**E-LEARNING MANAGEMENT SYSTEM**

**E-LEARNING MANAGEMENT SYSTEM**

A **Project Report**

Submitted in partial fulfilment of the

requirements for the award of the degree of

**BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)**

**By**

**Ashish Harish Shetty**

**Under the esteemed guidance of**

**Mrs. Meenakshi Dhande**

**&**

**Mrs. Sheetal Vekhande**



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MODEL COLLEGE OF SCIENCE AND COMMERCE**

**University of Mumbai**

**KALYAN (E) – 421306**

**MAHARASHTRA**

**2020-2021**

**PROFORMA FOR THE APPROVAL OF PROJECT PROPOSAL**

***(Note: All entries of the proforma of approval should be filled up with  
appropriate and complete information. Incomplete proforma of approval in  
any aspect will be summarily rejected.)***

**PNR No.: ...................................... Roll No.: ......................................**

**1. Name of the Student**: Ashish Harish Shetty  
**2. Title of the Project**: E-Learning Management System (e-LMS)  
**3. Name of the Guide(s)**: Mrs. Meenakshi Dhande & Mrs. Sheetal Vekhande   
**4. Is this your first submission**? □Yes □No

Signature of the Student Signature of the Guide  
Date: ………………… Date: ………………

Signature of the coordinator  
Date: …………………

**MODEL COLLEGE OF SCIENCE & COMMERCE   
*(Affiliated to University of Mumbai)*DEPARTMENT OF INFORMATION TECHNOLOGY**

****

**CERTIFICATE**

This is to certify that the project entitled, **"E-LEARING MANAGEMENT SYSTEM "**, is a bonafide work of **Ashish Harish Shetty** bearing Seat No: ( ) submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

**Internal Guide Coordinator**

**External Examiner**

**Date: College Seal**

**ACKNOWLEDGEMENT**

I WISH TO EXPRESS MY SINCERE GRATITUDE TO PROJECT **MR. K.S. BRAMHAWALE** PRINCIPAL OF MODEL COLLEGE OF SCIENCE & COMMERCE FOR PROVIDING ME THE OPPURTUNITY TO DO MY WEB PROJECT WORK ON E-LEARNING MANAGEMENT SYSTEM. I WANT TO SINCERELY THANK MY PROJECT GUIDES **MRS MINAKSHI DHANDE** AND **MRS SHEETAL VEKHANDE** FOR GUIDANCE AND ENCOURAGEMENT IN CARRYING OUT THIS PROJECT WORK.

SPECIAL THANKS TO ALL THE LAB SYSTEMS FOR SEEMINGLY SMALL BUT VALUABLE HELPS IN TERMS OF TIMELY INTERNET AND LAB ACCESS.

**DECLARATION**

I ASHISH HARISH SHETTY, STUDENT OF MODEL COLLEGE OF SCIENCE & COMMERCE, RAJBHAR NAGAR, CHINCHPADA ROAD, KATEMANIVALI NAKA, KALYAN (EAST) 421306. STUDYING IN 3RD YEAR B.SC IN INFORMATION TECHNOLOGY HEREBY DECLARE THAT I HAVE COMPLETED THIS PROJECT ON E-LEARNING MANAGEMENT SYSTEM DURING THE ACADEMIC YEAR 2020-21.

THE INFORMATION SUBMITTED IS TRUE AND ORIGINAL TO THE BEST OF MY KNOWLEDGE.

DATE:

PLACE:

**Ashish Harish Shetty**

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **INDEX** | **Pg.**  **No.** | **Sign** |
| **CHAPTER 1:** | **INTRODUCTION** |  |  |
| 1.1 | Background |  |  |
| 1.2 | Objectives |  |  |
| 1.3 | Purpose, Scope & Applicability |  |  |
| 1.3.1 | Purpose |  |  |
| 1.3.2 | Scope |  |  |
| 1.3.3 | Applicability |  |  |
| 1.4 | Organization of Report |  |  |
| **CHAPTER 2:** | **SURVEY OF TECHNOLOGIES** |  |  |
| **CHAPTER 3:** | **REQUIREMENTS AND ANALYSIS** |  |  |
| 3.1 | Problem Definition |  |  |
| 3.2 | Requirements Specification |  |  |
| 3.3 | Planning and Scheduling |  |  |
| 3.4 | Software and Hardware Requirements |  |  |
| 3.5 | Conceptual Models |  |  |
| **CHAPTER 4:** | **SYSTEM DESIGN** |  |  |
| 4.1 | Basic Modules |  |  |
| 4.2 | Data Design |  |  |
| 4.2.1 | Schema Design |  |  |
| 4.2.2 | Data Integrity and Constraints |  |  |
| 4.3 | Procedural Design |  |  |
| 4.3.1 | Logic Diagrams |  |  |
| 4.3.2 | Algorithm Design |  |  |
| 4.4 | User Interface Design |  |  |
| 4.5 | Security Issues |  |  |
| 4.6 | Test Case Design |  |  |

**ABSTRACT**

The SYB e-Learning Management System is developed as a solution to the modern e-LMS problems. The LMSs currently available in the market has certain shortcomings which need to be addressed as it hinders the path of the knowledge for the students. This system is developed for country like India but can be scaled up easily to handle other regions as well. In these testing times of covid, all colleges/schools/universities are shutdown and lectures are conducted online. This is developed as a SAAS platform so colleges can buy any package suitable for their needs out of the three available packs. There are four main users of this system: SYB admin (our admin who will manage the platform itself), college admin (add classes/departments/teachers, manage subscription), teacher and student both of whom are core to the system.

Teachers can post documents, videos, images, text, links, etc for students which will appear in the students’ main activity feed. Detailed descriptions of the system and its core functionalities are mentioned in the forthcoming chapters.

**CHAPTER 1: INTRODUCTION**

An e-Learning Management System (e-LMS) is a software application for the administration, tracking, reporting, automation and delivery of educational courses. The LMS concept emerged directly from the e-Learning. Although the first LMS appeared in the higher education sector, the majority of the LMSs today focus on the corporate market. Learning Management Systems make up the largest segment of the learning system market. The first introduction of the LMS was in the late 1990s.

Learning management systems were designed to identify training and learning gaps, utilizing analytical data and reporting. LMSs are focused on online learning delivery but support a range of uses, acting as a platform for online content, including courses, both asynchronous based and synchronous based. An LMS may offer classroom management for instructor-led training or a flipped classroom, used in higher education, but not in corporate space. Modern LMSs include intelligent algorithms to make automated recommendations for courses based on a user’s skill profile as well as extract meta-data from learning materials in order to make such recommendations even more accurate.

* 1. **BACKGROUND**

There are several historical phases of distance education that preceded the development of LMS:

**Correspondence teaching:** The first document of correspondence teaching dates back to 1723, through the advertisement of Boston Gazette of Caleb Phillips, professor of Shorthand, offering teaching materials and tutorials.

**Multimedia teaching:** The concept of e-Learning began developing in the early 20th century, marketed by the appearance of audio-video communication systems used for remote teaching. In 1909, E.M. Forster published his story ‘The Machine Stops’ and explained the benefits of using audio communication to deliver lectures to remote audiences. The earliest networked learning system was the Plato Learning Management system developed in the 1970s by Control Data Corporation.

The history of application of computers to education is filled with broadly descriptive terms such as computer-managed instruction (CMI) and integrated learning systems (ILS), computer-based instruction (CBI), computer-assisted instruction (CAI), and computer-assisted learning (CAL). These terms describe drill-and-practice programs, more sophisticated tutorials, and more individualized instruction, respectively.

The first fully-featured Learning Management System (LMS) was called EKKO, developed and released by Norway’s NKI Distance Education Network in 1991. Three years later, New Brunswick’s NB Learning Network presented a similar system designed for DOS-based teaching, and devoted exclusive to business learners.

Most modern LMSs are web-based applications like Google’s Classroom. In these systems learners can see real-time progress of their learning and get learning materials and content directly to their device’s feed and email inbox.

**1.2 OBJECTIVES**

The objectives of this system are as follows:-

* To provide education opportunity for every student even if they are not able to physically attend the educational institution.
* To streamline the learning process and make productive use of both learner’s and instructor’s time.
* To make the learning process more engaging and less robotic. Learners can share their feedback about any specific topic posted by instructors.
* To reduce infrastructure costs for the institutions providing the educational courses.
* To increase efficiency of both instructors and students.
* To leverage technology and make learning future ready.
* To provide a centralized learning platform for multiple users.
* To provide anywhere, anytime learning. Centralized nature of e-LMS makes this possible.

**1.3.1 PURPOSE**

This is a web-based e-Learning Management System built with the purpose of delivering and managing all types of content, including video, documents, articles, etc. In the education and higher education markets, an LMS will include a variety of functions that is similar to corporate but will have features such as rubrics (scoring system), teacher and instructor facilitated learning, a discussion board, and often use of a syllabus.

SYB e-LMS has the following purposes:-

1. To provide an easy to use mobile learning experience
2. To make learning inclusive and engaging
3. To provide a centralized platform for sharing learning resources
4. To reduce infrastructure cost

After learning about most of the popular existing systems and understanding the need for developing a system different people involved in related activities have been consulted. The data required for this study has been collected from company records and reputed internet sources.

**1.3.2 SCOPE**

A learning management system (LMS) is a software application for the administration, documentation, tracking, reporting and delivery of educational courses or training programs.

LMS tools were created for making knowledge-sharing easier. The whole system was created to work across different web-based platforms. The technology has found its way in both education and the business world.

With LMS tools, businesses receive a lot of benefits, as their employees can acquire knowledge without burning a hole in the pocket.

On the education side of things, educational institutes are now in a position to work with students who are thousands of miles away. Plus, the functionality of these platforms makes the learning process easier than ever before.

**1.3.3 APPLICABILITY**

SYB e-learning management platform makes learning efficient and easier for people who are not able to physically attend the institutions providing the educational/training courses. The mobile first approach of design focuses on the broad user base especially in a country like India where majority of the Internet users are mobile users.

Year 2020 has been a very difficult time for everyone due to the global covid-19 pandemic, but it has been especially true for the education system. As colleges and schools remain shut many students are facing difficulty in their education. And with the low availability of a local e-LMS many colleges depend on big alternatives like Google Classroom. While Google’s Classroom is a good option there are certain serious privacy concerns from student’s data privacy point of view.

The SYB e-LMS provides a safe platform with zero privacy concerns as we do not use our customer data outside of the scope of this product, nor do we share any private information (not even email).

**1.4 ORGANISATION OF REPORT**

* SURVEY OF TECHNOLOGIES: In this chapter we will discuss the student’s awareness and understanding of available technologies related to the topic.
* REQUIREMENTS AND ANALYSIS: In this chapter we will discuss the requirements specification of the system i.e. hardware and software, problem definition, planning and scheduling.
* SYSTEM DESIGN: In this chapter we will discuss the features and operation of this system in detail, including screen layout, business rule, process diagram, pseudo code and other documentation.
* The chapter 5 to 7 include the IMPLEMENTATION AND TESTING, RESULTS AND DISCUSSION ,CONCLUSIONS, REFERENCES and will be submitted in the next semester i.e. semester VI
* IMPLEMENTATION AND TESTING: Inside this chapter we will discuss coding details and code efficiency, types of testing, testing approaches, modifications and improvements of this project.
* RESULTS AND DISCUSSION: We will discuss the test reports and user documentation in this chapter.
* CONCLUSIONS: The conclusions will be summarized in a fairly short chapter (2 or 3 pages). This chapter brings together many of the points that would have made in other chapters.
* REFERENCES: In this chapter we will discuss the bibliography and website used to create the project.

**CHAPTER 2: SURVEY OF TECHNOLOGIES**

**Front End: HTML, CSS, Bootstrap, JavaScript, jQuery**

As SYB e-LMS is a web application, we will use HTML for frontend markup and CSS along with Bootstrap (CSS framework) for styling. We will use JavaScript and jQuery in the front end for handling events and sending AJAX requests. These entire front end technologies are very popular and robust and are being used by several organizations as their front end choices thus proving their ability.

**HTML** is the standard markup language for web pages.

**CSS** is the language we use to style an HTML document. They can be inline code, internal code (using <style> tags) or external files. CSS along with HTML go hand in hand to specify the layout of web pages.

**Bootstrap** is a free CSS framework i.e. it provides a set of pre-written styles for us to directly use in our project. Bootstrap was created by twitter and is an open-source project. We are going to use Bootstrap 4 in this project.

**JavaScript** is the programming language of the web. It is an easy to learn scripting language.

Just like how Bootstrap is a CSS library (framework), **jQuery** is a JavaScript language library (more of a framework). jQuery greatly simplifies JavaScript programming especially some of the event handling part.

**Back End: Django (python) and MySQL (or MariaDB) database server**

**Django** is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source. Django will be the backend for our e-LMS web application. The code will be written in python language.

**MySQL** is the world’s most popular open-source relational database management system. This will be our data storage solution. We will use either MySQL server or MariaDB server, either one will work fine the only difference is MySQL server is commercially distributed and MariaDB is community-developed and commercially supported fork of MySQL server.

**CHAPTER 3: REQUIREMENTS AND ANALYSIS**

**3.1 PROBLEM DEFINITION**

There are several problems that are indentified in the current learning systems. The proposed enhancement system can prevent and overcome the existing problems. First of all, current distance learning systems are a mess when it comes to efficiency and productivity. There are a very few good systems existing today which focus exclusively on college and universities. Current system has structural problems such as ungrouped listing of content, no subject categories to separate study materials, weak multimedia support especially when it comes to video. The most popular system used today is Google’s classroom for which they force you to use Google’s own email system i.e. Gmail for signing up or logging in. This forces users to open and use Gmail account even if they already have another email provider. Moreover this existing system promotes other products of Google alongside classroom, so it becomes sort of a promotional platform rather than an educational platform.

Another major issue in existing systems is privacy. Most current systems use the personal data of its users for promotional activities and also share these details with the SAAS provider’s partners. Recently in 2020 itself the attorney general of New Mexico state in the US has revealed that Google and other major LMS providers track children across the internet, across devices, in their homes, and well outside the educational sphere, all without obtaining parental consent. These tracking activities are performed using cookies and other such technologies. Our system will not share any personal data of its users with anyone nor will it track the user’s internet activity outside the sphere of our application’s usage (by the user).

Account management and the overall administration of the current existing system are complicated. There are several other issues with the current system all of which are listed below:

1. Difficult account management
2. Complex administration
3. No automated feed updates
4. Difficult learner sharing
5. Content management problems
6. Unstable file handling
7. Privacy concerns
8. Shady marketing gimmicks

This is why the current system is proposed as a replacement for the existing system.

**3.2 REQUIREMENTS SPECIFICATION**

The project has been planned to have a distributed architecture, with centralized storage of the database. The database application to be used for storage has been planned to be MySQL server or MariaDB server. Both are same with MariaDB being more popular and stable. Django web framework will be used as the main backend which will handle the database server connectivity as well as the business logic and URL view management. It is an open-source framework written in Python and is regularly updated with security patches and new and improved features. Django uses the MVC pattern/architecture for development of the applications. It is a very high performance framework and is used by big corporations such as Facebook (for Instagram), Disqus, Spotify, YouTube, etc. Django uses ORM for database management and CRUD operations i.e. the developer does not have to manually write the SQL statements, the framework will generate these statements instead. We just have to provide the data Models as classes and also define the relationships between these models. This helps in developing a robust and stable application under less time.

**3.3 PLANNING AND SCHEDULING**

* **PLANNING**

Project planning defines the project activities that will be performed and describes how the activities will be accomplished. The purpose of project planning is to define each major task, estimate the time and resources required, and provide a framework for management review and control. The project planning activities and goals include defining:

* + The specific work to be performed and goals that define and bind the project.
  + Estimates to be documented for planning, tracking, and controlling the project.
  + Commitments that are planned, documented, and agreed to by affected groups.
  + Project alternatives, assumptions, and constraints.
* **SCHEDULING**

The project schedule provides a graphical representation of predicted tasks, milestones, dependencies, resource requirements, task duration, and deadlines. The project’s master schedule inter-relates all tasks on a common time scale. The project schedule should be detailed enough to show each WB Stack to be performed, the name of the person responsible for completing the task, the start and end date of each task, and the expected duration of the task.

* Define the type of schedule
* Define precise and measurable milestones
* Estimate task duration
* Define priorities
* Define the critical path
* Document assumptions
* Identify risks
* **GANTT CHART**

A Gantt chart is a horizontal bar chart used in project management as a tool for graphically representing the schedule of a set of specific activities or tasks. The horizontal bars indicate the length of time allocated to each activity, so the x-axis of a Gantt chart is subdivided into equal units of time, e.g., days, weeks, months. The y-axis of a Gantt chart, on the other hand, simply lists all the activities or tasks being monitored by the Gantt chart. A simple look at a Gantt chart should enable its user to determine which tasks take the longest time to complete, which tasks are overlapping with each other, etc.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Month** | **July** | | | | **August** | | | | **September** | | | |
| **Weeks** | | | | **Weeks** | | | | **Weeks** | | | |
| **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** |
| **Planning** |  |  |  |  |  |  |  |  |  |  |  |  |
| **Requirement**  **gathering** |  |  |  |  |  |  |  |  |  |  |  |  |
| **Analysis** |  |  |  |  |  |  |  |  |  |  |  |  |
| **Design** |  |  |  |  |  |  |  |  |  |  |  |  |

**3.4 HARDWARE AND SOFTWARE REQUIREMENTS**

**HARDWARE**:

Processor : Intel dual core and above

RAM : 2GB and above

Hard Disk : 120 GB

Monitor : LCD/LED

Keyboard : Normal or Multimedia

Mouse : Compatible Mouse

**SOFTWARE**

Front end : HTML, CSS, Bootstrap, JavaScript (AJAX) and jQuery

Back end : Django (Python), MySQL server (or MariaDB server)

Operating system : Windows 7/8/10 (Linux distro or Mac OS will also work fine)

**3.5 CONCEPTUAL MODELS**

**E-R DIAGRAMS:** A graphical model of the data needed by a system, including things about which information is stored & the relationships among them, produced in structured analysis & information engineering. ER Diagram represents entities or tables and their relationships with one another.

**Symbols:**

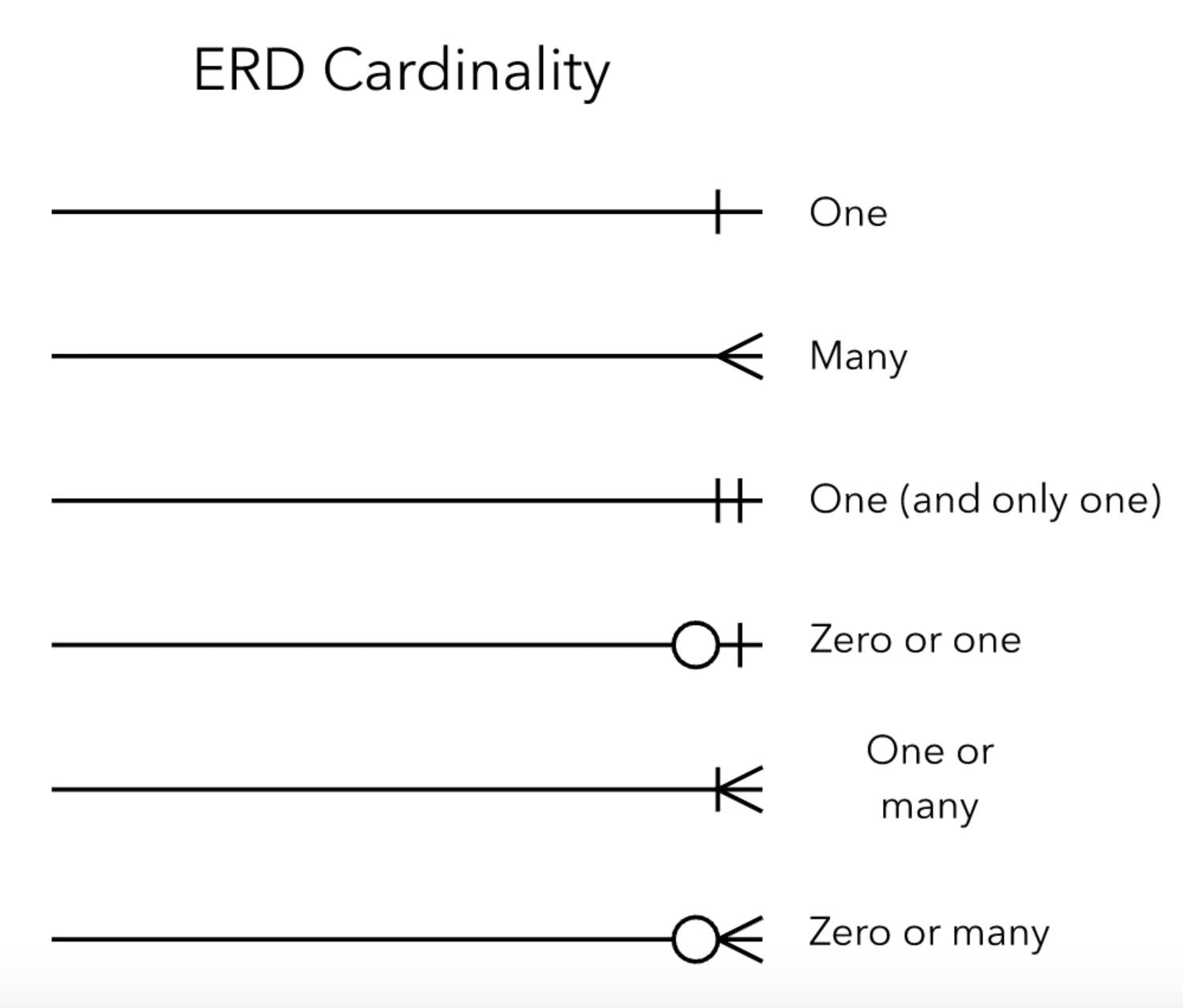


Fig 3.1 E-R cardinality symbols

**E-R DIAGRAM**

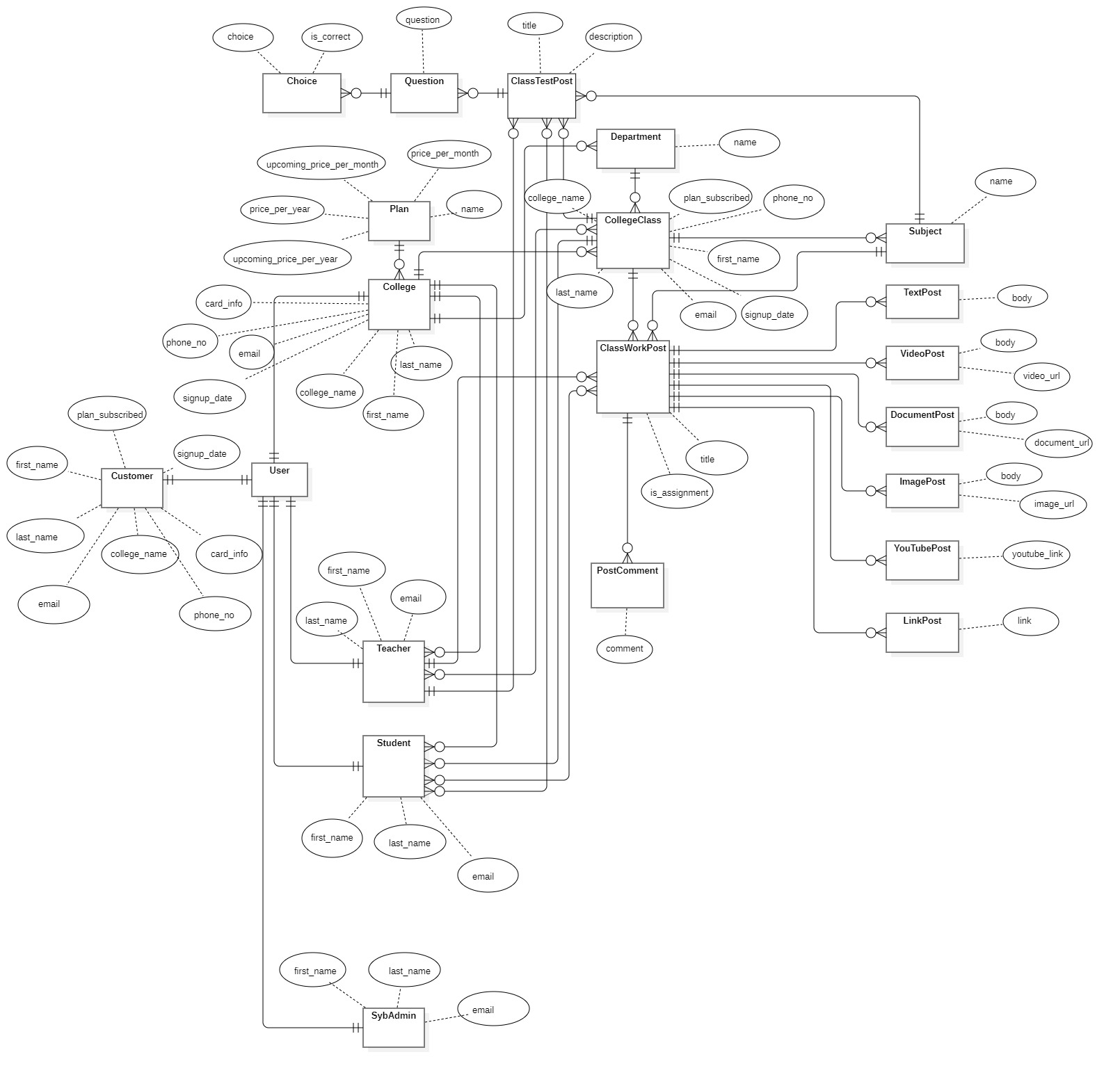


Fig 3.2 E-R diagram

**DATA FLOW DIAGRAM**

* A **Data Flow Diagram (DFD)** is a graphical representation of the “flow” of data through an information system.
* DFDs can also be used for the visualization of data processing (structured design).
* It views a system as a function that transforms the input into desired output.

**SYMBOLS:**

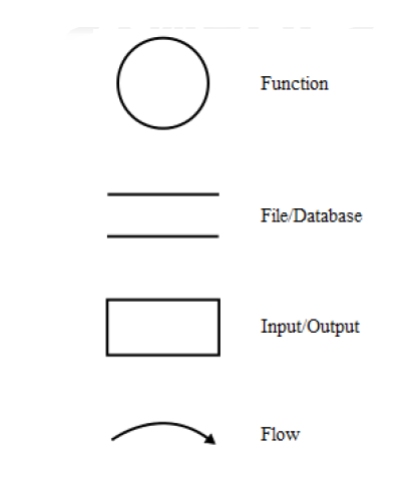


Fig 3.3 DFD symbols

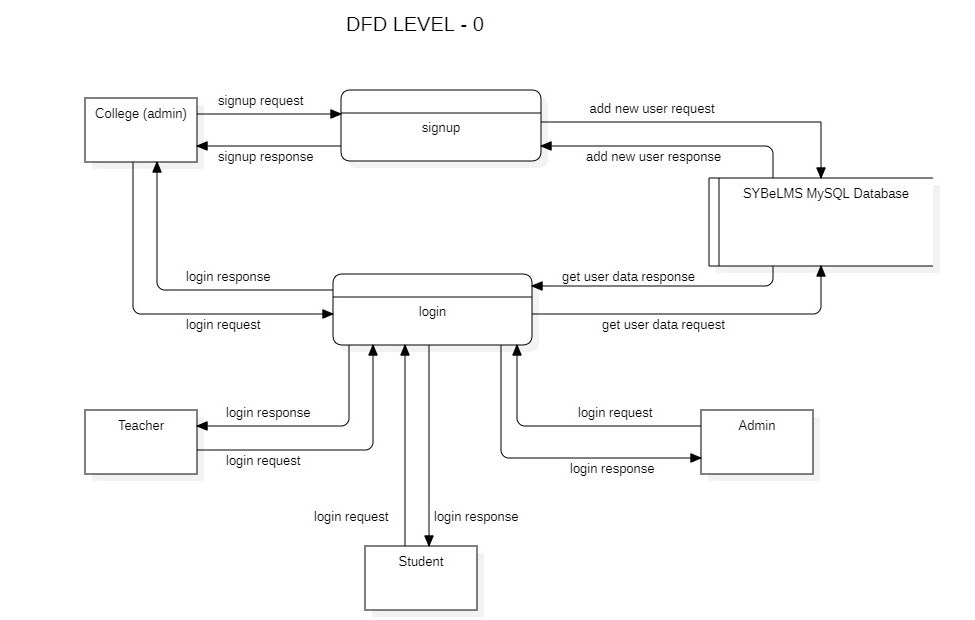


Fig 3.4 DFD Level 0

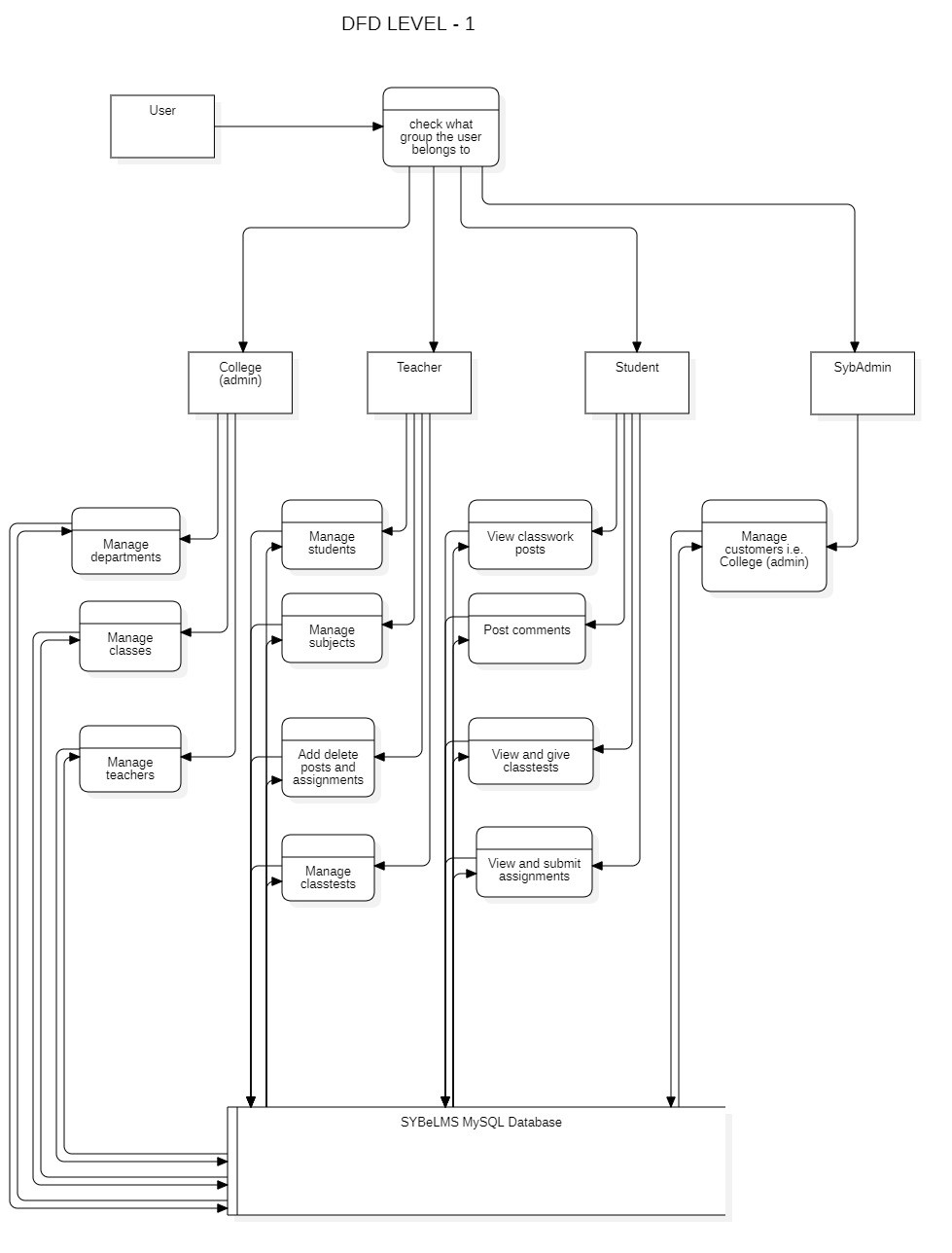


Fig 3.5 DFD Level 1

**OBJECT ORIENTED DIAGRAM**

**CLASS DIAGRAM**

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modelling Language (UML). In this context, a class defines the methods and variables in an object which is a specific entity in a program or the unit of code representing the entity. Class diagrams are useful in all forms of object-oriented programming.

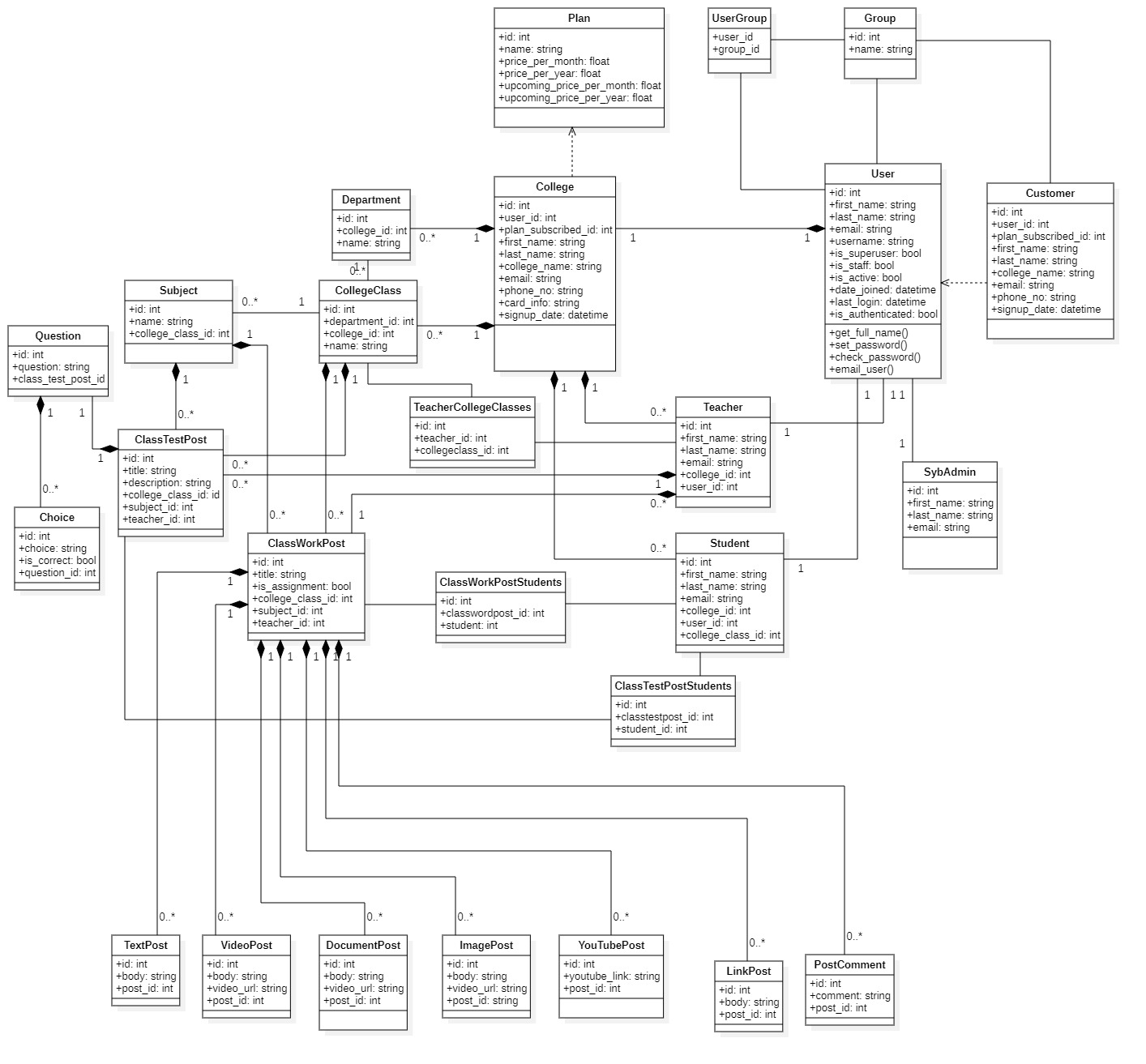


Fig 3.6 Class Diagram

**COMPONENT DIAGRAM**

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

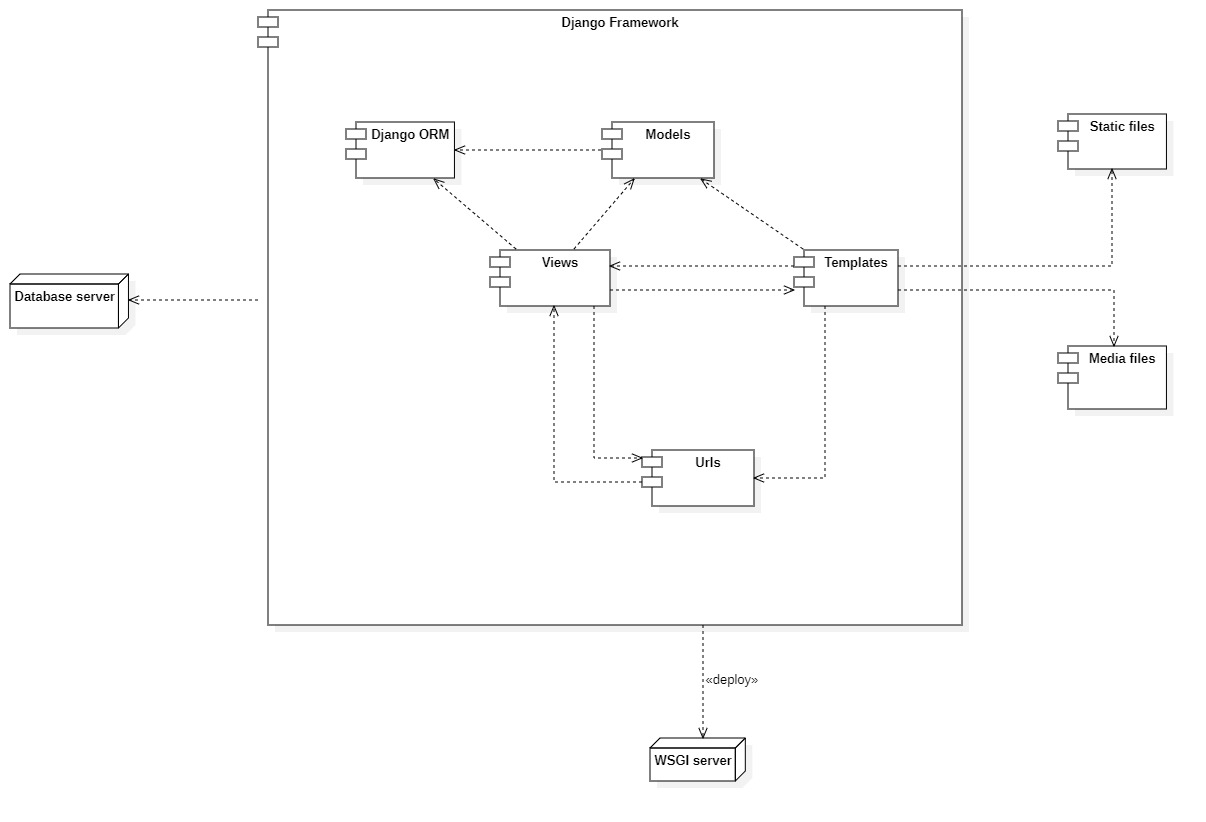


Fig 3.7 Component Diagram

**COLLABORATION DIAGRAM**

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.

Collaboration diagrams are created by first identifying the structural elements required to carry out the functionality of an interaction. A model is then built using the relationships between those elements. Several vendors offer software for creating and editing collaboration diagrams.

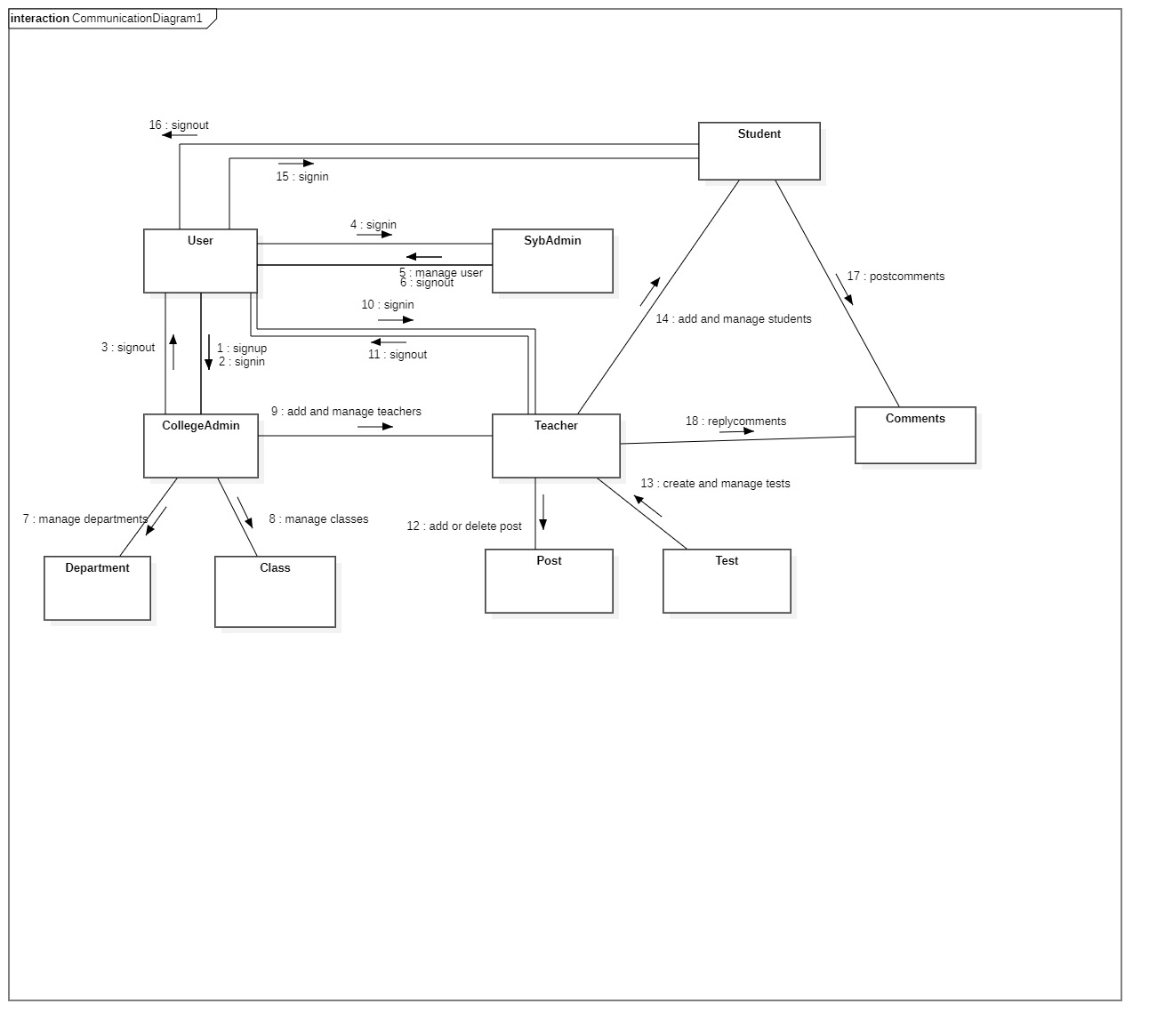


Fig 3.8 Collaboration (Communication) Diagram

**PACKAGE DIAGRAM**

Package diagram is UML structure diagram which shows packages and dependencies between the packages. Model diagrams allow to show different views of a system, for example, as multi-layered (aka multi-tiered) application - multi-layered application model. The following nodes and edges are typically drawn in a package diagram: package, packageable element, dependency, element import, package import, package merge.

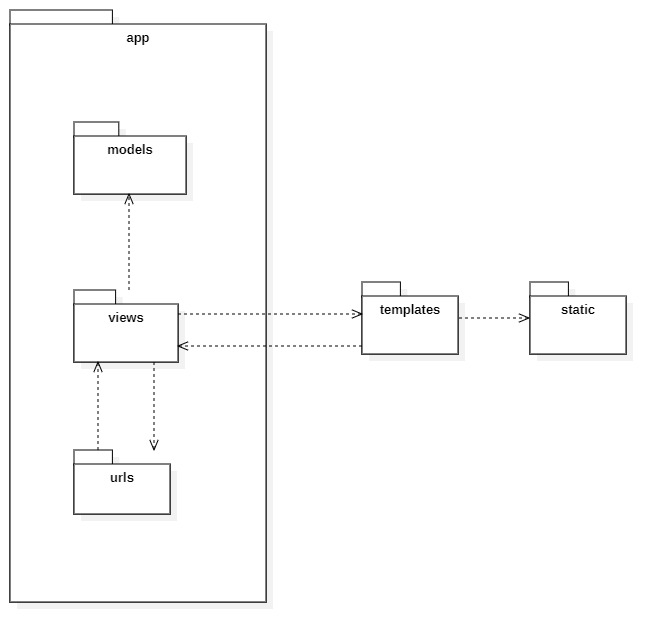


Fig 3.9 Package Diagram

**CHAPTER 4: SYSTEM DESIGN**

**4.1 BASIC MODULES**

BASIC MODULES: -

* SybAdmin module
* College module
* Teacher module
* Student module

**SybAdmin module:** This module oversees all the College users (customers) who have subscribed and are currently using the e-LMS service. This module is integrated to the whole e-LMS system and the ‘sybadmin’ user has control over each and every component of each and every College (customer) user.

**College module:** This module takes the responsibility of management of its staff and courses. This module can manage (add, modify or delete) Teachers, Departments, Classes. This module also has its own dashboard so that a ‘collegeuser’ can see and manage their respective college resources. This module is only for College (collegeadmin) users.

**Teacher module:** This module is only visible to Teacher user and has the responsibility of managing Student users and Subjects as well as creating, updating or deleting posts which are viewable to the Student users on their post feed. This module also has a dashboard from where Teachers can manage their responsibilities.

**Student module:** This module is visible only to the Student user and has a post feed where students can check and interact with the content posted by their respective Teachers. It also has a separate section for class tests and assignments.

**4.2 DATA DESIGN**

**EVENT TABLE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Event** | **Trigger** | **Source** | **Use case** | **Response** | **Destination** |
| User signup | Signup request sent | College user | Validate payment and user-data and create user | Signup successful or failed | College user |
| User login | Login request sent | User | Validate username and password | Login successful or failed | User |
| Add teacher | Add teacher request | College user | Validate form data and add user | Creation successful or failed | College user |
| Update teacher | Update teacher request | College user | Validate from data and update teacher | Update successful or failed | College user |
| Delete teacher | Delete teacher request | College user | Delete a particular teacher | Deletion successful or failed | College user |
| Add department | Add department request | College user | Validate form data and add department | Creation successful or failed | College user |
| Update department | Update department request | College user | Validate from data and update department | Update successful or failed | College user |
| Delete department | Delete department request | College user | Delete a particular department | Deletion successful or failed | College user |
| Add collegeclass | Add collegeclass request | College user | Validate form data and add collegeclass | Creation successful or failed | College user |
| Update collegeclass | Update collegeclass request | College user | Validate from data and update collegeclass | Update successful or failed | College user |
| Delete collegeclass | Delete collegeclass request | College user | Delete a particular collegeclass | Deletion successful or failed | College user |
| Add student | Add student request | Teacher user | Validate form data and add student | Creation successful or failed | Teacher user |
| Update student | Update student request | Teacher user | Validate from data and update student | Update successful or failed | Teacher user |
| Delete student | Delete student request | Teacher user | Delete a particular student | Deletion successful or failed | Teacher user |
| Add subject | Add subject request | Teacher user | Validate form data and add subject | Creation successful or failed | Teacher user |
| Update subject | Update subject request | Teacher user | Validate from data and update subject | Update successful or failed | Teacher user |
| Delete subject | Delete subject request | Teacher user | Delete a particular subject | Deletion successful or failed | Teacher user |
| Add post | Add post request | Teacher user | Validate form data and add post | Creation successful or failed | Teacher user |
| Update post | Update post request | Teacher user | Validate from data and update post | Update successful or failed | Teacher user |
| Delete post | Delete post request | Teacher user | Delete a particular post | Deletion successful or failed | Teacher user |
| Add classtest | Add classtest request | Teacher user | Validate form data and add classtest | Creation successful or failed | Teacher user |
| Delete classtest | Delete classtest request | Teacher user | Delete a particular classtest | Deletion successful or failed | Teacher user |
| Get classtest | Get classtest data request | Student user | Get classtest data for this particular student | Retrieval successful or failed | Student user |
| Submit classtest | Submit classtest request | Student user | Submit this student’s finished classtest | Submission successful or failed | Student user |
| Add comment | Add comment request | Student user | Validate form data and add comment | Creation successful or failed | Student user |
| Edit comment | Edit comment request | Student user | Validate form data and update comment | Update successful or failed | Student user |
| Delete comment | Delete comment request | Student user | Delete a particular comment | Deletion successful or failed | Student user |
| Update customer (College user) data | Update college data request | SybAdmin user | Validate form data and update college data | Update successful or failed | SybAdmin user |
| Delete customer (College user) data | Delete college data request | SybAdmin user | Delete a particular college data | Deletion successful or failed | SybAdmin user |

**4.2.1 SCHEMA DESIGN**

A **database** is an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques.

The **database schema** of a database is its structure described in a formal language supported by the database management system (DBMS). The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases). The formal definition of a database schema is a set of formulas (sentences) called integrity constraints imposed on a database. These integrity constraints ensure compatibility between parts of the schema. All constraints are expressible in the same language. A database can be considered a structure in realization of the database language. The states of a created conceptual schema are transformed into an explicit mapping, the database schema. This describes how real-world entities are modeled in the database.

**Normalization**

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

* Eliminating redundant (useless) data.
* Ensuring data dependencies make sense i.e. data is logically stored.

There are different forms of normal forms:

* First normal form (1NF)
* Second normal form (2NF)
* Third normal form (3NF)
* Boyce code normal form (BCNF)
* Fourth normal form (4NF)
* Fifth normal Form (5NF)

**group table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| name | varchar(150) | Not Null |  |

**user table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| password | varchar(128) | Not Null |  |
| last\_login | datetime(6) | Null |  |
| is\_superuser | tinyint(1) | Not Null |  |
| username | varchar(150) | Not Null |  |
| first\_name | varchar(150) | Not Null |  |
| last\_name | varchar(150) | Not Null |  |
| email | varchar(254) | Not Null |  |
| is\_staff | tinyint(1) | Not Null |  |
| is\_active | tinyint(1) | Not Null |  |
| date\_joined | datetime(6) | Not Null |  |

**user\_groups table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| user\_id | int | Not Null | Foreign Key |
| group\_id | int | Not Null | Foreign Key |

**elearn\_choice table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| choice | varchar(256) | Null |  |
| is\_correct | tinyint(1) | Not Null |  |
| question\_id | int | Not Null | Foreign Key |

**elearn\_classtestpost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| title | varchar(256) | Not Null |  |
| description | varchar(500) | Null |  |
| college\_class\_id | int | Not Null | Foreign Key |
| subject\_id | int | Not Null | Foreign Key |
| teacher\_id | int | Not Null | Foreign Key |

**elearn\_classtestpost\_students table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| classtestpost\_id | int | Not Null | Foreign Key |
| student\_id | int | Not Null | Foreign Key |

**elearn\_classworkpost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| title | varchar(256) | Not Null |  |
| is\_assignment | tinyint(1) | Not Null |  |
| college\_class\_id | int | Not Null | Foreign Key |
| subject\_id | int | Not Null | Foreign Key |
| teacher\_id | int | Not Null | Foreign Key |

**elearn\_classworkpost\_students table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| classworkpost\_id | int | Not Null | Foreign Key |
| student\_id | int | Not Null | Foreign Key |

**elearn\_college table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| first\_name | varchar(256) | Not Null |  |
| last\_name | varchar(256) | Not Null |  |
| college\_name | varchar(500) | Not Null |  |
| email | varchar(256) | Not Null |  |
| phone\_no | varchar(13) | Not Null |  |
| card\_info | varchar(16) | Not Null |  |
| signup\_date | datetime(6) | Not Null |  |
| plan\_subscribed\_id | int | Null | Foreign Key |
| user\_id | int | Not Null | Foreign Key |

**elearn\_collegeclass table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| name | varchar(50) | Not Null |  |
| college\_id | int | Null | Foreign Key |
| department\_id | int | Null | Foreign Key |

**elearn\_customer table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| first\_name | varchar(256) | Not Null |  |
| last\_name | varchar(256) | Not Null |  |
| college\_name | varchar(500) | Not Null |  |
| email | varchar(256) | Not Null |  |
| phone\_no | varchar(13) | Not Null |  |
| signup\_date | datetime(6) | Not Null |  |
| plan\_subscribed\_id | int | Null | Foreign Key |
| user\_id | int | Null | Foreign Key |

**elearn\_department table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| name | varchar(50) | Not Null |  |
| college\_id | int | Not Null | Foreign Key |

**elearn\_documentpost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| body | varchar(500) | Null |  |
| document\_url | varchar(100) | Not Null |  |
| post\_id | int | Not Null | Foreign Key |

**elearn\_imagepost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| body | varchar(500) | Null |  |
| image\_url | varchar(100) | Not Null |  |
| post\_id | int | Not Null | Foreign Key |

**elearn\_linkpost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| link | varchar(500) | Null |  |
| post\_id | int | Not Null | Foreign Key |

**elearn\_plan table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| name | varchar(256) | Not Null |  |
| price\_per\_month | decimal(6,2) | Not Null |  |
| price\_per\_year | decimal(6,2) | Not Null |  |
| upcoming\_price\_per\_month | double | Null |  |
| upcoming\_price\_per\_year | double | Null |  |

**elearn\_postcomment table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| comment | varchar(500) | Not Null |  |
| post\_id | int | Not Null | Foreign Key |

**elearn\_question table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| question | varchar(500) | Null |  |
| class\_test\_post\_id | int | Not Null | Foreign Key |

**elearn\_student table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| first\_name | varchar(256) | Not Null |  |
| last\_name | varchar(256) | Not Null |  |
| email | varchar(256) | Not Null |  |
| college\_id | int | Not Null | Foreign Key |
| college\_class\_id | int | Null | Foreign Key |
| user\_id | int | Not Null | Foreign Key |

**elearn\_subject table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| name | varchar(256) | Not Null |  |
| college\_class\_id | int | Not Null | Foreign Key |

**elearn\_sybadmin table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| first\_name | varchar(256) | Not Null |  |
| last\_name | varchar(256) | Not Null |  |
| email | varchar(256) | Not Null |  |
| user\_id | int | Not Null | Foreign Key |

**elearn\_teacher table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| first\_name | varchar(256) | Not Null |  |
| last\_name | varchar(256) | Not Null |  |
| email | varchar(256) | Not Null |  |
| college\_id | int | Not Null | Foreign Key |
| user\_id | int | Not Null | Foreign Key |

**elearn\_teacher\_college\_classes table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| teacher\_id | int | Not Null | Foreign Key |
| collegeclass\_id | int | Not Null | Foreign Key |

**elearn\_textpost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| body | varchar(500) | Null |  |
| post\_id | int | Not Null | Foreign Key |

**elearn\_videopost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| body | varchar(500) | Null |  |
| video\_url | varchar(100) | Not Null |  |
| post\_id | int | Not Null | Foreign Key |

**elearn\_youtubepost table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Nullable** | **Key** |
| id | int | Not Null | Primary Key |
| youtube\_link | varchar(256) | Null |  |
| post\_id | int | Not Null | Foreign Key |

**4.2.2 DATA INTEGRITY AND CONSTRAINTS**

The term data integrity refers to the accuracy and consistency of data. When creating databases, attention needs to be given to integrity of the data and how to maintain it. A good database will enforce data integrity whenever possible.

For example, a user could accidentally try to enter a phone number into a date field. If the system enforces data integrity, it will prevent the user from making these mistakes.

Maintaining data integrity means making sure the data remains intact and unchanged throughout its entire life cycle. This includes the capture of the data, storage, updates, transfers, backups, etc. Every time data is processed there’s a risk that it could get corrupted (whether accidentally or maliciously).

**Risks to Data Integrity**

Some more examples of where data integrity is at risk:

* A user tries to enter a date outside an acceptable range.
* A user tries to enter a phone number in the wrong format.
* A bug in an application attempts to delete the wrong record.
* While transferring data between two databases, the developer accidentally tries to insert the data into the wrong table.
* While transferring data between two databases, the network went down.
* A user tries to delete a record in a table, but another table is referencing that record as part of a relationship.

**4 Types of Data Integrity**

In the database world, data integrity is often placed into the following types:

* Entity integrity
* Referential integrity
* Domain integrity
* User-defined integrity

**Entity Integrity**

Entity integrity defines each row to be unique within its table. No two rows can be the same. To achieve this, a primary key can be defined. The primary key field contains a unique identifier – no two rows can contain the same unique identifier.

**Referential Integrity**

Referential integrity is concerned with relationships. When two or more tables have a relationship, we have to ensure that the foreign key value matches the primary key value at all times. We don’t want to have a situation where a foreign key value has no matching primary key value in the primary table. This would result in an orphaned record.  
So referential integrity will prevent users from:

* Adding records to a related table if there is no associated record in the primary table.
* Changing values in a primary table that result in orphaned records in a related table.
* Deleting records from a primary table if there are matching related records.

**Domain Integrity**

Domain integrity concerns the validity of entries for a given column. Selecting the appropriate data type for a column is the first step in maintaining domain integrity. Other steps could include, setting up appropriate constraints and rules to define the data format and/or restricting the range of possible values.

**User-Defined Integrity**

User-defined integrity allows the user to apply business rules to the database that aren’t covered by any of the other three data integrity types.

Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the **data integrity** during an update/delete/insert into a table. In this tutorial we will learn several types of constraints that can be created in RDBMS.

**Types of constraints**

* NOT NULL
* UNIQUE
* DEFAULT
* CHECK
* Key Constraints – PRIMARY KEY, FOREIGN KEY
* Domain constraints
* Mapping constraints

**4.3 PROCEDURAL DESIGN**

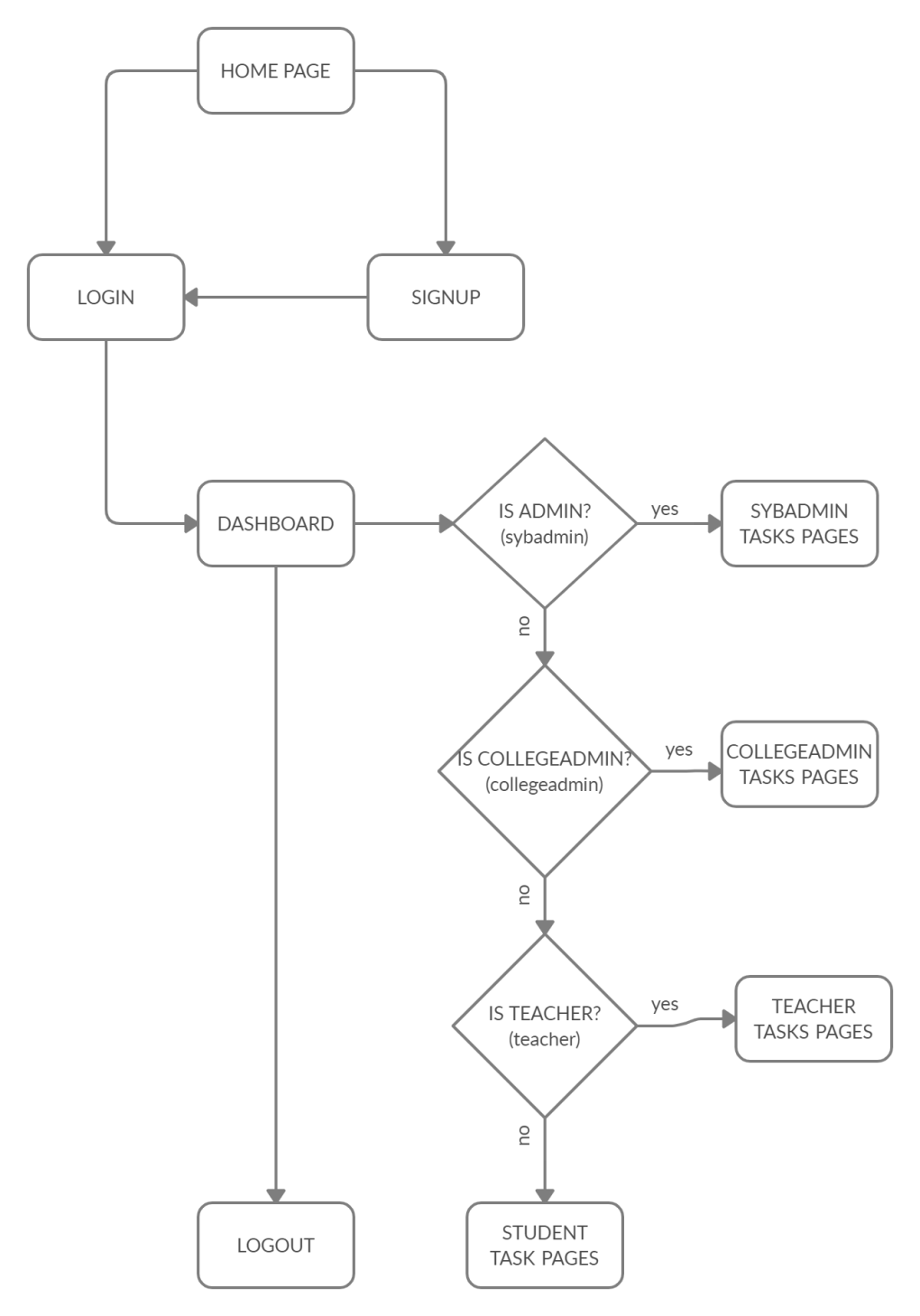


Fig 4.3.1 Procedural Diagram

**4.3.1 LOGIC DIAGRAMS**

**Logic diagrams** are diagrams in the field of logic, used for representation and to carry out certain types of reasoning.

**Nodes**

Nodes or elements are used to host graphical objects like paths and controls that can be arranged and manipulated on a diagram page.

* Many predefined standard shapes are included.
* Custom shapes can also be created and added easily.
* A node’s appearance can be fully customized.
* A node’s UI can also be converted into a template and re-used across multiple nodes.

**Connectors**

The relationship between two nodes is represented using a connector. Multiple instances of nodes and connectors form a diagram.

**Creating a Diagram**

To create a Diagram:

1. Select first an element where a new Diagram to be contained as a child in **Explorer**.
2. Select **Model | Add Diagram | [DiagramType]** in Menu Bar or select **Add Diagram | [DiagramType]** in Context Menu.

**ACTIVITY DIAGRAM**

We use **Activity Diagrams** to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.

UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams. An activity diagram is a **behavioral diagram** i.e. it depicts the behavior of a system. An activity diagram is very **similar to a flowchart**.

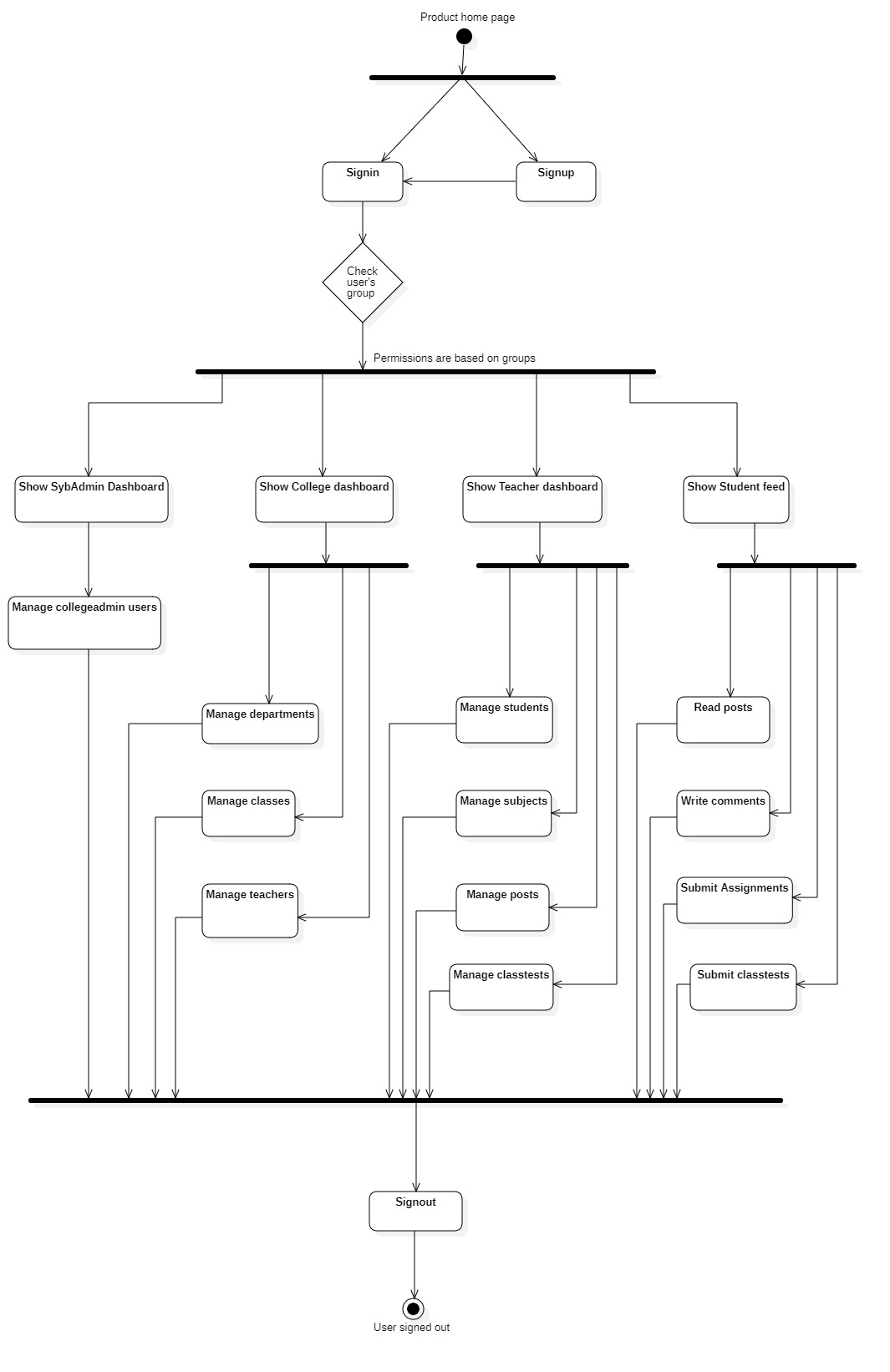
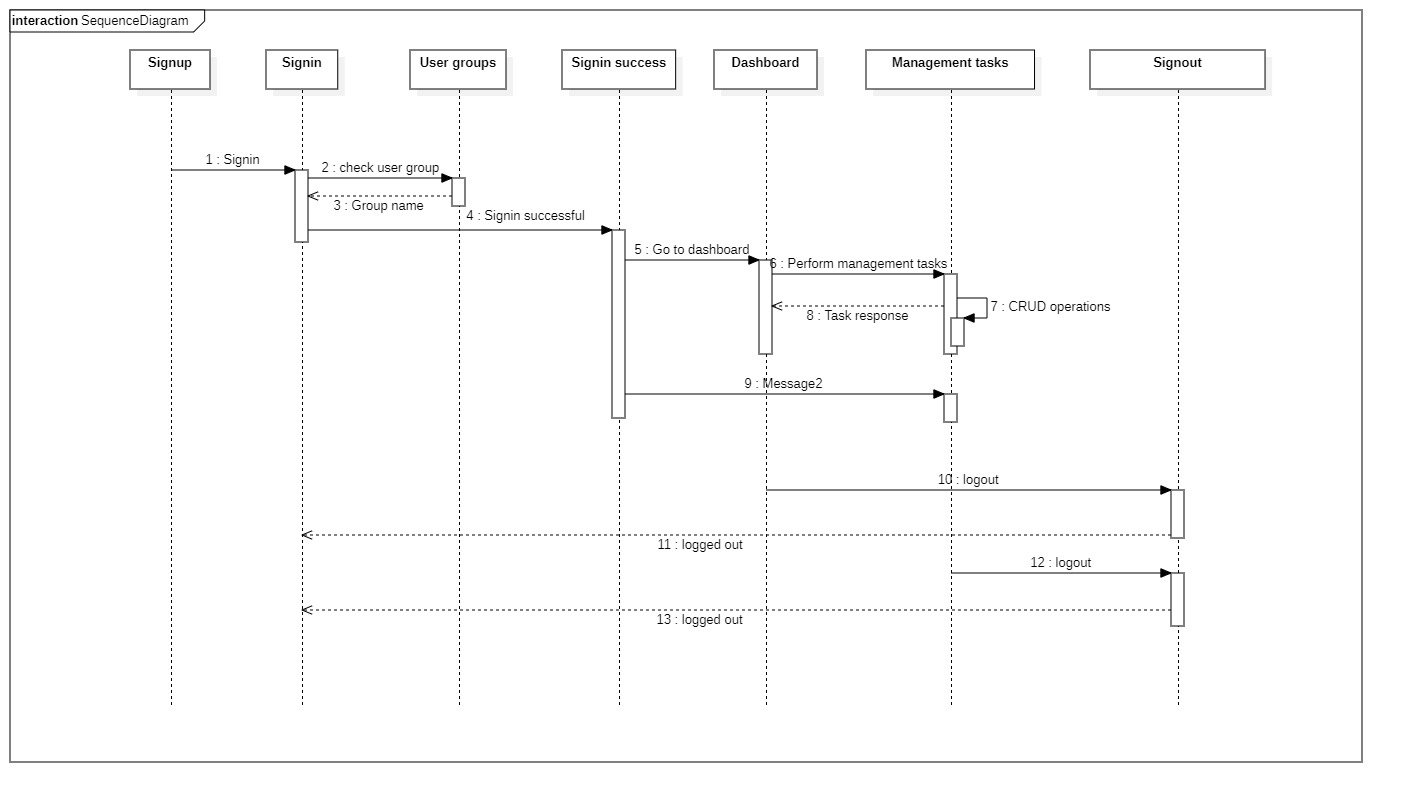


Fig 4.3.2 Activity diagram for e-LMS

**SEQUENCE DIAGRAM**

A **sequence diagram** shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams** or **event scenarios**.

Fig 4.3.3 Sequence diagram



**4.3.2 ALGORITHM DESIGN**

An algorithm is a set of steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks. An algorithm is an efficient method that can be expressed within finite amount of time and space.

An algorithm is the best way to represent the solution of a particular problem in a very simple and efficient way. If we have an algorithm for a specific problem, then we can implement it in any programming language, meaning that the **algorithm is independent from any programming languages**.

The important aspects of algorithm design include creating an efficient algorithm to solve a problem in an efficient way using minimum time and space.

To solve a problem, different approaches can be followed. Some of them can be efficient with respect to time consumption, whereas other approaches may be memory efficient. However, one has to keep in mind that both time consumption and memory usage cannot be optimized simultaneously. If we require an algorithm to run in lesser time, we have to invest in more memory and if we require an algorithm to run with lesser memory, we need to have more time.

The main characteristics of algorithms are as follows –

* Algorithms must have a unique name
* Algorithms should have explicitly defined set of inputs and outputs
* Algorithms are well-ordered with unambiguous operations
* Algorithms halt in a finite amount of time. Algorithms should not run for infinity, i.e., an algorithm must end at some point

**4.4 USER INTERFACE DIAGRAM**

**USE CASE DIAGRAM**

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

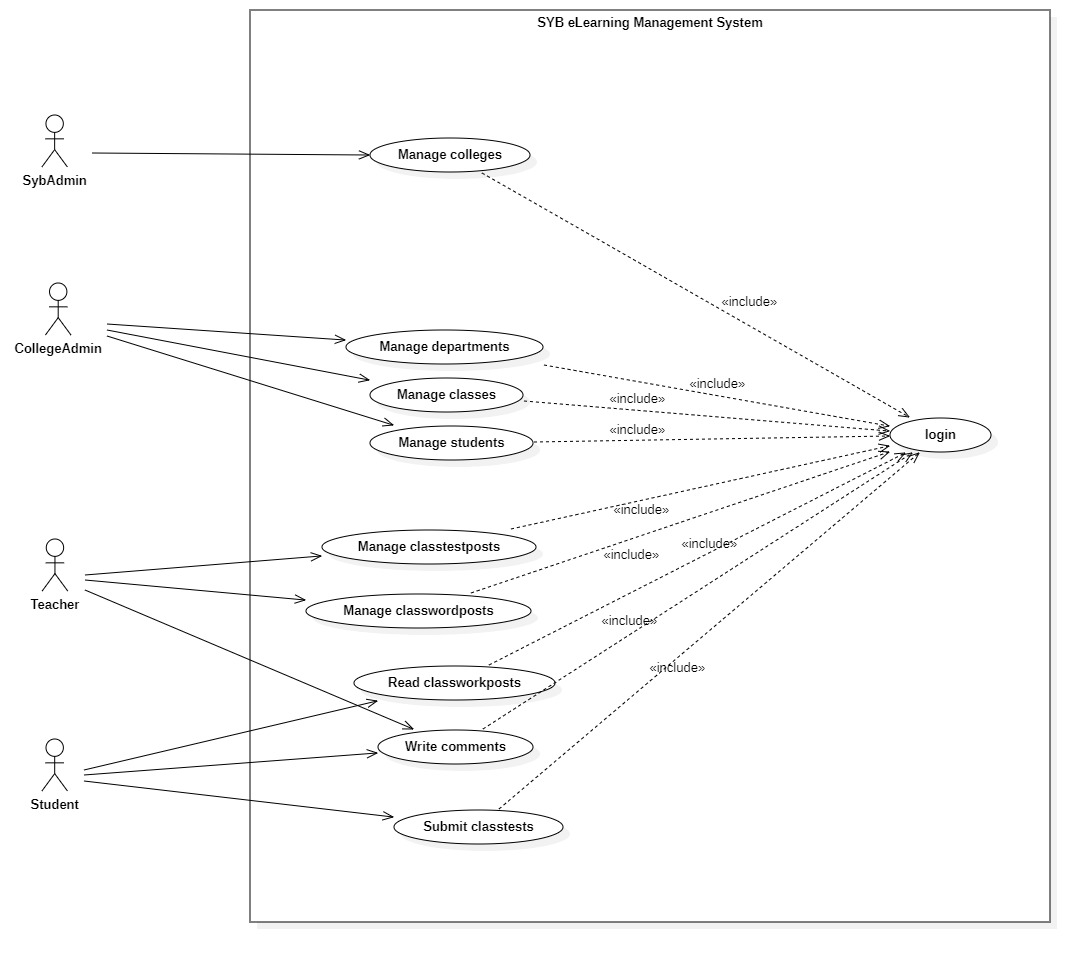


Fig 4.3.4 Use case diagram

**4.5 SECURITY ISSUE**

One of the top concerns organizations have about Learning Management Systems is data safety. Will their sensitive information be vulnerable? Who will be able to access their valuable online training resources?

This project is being implemented in Django, a very popular Python library for creating robust and secure web applications under less time.

Out-of-the-box Django provides various features which the developer can use to build a secure application. Some of these features include:

* Cross site scripting (XSS) protection
* Cross site request forgery (CSRF) protection
* SQL injection protection
* Clickjacking protection
* SSL/HTTPS redirection
* Host header validation
* Referrer policy
* Session security

All of these played a vital role in building this secure web application.

**CHAPTER 5: IMPLEMENTATION AND TESTING**

**5.1 IMPLEMENTATION APPROACHES**

An implementation method tailored to the project is a prerequisite for successful software implementation. This involves costing, planning, controlling, and monitoring the necessary tasks, including resource.

**5.1 CODING DETAILS**

The basic project structure of any Django application will have at least one project which will contain at least one application but we can create more. Django applications work on the basis of Model-View-Controller principle. So basically the application is divided into three main components and these three components work together to provide our application’s functionality.

There is a slight difference in naming of these components from traditional MVCs. In Django, Models (M of MVC) are called Models and are written in models.py file inside the app directory. The templates directory contains the .html template files which are considered as Views (V of MVC). Controllers (C of MVC) consist of the url\_patterns written in the urls.py file as well as the functional or class-based views inside the views.py file.

The MVC architecture works as follows:

1. User requests something (a page or resource)
2. The request is received by the Controller (url\_patterns and views of django).
3. If there is a need for data access or manipulation then the Controllers communicate with the required Models.
4. Response (View) is sent back to the users in form of html pages (templates) or JSON/XML.

**Code for our home page view**:

def home(request):

plans = Plan.objects.all()

context\_dict = {

'plans': plans,

}

return render(request, template\_name='home.html', context=context\_dict)

In the above code, home() function defines the name of our view. It takes a request object as a parameter. This request object will contain all the form data is any form was submitted in the request.

The return statement returns the response, which in this case is a view template called home.html. Templates in Django contain normal html code along with some place holder variables which are part of a syntax called as **Django template language**. Now we must pass the values to these variables so that the page will be rendered properly with all data in place. In order to do so we pass in a context dictionary (***context=context\_dict****)* in our **render()** method. This dictionary contains the plans object which is nothing but a collection of all the plans we retrieved from the database in the line ***plans = Plan.objects.all()***

Now let’s see the portion of the template where the plans will be used.

<table class="table">

<thead class="thead-dark">

<tr>

<th></th>

{% for plan in plans %}

<th>{{ plan.name }}</th>

{% endfor %}

</tr>

</thead>

<tbody>

<tr>

<td class="row-desc">Price per month</td>

{% for plan in plans %}

<td>₹{{ plan.price\_per\_month|floatformat }} / month<sup>\*</sup></td>

{% endfor %}

</tr>

<tr>

<td class="row-desc">Price you will be paying per year</td>

{% for plan in plans %}

<td>₹{{ plan.price\_per\_year|floatformat }} / year</td>

{% endfor %}

</tr>

<tr>

<td class="row-desc">Storage</td>

{% for plan in plans %}

{% if plan.allotted\_storage\_space > 999 %}

<td>{{ plan.allotted\_storage\_space|div:1000|floatformat }} TB</td>

{% else %}

<td>{{ plan.allotted\_storage\_space|floatformat }} GB</td>

{% endif %}

{% endfor %}

</tr>

<tr>

<td colspan="4">

Payment will be on yearly basis and not monthly.<sup>\*</sup>

</td>

</tr>

<tr>

<td></td>

<td><a class="btn btn-warning buy-btns" href="{% url 'signup' plan\_subscribed='Basic' %}">Buy</a>

</td>

<td><a class="btn btn-warning buy-btns" href="{% url 'signup' plan\_subscribed='Standard' %}">Buy</a>

</td>

<td><a class="btn btn-warning buy-btns" href="{% url 'signup' plan\_subscribed='Pro' %}">Buy</a></td>

</tr>

</tbody>

</table>

As we can see from the above code, the plans dictionary that we passed through our view into this template is being used to fill up the data for our home page plans table. This table contains all the available plans that we are providing for our customers. The things that start with a {% or {{ are part of the Django template language syntax and are interpreted and processed before the page is finally sent to the requesting user’s browser.

In Django template language, Variables look like this: {{ variable }}. When the template engine encounters a variable, it evaluates that variable and replaces it with the result. The dot (".") also appears in variable sections, although that has a special meaning, as indicated below. Importantly, you cannot have spaces or punctuation characters in variable names.

**Behind the scenes**:

Technically, when the template system encounters a dot, it tries the following lookups, in this order:

* Dictionary lookup
* Attribute or method lookup
* Numeric index lookup

If the resulting value is callable, it is called with no arguments. The result of the call becomes the template value.

This lookup order can cause some unexpected behaviour with objects that override dictionary lookup. For example, consider the following code snippet that attempts to loop over a collections.defaultdict:

{% for k, v in defaultdict.items %}

Do something with k and v here...

{% endfor %}

Because dictionary lookup happens first, that behaviour kicks in and provides a default value instead of using the intended .items() method. In this case, consider converting to a dictionary first.

**Tags** look like this: {% tag %}. Tags are more complex than variables: Some create text in the output, some control flow by performing loops or logic, and some load external information into the template to be used by later variables.

Some tags require beginning and ending tags (i.e. {% tag %} ... tag contents ... {% endtag %}).

Django ships with about two dozen built-in template tags. You can read all about them in the built-in tag reference. To give you a taste of what’s available, here are some of the more commonly used tags:

* for
* if, elif, and else
* block and extends

& many more…

The functionality of these are similar to their counterparts in other languages albeit they differ in the syntax.

**Authentication mechanism**: In our app every user be it College or Teacher or Student or Sybadmin, all have a One-To-One relationship with the Django’s built-in User model. This saves us some development time as this User model comes with its own authentication mechanism. If we chose to create all of these users custom, without this One-To-One relationship, we could have done so. But that approach would have caused us to write custom authentication mechanisms for each users.

@unauthenticated\_user

def sign\_up(request, plan\_subscribed=''):

if request.method == 'POST':

data = json.loads(request.body)

first\_name = data['first\_name']

last\_name = data['last\_name']

college\_name = data['college\_name']

email\_id = data['email\_id']

password1 = data['password']

phone\_no = data['phone\_no']

card\_no = data['card\_no']

card\_cvv = data['card\_cvv']

plan\_subscribed = data['plan\_subscribed']

# Dummy payment processing

pay\_status\_failed = False

if pay\_status\_failed:

return redirect(payment\_failed)

try:

# register the User

new\_user = User.objects.create\_user(

first\_name=first\_name,

last\_name=last\_name,

email=email\_id,

username=email\_id,

)

new\_user.set\_password(password1)

new\_user.save()

# Add this user to collegeadmin group

collegeadmin\_group = Group.objects.get(name='collegeadmin')

collegeadmin\_group.user\_set.add(new\_user)

# get the plan\_subscribed object

plan = Plan.objects.get(name=plan\_subscribed)

# register the User as College

college = College.objects.create(

user=new\_user,

plan\_subscribed=plan,

first\_name=first\_name,

last\_name=last\_name,

college\_name=college\_name,

email=email\_id,

phone\_no=phone\_no,

card\_info=card\_no,

)

college.set\_initial\_subscription\_dates()

college.save()

# NOTE: Here you can add backend payment processing

# process the payment here and generate invoice for it

invoice = Invoice.objects.create(

college=college,

plan\_subscribed=plan,

)

invoice.pay()

invoice.save()

# Now log the user in

auth\_user = authenticate(request, username=email\_id, password=password1)

if auth\_user is not None:

login(request, auth\_user)

return JsonResponse({'process': 'success'})

else:

return JsonResponse({'process': 'failed', 'msg': 'User authentication system failed! Please sign in'})

except IntegrityError:

return JsonResponse({'process': 'failed', 'msg': 'User already exists'})

except Exception as err:

return JsonResponse({'process': 'failed', 'msg': f'{str(err)}'})

plans = Plan.objects.all().values\_list('name', flat=True)

if plan\_subscribed not in plans:

return redirect(home)

cost\_of\_selected\_plan = Plan.objects.get(name=plan\_subscribed).price\_per\_year

context\_dict={

'plan\_selected': plan\_subscribed,

'plan\_cost': f'₹{int(cost\_of\_selected\_plan)} / year'

}

return render(request, template\_name='sign\_up.html', context=context\_dict)

In the above code a College is signing up for a plan. The ***json.loads(request.body)*** loads the posted form data into python’s dictionary format for further processing. We are using json.loads() here because the form in this view was posted via browser’s fetchApi (AJAX). Down the view we create the required objects and store the data in the database using Django ORM. We also create an invoice for the College.

Note how the User object was created first then it was passed to the College’s user attribute. This college\_object.user defines the One-To-One relationship between the User and the College. Also make a note of how the User’s password was set. Instead of doing new\_user.password=password1 we called a method on the new\_user object: ***new\_user.set\_password(password1).*** This is the preferred way of storing passwords in Django. The ***set\_password()*** method generates a hash for the password provided and stores the hash in the database. In Django (and in any kind of application in general) we never store credentials directly/as it is in the database, instead we generate a hash for the password using hashing algorithms provided by various libraries and then store the hash in the database. Then at the time of login we hash the password provided at the login form and then compare the hashes.

@unauthenticated\_user

def sign\_in(request):

if request.method == 'POST':

username = request.POST.get('email') # username and email are one and the same for the context of this app

password = request.POST.get('password')

try:

# Log the user in

auth\_user = authenticate(request, username=username, password=password)

if auth\_user is not None:

login(request, auth\_user)

# Now redirect based on what group user is a part of

login\_user = User.objects.get(username=username)

login\_user\_group = login\_user.groups.all()[0].name

redirect\_dict = {

'collegeadmin': college\_page,

'teacher': college\_teacher,

'student': college\_student,

'sybadmin': syb\_admin\_page,

}

return redirect(redirect\_dict[login\_user\_group])

else:

if User.objects.filter(username=username).exists():

context\_dict = {'msg': 'Email id or password is wrong'}

return render(request, template\_name='sign\_in.html', context=context\_dict)

context\_dict = {'msg': 'User does not exist please sign up first'}

print(1)

return render(request, template\_name='sign\_in.html', context=context\_dict)

except Exception as err:

context\_dict = {'msg': f'{err}'}

return render(request, template\_name='sign\_in.html', context=context\_dict)

context\_dict = {}

return render(request, template\_name='sign\_in.html', context=context\_dict)

In the above code we are signing the user in. The signin page is common for all types of users be it collegeadmins, teachers, students, or even sybadmins. Take a close look at the line containing the *authenticate()* method: ***authenticate(request, username=username, password=password)***. This method is used for checking user’s credentials in Django. We only need to call this method and provide the required credentials i.e. username and password. In our application we are using email itself as our username thus making it possible for us to sign in by using our email id. If the credentials are invalid then the ***authenticate()*** method will return **None**. If the result is not **None** that means that the provided credentials are correct and we can login the user via the ***login()*** method of Django and passing the **request** object as well as the **auth\_user** object returned by the ***authenticate()*** method. This will do all the required stuff required for login, like setting the sessions/cookies, etc. in the background.

def sign\_out(request):

# sign out logic here

logout(request)

context\_dict = {'msg': 'Logged out successfully'}

return render(request, template\_name='sign\_in.html', context=context\_dict)

The above code is from the **sign\_out** view. Here in this view we are using the ***logout()*** method provided by Django for logging the user out of the application. Behind the curtains this method clears all the session and cookie data set during the signup process. After signing the user out we then render the **sign\_in** template /page.

The ***@login\_required*** and ***@unauthenticated\_user*** are python decorators (special functions) which we use for user authorization or access restriction. The ***@login\_required*** is provided by Django itself while ***@unauthenticated\_user*** is custom user written decorator.

All other views and templates use these same techniques and methodologies for providing the functionality of our application.

**5.3 TESTING APPROACH**

Automated testing is an extremely useful bug-killing tool for the modern Web developer. You can use a collection of tests – a test suite – to solve, or avoid, a number of problems:

* When you’re writing new code, you can use tests to validate your code works as expected.
* When you’re refactoring or modifying old code, you can use tests to ensure your changes haven’t affected your application’s behavior unexpectedly.

Testing a Web application is a complex task, because a Web application is made of several layers of logic – from HTTP-level request handling, to form validation and processing, to template rendering. With Django’s test-execution framework and assorted utilities, you can simulate requests, insert test data, inspect your application’s output and generally verify your code is doing what it should be doing.

**5.3.1 UNIT TESTING**

Django’s unit tests use a Python standard library module: unittest. This module defines tests using a class-based approach.

Here is an example which subclasses from **django.test.TestCase**, which is a subclass of **unittest.TestCase** that runs each test inside a transaction to provide isolation:

Once you’ve written tests, run them using the test command of your project’s manage.py utility:

>python manage.py test

Tests that require a database (namely, model tests) will not use your “real” (production) database. Separate, blank databases are created for the tests.

Regardless of whether the tests pass or fail, the test databases are destroyed when all the tests have been executed.

**5.3.2. INTEGRATION TESTING**

Integration testing implements a module code for a system and test for a work correctly or not. And error are occurs this time solve problems of an error. Integration testing is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire programmer is tested as a whole. In the integration-testing step, all the error uncovered is corrected for the next testing steps.

**5.4. MODIFICATIONS AND IMPROVEMENTS**

The system modification is depends on their implementation which is done by using software which we use for project development. In the development of project we finish the testing and then find the bugs, errors and after that we improve it. But for implementation we have to focus on problems also decisions necessary to solve the problems. Information needs to make these decisions.

**5.5 TEST CASES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Test Steps** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| 1 | Test signup form fields with invalid data | 1. Enter invalid data into all form fields  2. Press Signup button | Invalid data message should be displayed. | Invalid data message is displayed. | Pass |
| 2 | Test signup form fields with valid data | 1. Enter invalid data into all form fields  2. Press Signup button | User must be signed up and taken to the college dashboard page | User is signed up and taken to the college dashboard page | Pass |
| 3 | Test add teacher form fields with invalid data | 1. Enter invalid data into all form fields  2. Press add button | Invalid data message should be displayed. | Invalid data message is displayed. | Pass |
| 4 | Test add teacher form fields with valid data | 1. Enter valid data into all form fields  2. Press add button | New teacher must be added to the database and the user must be notified | New teacher is added to the database and the user is notified | Pass |
| 5 | Test add department form fields with invalid data | 1. Enter invalid data into all form fields  2. Press add button | Invalid data message should be displayed. | Invalid data message is displayed. | Pass |
| 6 | Test add department form fields with valid data | 1. Enter valid data into all form fields  2. Press add button | New department must be added to the database and the user must be notified | New department is added to the database and the user is notified | Pass |
| 7 | Test add class form fields with invalid data | 1. Enter invalid data into all form fields  2. Press add button | Invalid data message should be displayed. | Invalid data message is displayed. | Pass |
| 8 | Test add class form fields with valid data | 1. Enter valid data into all form fields  2. Press add button | New class must be added to the database and the user must be notified | New class is added to the database and the user is notified | Pass |
| 9 | Test edit department data with invalid data | 1. Press edit button  2. Enter invalid data  3. Press save button | Old data should not be updated and invalid data message should be displayed | Old data is not updated and invalid data message is displayed | Pass |
| 10 | Test edit department data with valid data | 1. Press edit button  2. Enter valid data  3. Press save button | Old department data should be updated with the new one | Old department data is updated with the new one | Pass |
| 11 | Test edit class data with invalid data | 1. Press edit button  2. Enter invalid data  3. Press save button | Old data should not be updated and invalid data message should be displayed | Old data is not updated and invalid data message is displayed | Pass |
| 12 | Test edit class data with valid data | 1. Press edit button  2. Enter valid data  3. Press save button | Old class data should be updated with the new one | Old class data is updated with the new one | Pass |
| 13 | Test edit teacher data with invalid data | 1. Press edit button  2. Enter invalid data  3. Press save button | Old data should not be updated and invalid data message should be displayed | Old data is not updated and invalid data message is displayed | Pass |
| 14 | Test edit teacher data with valid data | 1. Press edit button  2. Enter valid data  3. Press save button | Old teacher data should be updated with the new one | Old teacher data is updated with the new one | Pass |
| 15 | Delete teacher data | 1. Press delete button for respective teacher (row) | Teacher data should be deleted from the table | Teacher data deleted from the table | Pass |
| 16 | Delete department data | 1. Press delete button for respective department (row) | Department data should be deleted from the table | Department data deleted from the table | Pass |
| 17 | Delete class data | 1. Press delete button for respective class (row) | Class data should be deleted from the table | Class data deleted from the table | Pass |
| 18 | Test sign in form with invalid data | 1. Enter username and password  2. Press signin button | Invalid username or password message should be displayed | Invalid username or password message is displayed | Pass |
| 19 | Test sign in form with valid data | 1. Enter username and password  2. Press signin button | User must be signed in and taken to their respective dashboard page | User is signed in and taken to their respective dashboard page | Pass |
| 20 | Test add student form fields with invalid data | 1. Enter invalid data into all form fields  2. Press add button | Invalid data message should be displayed | Invalid data message is displayed. | Pass |
| 21 | Test add student form fields with valid data | 1. Enter valid data into all form fields  2. Press add button | New student must be added to the database and the user must be notified | New student is added to the database and the user is notified | Pass |
| 22 | Test add subject form fields with invalid data | 1. Enter invalid data into all form fields  2. Press add button | Invalid data message should be displayed | Invalid data message is displayed. | Pass |
| 23 | Test add subject form fields with valid data | 1. Enter valid data into all form fields  2. Press add button | New subject must be added to the database and the user must be notified | New subject is added to the database and the user is notified | Pass |
| 24 | Test add post | 1. Click on add post button  2. Select post type  3. Submit the post | If the post meets all requirements and is within constraints then it should be successfully submitted and visible in the feed else display error message | If the post meets all requirements and is within constraints then it is successfully submitted and visible in the feed else error message is displayed | Pass |
| 25 | Test add classtestpost | 1. Click on add classtest button  2. Enter the required data  3. Submit the test post | If all the data is valid then create the classtestpost else display error message | If all the data is valid then the classtestpost is created else error message is displayed | Pass |
| 26 | Test answer and submit classtest | 1. Click on test post  2. Answer all question  3. Press the submit button | If all questions are answered then the test must be submitted else display error message | If all questions are answered the submit the test else display error message | Pass |
| 27 | Test add comment | 1. Type the comment in the textbox  2. Press submit button | If the comment text is valid then the comment must be submitted else display the error message | If the comment text is valid then the comment is submitted else the error message is displayed | Pass |

**CHAPTER 6: RESULTS AND DISCUSSION**

**6.1. SCREENSHOTS AND FUNCTIONALITY**