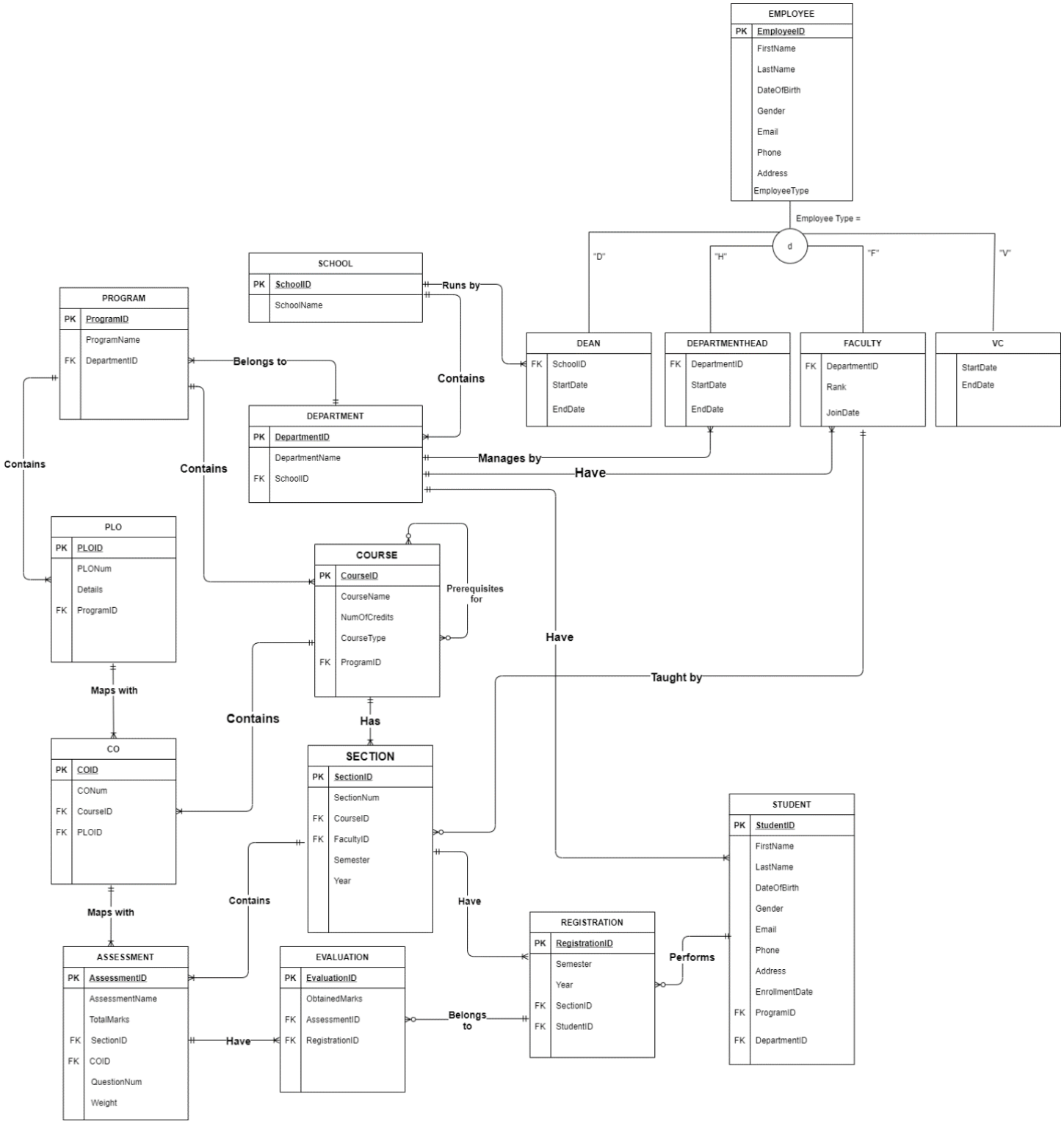


Figure 2.8: Entity relationship diagram

C. ENTITY RELATIONSHIP DIAGRAM TO RELATIONAL SCHEMA:

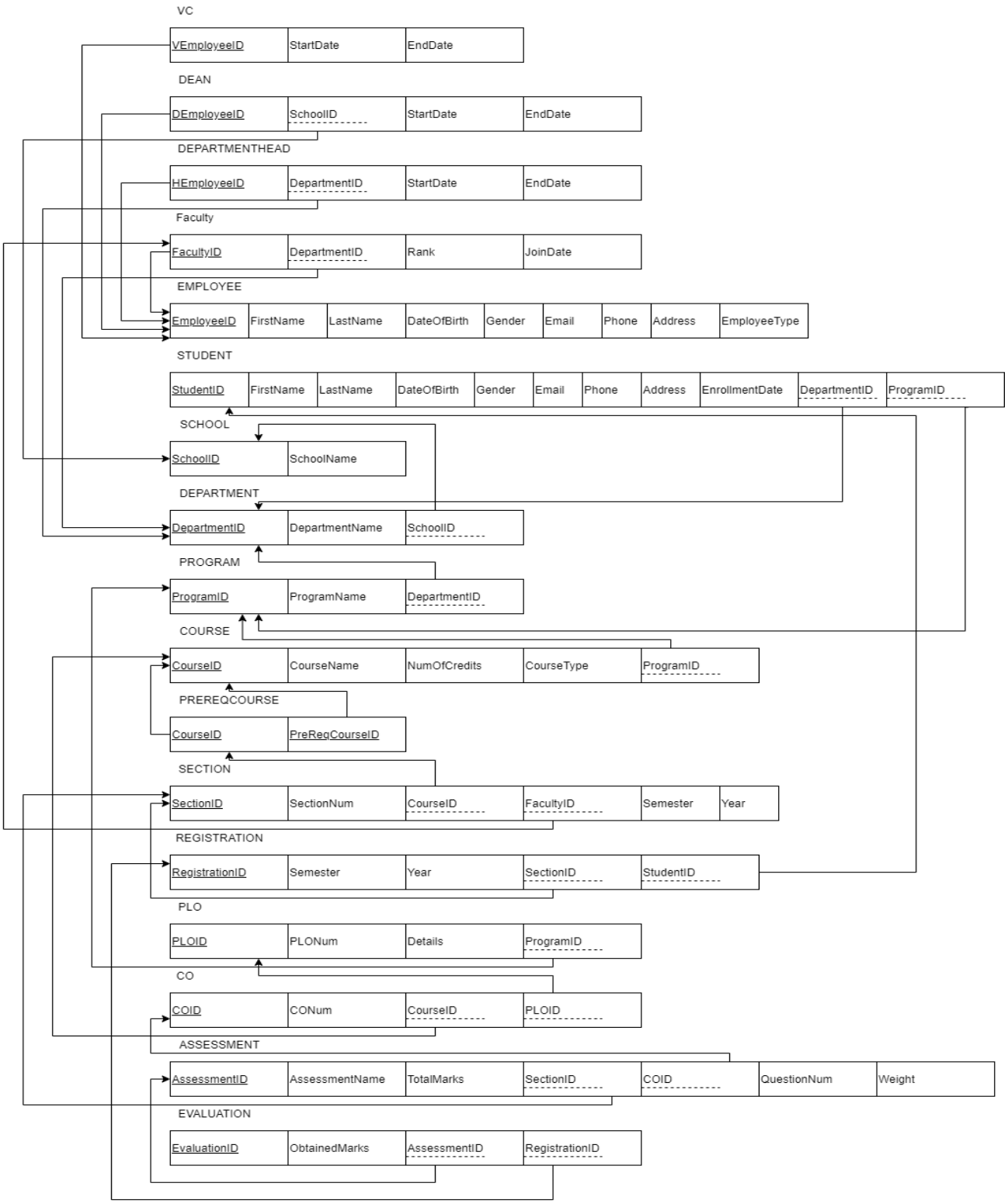


Figure 2.9: Entity relationship diagram

D. NORMALIZATION:

Registration	RegistrationID	r1	Evaluation	EvaluationID	e1
	Semester	r2		ObtainedMarks	e2
	Year	r3		AssessmentID	a1
	StudentID	s1		RegistrationID	r1
	SectionID	q1	Student	StudentID	s1
Section	SectionID	q1		FirstName	s2
	SectionNum	q2		LastName	s3
	Semester	q3		DateOfBirth	s4
	Year	q4		Gender	s5
	CourseID	o1		Email	s6
	FacultyID	f1		Phone	s7
Course	CourseID	o1		Address	s8
	CourseName	o2		EnrollmentDate	s9
	NumOfCredits	o3		ProgramID	g1
	CourseType	o4		DepartmentID	d1
	ProgramID	g1	Employee	EmployeeID	m1
Program	ProgramID	g1		FirstName	m2
	ProgramName	g2		LastName	m3
	DepartmentID	d1		DateOfBirth	m4
School	SchoolID	l1		Gender	m5
	SchoolName	l2		Email	m6

Department	DepartmentID	d1		Phone	m7
	DepartmntName	d2		Address	m8
	SchoolID	l1		EmployeeType	m9
CO	COID	c1	VC	VEmployeeID	v1
	CONum	c2		StartDate	v2
	CourseID	o1		EndDate	v3
	PLOID	p1	Dean	DEmployeeID	n1
Assessment	AssessmentID	a1		SchoolID	l1
	AssessmentName	a2		StartDate	n2
	TotalMarks	a3		EndDate	n3
	SectionID	q1	Department Head	HEmployeeID	h1
	COID	c1		DepartmentID	d1
	QuestionNum	a4		StartDate	h2
	Weight	a5		EndDate	h3
PreReqCourse	CourseID	j1	Faculty	FacultyID	f1
	PreReqCourseID	j2		DepartmentID	d1
PLO	PLOID	p1		Rank	f2
	PLONum	p2		JoinDate	f3
	Details	p3			
	ProgramID	g1			

l1➔	l2	j1➔	j2
d1➔	d2, l1	o1➔	o2, o3, o4, g1
g1➔	g2, d1	q1➔	q2, q3, q4, o1, f1
m1➔	m2, m3, m4, m5, m6, m7, m8, m9	p1➔	p2, p3, g1
v1➔	v2, v3	c1➔	c2, o1, p1
n1➔	n2, n3, l1	r1➔	r2, r3, s1, q1
h1➔	h2, h3, d1	a1➔	a2, a3, a4,a5, q1, c1
f1➔	f2, f3, d1	e1➔	e2, a1, r1
s1➔	s2, s3, s4, s5, s6, s7, s8, s9, g1, d1		

SchoolID➔	SchoolName
DepartmentID➔	DepartmentName, SchoolID
ProgramID➔	ProgramName, DepartmentID
EmployeeID➔	FirstName, LastName, Gender, DateOfBirth, Email, Phone, Address, EmployeeType
VEmployeeID➔	StartDate, EndDate
DEmployeeID➔	SchoolID, StartDate, EndDate
HEmployeeID➔	DepartmentID, StartDate, EndDate
FacultyID➔	DepartmentID, Rank, JoinDate
StudentID➔	FirstName, LastName, DateOfBirth, Gender, Email, Phone, Address, Enrollmentdate, DepartmentID, ProgramID
CourseID➔	CourseName, NumOfCredits, CourseType, ProgramID
CourseID➔	PreReqCourseID
SectionID➔	SectionNum, Semester, Year, CourseID, FacultyID
PLOID➔	PLONum, Details, ProgramID
COID➔	CONum, PLOID, CourseID
RegistrationID➔	Semester, Year, SectionID, StudentID

AssessmentID➔	AssessmentName, QuestionNum, TotalMarks, COID, SectionID, Weight
EvaluationID➔	ObtainedMarks, AssesmentID, RegistrationID

1NF: A relation that has a primary key and in which there are no repeating groups.

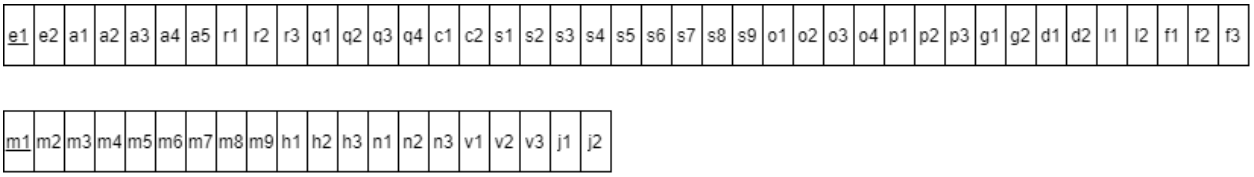


Figure 3.0: 1NF

2NF: A relation in first normal form in which every non-key attribute is fully functionally dependent on the primary key.

Figure 3.1: 2NF

3NF: A relation that is in second normal form and has no transitive dependencies.

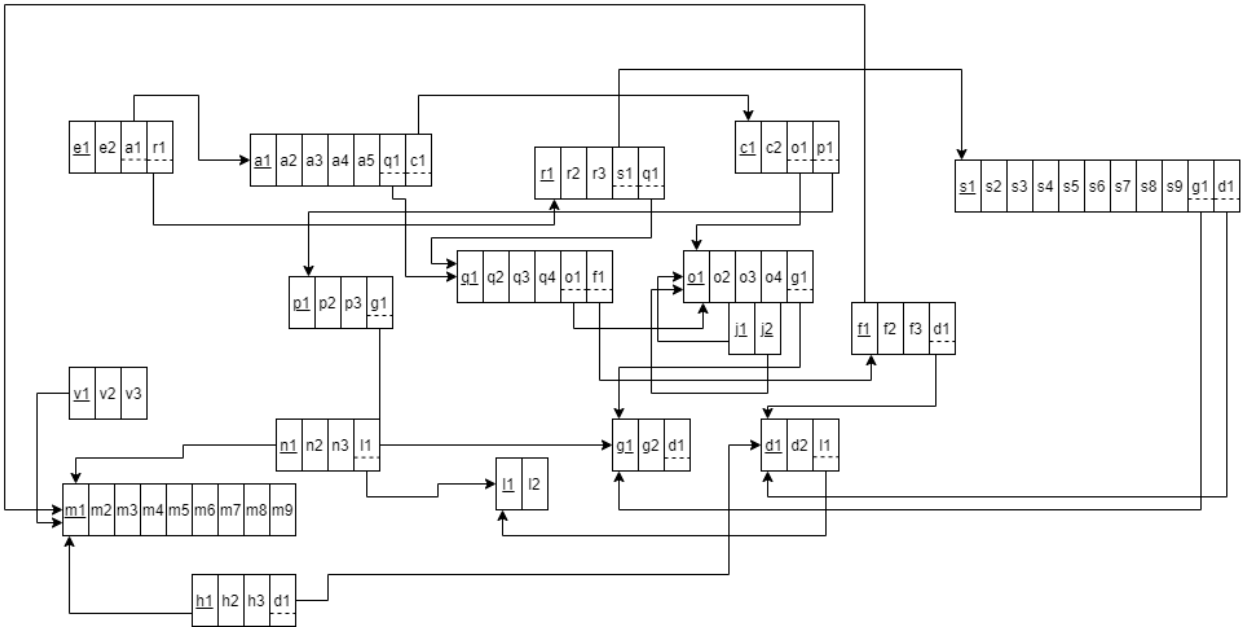


Figure 3.2: 3NF

BCNF: All determinants are candidate keys. There is no determinant that is not a unique identifier. Here, all the relations already are in BCNF.

E. DATA DICTIONARY:

School_T

Name	Data Type	Size	Remarks
cSchoolID	VARCHAR	5	This is the primary key of School. E.g: “SETS”
cSchoolName	VARCHAR	50	This is the name of the School. E.g: “School of Engineering, Technology & Science”.

Program_T

Name	Data Type	Size	Remarks
cProgramID	INTEGER		This is the primary key for a program. E.g: “1”
cProgramName	VARCHAR	50	This is the name of the program. E.g: “Bachelor of Science”
cDepartmentID	VARCHAR	3	This is the foreign key from the Department table. E.g: “CSE”

Department_T

Name	Data Type	Size	Remarks
cDepartmentID	VARCHAR	3	This is the primary key for the Department table. E.g: “CSE”
cDepartmentName	VARCHAR	50	This is the name of the department. E.g: “Computer Science and Engineering”.
cSchoolID	VARCHAR	5	This is a foreign key from the School table. E.g: “SETS”.

Student_T

Name	Data Type	Size	Remarks
nStudentID	INTEGER		This is the primary key for the Student table. E.g: “1921834”.
cFirstName	VARCHAR	30	This is the first name of the student. E.g: “Rakibul”.
cLastName	VARCHAR	30	This is the last name of the student. E.g: “Hasan”.
dDateOfBirth	DATE	DD-MM-YYYY	This is the birth date of the student. E.g: “21-12-1996”.
cGender	VARCHAR	6	This is the gender of the student. E.g: “Female”.
cEmail	VARCHAR	30	This is the email of the student. E.g: “1921834@iub.edu.bd”
nPhone	NUMERIC	11	This is the phone of the student. E.g: “01XXXXXXXXX”.

cAddress	VARCHAR	50	This is the address of the student. E.g: “House 1,Road 4,Block D, Bashundhara RA
cDepartmentID	VARCHAR	3	This is the foreign key from the Department table. E.g: “CSE”
cProgramID	INTEGER		This is the foreign key from the Program table. E.g: “1”
dEnrollmentDate	DATE	DD-MM-YYYY	This is enrollment date of the student. E.g.: “1-1-2019”

CO_T

Name	Data Type	Size	Remarks
cCOID	VARCHAR	9	This is the primary key for the CO table. E.g: “CO1”.
nCONum	INTEGER		This is the CO number. E.g: 1,2 etc.
cCourseID	VARCHAR	6	This is the foreign key from the Course table. E.g: “CSE303”
cPLOID	VARCHAR	5	This is the foreign key from the PLO table. E.g: “PLO1”

PLO_T

Name	Datatype	Size	Remarks
cPLOID	VARCHAR	5	This is the primary key for Program Learning Outcome. E.g: “PLO1”
nPLONum	INTEGER		This is the PLO number. E.g: “1”
cDetails	VARCHAR	50	This is the details for Program Learning Outcome. E.g: “An ability to select and apply the knowledge, technique, skills and modern tools of the computer science and engineering discipline ”
cProgramID	INTEGER		This is a foreign key from Program table. E.g: “1”

Employee_T

Name	Datatype	Size	Remarks
nEmployeeID	INTEGER		This is the primary key for Employee table. E.g: “1801”
cFirstName	VARCHAR	30	This is the first name of the faculty. E.g: “Sadita”
cLastName	VARCHAR	30	This is the last name of the faculty. E.g: “Ahmed”
dDateofbirth	DATE	DD-MM-YYYY	This is the date of Birth of the faculty. E.g:01-01-1992
cGender	VARCHAR	6	This is the gender of the faculty. Eg: “Female”
cEmail	VARCHAR	30	This is the email address of the Student. E.g: “1675231@iub.edu.bd”

nPhone	NUMERIC	11	This is the phone number of the Faculty. E.g: “01292383111”
cAddress	VARCHAR	30	This is the address of the Faculty. E.g: “House 14, Road 21, Sector 11,Baridara,Dhaka, Bangladesh”
cEmployeeType	CHAR	1	This is the type of the employee. E.g: “F”

Course_T

Name	Datatype	Size	Remarks
cCourseID	VARCHAR	6	This is the Primary Key for the Course. E.g: “CSE203”
cCourseName	VARCHAR	40	This is the name of the Course. E.g: “Discreet Mathematics”
nNumOfCredits	INTEGER		This is the number of credits for the Course. E.g: “3”
cCourseType	VARCHAR	10	This is the type of the Course. E.g: “Core”
cProgramID	INTEGER		This is the foreign key from the program table. E.g: “1”

Section_T

Name	Datatype	Size	Remarks
nSectionID	INTEGER		This is the Primary Key for Section. E.g: "1"
nSectionNum	INTEGER		This is the section number. E.g: "1"
cCourseID	VARCHAR	6	This is the foreign key from the Course table. E.g: "CSE101"
cFacultyID	NUMERIC	4	This is the foreign key from Faculty table. E.g: "1801"
cSemester	VARCHAR	6	This is the semester of the section. E.g: "Summer"

Registration_T

Name	Datatype	Size	Remarks
nRegistrationID	INTEGER		This is the Primary Key for Registration. E.g: "0101010101"
cSemester	VARCHAR	6	This is the semester of registration. E.g: "Spring"
dYear	YEAR	yyyy	This is the year of registration. E.g: "2019"
nSectionID	INTEGER		This is the Foreign Key from Section table E.g: "1"
nStudentID	INTEGER		This is the Foreign key from the Student Table. E.g: "1800001"

Assessment_T

Name	Datatype	Size	Remarks
nAssessmentID	INTEGER		This is the Primary Key for Assessment.
cAssessmentName	VARCHAR	30	This is the name of the assessment. E.g: “Mid”
cTotalMarks	NUMBER		This is the total marks of the assessment. E.g: “30”
nSectionID	INTEGER		This is the Foreign Key from Section table.
nCOID	INTEGER		This is the Foreign Key from the Course Outcome table.
nQuestionNum	INTEGER		This is the question number for assessment. E.g: “1,2,3….”
nWeight	INTEGER		This is the percentage range for assessment. E.g: “Project- 50%, Assessment-50%”.

Evaluation_T

Name	Datatype	Size	Remarks
nEvaluationID	INTEGER		This is the Primary Key for Enrollment.
cObtainedMarks	NUMBER		This is the obtained marks of the student. E.g: “24.5”
cAssessmentID	INTEGER		This is the foreign key from the assessment table.
nRegistrationID	INTEGER		This is the Foreign Key from Registration table.

VC_T

Name	Datatype	Size	Remarks
nVEmployeeID	INTEGER		This is the foreign key from the Employee table. E.g: “4250”
dStartDate	DATE	dd-mm-yyyy	This is starting date for the VC. E.g: “01-03-2020”
dEndDate	DATE	dd-mm-yyyy	This is the date VC retire from his post. E.g: “01-03-2024”

Dean_T

Name	Datatype	Size	Remarks
nDEmployeeID	INTEGER		This is the foreign key from the Employee table. E.g: “4250”
cSchoolID	VARCHAR	5	This is the SchoolID of the school DEAN manages. E.g: “SETS”
dStartDate	DATE	dd-mm-yyyy	This is starting date. E.g: “01-03-2020”
dEndDate	DATE	dd-mm-yyyy	This is the date DEAN retire from his post. E.g: “01-03-2024”

DepartmentHead_T

Name	Datatype	Size	Remarks
nHEmployeeID	INTEGER		This is the foreign key from the Employee table. E.g: “4250”
cDepartmentID	VARCHAR	3	This is the DepartmentID of the department HEAD manages. E.g: “CSE”
dStartDate	DATE	dd-mm-yyyy	This is starting date. E.g: “01-03-2020”
dEndDate	DATE	dd-mm-yyyy	This is the date HEAD retire from his post. E.g: “01-03-2024”

Faculty_T

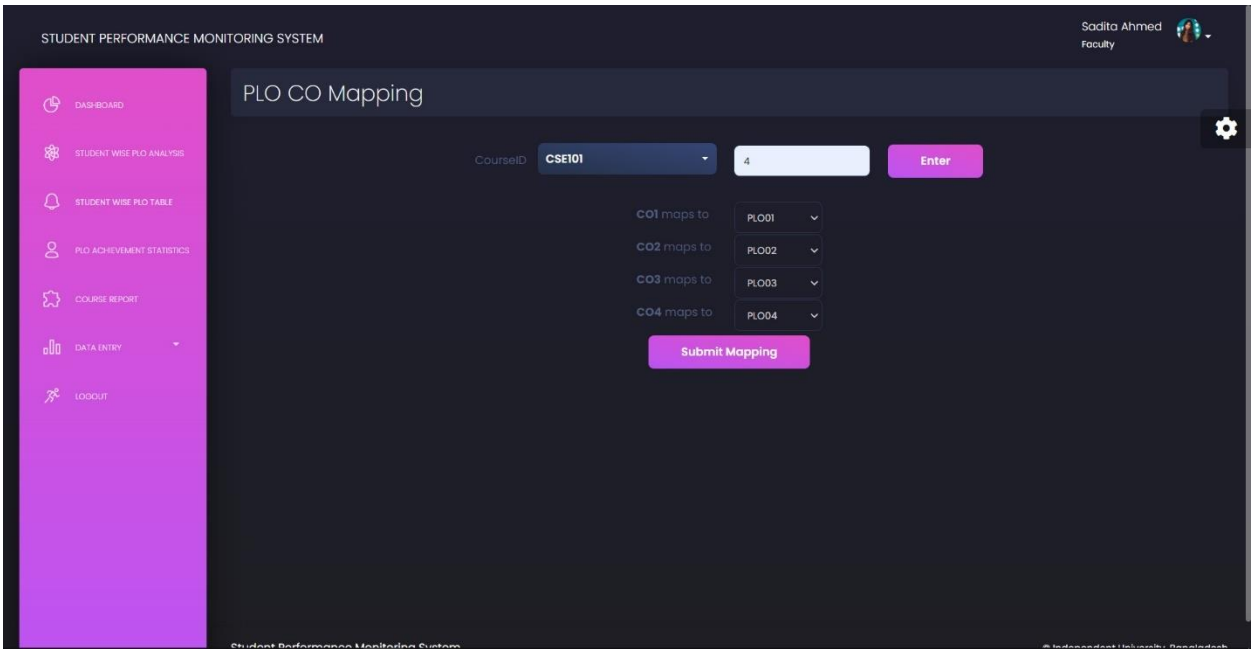
Name	Datatype	Size	Remarks
nFacultyID	INTEGER		This is the foreign key from the Employee table. E.g: “4250”
cDepartmentID	VARCHAR	3	This is the DepartmentID of the department faculty belongs to. E.g: “CSE”
dJoinDate	DATE	dd-mm-yyyy	This is starting date. E.g: “01-03-2020”
cRank	VARCHAR	30	This is the rank of the faculty. E.g: “Assistant Professor”

PreReqCourse_T

Name	Datatype	Size	Remarks
cCourseID	VARCHAR	6	This is the foreign key from the Course table. E.g: “CSE303”
cPreReqCourseID	VARCHAR	6	This is the foreign key from the Course table . E.g: CSE203

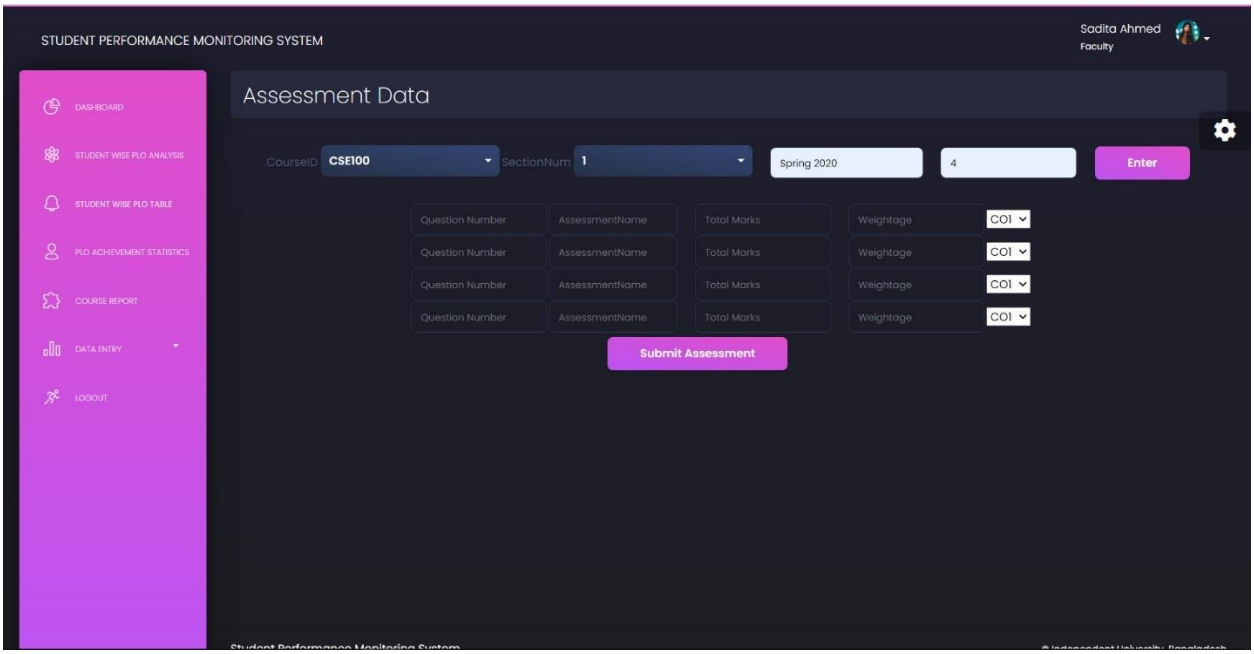
CHAPTER 4 - PHYSICAL SYSTEM DESIGN:

A. INPUT FORM:



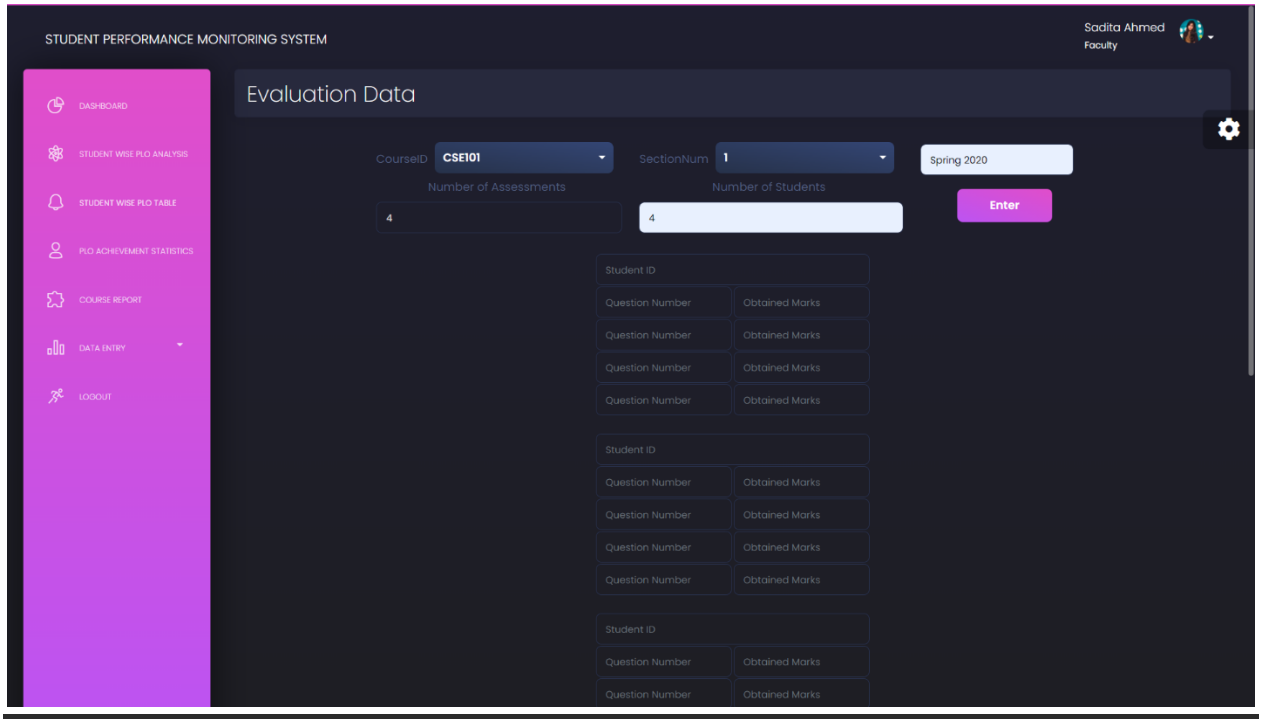

```
def plocomapping(request):
    usertype = request.user.groups.all()[0].name
    if request.method == 'POST':
        courseID = request.POST.get('course')
        coMaps = request.POST.getlist('coMaps')

        course = Course_T.objects.get(pk=courseID)

        plist = PLO_T.objects.all()

        plolist = []

        for p in plist:
            if p.program_id == course.program_id:
                plolist.append(p)

        for i in range(len(coMaps)):
            k = -1

            for p in range(len(plolist)):
                if plolist[p].ploNum == coMaps[i]:
                    k = p
            co = CO_T(coNum="CO" + str(i + 1), course=course, plo=plolist[k])
            co.save()

        return redirect('plocomapping')
    else:
        return render(request, 'plocomapping.html', {
            'usertype': usertype,
            'clist': courselist,
```

```
        })
```




```
def assessmentdataentry(request):
    usertype = request.user.groups.all()[0].name

    sections = [1, 2, 3]

    if request.method == 'POST':
        faculty_id = request.user.username
        course_id = request.POST.get('course')
        sectionNo = request.POST.get('section')
        semester = request.POST.get('semester')
        totalMarks = request.POST.getlist('totalMarks')
        weightage = request.POST.getlist('weightAge')
        assessmentName = request.POST.getlist('assessmentName')
        questions = request.POST.getlist('questions')
        cos = request.POST.getlist('co')

        section_id = None
        try:
            sections = Section_T.objects.raw('''
                SELECT *
                FROM spmapp_section_t
                WHERE course_id = '{}'
                    AND sectionNum = {};
                    AND semester = '{}'

            '''.format(course_id, sectionNo,semester))
            section_id = sections[0].sectionID
        except:
            section_id = None

        if section_id is None:
            section = Section_T(sectionNum=sectionNo, course_id=course_id,
faculty_id=faculty_id,semester=semester)
```

```
            section.save()
            section_id = section.sectionID

        for j in range(1, len(totalMarks) + 1):
            conum = cos[j-1]
            co_id = CO_T.objects.raw('''
                SELECT *
                FROM spmapp_co_t
                WHERE course_id = '{}'
                    AND coNum = '{}'
            '''.format(course_id, conum))

            assessment = Assessment_T(section_id=section_id,
co_id=co_id[0].coID, totalMarks=totalMarks[j - 1],
                                      assessmentName=assessmentName[j-
1],questionNum=questions[j-1],weight=weightage[j-1])
            assessment.save()

        return redirect('assessmentdataentry')

    else:
        return render(request, 'assessmentdataentry.html', {
            'usertype': usertype,
            'clist': courselist,
            'semesters': semesters,
            'sections': sections,

        })
```




```
def evaluationdataentry(request):
    usertype = request.user.groups.all()[0].name
    section = [1, 2, 3]

    if request.method == 'POST':
        course_id = request.POST.get('course')
```

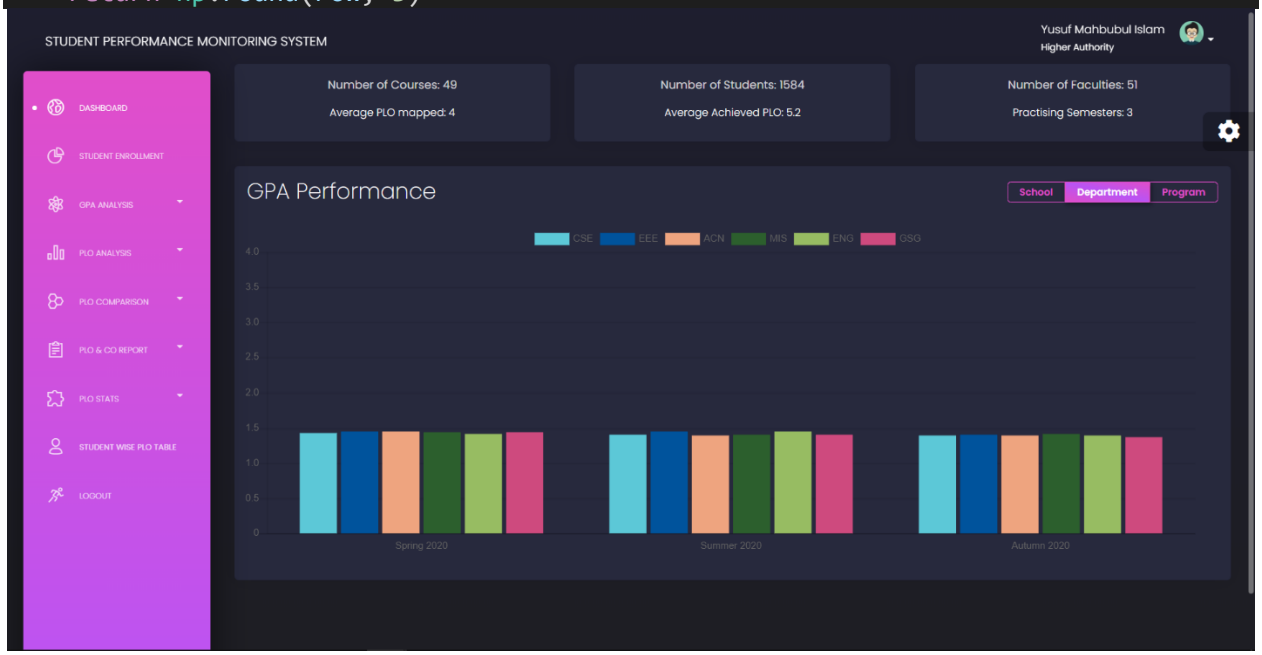
```
        section = request.POST.get('section')
        semester = request.POST.get('semester')

        print(course_id)
        print(section)
        print(semester)

        student_id = request.POST.getlist('student')
        obtainedMarks = []
        questions = []
        for i in range(len(student_id)):
            obtainedMarks.append(request.POST.getlist(f'obtainedMarks{i}'))
            questions.append(request.POST.getlist(f'questions'))

        section_id = None
        try:
            section_id = Section_T.objects.raw('''
                SELECT *
                FROM spmapp_section_t
                WHERE course_id = '{}'
                    AND sectionNum = '{}'
                    AND semester ='{}';
            '''.format(course_id, section,semester))
            section_id = section_id[0].sectionID
            print(section_id)
        except:
            section_id = None
        assessment_list = []
        coLength = 0
        try:
            col = CO_T.objects.raw('''
                SELECT count(*)
                FROM spmapp_co_t
                WHERE course_id ='{}'
                '''.format(course_id))
            coLength = col[0][0]+1
        except:
            coLength = 0
        for j in range(1, len(questions[0])+1):
            assessment_id = None
            try:
                assessment_id = Assessment_T.objects.raw('''
                    SELECT *
                    FROM spmapp_assessment_t
                    WHERE section_id = '{}'
                        AND co_id IN (
                        SELECT coID
                        FROM spmapp_co_t
                        WHERE course_id = '{}'
                            AND questionNum = '{}'
                    )
                '''.format(section_id, course_id, j))
                assessment_list.append(assessment_id[0].assessmentID)
            except:
                assessment_id = None
                assessment_list.append(assessment_id)

        for i in range(len(student_id)):
```

```
            registration_id = None
            try:
                registration_id = Registration_T.objects.raw('''
                    SELECT *
                    FROM spmapp_registration_t
                    WHERE student_id = '{}'
                        AND section_id = '{}'
                '''.format(student_id[i], section_id))
                registration_id = registration_id[0].registrationID
            except:
                registration_id = None

            if registration_id is None:
                print(section_id)
                print(student_id[i])
                registration = Registration_T(student_id=student_id[i],
section_id=section_id, semester=semester)
                registration.save()
                registration_id = registration.registrationID

            for j in range(len(assessment_list)):
                evaluation = Evaluation_T(registration_id=registration_id,
assessment_id=assessment_list[j],
                                          obtainedMarks=obtainedMarks[i][j])
                evaluation.save()
        return redirect('evaluationdataentry')
    else:
        return render(request, 'evaluationdataentry.html', {
            'usertype': usertype,
            'clist': courselist,
            'semesters': semesters,
            'sections': section,

        })
```

B. OUTPUT FORMS:


```
def getSchoolWiseGPA(school, semester):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT AVG(grade) as avgGrade
                FROM(
                    SELECT StudentID,sum(Credits*gradepoint)/sum(Credits) as grade
                    FROM(
                        SELECT  StudentID,Credits,
                            CASE
                                WHEN sum(Marks) >= 85 THEN 4.0
                                WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                                WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                                WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                                WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                                WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                                WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                                WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                                WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                                WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                                ELSE 0.0
                            END as gradepoint
                        FROM(
                            SELECT st.studentID as StudentID,c.courseID as CourseI
D,
                                a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks))
as Marks, c.numOfCredits as Credits
                            FROM spmapp_student_t st,
```

```
                                spmapp_department_t d,
                                spmapp_school_t s,
                               spmapp_registration_t r,
                               spmapp_section_t sc,
                               spmapp_course_t c,
                               spmapp_assessment_t a,
                               spmapp_evaluation_t e
                          WHERE st.studentID = r.student_id
                               and st.department_id = d.departmentID
                               and d.school_id = s.schoolID
                               and r.section_id = sc.sectionID
                               and sc.course_id = c.courseID
                               and r.registrationID = e.registration_id
                               and e.assessment_id = a.assessmentID
                               and s.schoolID = '{}'
                               and r.semester='{}'
                          GROUP BY  st.studentID,c.courseID,a.assessmentName) De
rived1
                     GROUP BY StudentID,CourseID) Derived2
                GROUP BY StudentID)
                    '''.format(school, semester))

        row = cursor.fetchall()[0][0]
    return np.round(row, 3)
```

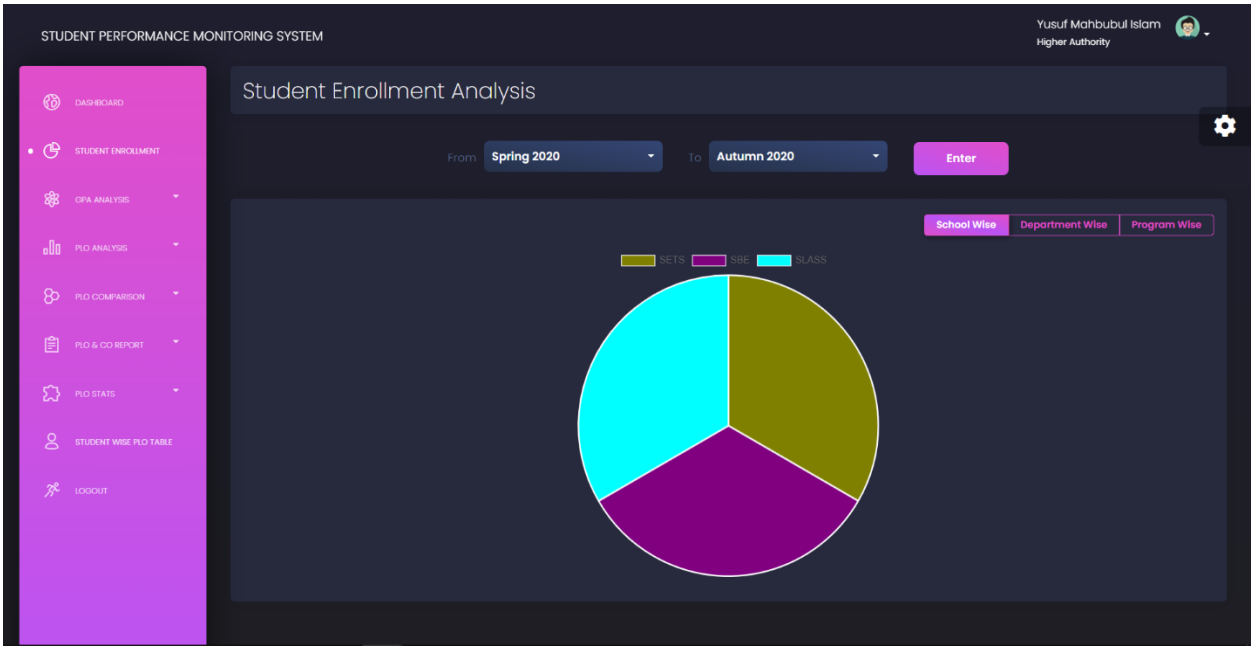



```
def getDeptWiseGPA(dept, semester):
    with connection.cursor() as cursor:
        cursor.execute('''
            SELECT AVG(grade) as avgGrade
            FROM(
                SELECT StudentID,sum(Credits*gradepoint)/sum(Credits) as grade
```

```
                FROM(
                    SELECT  StudentID,Credits,
                        CASE
                            WHEN sum(Marks) >= 85 THEN 4.0
                            WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                            WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                            WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                            WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                            WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                            WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                            WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                            WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                            WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                            ELSE 0.0
                        END as gradepoint
                    FROM(
                        SELECT st.studentID as StudentID,c.courseID as CourseID,
                            a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks)) as
Marks, c.numOfCredits as Credits
                        FROM spmapp_student_t st,
                            spmapp_registration_t r,
                            spmapp_section_t sc,
                            spmapp_course_t c,
                            spmapp_assessment_t a,
                            spmapp_evaluation_t e
                        WHERE st.studentID = r.student_id
                            and r.section_id = sc.sectionID
                            and sc.course_id = c.courseID
                            and r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and st.department_id = '{}'
                            and r.semester='{}'
                        GROUP BY  st.studentID,c.courseID,a.assessmentName) Deriv
ed1
                    GROUP BY StudentID,CourseID) Derived2
                GROUP BY StudentID)
                    '''.format(dept, semester))

        row = cursor.fetchall()[0][0]
    return np.round(row, 3)
```

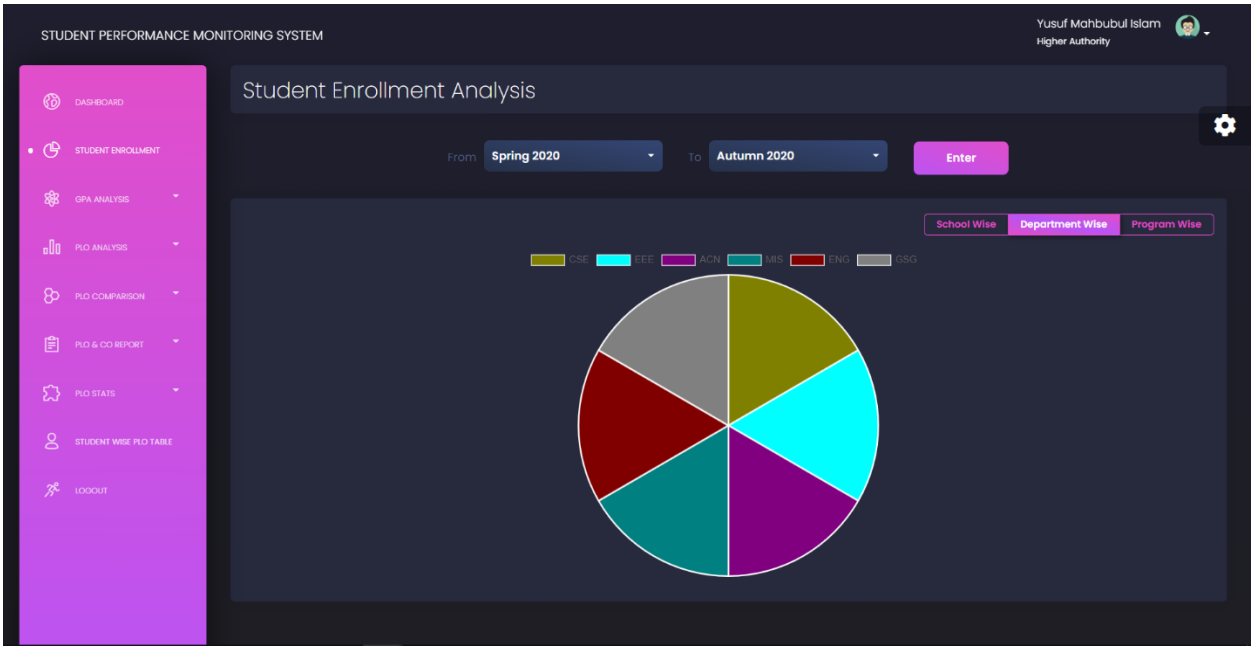

```
def getProgramWiseGPA(program, semester):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT AVG(grade) as avgGrade
                FROM(
                    SELECT StudentID,sum(Credits*gradepoint)/sum(Credits) as grade
                    FROM(
                        SELECT  StudentID,Credits,
                            CASE
                                WHEN sum(Marks) >= 85 THEN 4.0
                                WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                                WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                                WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                                WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                                WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                                WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                                WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                                WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                                WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                                ELSE 0.0
                            END as gradepoint
                        FROM(
                            SELECT st.studentID as StudentID,c.courseID as CourseI
D,
                                a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks))
as Marks, c.numOfCredits as Credits
                            FROM spmapp_student_t st,
                                spmapp_registration_t r,
                                spmapp_section_t sc,
                                spmapp_course_t c,
```

```
                            spmapp_assessment_t a,
                            spmapp_evaluation_t e
                        WHERE st.studentID = r.student_id
                            and r.section_id = sc.sectionID
                            and sc.course_id = c.courseID
                            and r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and st.program_id = '{}'
                            and r.semester='{}'
                        GROUP BY  st.studentID,c.courseID,a.assessmentName) De
rived1
                    GROUP BY StudentID,CourseID) Derived2
                GROUP BY StudentID)
                    '''.format(program, semester))

        row = cursor.fetchall()[0][0]
    return np.round(row, 3)
```



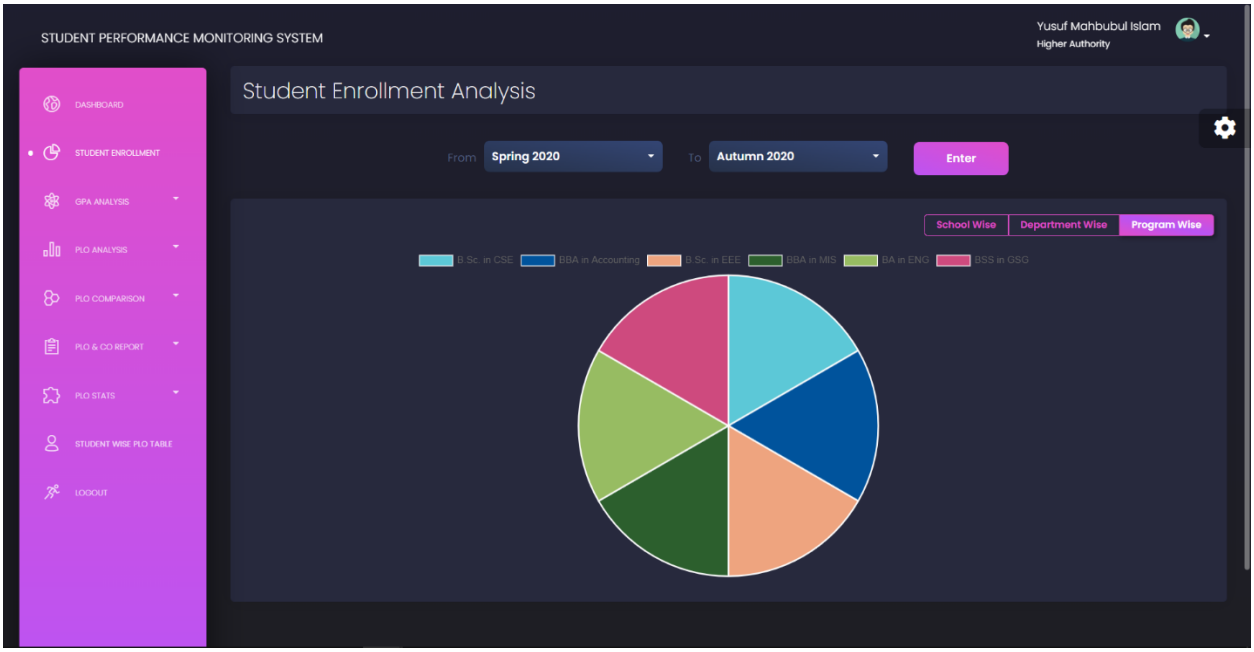

```
def getSchoolWiseEnrolledStudents(school, semesters):
    cursor = connection.cursor()

    if len(semesters) == 1:
        cursor.execute('''
            SELECT count( distinct st.studentID)
            FROM spmapp_school_t s,
                spmapp_department_t d,
                spmapp_student_t st,
                spmapp_registration_t r
            WHERE r.student_id = st.studentID
                and st.department_id = d.departmentID
```

```
                and d.school_id = s.schoolID
                 and s.schoolID = '{}'
                and r.semester = '{}'
            '''.format(school, semesters[0]))
        row = cursor.fetchall()
    else:
        cursor.execute('''
                    SELECT count( distinct st.studentID)
                    FROM spmapp_school_t s,
                        spmapp_department_t d,
                        spmapp_student_t st,
                        spmapp_registration_t r
                    WHERE r.student_id = st.studentID
                        and st.department_id = d.departmentID
                        and d.school_id = s.schoolID
                         and s.schoolID = '{}'
                        and r.semester in {}
                    '''.format(school, str(tuple(semesters))))
        row = cursor.fetchall()

    return row[0][0]
```



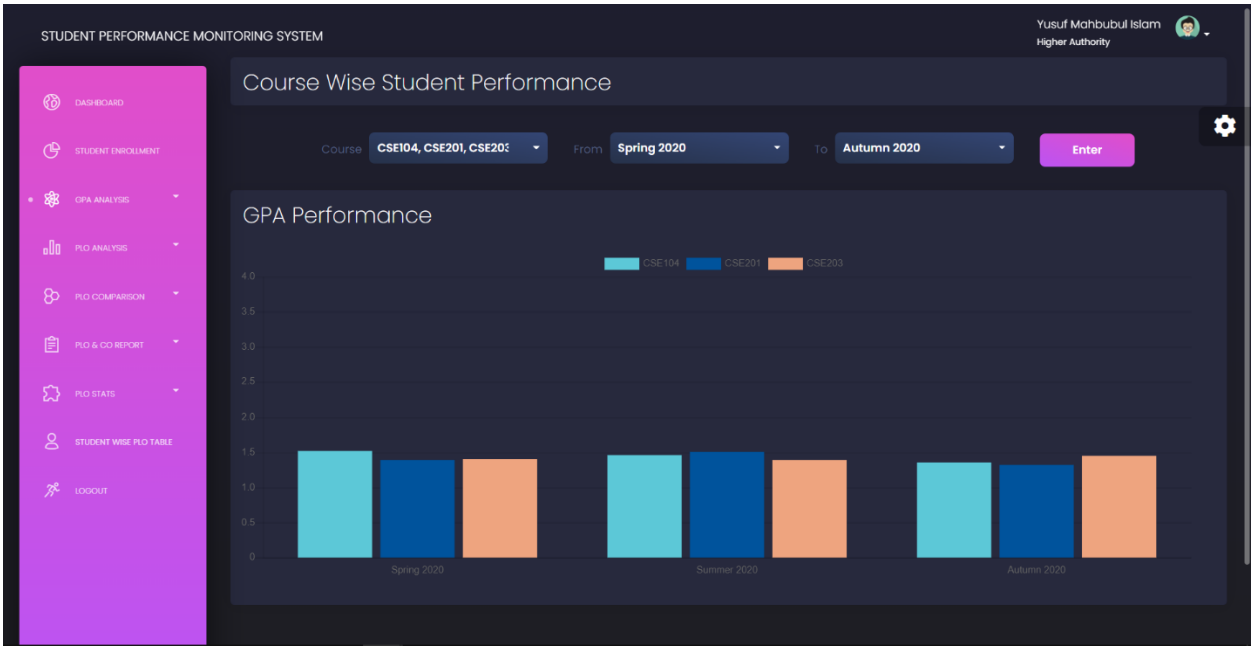

```
def getDeptWiseEnrolledStudents(dept, semesters):
    cursor = connection.cursor()
    if (len(semesters) == 1):
        cursor.execute('''
            SELECT count(distinct st.studentID)
            FROM spmapp_department_t d,
                spmapp_student_t st,
                spmapp_registration_t r
```

```
            WHERE r.student_id = st.studentID
                and st.department_id = '{}'
                and r.semester = '{}'
            '''.format(dept, semesters[0]))
        row = cursor.fetchall()
    else:
        cursor.execute('''
                    SELECT count(distinct st.studentID)
                    FROM spmapp_department_t d,
                        spmapp_student_t st,
                        spmapp_registration_t r
                    WHERE r.student_id = st.studentID
                        and st.department_id = '{}'
                        and r.semester in {}
                    '''.format(dept, str(tuple(semesters))))
        row = cursor.fetchall()

    return row[0][0]
```



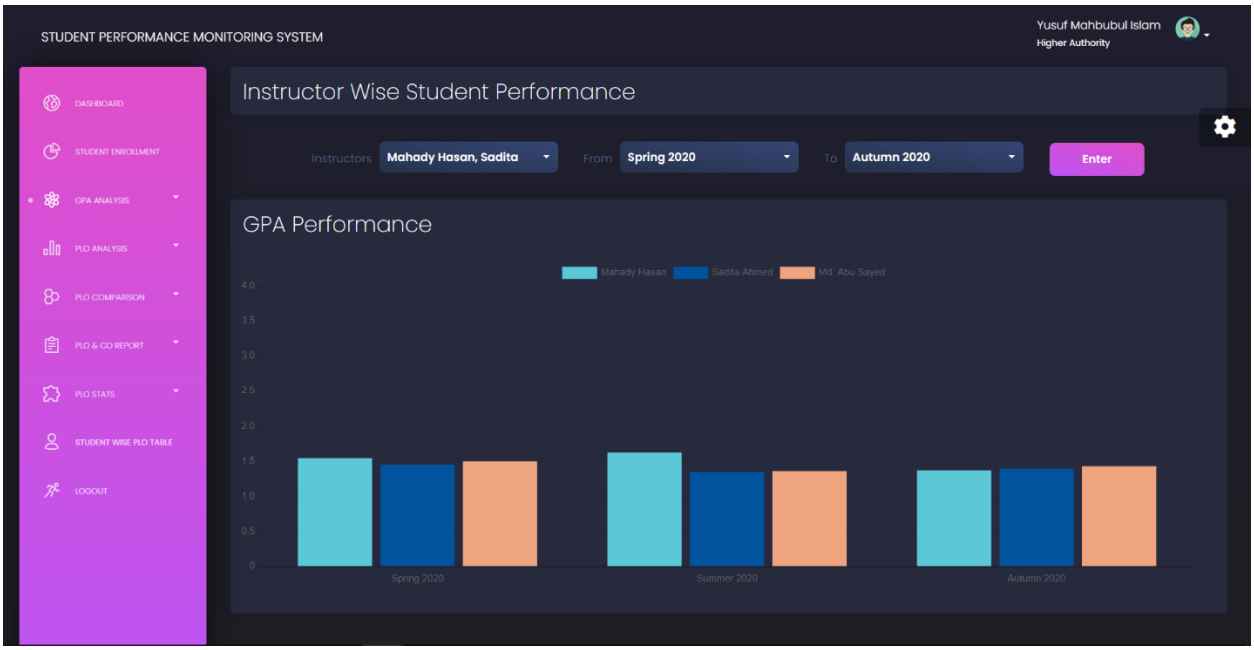

```
def getProgramWiseEnrolledStudents(program, semesters):
    cursor = connection.cursor()

    if len(semesters) == 1:
        cursor.execute('''
            SELECT count( distinct st.studentID)
            FROM spmapp_program_t p,
                spmapp_student_t st,
                spmapp_registration_t r
            WHERE r.student_id = st.studentID
                and st.program_id = p.programID
```

```
                and st.program_id = '{}'
                and r.semester = '{}'
            '''.format(program, semesters[0]))
        row = cursor.fetchall()

    else:
        cursor.execute('''
                    SELECT count( distinct st.studentID)
                    FROM spmapp_program_t p,
                        spmapp_student_t st,
                        spmapp_registration_t r
                    WHERE r.student_id = st.studentID
                        and st.program_id = p.programID
                        and st.program_id = '{}'
                        and r.semester in {}
                    '''.format(program, str(tuple(semesters))))
        row = cursor.fetchall()

    return row[0][0]
```



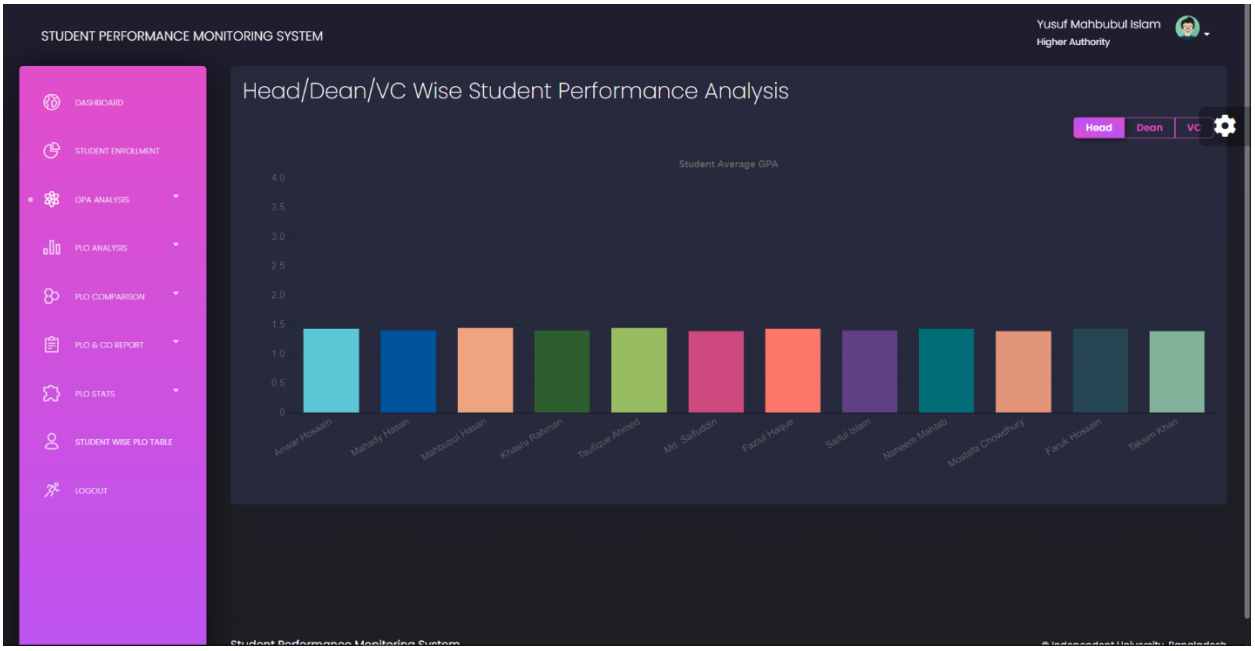

```
def getCourseWiseGPA(course, semester):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT AVG(gradepoint) as avgGrade
                FROM(
                        SELECT  StudentID,
                            CASE
                                WHEN sum(Marks) >= 85 THEN 4.0
                                WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                                WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
```

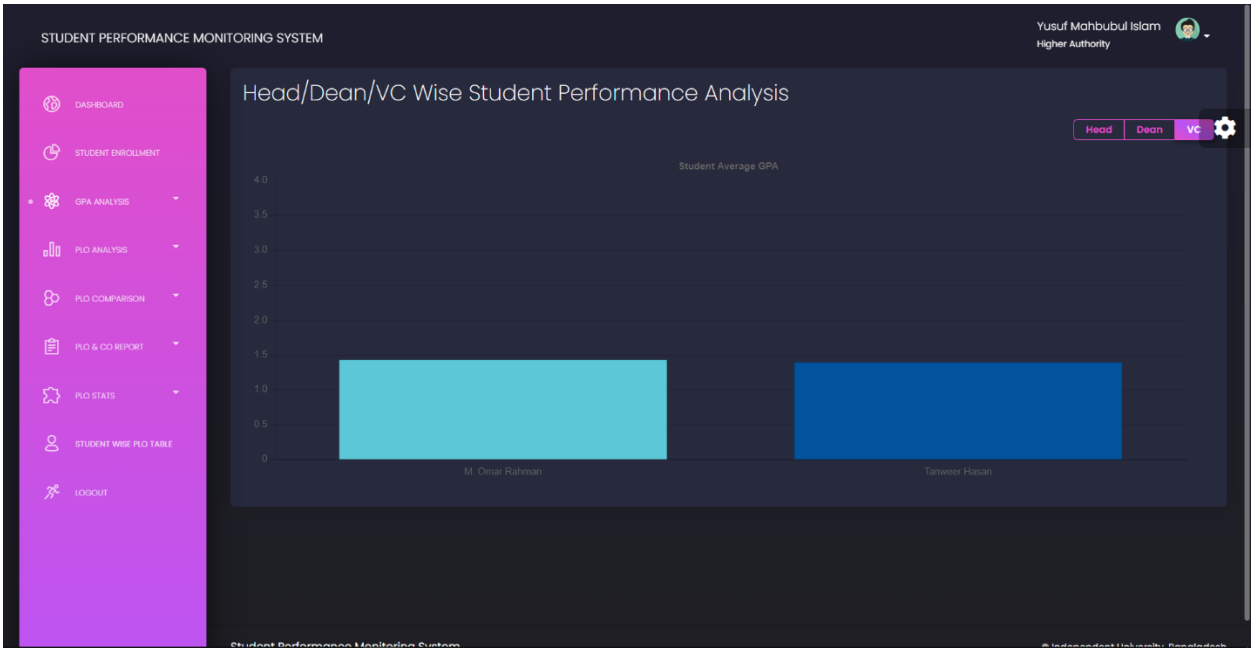
```
                            WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                            WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                            WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                            WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                            WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                            WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                            WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                            ELSE 0.0
                        END as gradepoint
                    FROM(
                        SELECT st.studentID as StudentID,c.courseID as CourseI
D,
                            a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks))
as Marks
                        FROM spmapp_student_t st,
                            spmapp_registration_t r,
                            spmapp_section_t sc,
                            spmapp_course_t c,
                            spmapp_assessment_t a,
                            spmapp_evaluation_t e
                        WHERE st.studentID = r.student_id
                            and r.section_id = sc.sectionID
                            and sc.course_id = c.courseID
                            and r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and c.courseID = '{}'
                            and r.semester='{}'
                        GROUP BY  st.studentID,a.assessmentName) Derived
                    GROUP BY StudentID) Derived2
                    '''.format(course, semester))

        row = cursor.fetchall()[0][0]
    return np.round(row, 3)
```




```
def getInstructorWiseGPA(instructor, semester):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT AVG(gradepoint) as avgGrade
                FROM(
                        SELECT  StudentID,
                            CASE
                                WHEN sum(Marks) >= 85 THEN 4.0
                                WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                                WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                                WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                                WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                                WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                                WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                                WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                                WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                                WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                                ELSE 0.0
                            END as gradepoint
                        FROM(
                            SELECT st.studentID as StudentID,c.courseID as CourseI
D,
                                a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks))
as Marks
                            FROM spmapp_student_t st,
                                spmapp_registration_t r,
                                spmapp_section_t sc,
                                spmapp_course_t c,
                                spmapp_assessment_t a,
                                spmapp_evaluation_t e
```

```
                        WHERE st.studentID = r.student_id
                            and r.section_id = sc.sectionID
                            and r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and sc.faculty_id = '{}'
                            and r.semester='{}'
                        GROUP BY  st.studentID,a.assessmentName) Derived
                    GROUP BY StudentID) Derived2
                    '''.format(instructor, semester))

    row = cursor.fetchall()[0][0]
return np.round(row, 3)
```



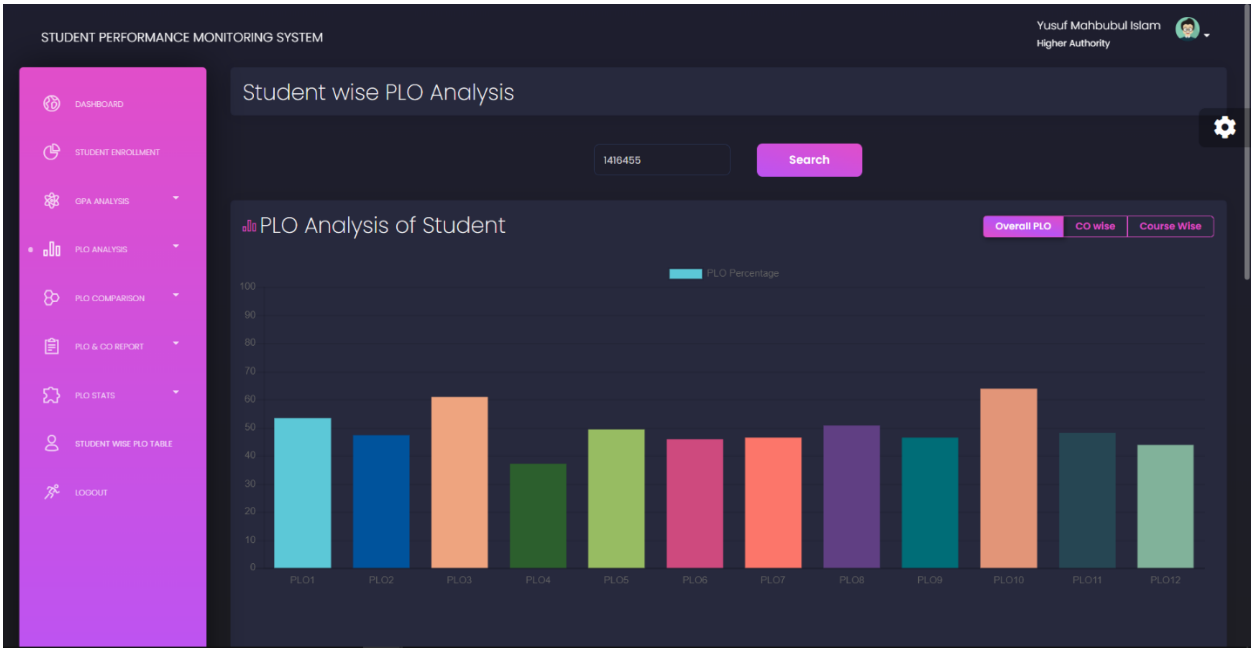



```
def getVCWiseGPA(vc):
    semlist = getAllSemesters()

    b = -1
    e = -1

    for s in range(0, len(semlist)):
        if semlist[s][0] == vc.startDate:
            b = s
        if vc.endDate == 'N/A':
            e = len(semlist) - 1
        elif semlist[s][0] == vc.endDate:
            e = s
    semesters = []

    for i in range(b, e + 1):
        semesters.append(semlist[i][0])

    cursor = connection.cursor()

    if len(semesters) == 1:
        cursor.execute('''
                SELECT AVG(grade) as avgGrade
                FROM(
                   SELECT StudentID,sum(Credits*gradepoint)/sum(Credits) as grade
                    FROM(
                        SELECT  StudentID,Credits,
                            CASE
                                WHEN sum(Marks) >= 85 THEN 4.0
                                WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
```

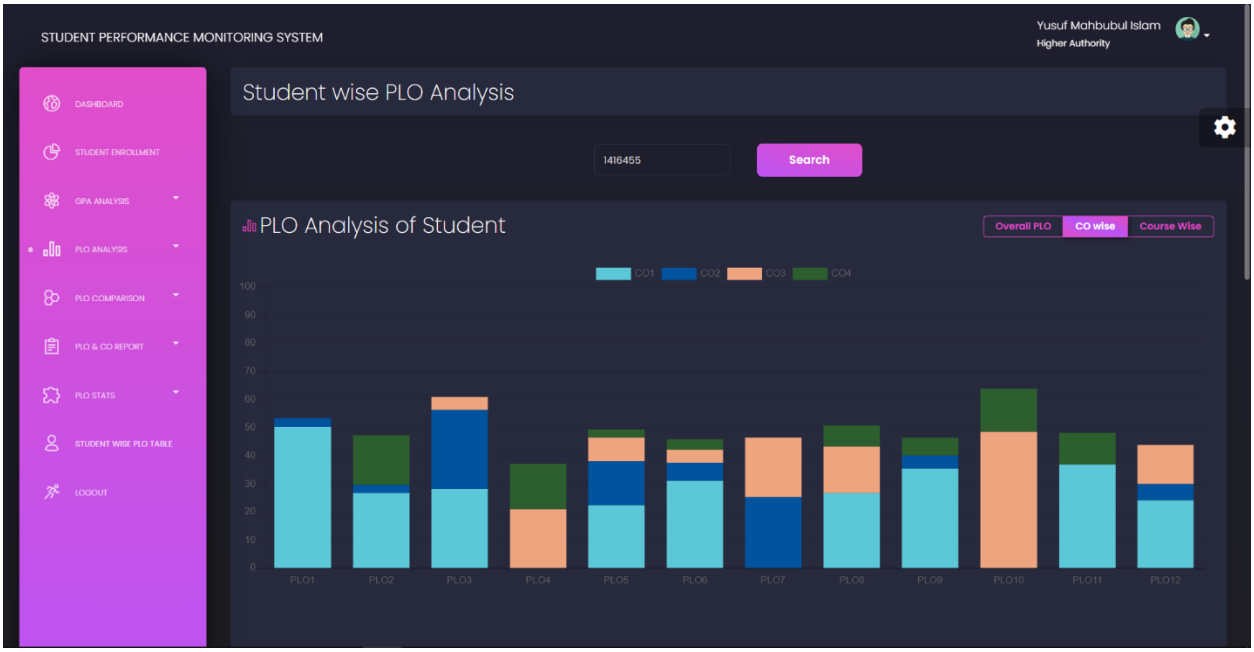
```
                                WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                                WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                                WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                                WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                                WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                                WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                                WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.
                                WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                                  ELSE 0.0
                           END as gradepoint
                       FROM(
                         SELECT st.studentID as StudentID,c.courseID as CourseID,
a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks)) as Marks, c.numOfCredits as Cre
dits
                           FROM spmapp_student_t st,
                               spmapp_registration_t r,
                               spmapp_section_t sc,
                               spmapp_course_t c,
                               spmapp_assessment_t a,
                               spmapp_evaluation_t e
                           WHERE st.studentID = r.student_id
                                and r.section_id = sc.sectionID
                                and sc.course_id = c.courseID
                                and r.registrationID = e.registration_id
                                and e.assessment_id = a.assessmentID
                                and r.semester='{}'
                     GROUP BY  st.studentID,c.courseID,a.assessmentName) Derived1
                       GROUP BY StudentID,CourseID) Derived2
                  GROUP BY StudentID)
                      '''.format(semesters[0]))
    else:
        print(semesters)
        print(b, e)
        cursor.execute('''
                       SELECT AVG(grade) as avgGrade
                FROM(
                   SELECT StudentID,sum(Credits*gradepoint)/sum(Credits) as grade
                    FROM(
                        SELECT  StudentID,Credits,
                            CASE
                                WHEN sum(Marks) >= 85 THEN 4.0
                                WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                                WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                                WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                                WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                                WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                                WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
```

```
                                WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                                WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                                WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                                  ELSE 0.0
                            END as gradepoint
                        FROM(
                          SELECT st.studentID as StudentID,c.courseID as CourseID,
a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks)) as Marks, c.numOfCredits as Cre
dits
                            FROM spmapp_student_t st,
                                spmapp_registration_t r,
                                spmapp_section_t sc,
                                spmapp_course_t c,
                                spmapp_assessment_t a,
                                spmapp_evaluation_t e
                            WHERE st.studentID = r.student_id
                                 and r.section_id = sc.sectionID
                                 and sc.course_id = c.courseID
                                 and r.registrationID = e.registration_id
                                 and e.assessment_id = a.assessmentID
                                 and r.semester in {}
                      GROUP BY  st.studentID,c.courseID,a.assessmentName) Derived1
                        GROUP BY StudentID,CourseID) Derived2
                    GROUP BY StudentID)
                        '''.format(str(tuple(semesters))))
    row = cursor.fetchall()[0][0]

    return row
```

```
def getInstructorWiseGPAForCourse(course, semester):
    with connection.cursor() as cursor:
        cursor.execute('''
               SELECT FacultyID, AVG(gradepoint) as avgGrade
               FROM(
                       SELECT  FacultyID,StudentID,
                           CASE
                               WHEN sum(Marks) >= 85 THEN 4.0
                               WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                               WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                               WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                               WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                               WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                               WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                               WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                               WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                               WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                               ELSE 0.0
                           END as gradepoint
                       FROM(
SELECT sc.faculty_id as FacultyID, st.studentID as StudentID,c.courseID as Course
ID,
                      a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks)) as Marks
                           FROM spmapp_student_t st,
                               spmapp_registration_t r,
                               spmapp_section_t sc,
                               spmapp_course_t c,
                               spmapp_assessment_t a,
                               spmapp_evaluation_t e
                           WHERE st.studentID = r.student_id
                               and r.section_id = sc.sectionID
                               and r.registrationID = e.registration_id
                               and e.assessment_id = a.assessmentID
                               and sc.course_id = '{}'
                               and r.semester='{}'
                   GROUP BY  sc.faculty_id,st.studentID,a.assessmentName) Derived
                       GROUP BY FacultyID,StudentID) Derived2
               GROUP BY FacultyID
                       '''.format(course, semester))

        row = cursor.fetchall()
    return row
```

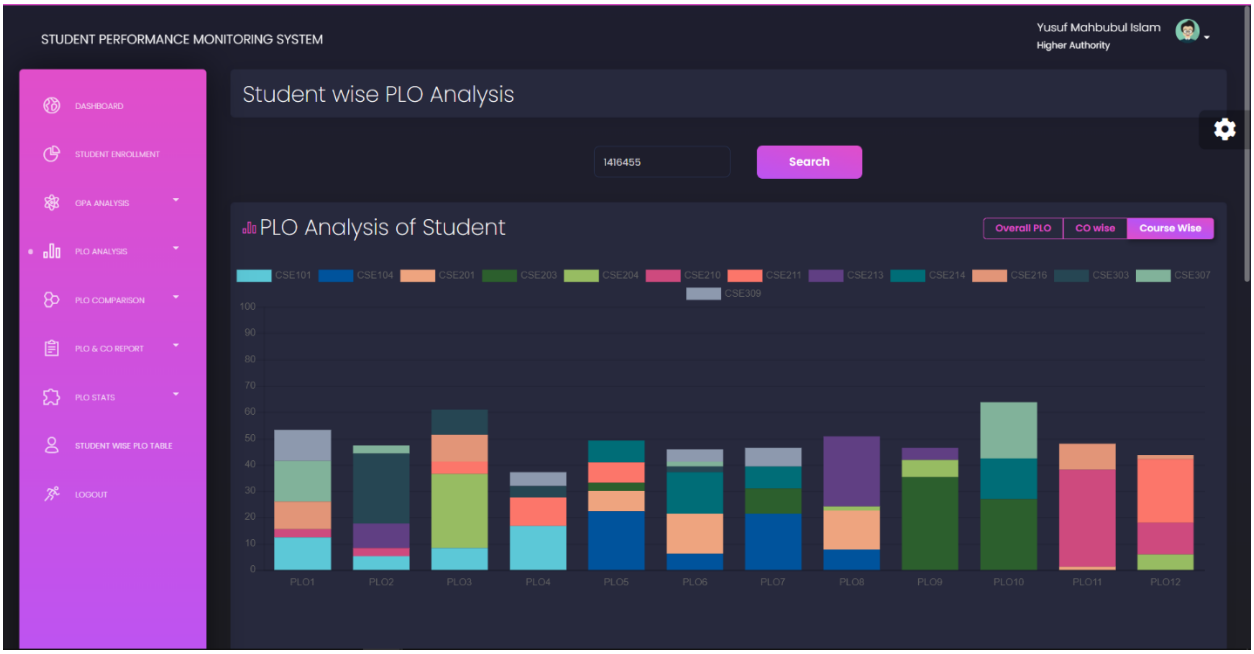



```
def getStudentWisePLO(studentID):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT p.ploNum as plonum,100*(sum( e.obtainedMarks)/sum( a.total
Marks)) as plopercent
                FROM spmapp_registration_t r,
                    spmapp_assessment_t a,
                    spmapp_evaluation_t e,
                    spmapp_co_t co,
                    spmapp_plo_t p
                WHERE  r.registrationID = e.registration_id
                    and e.assessment_id = a.assessmentID
                    and a.co_id=co.coID
                    and co.plo_id = p.ploID
                    and  r.student_id = '{}'
                GROUP BY  p.ploID

                '''.format(studentID))
        row = cursor.fetchall()
    return row
```



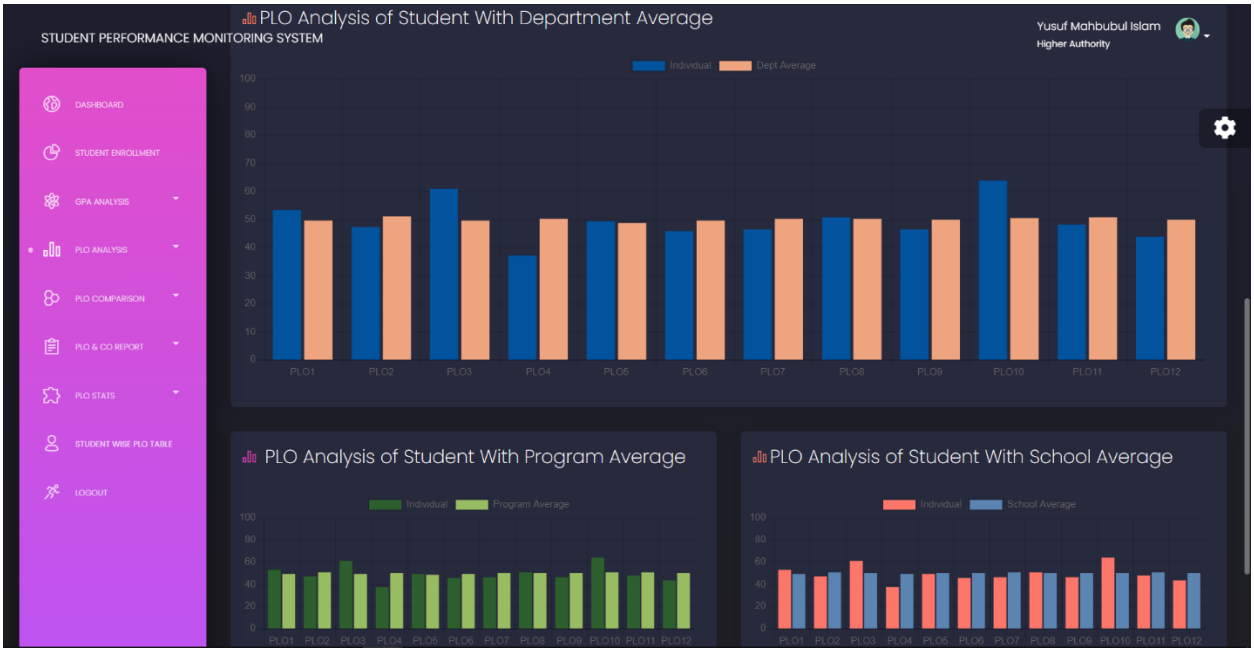

```
def getCOWiseStudentPLO(studentID, cat):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT p.ploNum as ploNum,co.coNum, sum(e.obtainedMarks),sum(a.tot
alMarks),derived.Total
                FROM spmapp_registration_t r,
                    spmapp_assessment_t a,
                    spmapp_evaluation_t e,
                    spmapp_co_t co,
                    spmapp_plo_t p,
                    (
                        SELECT p.ploNum as ploNum,sum(a.totalMarks) as Total, r.s
tudent_id as StudentID
                        FROM spmapp_registration_t r,
                            spmapp_assessment_t a,
                            spmapp_evaluation_t e,
                            spmapp_co_t co,
                            spmapp_plo_t p
                        WHERE r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and a.co_id=co.coID
                            and co.plo_id = p.ploID
                            and r.student_id = '{}'
                        GROUP BY  r.student_id,p.ploID) derived
                WHERE r.student_id = derived.StudentID
                     and e.registration_id = r.registrationID
                     and e.assessment_id = a.assessmentID
                     and a.co_id=co.coID
                     and co.plo_id = p.ploID
                     and p.ploNum = derived.ploNum
```

```
                GROUP BY  p.ploID,co.coNum

                '''.format(studentID))
        row = cursor.fetchall()

    table = []
    cos = []

    for entry in row:
        if entry[1] not in cos:
            cos.append(entry[1])
    cos.sort()
    plo = ["PLO1", "PLO2", "PLO3", "PLO4", "PLO5", "PLO6", "PLO7", "PLO8", "PLO9"
, "PLO10", "PLO11", "PLO12"]

    for i in cos:
        temptable = []
        if cat == 'report':
            temptable = [i]

        for j in plo:
            found = False
            for k in row:
                if j == k[0] and i == k[1]:
                    if cat == 'report':
                        temptable.append(np.round(100 * k[2] / k[3], 2))
                    elif cat == 'chart':
                        temptable.append(np.round(100 * k[2] / k[4], 2))
                    found = True
            if not found:
                if cat == 'report':
                    temptable.append('N/A')
                elif cat == 'chart':
                    temptable.append(0)
        table.append(temptable)
    return plo, cos, table
```




```
def getCourseWiseStudentPLO(studentID, cat):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT p.ploNum as ploNum,co.course_id,sum(e.obtainedMarks),sum(a.
totalMarks), derived.Total
                FROM spmapp_registration_t r,
                    spmapp_assessment_t a,
                    spmapp_evaluation_t e,
                    spmapp_co_t co,
                    spmapp_plo_t p,
                    (
                        SELECT p.ploNum as ploNum,sum(a.totalMarks) as Total, r.s
tudent_id as StudentID
                        FROM spmapp_registration_t r,
                            spmapp_assessment_t a,
                            spmapp_evaluation_t e,
                            spmapp_co_t co,
                            spmapp_plo_t p
                        WHERE r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and a.co_id=co.coID
                            and co.plo_id = p.ploID
                            and r.student_id = '{}'
                        GROUP BY  r.student_id,p.ploID) derived
                WHERE r.student_id = derived.StudentID
                     and e.registration_id = r.registrationID
                     and e.assessment_id = a.assessmentID
                     and a.co_id=co.coID
                     and co.plo_id = p.ploID
                     and p.ploNum = derived.ploNum
```

```
                GROUP BY  p.ploID,co.course_id

                '''.format(studentID))
        row = cursor.fetchall()

    table = []
    courses = []

    for entry in row:
        if entry[1] not in courses:
            courses.append(entry[1])
    courses.sort()
    plo = ["PLO1", "PLO2", "PLO3", "PLO4", "PLO5", "PLO6", "PLO7", "PLO8", "PLO9"
, "PLO10", "PLO11", "PLO12"]

    for i in courses:
        temptable = []
        if cat == 'report':
            temptable = [i]

        for j in plo:
            found = False
            for k in row:
                if j == k[0] and i == k[1]:
                    if cat == 'report':
                        temptable.append(np.round(100 * k[2] / k[3], 2))
                    elif cat == 'chart':
                        temptable.append(np.round(100 * k[2] / k[4], 2))
                    found = True
            if not found:
                if cat == 'report':
                    temptable.append('N/A')
                elif cat == 'chart':
                    temptable.append(0)
        table.append(temptable)
    return plo, courses, table
```




```
def getDeptWisePLO(dept):
    with connection.cursor() as cursor:
        cursor.execute('''
             SELECT derived.plonum, avg(per)
             FROM(
                SELECT p.ploID as PLOID,p.ploNum as ploNum, 100*sum(e.obtainedMar
ks)/sum(a.TotalMarks) as per
                FROM spmapp_registration_t r,
                    spmapp_evaluation_t e,
                    spmapp_student_t st,
                    spmapp_department_t d,
                    spmapp_assessment_t a,
                    spmapp_co_t c,
                    spmapp_plo_t p
                WHERE r.student_id = st.studentID
                    and st.department_id = d.departmentID
                    and e.registration_id = r.registrationID
                    and a.assessmentID = e.assessment_id
                    and a.co_id = c.coID
                    and c.plo_id = p.ploID
                    and st.department_id = '{}'
                    GROUP BY p.ploNum,r.student_id) derived
             GROUP BY derived.ploNum
                   '''.format(dept))
        row = cursor.fetchall()
        row.sort(key=len)
    return row


def getProgramWisePLO(program):
```

```
    with connection.cursor() as cursor:
        cursor.execute('''
             SELECT derived.plonum, avg(per)
             FROM(
                SELECT p.ploID as PLOID, p.ploNum as ploNum, 100*sum(e.obtainedMa
rks)/sum(a.TotalMarks) as per
                FROM spmapp_registration_t r,
                    spmapp_evaluation_t e,
                    spmapp_student_t st,
                    spmapp_program_t p,
                    spmapp_assessment_t a,
                    spmapp_co_t c,
                    spmapp_plo_t p
                WHERE r.student_id = st.studentID
                    and st.program_id = p.programID
                    and e.registration_id = r.registrationID
                    and a.assessmentID = e.assessment_id
                    and a.co_id = c.coID
                    and c.plo_id = p.ploID
                    and st.program_id = '{}'
                    GROUP BY p.ploID,r.student_id) derived
             GROUP BY derived.PLOID
                  '''.format(program))
        row = cursor.fetchall()

    return row
```


```
def getStudentWisePLOComp(student, semester):
    cursor = connection.cursor()
```

```
    cursor.execute('''
        SELECT  COUNT(marks)
        FROM(
            SELECT p.ploNum as ploNum,100*sum(e.obtainedMarks)/sum(a.totalMarks)
as marks
            FROM spmapp_registration_t r,
                spmapp_evaluation_t e,
                spmapp_assessment_t a,
                spmapp_co_t c,
                spmapp_plo_t p
            WHERE r.registrationID = e.registration_id
                and e.assessment_id = a.assessmentID
                and a.co_id = c.coID
                and c.plo_id = p.ploID
                and r.student_id='{}'
                and r.semester ='{}'
            GROUP BY p.ploNum,c.course_id) derived1


    '''.format(student, semester))

    expected = cursor.fetchall()

    cursor.execute('''
           SELECT COUNT(marks)
           FROM(
               SELECT p.ploNum as ploNum,100*sum(e.obtainedMarks)/sum(a.totalMark
s) as marks
               FROM spmapp_registration_t r,
                   spmapp_evaluation_t e,
                   spmapp_assessment_t a,
                   spmapp_co_t c,
                   spmapp_plo_t p
               WHERE r.registrationID = e.registration_id
                   and e.assessment_id = a.assessmentID
                   and a.co_id = c.coID
                   and c.plo_id = p.ploID
                   and r.student_id = '{}'
                   and r.semester ='{}'
               GROUP BY p.ploNum,c.course_id
               HAVING 100*sum(e.obtainedMarks)/sum(a.totalMarks)>=40) derived1

       '''.format(student, semester))

    actual = cursor.fetchall()
```

```
    return expected[0][0], actual[0][0]
```


```
def getCourseWisePLOComp(course, semester):
    cursor = connection.cursor()

    cursor.execute('''
        SELECT ploNum, COUNT(marks)
        FROM(
            SELECT p.ploNum as ploNum,100*sum(e.obtainedMarks)/sum(a.totalMarks)
as marks
            FROM spmapp_registration_t r,
                spmapp_evaluation_t e,
                spmapp_assessment_t a,
                spmapp_co_t c,
                spmapp_plo_t p
            WHERE r.registrationID = e.registration_id
                and e.assessment_id = a.assessmentID
                and a.co_id = c.coID
                and c.plo_id = p.ploID
                and c.course_id = '{}'
                and r.semester ='{}'
            GROUP BY p.ploNum,r.student_id) derived1
        GROUP BY ploNum

    '''.format(course, semester))

    temp1 = cursor.fetchall()
    temp1.sort(key=lambda t: len(t[0]))

    expected = temp1[0][1]
```

```
    cursor.execute('''
           SELECT ploNum, COUNT(marks)
           FROM(
               SELECT p.ploNum as ploNum,100*sum(e.obtainedMarks)/sum(a.totalMark
s) as marks
               FROM spmapp_registration_t r,
                   spmapp_evaluation_t e,
                   spmapp_assessment_t a,
                   spmapp_co_t c,
                   spmapp_plo_t p
               WHERE r.registrationID = e.registration_id
                   and e.assessment_id = a.assessmentID
                   and a.co_id = c.coID
                   and c.plo_id = p.ploID
                   and c.course_id = '{}'
                   and r.semester ='{}'
               GROUP BY p.ploNum,r.student_id
               HAVING 100*sum(e.obtainedMarks)/sum(a.totalMarks)>=40) derived1
           GROUP BY ploNum
       '''.format(course, semester))

    actual = []
    temp2 = cursor.fetchall()
    temp1.sort(key=lambda t: len(t[0]))

    plo = []

    for i in temp2:
        plo.append(i[0])
        actual.append(i[1])

    return plo, expected, actual
```


```
def getProgramWisePLOComp(program, semester):
    cursor = connection.cursor()
    cursor.execute('''
        SELECT ploNum,COUNT(*)
        FROM(
            SELECT p.ploNum as ploNum, c.course_id, r.student_id, 100*(sum(e.obta
inedMarks)/sum(a.totalMarks))
            FROM spmapp_student_t st,
                spmapp_registration_t r,
                spmapp_program_t pr,
                spmapp_evaluation_t e,
                spmapp_assessment_t a,
                spmapp_co_t c,
                spmapp_plo_t p
            WHERE st.studentID = r.student_id
                and e.registration_id = r.registrationID
                and a.assessmentID = e.assessment_id
                and a.co_id = c.coID
                and c.plo_id = p.ploID
                and st.program_id = pr.programID
                and pr.programID = '{}'
                and r.semester = '{}'
            GROUP BY p.ploNum, c.course_id, r.student_id) derived
        GROUP BY  derived.ploNum

    '''.format(program, semester))

    row1 = cursor.fetchall()
    row1.sort(key=lambda t: len(t[0]))
```

```
    cursor.execute('''
            SELECT ploNum,COUNT(*)
            FROM(
                SELECT p.ploNum as ploNum, c.course_id, r.student_id, 100*(sum(e.
obtainedMarks)/sum(a.totalMarks))
                FROM spmapp_student_t st,
                    spmapp_registration_t r,
                    spmapp_program_t pr,
                    spmapp_evaluation_t e,
                    spmapp_assessment_t a,
                    spmapp_co_t c,
                    spmapp_plo_t p
                WHERE st.studentID = r.student_id
                    and e.registration_id = r.registrationID
                    and a.assessmentID = e.assessment_id
                    and a.co_id = c.coID
                    and c.plo_id = p.ploID
                    and st.program_id = pr.programID
                    and pr.programID = '{}'
                    and r.semester = '{}'
                GROUP BY p.ploID, c.course_id, r.student_id
                HAVING  100*(sum(e.obtainedMarks)/sum(a.totalMarks))>=40) derived
            GROUP BY  derived.ploNum

        '''.format(program, semester))

    row2 = cursor.fetchall()
    row2.sort(key=lambda t: len(t[0]))

    plo = []
    expected = []
    actual = []

    for r in row1:
        plo.append(r[0])
        expected.append(r[1])

    for r in row2:
        actual.append(r[1])

    return plo, expected, actual
```


```
def getDeptWisePLOComp(dept, semester):
    cursor = connection.cursor()
    cursor.execute('''
        SELECT ploNum,COUNT(*)
        FROM(
            SELECT p.ploNum as ploNum, c.course_id, r.student_id, 100*(sum(e.obta
inedMarks)/sum(a.totalMarks))
            FROM spmapp_student_t st,
                spmapp_registration_t r,
                spmapp_department_t d,
                spmapp_evaluation_t e,
                spmapp_assessment_t a,
                spmapp_co_t c,
                spmapp_plo_t p
            WHERE st.studentID = r.student_id
                and e.registration_id = r.registrationID
                and a.assessmentID = e.assessment_id
                and a.co_id = c.coID
                and c.plo_id = p.ploID
                and st.department_id = d.departmentID
                and d.departmentID = '{}'
                and r.semester = '{}'
            GROUP BY p.ploNum, c.course_id, r.student_id) derived
        GROUP BY  derived.ploNum

    '''.format(dept, semester))

    row1 = cursor.fetchall()
    row1.sort(key=lambda t: len(t[0]))
```

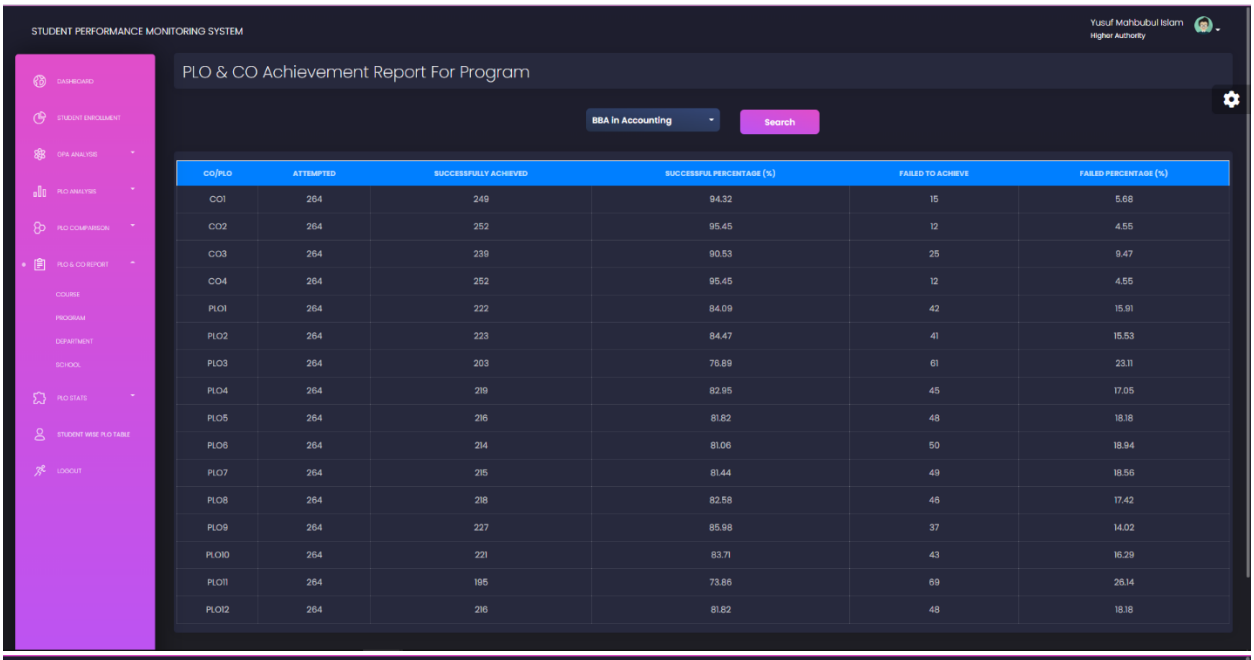
```
    cursor.execute('''
            SELECT ploNum,COUNT(*)
            FROM(
                SELECT p.ploNum as ploNum, c.course_id, r.student_id, 100*(sum(e.
obtainedMarks)/sum(a.totalMarks))
                FROM spmapp_student_t st,
                    spmapp_registration_t r,
                    spmapp_department_t d,
                    spmapp_evaluation_t e,
                    spmapp_assessment_t a,
                    spmapp_co_t c,
                    spmapp_plo_t p
                WHERE st.studentID = r.student_id
                    and e.registration_id = r.registrationID
                    and a.assessmentID = e.assessment_id
                    and a.co_id = c.coID
                    and c.plo_id = p.ploID
                    and st.department_id = d.departmentID
                    and d.departmentID = '{}'
                    and r.semester = '{}'
                GROUP BY p.ploNum, c.course_id, r.student_id
                HAVING  100*(sum(e.obtainedMarks)/sum(a.totalMarks))>=40) derived
            GROUP BY  derived.ploNum

        '''.format(dept, semester))

    row2 = cursor.fetchall()
    row2.sort(key=lambda t: len(t[0]))

    plo = []
    expected = []
    actual = []

    for r in row1:
        plo.append(r[0])
        expected.append(r[1])

    for r in row2:
        actual.append(r[1])

    return plo, expected,actual
```


```
def getSchoolWisePLOComp(school, semester):
    cursor = connection.cursor()
    cursor.execute('''
        SELECT ploNum,COUNT(*)
        FROM(
            SELECT p.ploNum as ploNum, c.course_id, r.student_id, 100*(sum(e.obta
inedMarks)/sum(a.totalMarks))
            FROM spmapp_student_t st,
                spmapp_registration_t r,
                spmapp_department_t d,
                spmapp_school_t s,
                spmapp_evaluation_t e,
                spmapp_assessment_t a,
                spmapp_co_t c,
                spmapp_plo_t p
            WHERE st.studentID = r.student_id
                and e.registration_id = r.registrationID
                and a.assessmentID = e.assessment_id
                and a.co_id = c.coID
                and c.plo_id = p.ploID
                and st.department_id = d.departmentID
                and d.school_id = s.schoolID
                and s.schoolID = '{}'
                and r.semester = '{}'
            GROUP BY p.ploNum, c.course_id, r.student_id) derived
        GROUP BY  derived.ploNum

    '''.format(school, semester))

    row1 = cursor.fetchall()
```

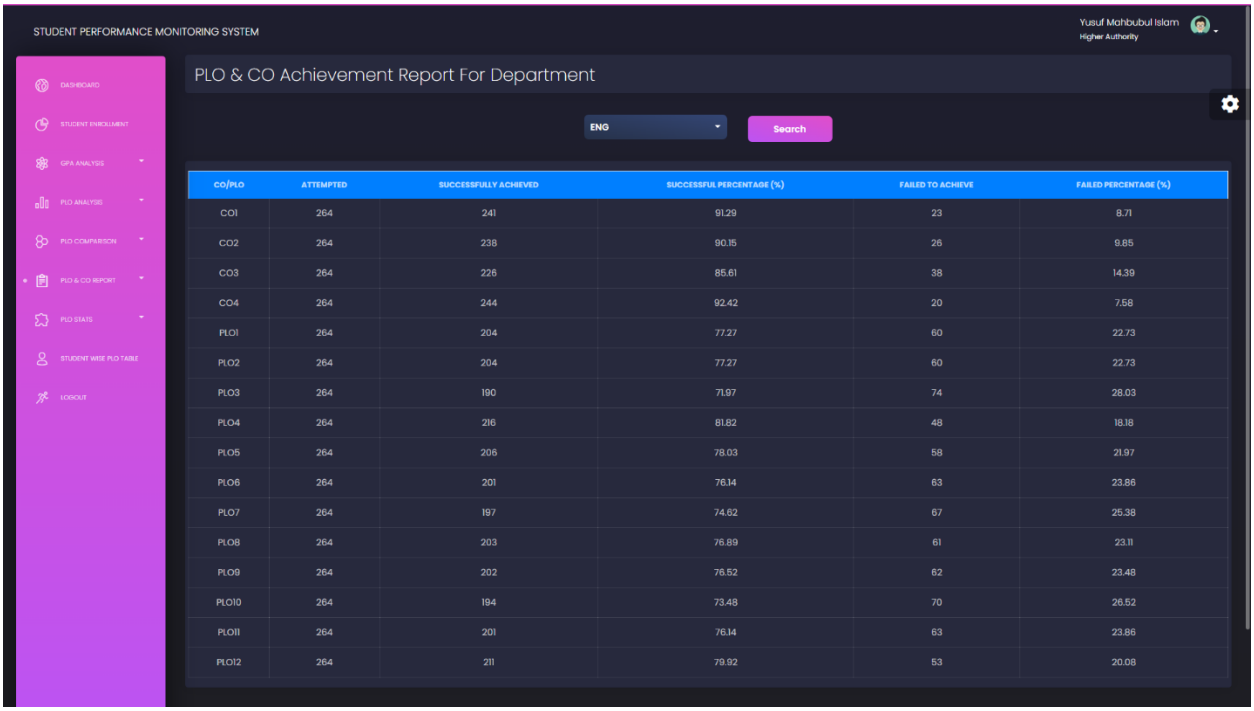
```
    row1.sort(key=lambda t: len(t[0]))

    cursor.execute('''
            SELECT ploNum,COUNT(*)
            FROM(
                SELECT p.ploNum as ploNum, c.course_id, r.student_id, 100*(sum(e.
obtainedMarks)/sum(a.totalMarks))
                FROM spmapp_student_t st,
                    spmapp_registration_t r,
                    spmapp_department_t d,
                    spmapp_school_t s,
                    spmapp_evaluation_t e,
                    spmapp_assessment_t a,
                    spmapp_co_t c,
                    spmapp_plo_t p
                WHERE st.studentID = r.student_id
                    and e.registration_id = r.registrationID
                    and a.assessmentID = e.assessment_id
                    and a.co_id = c.coID
                    and c.plo_id = p.ploID
                    and st.department_id = d.departmentID
                    and d.school_id = s.schoolID
                    and s.schoolID = '{}'
                    and r.semester = '{}'
                GROUP BY p.ploNum, c.course_id, r.student_id
                HAVING  100*(sum(e.obtainedMarks)/sum(a.totalMarks))>=40) derived
            GROUP BY  derived.ploNum
        '''.format(school, semester))

    row2 = cursor.fetchall()
    row2.sort(key=lambda t: len(t[0]))

    plo = []
    expected = []
    actual = []

    for r in row1:
        plo.append(r[0])
        expected.append(r[1])

    for r in row2:
        actual.append(r[1])

    return plo, expected, actual
```


```
def getCourseReport(course):
    row = []
    total = 0
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT coNum, ploNum, COUNT(marks)
                FROM(
                        SELECT c.coNum as coNum,p.ploNum as ploNum,100*sum(e.obtai
nedMarks)/sum(a.totalMarks) as marks
                        FROM spmapp_registration_t r,
                            spmapp_evaluation_t e,
                            spmapp_assessment_t a,
                            spmapp_co_t c,
                            spmapp_plo_t p
                        WHERE r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and a.co_id = c.coID
                            and c.plo_id = p.ploID
                             and c.course_id = '{}'
                        GROUP BY r.student_id,c.course_id,c.coID, p.ploID
                        )derived

                WHERE marks>=40
                GROUP BY coNum,ploNum

                '''.format(course))
        row = cursor.fetchall()
        if row is None:
            row = []
```

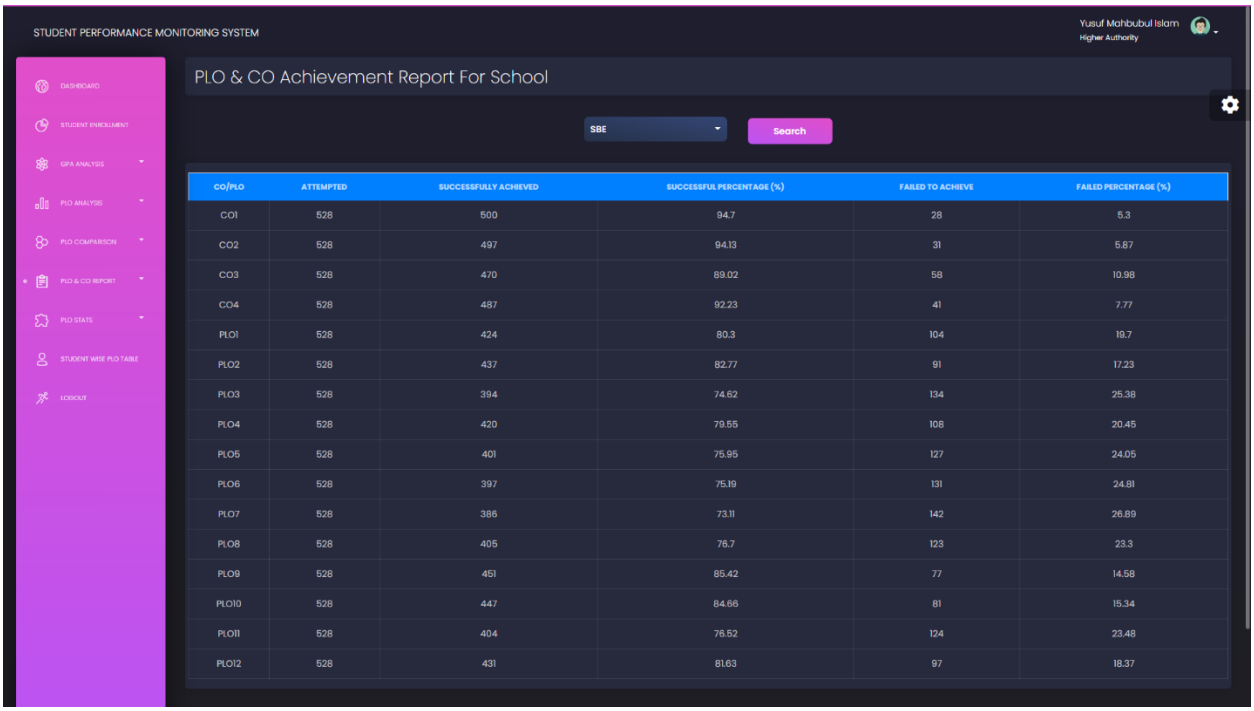
```
    with connection.cursor() as cursor:
        cursor.execute('''
               SELECT coNum, ploNum, COUNT(marks)
               FROM(
                       SELECT r.student_id as StudentID,c.course_id as CourseID,c
.coNum as coNum,
                       p.ploNum as ploNum,100*sum(e.obtainedMarks)/sum(a.totalMar
ks) as marks
                       FROM spmapp_registration_t r,
                           spmapp_evaluation_t e,
                           spmapp_assessment_t a,
                           spmapp_co_t c,
                           spmapp_plo_t p
                       WHERE r.registrationID = e.registration_id
                           and e.assessment_id = a.assessmentID
                           and a.co_id = c.coID
                           and c.plo_id = p.ploID
                            and c.course_id = '{}'
                       GROUP BY r.student_id,c.course_id,c.coID, p.ploID
                       )derived
                GROUP BY CourseID,coNum,ploNum
               '''.format(course))
        total = cursor.fetchone()[2]

    coplo = []
    temp = []
    for i in row:
        temp.append(i[2])
        coplo.append([i[0], i[1]])
    temp = np.array(temp)

    success = np.round(temp / total * 100, 3)
    failCount = total - temp
    fail = np.round(failCount / total * 100, 3)
    row = np.column_stack((temp, success, failCount, fail)).tolist()

    finalRow = []
    for i in range(len(row)):
        tempRow = coplo[i]
        for j in range(len(row[i])):
            tempRow.append(row[i][j])
        finalRow.append(tempRow)

    return (finalRow, total)
```



CO/PLO	ATTEMPTED	SUCCESSFULLY ACHIEVED	SUCCESSFUL PERCENTAGE (%)	FAILED TO ACHIEVE	FAILED PERCENTAGE (%)
CO1	264	249	94.32	15	5.68
CO2	264	252	95.45	12	4.55
CO3	264	239	90.53	25	9.47
CO4	264	252	95.45	12	4.55
PLO1	264	222	84.09	42	15.91
PLO2	264	223	84.47	41	15.53
PLO3	264	203	76.89	61	23.11
PLO4	264	219	82.95	45	17.05
PLO5	264	216	81.82	48	18.18
PLO6	264	214	81.06	50	18.94
PLO7	264	215	81.44	49	18.56
PLO8	264	218	82.58	46	17.42
PLO9	264	227	85.98	37	14.02
PLO10	264	221	83.71	43	16.29
PLO11	264	195	73.86	69	26.14
PLO12	264	216	81.82	48	18.18



CO/PLO	ATTEMPTED	SUCCESSFULLY ACHIEVED	SUCCESSFUL PERCENTAGE (%)	FAILED TO ACHIEVE	FAILED PERCENTAGE (%)
CO1	264	241	91.29	23	8.71
CO2	264	238	90.15	26	9.85
CO3	264	226	85.61	38	14.39
CO4	264	244	92.42	20	7.58
PLO1	264	204	77.27	60	22.73
PLO2	264	204	77.27	60	22.73
PLO3	264	190	71.97	74	28.03
PLO4	264	216	81.82	48	18.18
PLO5	264	206	78.03	58	21.97
PLO6	264	201	76.14	63	23.86
PLO7	264	197	74.62	67	25.38
PLO8	264	203	76.89	61	23.11
PLO9	264	202	76.52	62	23.48
PLO10	264	194	73.48	70	26.52
PLO11	264	201	76.14	63	23.86
PLO12	264	211	79.92	53	20.08



CO/PLO	ATTEMPTED	SUCCESSFULLY ACHIEVED	SUCCESSFUL PERCENTAGE (%)	FAILED TO ACHIEVE	FAILED PERCENTAGE (%)
CO1	528	500	94.7	28	5.3
CO2	528	497	94.13	31	5.87
CO3	528	470	89.02	58	10.98
CO4	528	487	92.23	41	7.77
PLO1	528	424	80.3	104	19.7
PLO2	528	437	82.77	91	17.23
PLO3	528	394	74.62	134	25.38
PLO4	528	420	79.55	108	20.45
PLO5	528	401	75.95	127	24.05
PLO6	528	397	75.19	131	24.81
PLO7	528	386	73.11	142	26.89
PLO8	528	405	76.7	123	23.3
PLO9	528	451	85.42	77	14.58
PLO10	528	447	84.66	81	15.34
PLO11	528	404	76.52	124	23.48
PLO12	528	431	81.63	97	18.37

```
def getSchoolReport(school):
    cursor = connection.cursor()

    cursor.execute('''
        SELECT coNum, COUNT(marks)
        FROM(
            SELECT c.coNum as coNum,100*sum(e.obtainedMarks)/sum(a.totalMarks) as
 marks
            FROM spmapp_student_t st,
                spmapp_department_t d,
                spmapp_school_t s,
                spmapp_registration_t r,
                spmapp_evaluation_t e,
                spmapp_assessment_t a,
                spmapp_co_t c
            WHERE st.studentID = r.student_id
                and st.department_id = d.departmentID
                and d.school_id = s.schoolID
                and r.registrationID = e.registration_id
                and e.assessment_id = a.assessmentID
                and a.co_id = c.coID
                and s.schoolID = '{}'
            GROUP BY c.coNum,r.student_id) derived
        GROUP BY coNum
    '''.format(school))

    row1 = cursor.fetchall()
```

```
    cursor.execute('''
            SELECT coNum, COUNT(marks)
            FROM(
                SELECT c.coNum as coNum,100*sum(e.obtainedMarks)/sum(a.totalMarks
) as marks
                FROM spmapp_student_t st,
                    spmapp_department_t d,
                    spmapp_school_t s,
                    spmapp_registration_t r,
                    spmapp_evaluation_t e,
                    spmapp_assessment_t a,
                    spmapp_co_t c
                WHERE st.studentID = r.student_id
                    and st.department_id = d.departmentID
                    and d.school_id = s.schoolID
                    and r.registrationID = e.registration_id
                    and e.assessment_id = a.assessmentID
                    and a.co_id = c.coID
                    and s.schoolID = '{}'
                    GROUP BY c.coNum,r.student_id) derived
            WHERE marks>=40
            GROUP BY coNum

        '''.format(school))

    row2 = cursor.fetchall()

    cursor.execute('''
            SELECT ploNum, COUNT(marks)
            FROM(
                SELECT p.ploNum as ploNum,100*sum(e.obtainedMarks)/sum(a.totalMar
ks) as marks
                FROM spmapp_student_t st,
                    spmapp_department_t d,
                    spmapp_school_t s,
                    spmapp_registration_t r,
                    spmapp_evaluation_t e,
                    spmapp_assessment_t a,
                    spmapp_co_t c,
                    spmapp_plo_t p
                WHERE st.studentID = r.student_id
                    and st.department_id = d.departmentID
                    and d.school_id = s.schoolID
                    and r.registrationID = e.registration_id
                    and e.assessment_id = a.assessmentID
                    and a.co_id = c.coID
                    and c.plo_id = p.ploID
```

```
                    and s.schoolID = '{}'
                    GROUP BY p.ploNum,r.student_id) derived
                GROUP BY ploNum
            '''.format(school))

        row3 = cursor.fetchall()

        row3.sort(key=lambda t: len(t[0]))

        cursor.execute('''
                    SELECT ploNum, COUNT(marks)
                    FROM(
                        SELECT p.ploNum as ploNum,100*sum(e.obtainedMarks)/sum(a.tota
lMarks) as marks
                        FROM spmapp_student_t st,
                            spmapp_department_t d,
                            spmapp_school_t s,
                            spmapp_registration_t r,
                            spmapp_evaluation_t e,
                            spmapp_assessment_t a,
                            spmapp_co_t c,
                            spmapp_plo_t p
                        WHERE st.studentID = r.student_id
                            and st.department_id = d.departmentID
                            and d.school_id = s.schoolID
                            and r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and a.co_id = c.coID
                            and c.plo_id = p.ploID
                            and s.schoolID = '{}'
                        GROUP BY p.ploNum,r.student_id) derived
                    WHERE marks>=40
                    GROUP BY ploNum
                '''.format(school))

        row4 = cursor.fetchall()

        row4.sort(key=lambda t: len(t[0]))

        finalrow = []

        for i in range(len(row1)):
            temp = []
            tot = row1[i][1]
            suc = row2[i][1]

            temp.append(row1[i][0])
```

```
        temp.append(tot)
        temp.append(suc)
        temp.append(np.round(100 * suc / tot, 2))
        temp.append(tot - suc)
        temp.append(np.round(100 * (tot - suc) / tot, 2))

        finalrow.append(temp)

    for i in range(len(row3)):
        temp = []
        tot = row3[i][1]
        suc = row4[i][1]

        temp.append(row3[i][0])
        temp.append(tot)
        temp.append(suc)
        temp.append(np.round(100 * suc / tot, 2))
        temp.append(tot - suc)
        temp.append(np.round(100 * (tot - suc) / tot, 2))

        finalrow.append(temp)

    return finalrow
```


```
def getProgramWisePLOStats(program):
    plo = ['PLO1', 'PLO2', 'PLO3', 'PLO4', 'PLO5', 'PLO6', 'PLO7', 'PLO8', 'PLO9'
, 'PLO10', 'PLO11', 'PLO12']
```

```
    achieved = []
    attempted = []

    for p in plo:
        with connection.cursor() as cursor:
            cursor.execute('''SELECT COUNT(*)
                FROM(SELECT AVG(percourse) as actual
                    FROM (SELECT r.student_id as StudentID, 100*sum(e.obtainedMar
ks)/sum(a.totalMarks) as percourse
                        FROM spmapp_registration_t r,
                            spmapp_evaluation_t e,
                            spmapp_assessment_t a,
                            spmapp_co_t c,
                            spmapp_plo_t p,
                            spmapp_program_t pr
                        WHERE r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and a.co_id = c.coID
                            and c.plo_id = p.ploID
                            and p.program_id = pr.programID
                            and pr.programID='{}'
                            and p.ploNum = '{}'
                        GROUP BY r.student_id,c.coID) per
                    GROUP BY per.StudentID) avgTable
            '''.format(program, p))
            row = cursor.fetchall()

            if row is not None:
                attempted.append(row[0][0])
            else:
                attempted.append(0)

    for p in plo:
        with connection.cursor() as cursor:
            cursor.execute('''SELECT COUNT(*)
               FROM(
                SELECT StudentID, AVG(percourse) as actual
                FROM(
                            SELECT r.student_id as StudentID, 100*sum(e.obtainedMa
rks)/sum(a.totalMarks) as percourse
                                FROM spmapp_registration_t r,
                                    spmapp_evaluation_t e,
                                    spmapp_assessment_t a,
                                    spmapp_co_t c,
                                    spmapp_plo_t p,
                                    spmapp_program_t pr
                                WHERE r.registrationID = e.registration_id
```

```
                                and e.assessment_id = a.assessmentID
                                and a.co_id = c.coID
                                and c.plo_id = p.ploID
                                and p.program_id = pr.programID
                                and pr.programID='{}'
                                and p.ploNum ='{}'
                            GROUP BY r.student_id,r.registrationID) d1
                        GROUP BY StudentID)d2
                        WHERE actual>=40
                ''.format(program, p))
            row = cursor.fetchall()

            if row is not None:
                achieved.append(row[0][0])
            else:
                achieved.append(0)

    return plo, achieved, attempted


    return plo, achieved, attempted
```


```
def getDeptWisePLOStats(dept):
    cursor = connection.cursor()
```

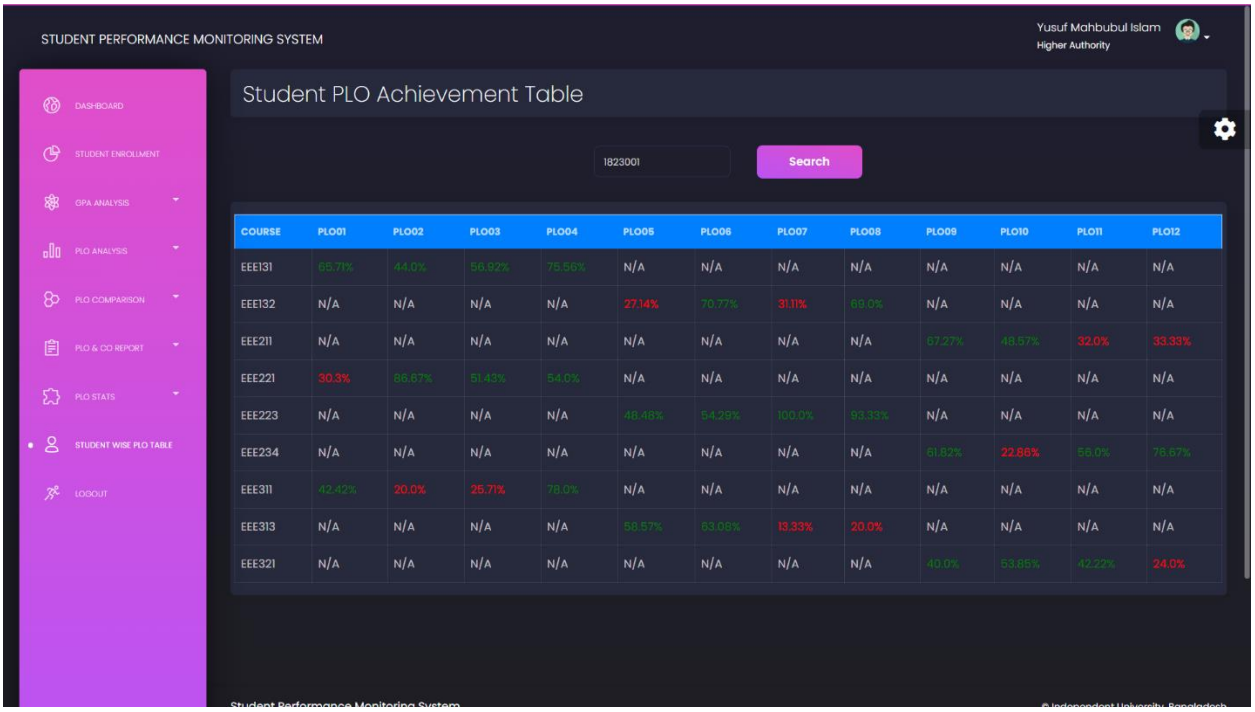
```
    cursor.execute('''
              SELECT ploNum,COUNT(Marks)
              FROM(
                    SELECT ploNum, StudentID, avg(coursemarks) as Marks
                    FROM(
                          SELECT p.ploNum as ploNum, r.student_id as StudentID,c.
course_id,
                                100*(sum(e.obtainedMarks)/sum(a.totalMarks)) as c
oursemarks
                          FROM spmapp_student_t st,
                              spmapp_registration_t r,
                              spmapp_department_t d,
                              spmapp_evaluation_t e,
                              spmapp_assessment_t a,
                              spmapp_co_t c,
                              spmapp_plo_t p
                          WHERE st.studentID = r.student_id
                              and e.registration_id = r.registrationID
                              and a.assessmentID = e.assessment_id
                              and a.co_id = c.coID
                              and c.plo_id = p.ploID
                              and st.department_id = d.departmentID
                              and d.departmentID = '{}'
                          GROUP BY p.ploNum, r.student_id,c.course_id) derived1
                    GROUP BY  ploNum,StudentID) derived2
                  GROUP BY ploNum

          '''.format(dept))

    row1 = cursor.fetchall()

    row1.sort(key=lambda t: len(t[0]))

    cursor.execute('''
                SELECT ploNum,COUNT(Marks)
                FROM(
                      SELECT ploNum, StudentID, avg(coursemarks) as Marks
                      FROM(
                            SELECT p.ploNum as ploNum, r.student_id as StudentI
D,c.course_id,
                                  100*(sum(e.obtainedMarks)/sum(a.totalMarks))
as coursemarks
                            FROM spmapp_student_t st,
                                spmapp_registration_t r,
                                spmapp_department_t d,
                                spmapp_evaluation_t e,
```

```
                                    spmapp_assessment_t a,
                                    spmapp_co_t c,
                                    spmapp_plo_t p
                                WHERE st.studentID = r.student_id
                                    and e.registration_id = r.registrationID
                                    and a.assessmentID = e.assessment_id
                                    and a.co_id = c.coID
                                    and c.plo_id = p.ploID
                                    and st.department_id = d.departmentID
                                    and d.departmentID = '{}'
                                GROUP BY p.ploNum, r.student_id,c.course_id) derive
d1
                            GROUP BY  ploNum,StudentID
                            HAVING avg(coursemarks)>=40) derived2
                        GROUP BY ploNum

                ''').format(dept))

    row2 = cursor.fetchall()
    row2.sort(key=lambda t: len(t[0]))

    plo = []
    attempted = []
    achieved = []

    for i in row1:
        plo.append(i[0])
        attempted.append(i[1])

    for i in row2:
        achieved.append(i[1])

    return plo, achieved, attempted
```


```
def getSchoolWisePLOStats(school):
    cursor = connection.cursor()

    cursor.execute('''
              SELECT ploNum,COUNT(Marks)
              FROM(
                    SELECT ploNum, StudentID, avg(coursemarks) as Marks
                    FROM(
                          SELECT p.ploNum as ploNum, r.student_id as StudentID,c.
course_id,
                                100*(sum(e.obtainedMarks)/sum(a.totalMarks)) as c
oursemarks
                          FROM spmapp_student_t st,
                              spmapp_registration_t r,
                              spmapp_department_t d,
                              spmapp_school_t s,
                              spmapp_evaluation_t e,
                              spmapp_assessment_t a,
                              spmapp_co_t c,
                              spmapp_plo_t p
                          WHERE st.studentID = r.student_id
                              and e.registration_id = r.registrationID
                              and a.assessmentID = e.assessment_id
                              and a.co_id = c.coID
                              and c.plo_id = p.ploID
                              and st.department_id = d.departmentID
                              and d.school_id = s.schoolID
                              and s.schoolID = '{}'
```

```
                        GROUP BY p.ploNum, r.student_id,c.course_id) derived1
                    GROUP BY  ploNum,StudentID) derived2
                GROUP BY ploNum

        '''.format(school))

    row1 = cursor.fetchall()
    row1.sort(key=lambda t: len(t[0]))

    cursor.execute('''
                SELECT ploNum,COUNT(Marks)
                FROM(
                      SELECT ploNum, StudentID, avg(coursemarks) as Marks
                      FROM(
                            SELECT p.ploNum as ploNum, r.student_id as StudentI
D,c.course_id,
                                  100*(sum(e.obtainedMarks)/sum(a.totalMarks))
as coursemarks
                            FROM spmapp_student_t st,
                                spmapp_registration_t r,
                                spmapp_department_t d,
                                spmapp_school_t s,
                                spmapp_evaluation_t e,
                                spmapp_assessment_t a,
                                spmapp_co_t c,
                                spmapp_plo_t p
                            WHERE st.studentID = r.student_id
                                and e.registration_id = r.registrationID
                                and a.assessmentID = e.assessment_id
                                and a.co_id = c.coID
                                and c.plo_id = p.ploID
                                and st.department_id = d.departmentID
                                and d.school_id = s.schoolID
                                and s.schoolID = '{}'
                            GROUP BY p.ploNum, r.student_id,c.course_id) derive
d1
                        GROUP BY  ploNum,StudentID
                        HAVING avg(coursemarks)>=40) derived2
                    GROUP BY ploNum

            '''.format(school))

    row2 = cursor.fetchall()
    row2.sort(key=lambda t: len(t[0]))

    plo = []
    attempted = []
```

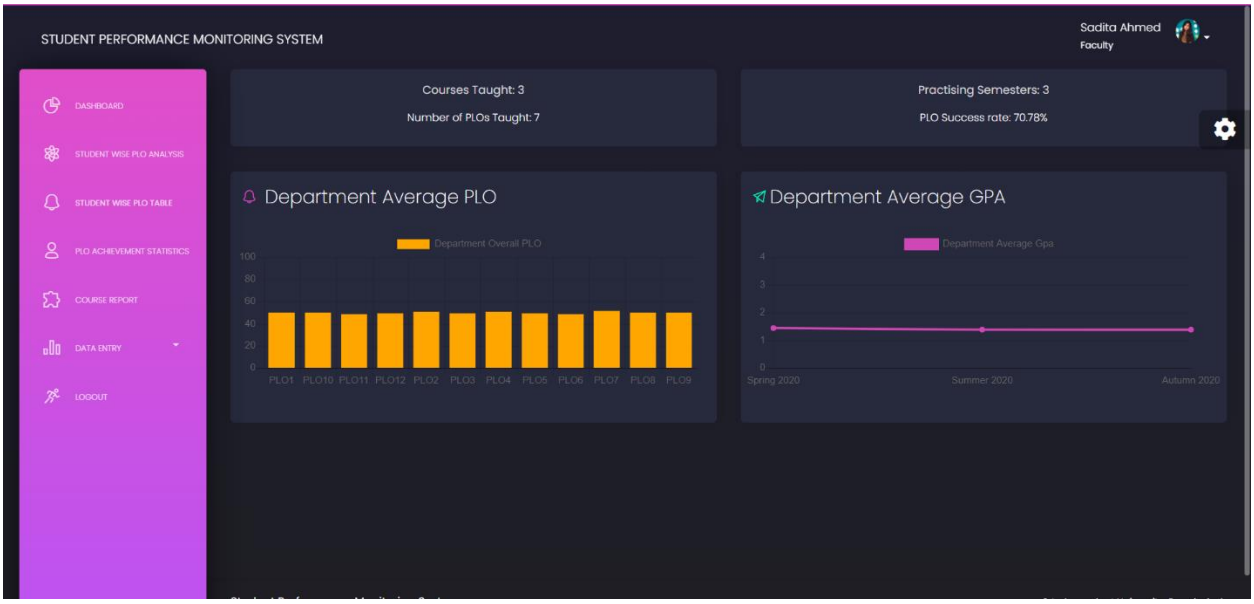
```
    achieved = []

    for i in row1:
        plo.append(i[0])
        attempted.append(i[1])

    for i in row2:
        achieved.append(i[1])

    return plo, achieved, attempted
```




```
def getCourseWiseStudentPLO(studentID, cat):
    with connection.cursor() as cursor:
        cursor.execute('''
                SELECT p.ploNum as ploNum,co.course_id,sum(e.obtainedMarks),sum(a.
totalMarks), derived.Total
                FROM spmapp_registration_t r,
                    spmapp_assessment_t a,
                    spmapp_evaluation_t e,
                    spmapp_co_t co,
                    spmapp_plo_t p,
                    (
                        SELECT p.ploNum as ploNum,sum(a.totalMarks) as Total, r.s
tudent_id as StudentID
                        FROM spmapp_registration_t r,
                            spmapp_assessment_t a,
                            spmapp_evaluation_t e,
                            spmapp_co_t co,
```

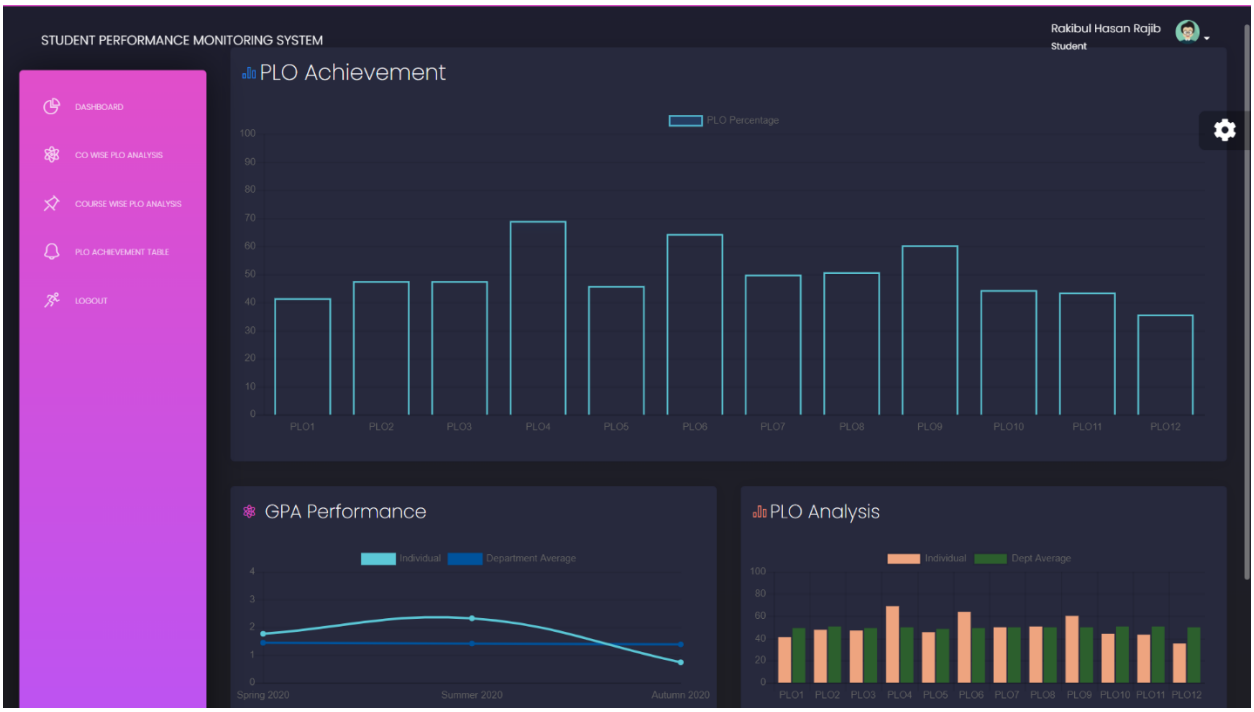
```
                        spmapp_plo_t p
                    WHERE r.registrationID = e.registration_id
                        and e.assessment_id = a.assessmentID
                        and a.co_id=co.coID
                        and co.plo_id = p.ploID
                        and r.student_id = '{}'
                    GROUP BY  r.student_id,p.ploID) derived
            WHERE r.student_id = derived.StudentID
                and e.registration_id = r.registrationID
                and e.assessment_id = a.assessmentID
                and a.co_id=co.coID
                and co.plo_id = p.ploID
                and p.ploNum = derived.ploNum

            GROUP BY  p.ploID,co.course_id

            '''.format(studentID))
        row = cursor.fetchall()

    table = []
    courses = []

    for entry in row:
        if entry[1] not in courses:
            courses.append(entry[1])
    courses.sort()
    plo = ["PLO1", "PLO2", "PLO3", "PLO4", "PLO5", "PLO6", "PLO7", "PLO8", "PLO9"
, "PLO10", "PLO11", "PLO12"]

    for i in courses:
        temptable = []
        if cat == 'report':
            temptable = [i]

        for j in plo:
            found = False
            for k in row:
                if j == k[0] and i == k[1]:
                    if cat == 'report':
                        temptable.append(np.round(100 * k[2] / k[3], 2))
                    elif cat == 'chart':
                        temptable.append(np.round(100 * k[2] / k[4], 2))
                    found = True
            if not found:
                if cat == 'report':
                    temptable.append('N/A')
                elif cat == 'chart':
```

```
                    temptable.append(0)
        table.append(temptable)
    return plo, courses, table
```




```
def getDeptWiseGPA(dept, semester):
    with connection.cursor() as cursor:
        cursor.execute('''
            SELECT AVG(grade) as avgGrade
            FROM(
                SELECT StudentID,sum(Credits*gradepoint)/sum(Credits) as grade
                FROM(
                    SELECT  StudentID,Credits,
                        CASE
                            WHEN sum(Marks) >= 85 THEN 4.0
                            WHEN sum(Marks) >= 80 AND sum(Marks)<85 THEN 3.7
                            WHEN sum(Marks) >= 75 AND sum(Marks)<80 THEN 3.3
                            WHEN sum(Marks) >= 70 AND sum(Marks)<75 THEN 3.0
                            WHEN sum(Marks) >= 65 AND sum(Marks)<70 THEN 2.7
                            WHEN sum(Marks) >= 60 AND sum(Marks)<65 THEN 2.3
                            WHEN sum(Marks) >= 55 AND sum(Marks)<60 THEN 2.0
                            WHEN sum(Marks) >= 50 AND sum(Marks)<55 THEN 1.7
                            WHEN sum(Marks) >= 45 AND sum(Marks)<50 THEN 1.3
                            WHEN sum(Marks) >= 40 AND sum(Marks)<45 THEN 1.0
                            ELSE 0.0
                        END as gradepoint
                    FROM(
                        SELECT st.studentID as StudentID,c.courseID as CourseID,
                            a.weight*(sum(e.obtainedMarks)/sum(a.totalMarks)) as
Marks, c.numOfCredits as Credits
                        FROM spmapp_student_t st,
                            spmapp_registration_t r,
```

```
                            spmapp_section_t sc,
                            spmapp_course_t c,
                            spmapp_assessment_t a,
                            spmapp_evaluation_t e
                        WHERE st.studentID = r.student_id
                            and r.section_id = sc.sectionID
                            and sc.course_id = c.courseID
                            and r.registrationID = e.registration_id
                            and e.assessment_id = a.assessmentID
                            and st.department_id = '{}'
                            and r.semester='{}'
                        GROUP BY  st.studentID,c.courseID,a.assessmentName) Deriv
ed1
                    GROUP BY StudentID,CourseID) Derived2
                GROUP BY StudentID)
                    '''.format(dept, semester))

        row = cursor.fetchall()[0][0]
    return np.round(row, 3)
def getDeptWisePLO(dept):
    with connection.cursor() as cursor:
        cursor.execute('''
             SELECT derived.plonum, avg(per)
             FROM(
                SELECT p.ploID as PLOID,p.ploNum as ploNum, 100*sum(e.obtainedMar
ks)/sum(a.TotalMarks) as per
                FROM spmapp_registration_t r,
                    spmapp_evaluation_t e,
                    spmapp_student_t st,
                    spmapp_department_t d,
                    spmapp_assessment_t a,
                    spmapp_co_t c,
                    spmapp_plo_t p
                WHERE r.student_id = st.studentID
                    and st.department_id = d.departmentID
                    and e.registration_id = r.registrationID
                    and a.assessmentID = e.assessment_id
                    and a.co_id = c.coID
                    and c.plo_id = p.ploID
                    and st.department_id = '{}'
                    GROUP BY p.ploNum,r.student_id) derived
             GROUP BY derived.ploNum
                   '''.format(dept))
        row = cursor.fetchall()
        row.sort(key=len)
    return row
```


STUDENT PERFORMANCE MONITORING SYSTEM
Rakibul Hasan Rajib
Student
DASHBOARD
CO WISE PLO ANALYSIS
COURSE WISE PLO ANALYSIS
PLO ACHIEVEMENT TABLE
LOGOUT
PLO Achievement
PLO Percentage
PLO1 PLO2 PLO3 PLO4 PLO5 PLO6 PLO7 PLO8 PLO9 PLO10 PLO11 PLO12
GPA Performance
Individual
Department Average
Spring 2020
Summer 2020
Autumn 2020
PLO Analysis
Individual
Dept Average
PLO1 PLO2 PLO3 PLO4 PLO5 PLO6 PLO7 PLO8 PLO9 PLO10 PLO11 PLO12

CHAPTER 5 - CONCLUSION:

A. PROBLEM AND SOLUTION:

Analysis Phase

Building upon project SPM developers, most of the work assumptions and queries were made while working on the rich picture and six element analysis of operations of the organization as there was no discreet data present. For a better understanding scenario and to overcome such confusions, respected faculty members and stake holder interviews were made.

Designing Phase

Based upon descriptive research created entities were kept at their Significant levels, which was also introduced in the Relational Schema schematic. Instructor's feedback played a very valid and crucial role here as well.

Implementation Phase

All the Software System Requirement's (SSR's) reached successfully!

Front-End Developing tools: HTML, CSS, Bootstrap JavaScript, Chart Js

Back End Developing tools: Python, Django

Database-integration: SQLlite3

B. ADDITIONAL FEATURE AND FUTURE DEVELOPMENT:

Future Developing Purposes:

- Plans for the project is, to add another feature which can predict A candidate's grade based on his/her past grades and performances.
- Deployment.

REFERENCES-

[1] Independent University - Bangladesh, "Curriculum for B Sc. in Computer Science and Engineering (Version 2.2)," Independent University - Bangladesh, March 19, 2017.

[2] Independent University - Bangladesh, "www.iub.edu.bd," 2020. [Online]. Available: www.iub.edu.bd.

[3] I. Department of Computer Science and Engineering, "Department of Computer Science and Engineering, IUB," 2020. [Online]. Available: http://www.cse.iub.edu.bd/.

[4] D. o. C. S. &. S. Engineering, "The University of Western Australia," 2006. [Online]. Available: https://teaching.csse.uwa.edu.au/units/CITS3240/Lectures/db-er1-nup4.pdf.

[5] K. Stevens, "BetterEvaluation," 2020. [Online]. Available: https://www.betterevaluation.org/en/evaluation-options/richpictures.

[6] Wikipedia, "Wikipedia," 2020. [Online]. Available: https://en.wikipedia.org/wiki/Rich_picture.

[7] Wikipedia, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation.