

# CHAPTER-1

## INTRODUCTION

### 1.1 OBJECTIVES:

- The main objective of the project is to design and develop a user friendly-system
- Easy to use and an efficient computerized system.
- To develop an accurate and flexible system, it will eliminate data redundancy.
- To study the functioning of Students management System.
- To make a software fast in processing, with good user interface.
- To make software with good user interface so that user can change it and it should be used for a long time without error and maintenance.
- To provide synchronized and centralized farmer and seller database.
- Computerization can be helpful as a means of saving time and money.
- To provide better Graphical User Interface (GUI).
- Less chances of information leakage.
- Provides Security to the data by using login and password method.
- To provide immediate storage and retrieval of data and information.
- Improving arrangements for students coordination.
- Reducing paperwork.

### 1.2 LIMITATIONS:

- Time consumption in data entry as the records are to be manually maintained faculties a lot of time.
- Lot of paper work is involved as the records are maintained in the files and registers.
- Storage Requires as files and registers are used the storage space requirement is increased.
- Less Reliable use of papers for storing valuable data information is not at all reliable.
- Aadhar linkage with the official aadhar database has not been done.

## **CHAPTER-2**

### **STUDY OF EXISTING SYSTEM**

#### **2.1 CASE STUDY**

The success of any organization such as School of Public Health, University of Ghana hinges on its ability to acquire accurate and timely data about its operations, to manage this data effectively, and to use it to analyze and guide its activities. Integrated student database system offer users (Student, Registrar, HOD) with a unified view of data from multiple sources. To provide a single consistent result for every object represented in these data sources, data fusion is concerned with resolving data inconsistency present in the heterogeneous sources of data. The main objective of this project is to build a rigid and robust integrated student database system that will track and store records of students. This easy-to-use, integrated database application is geared towards reducing time spent on administrative tasks. The system is intended to accept process and generate report accurately and any user can access the system at any point in time provided internet facility is available. The system is also intended to provide better services to users, provide meaningful, consistent, and timely data and information and finally promotes efficiency by converting paper processes to electronic form. The system was developed using technologies such as, HTML, CSS ,JS and MySQL. PYTHON- FLASK, HTML and CSS are used to build the user interface and database was built using MySQL. The system is free of errors and very efficient and less time consuming due to the care taken to develop it. All the phases of software development cycle are employed and it is worthwhile to state that the system is very robust. Provision is made for future development in the system.

#### **2.2 PROPOSED SYSTEM**

While there has been no consensus on the definition of Students Management in the literature, they have proposed that researchers adopt the below definition to allow for the coherent development of theory in the colleges. In order to have a successful students management, we need to make many decisions related to the flow of marks, attendance, and data. Each records should be added in a way to increase the scalability. Student management is more complex in colleges and other universities because of the impact on people's number requiring adequate and accurate information of students need.

## **CHAPTER 3**

### **3. DATABASE DESIGN**

#### **3.1 SOFTWARE REQUIREMENTS SPECIFICATION**

##### **3.1.2 SOFTWARE REQUIREMENTS:**

Frontend- HTML, CSS, Java Script, Bootstrap

Backend-Python flask (Python 3.7) , SQLAlchemy,

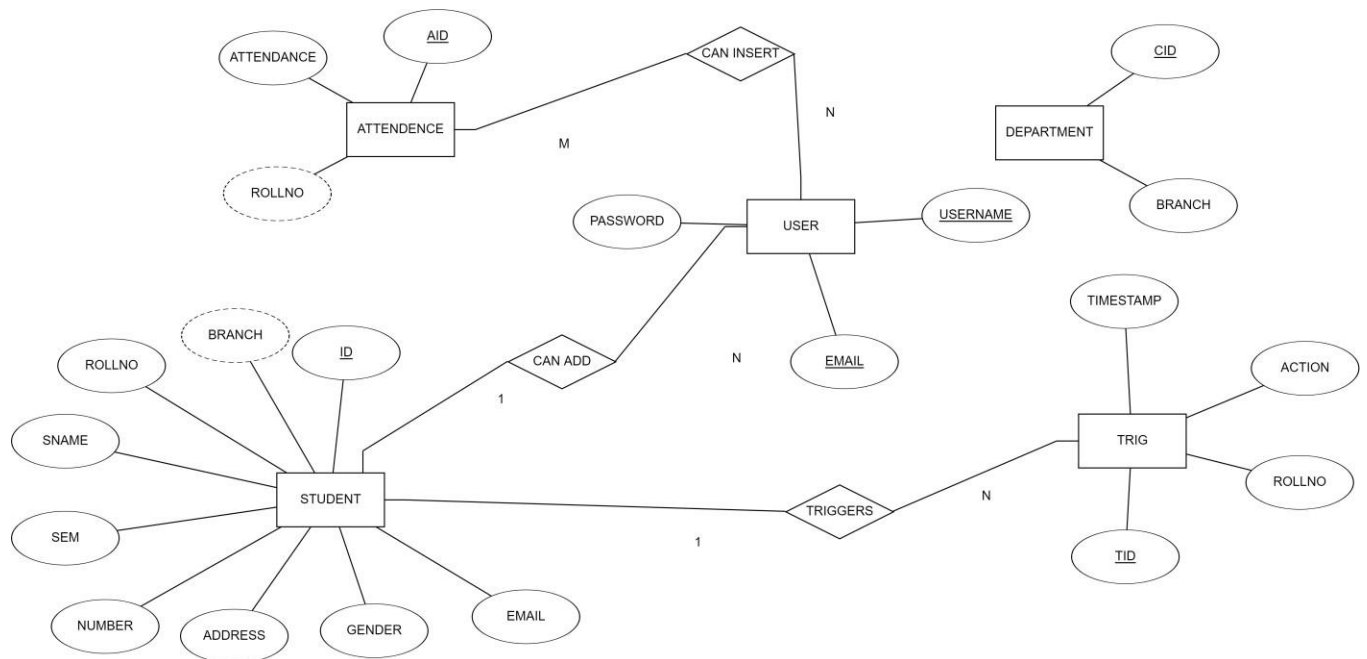
- Operating System: Windows 10
- Google Chrome/Internet Explorer
- XAMPP (Version-3.7)
- Python main editor (user interface): PyCharm Community
- workspace editor: Sublime text 3

##### **HARDWARE REQUIREMENTS:**

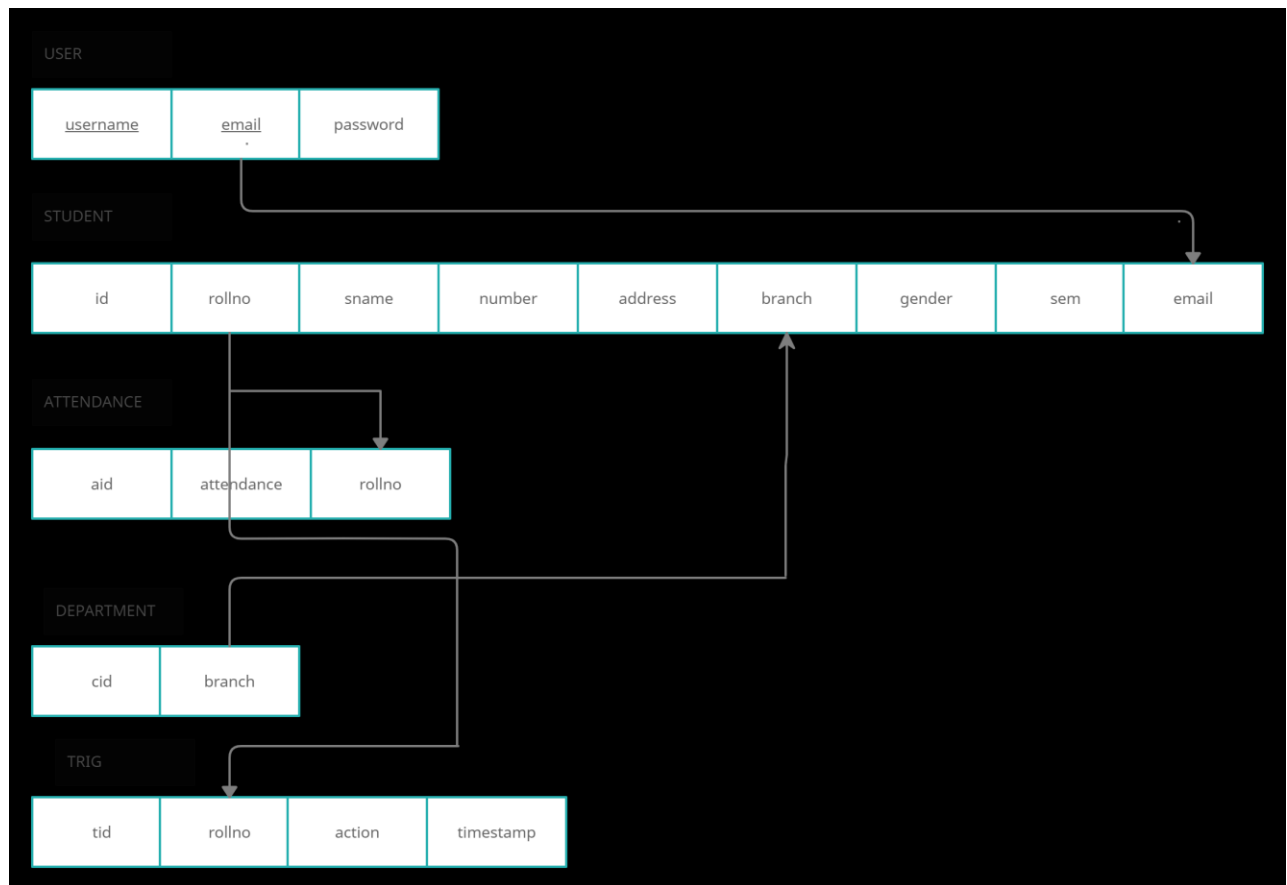
- Computer with a 1.1 GHz or faster processor
- Minimum 2GB of RAM or more
- 2.5 GB of available hard-disk space
- 5400 RPM hard drive
- 1366 × 768 or higher-resolution display
- DVD-ROM drive

## 3.2 CONCEPTUAL DESIGN:

### 3.2.1 E-R DIAGRAM:



### 3.2.2 SCHEMA DIAGRAM:



### 3.3 IMPLEMENTATION:

An "implementation" of Python should be taken to mean a program or environment which provides support for the execution of programs written in the Python language, as represented by the [CPython](#) reference implementation.

There have been and are several distinct software packages providing of what we all recognize as Python, although some of those are more like distributions or variants of some existing implementation than a completely new implementation of the language.

#### Back End (MySQL)

##### Database:

A Database Management System (DBMS) is computer software designed for the purpose of managing databases, a large set of structured data, and run operations on the data requested by numerous users. Typical examples of DBMSs include Oracle, DB2, Microsoft Access, Microsoft SQL Server, Firebird, PostgreSQL, MySQL, SQLite, FileMaker and Sybase Adaptive Server Enterprise. DBMSs are typically used by Database administrators in the creation of Database systems. Typical examples of DBMS use include accounting, human resources and customer support systems. Originally found only in large companies with the computer hardware needed to support large data sets, DBMSs have more recently emerged as a fairly standard part of any company back office.

A DBMS is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. A DBMS includes:

- A modeling language to define the schema of each database hosted in the DBMS, according to the DBMS data model.
- The dominant model in use today is the ad hoc one embedded in SQL, despite the objections of purists who believe this model is a corruption of the relational model, since it violates several of its fundamental principles for the sake of practicality and performance. Many DBMSs also support the Open Database Connectivity API that supports a standard way for programmers to access the DBMS.
- Data structures (fields, records, files and objects) optimized to deal with very large amounts of data stored on a permanent data storage device (which implies relatively slow access compared to volatile main memory).A database query language and report

writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.

- Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called sub schemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and student data.
  - If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.
- ✓ A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity, despite concurrent user accesses (concurrency control), and faults (fault tolerance).
- It also maintains the integrity of the data in the database.
  - The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database. See ACID properties for more information (Redundancy avoidance).

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. to the Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

### **SQL:**

Structured Query Language (SQL) is the language used to manipulate relational databases. SQL is tied very closely with the relational model.

- In the relational model, data is stored in structures called relations or tables.

SQL statements are issued for the purpose of:

- Data definition: Defining tables and structures in the database (DDL used to create, alter and drop schema objects such as tables and indexes)

#### **4.2 : Stored Procedure**

Routine name: proc

Type: procedure

Definition: Select \* from register;

#### **4.3: Triggers**

It is the special kind of stored procedure that automatically executes when an event occurs in the database.

Triggers used :

1: Trigger name: on insert

Table: register

Time: after

Event: insert

INSERT INTO trig VALUES(null,NEW.rid,'Farmer Inserted',NOW())

2: Trigger name: on delete

Table: register

Time: after

Event: delete

Definition: INSERT INTO trig VALUES(null,OLD.rid,'FARMER DELETED',NOW())

3: Trigger name: on update

Table: register

Time: after

Event: update

Definition: INSERT INTO trig VALUES(null,NEW.rid,'FARMER UPDATED',NOW())



## BACKEND PYHTON WITH MYSQL CODE

```
from flask import Flask,render_template,request,session,redirect,url_for,flash
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin

from werkzeug.security import generate_password_hash,check_password_hash
from flask_login import login_user,logout_user,login_manager,LoginManager
from flask_login import login_required,current_user
import json


# MY db connection
local_server= True
app = Flask(__name__)
app.secret_key='kusumachandashwini'


# this is for getting unique user access
login_manager=LoginManager(app)
login_manager.login_view='login'


@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


#
app.config['SQLALCHEMY_DATABASE_URL']='mysql://username:password@localhost/databas_table_
name'
app.config['SQLALCHEMY_DATABASE_URI']='mysql://root:@localhost/students'
db=SQLAlchemy(app)


# here we will create db models that is tables
class Test(db.Model):
    id=db.Column(db.Integer,primary_key=True)
    name=db.Column(db.String(100))
    email=db.Column(db.String(100))
```

## Students Management System

```
class Department(db.Model):  
    cid=db.Column(db.Integer,primary_key=True)  
    branch=db.Column(db.String(100))
```

```
class Attendance(db.Model):  
    aid=db.Column(db.Integer,primary_key=True)  
    rollno=db.Column(db.String(100))  
    attendance=db.Column(db.Integer())
```

```
class Trig(db.Model):  
    tid=db.Column(db.Integer,primary_key=True)  
    rollno=db.Column(db.String(100))  
    action=db.Column(db.String(100))  
    timestamp=db.Column(db.String(100))
```

```
class User(UserMixin,db.Model):  
    id=db.Column(db.Integer,primary_key=True)  
    username=db.Column(db.String(50))  
    email=db.Column(db.String(50),unique=True)  
    password=db.Column(db.String(1000))
```

```
class Student(db.Model):  
    id=db.Column(db.Integer,primary_key=True)  
    rollno=db.Column(db.String(50))  
    sname=db.Column(db.String(50))  
    sem=db.Column(db.Integer)  
    gender=db.Column(db.String(50))  
    branch=db.Column(db.String(50))  
    email=db.Column(db.String(50))  
    number=db.Column(db.String(12))  
    address=db.Column(db.String(100))
```

## Students Management System

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/studentdetails')
def studentdetails():
    query=db.engine.execute(f"SELECT * FROM `student`")
    return render_template('studentdetails.html',query=query)

@app.route('/triggers')
def triggers():
    query=db.engine.execute(f"SELECT * FROM `trig`")
    return render_template('triggers.html',query=query)

@app.route('/department',methods=['POST','GET'])
def department():
    if request.method=="POST":
        dept=request.form.get('dept')
        query=Department.query.filter_by(branch=dept).first()
        if query:
            flash("Department Already Exist","warning")
            return redirect('/department')
        dep=Department(branch=dept)
        db.session.add(dep)
        db.session.commit()
        flash("Department Addes","success")
    return render_template('department.html')

@app.route('/addattendance',methods=['POST','GET'])
def addattendance():
    query=db.engine.execute(f"SELECT * FROM `student`")
    if request.method=="POST":
        rollno=request.form.get('rollno')
        attend=request.form.get('attend')
        print(attend,rollno)
        atte=Attendance(rollno=rollno,attendance=attend)
        db.session.add(atte)
        db.session.commit()
```

## Students Management System

```
flash("Attendance added","warning")
return render_template('attendance.html',query=query)

@app.route('/search',methods=['POST','GET'])
def search():
    if request.method=="POST":
        rollno=request.form.get('roll')
        bio=Student.query.filter_by(rollno=rollno).first()
        attend=Attendance.query.filter_by(rollno=rollno).first()
        return render_template('search.html',bio=bio,attend=attend)

    return render_template('search.html')

@app.route("/delete/<string:id>",methods=['POST','GET'])
@login_required
def delete(id):
    db.engine.execute(f"DELETE FROM `student` WHERE `student`.`id`={id}")
    flash("Slot Deleted Successful","danger")
    return redirect('/studentdetails')

@app.route("/edit/<string:id>",methods=['POST','GET'])
@login_required
def edit(id):
    dept=db.engine.execute("SELECT * FROM `department`")
    posts=Student.query.filter_by(id=id).first()
    if request.method=="POST":
        rollno=request.form.get('rollno')
        sname=request.form.get('sname')
        sem=request.form.get('sem')
        gender=request.form.get('gender')
        branch=request.form.get('branch')
        email=request.form.get('email')
        num=request.form.get('num')
        address=request.form.get('address')
        query=db.engine.execute(f"UPDATE `student` SET
`rollno`='{rollno}',`sname`='{sname}',`sem`='{sem}',`gender`='{gender}',`branch`='{branch}',`email`='{em
ail}',`number`='{num}',`address`='{address}'")
```

## Students Management System

```
flash("Slot is Updates","success")
```

```
return redirect('/studentdetails')
```

```
return render_template('edit.html',posts=posts,dept=dept)
```

```
@app.route('/signup',methods=['POST','GET'])
```

```
def signup():
```

```
    if request.method == "POST":
```

```
        username=request.form.get('username')
```

```
        email=request.form.get('email')
```

```
        password=request.form.get('password')
```

```
        user=User.query.filter_by(email=email).first()
```

```
        if user:
```

```
            flash("Email Already Exist","warning")
```

```
            return render_template('/signup.html')
```

```
        encpassword=generate_password_hash(password)
```

```
        new_user=db.engine.execute(f"INSERT INTO `user` (`username`,`email`,`password`) VALUES  
('{'username}','{'email}','{'encpassword}'))")
```

```
        # this is method 2 to save data in db
```

```
        # newuser=User(username=username,email=email,password=encpassword)
```

```
        # db.session.add(newuser)
```

```
        # db.session.commit()
```

```
        flash("Signup Succes Please Login","success")
```

```
        return render_template('login.html')
```

```
return render_template('signup.html')
```

```
@app.route('/login',methods=['POST','GET'])
```

```
def login():
```

```
    if request.method == "POST":
```

```
        email=request.form.get('email')
```

```
        password=request.form.get('password')
```

```
        user=User.query.filter_by(email=email).first()
```

## Students Management System

```
if user and check_password_hash(user.password,password):
```

```
    login_user(user)
```

```
    flash("Login Success","primary")
```

```
    return redirect(url_for('index'))
```

```
else:
```

```
    flash("invalid credentials","danger")
```

```
    return render_template('login.html')
```

```
return render_template('login.html')
```

```
@app.route('/logout')
```

```
@login_required
```

```
def logout():
```

```
    logout_user()
```

```
    flash("Logout SuccessFul","warning")
```

```
    return redirect(url_for('login'))
```

```
@app.route('/addstudent',methods=['POST','GET'])
```

```
@login_required
```

```
def addstudent():
```

```
    dept=db.engine.execute("SELECT * FROM `department`")
```

```
    if request.method=="POST":
```

```
        rollno=request.form.get('rollno')
```

```
        sname=request.form.get('sname')
```

```
        sem=request.form.get('sem')
```

```
        gender=request.form.get('gender')
```

```
        branch=request.form.get('branch')
```

```
        email=request.form.get('email')
```

```
        num=request.form.get('num')
```

```
        address=request.form.get('address')
```

```
        query=db.engine.execute(f"INSERT INTO `student`
```

```
(`rollno`,`sname`,`sem`,`gender`,`branch`,`email`,`number`,`address`) VALUES
```

```
('{'rollno}','{'sname}','{'sem}','{'gender}','{'branch}','{'email}','{'num}','{'address}'))")
```

## Students Management System

```
flash("Booking Confirmed","info")
```

```
return render_template('student.html',dept=dept)
```

```
@app.route('/test')
```

```
def test():
```

```
    try:
```

```
        Test.query.all()
```

```
        return 'My database is Connected'
```

```
    except:
```

```
        return 'My db is not Connected'
```

```
app.run(debug=True)
```

## FRONT END CODE

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="utf-8">
```

```
<meta content="width=device-width, initial-scale=1.0" name="viewport">
```

```
<title>{ % block title % }
```

```
{ % endblock title % }</title>
```

```
<meta content="" name="description">
```

```
<meta content="" name="keywords">
```

```
{ % block style % }
```

```
{ % endblock style % }
```

```
<link
```

```
href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,700,700i|Raleway:300,400,500,700,800" rel="stylesheet">
```

```
<!-- Vendor CSS Files -->
```

```
<link href="static/assets/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
```

```
<link href="static/assets/vendor/venobox/venobox.css" rel="stylesheet">
```

```
<link href="static/assets/vendor/font-awesome/css/font-awesome.min.css" rel="stylesheet">
```

```
<link href="static/assets/vendor/owl.carousel/assets/owl.carousel.min.css" rel="stylesheet">
```

```
<link href="static/assets/vendor/aos/aos.css" rel="stylesheet">
```

```
<!-- Template Main CSS File -->
```

```
<link href="static/assets/css/style.css" rel="stylesheet">
```

```
</head>
```

```
<body>
```



```
<!-- ===== Header ===== -->
<header id="header">
  <div class="container">

    <div id="logo" class="pull-left">

      <a href="/" class="scrollto">S.M.S</a>
    </div>

    <nav id="nav-menu-container">
      <ul class="nav-menu">
        <li class="{ % block home % }
          { % endblock home % }"><a href="/">Home</a></li>

        <li><a href="/addstudent">Students</a></li>
        <li><a href="/addattendance">Attendance</a></li>
        <li><a href="/department">Department</a></li>
        <li><a href="/triggers">Records</a></li>
        <li><a href="/studentdetails">Student Details</a></li>
        <li><a href="/search">Search</a></li>

        <li><a href="/about">About</a></li>

        { % if current_user.is_authenticated % }
        <li class="buy-tickets"><a href="">Welcome</a></li>
        <li class="buy-tickets"><a href="/logout">Logout</a></li>
        { % else % }
        <li class="buy-tickets"><a href="/signup">Signin</a></li>

        { % endif % }
      </ul>
    </nav><!-- #nav-menu-container -->
```

```
</div>
</header><!-- End Header -->

<!-- ===== Intro Section ===== -->
<section id="intro">
  <div class="intro-container" data-aos="zoom-in" data-aos-delay="100">
    <h1 class="mb-4 pb-0">STUDENT MANAGEMENT SYSTEM </span> </h1>
    <p class="mb-4 pb-0">DBMS Mini Project Using Flask & MYSQL</p>

    <a href="" class="about-btn scrollto">View More</a>
  </div>
</section><!-- End Intro Section -->
<main id="main">

  { % block body % }

  { % with messages=get_flashed_messages(with_categories=true) % }
  { % if messages % }
  { % for category, message in messages % }

  <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
    {{ message }}

  </div>

  { % endfor % }
  { % endif % }
  { % endwith % }
  { % endblock body % }

  <a href="#" class="back-to-top"><i class="fa fa-angle-up"></i></a>
```

<!-- Vendor JS Files -->

<script src="static/assets/vendor/jquery/jquery.min.js"></script>

<script src="static/assets/vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

<script src="static/assets/vendor/jquery.easing/jquery.easing.min.js"></script>

<script src="static/assets/vendor/php-email-form/validate.js"></script>

<script src="static/assets/vendor/venobox/venobox.min.js"></script>

<script src="static/assets/vendor/owl.carousel/owl.carousel.min.js"></script>

<script src="static/assets/vendor/superfish/superfish.min.js"></script>

<script src="static/assets/vendor/hoverIntent/hoverIntent.js"></script>

<script src="static/assets/vendor/aos/aos.js"></script>

<!-- Template Main JS File -->

<script src="static/assets/js/main.js"></script>

</body>

</html>

## 2.Students.html

{% extends 'base.html' %}

{% block title %}

Add Students

{% endblock title %}

{% block body %}

<h3 class="text-center"><span>Add Student Details</span> </h3>

{% with messages=get\_flashed\_messages(with\_categories=true) %}

{% if messages %}

{% for category, message in messages %}

```
<div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
    {{ message }}
```

```
</div>
```

```
{% endfor %}
```

```
{% endif %}
```

```
{% endwith %}
```

```
<br>
```

```
<div class="container">
```

```
<div class="row">
```

```
<div class="col-md-4"></div>
```

```
<div class="col-md-4">
```

```
<form action="/addstudent" method="post">
```

```
<div class="form-group">
```

```
<label for="rollno">Roll Number</label>
```

```
<input type="text" class="form-control" name="rollno" id="rollno">
```

```
</div>
```

```
<br>
```

```
<div class="form-group">
```

```
<label for="sname">Student Name</label>
```

```
<input type="text" class="form-control" name="sname" id="sname">
```

```
</div>
```

```
<br>
```

```
<div class="form-group">
```

```
<label for="sem">Sem</label>
```

```
<input type="number" class="form-control" name="sem" id="sem">
```

```
</div>
```

```
<br>
```

```
<div class="form-group">
<select class="form-control" id="gender" name="gender" required>
  <option selected>Select Gender</option>

  <option value="male">Male</option>
  <option value="female">Female</option>

</select>
</div>
<br>

<div class="form-group">
<select class="form-control" id="branch" name="branch" required>
  <option selected>Select Branch</option>
  {% for d in dept %}
  <option value="{{ d.branch }}">{{ d.branch }}</option>
  {% endfor %}
</select>
</div>
<br>
<div class="form-group">

<label for="email">Email</label>
<input type="email" class="form-control" name="email" id="email">
</div>
<br>
<div class="form-group">
<label for="num">Phone Number</label>
<input type="number" class="form-control" name="num" id="num">
</div>
<br>

<div class="form-group">
<label for="address">Address</label>
<textarea class="form-control" name="address" id="address"></textarea>
```

</div>

<br>

<button type="submit" class="btn btn-danger btn-sm btn-block">Add Record</button>

</form>

<br>

<br>

</div>

<div class="col-md-4"></div>

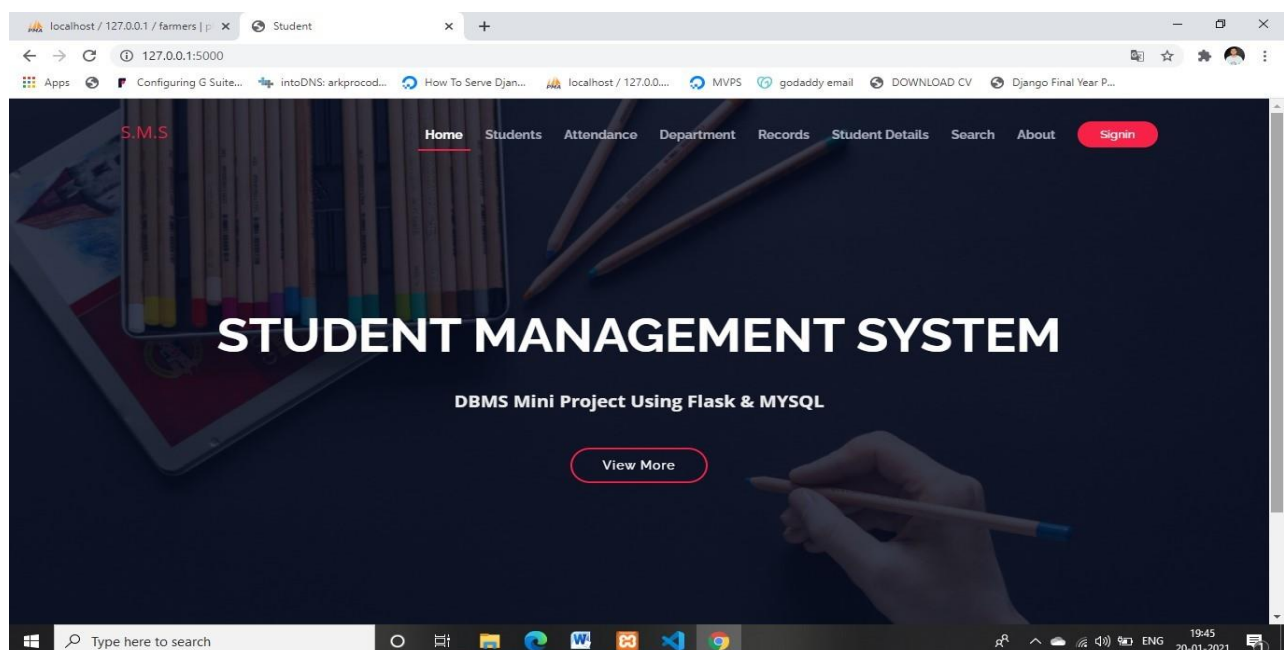
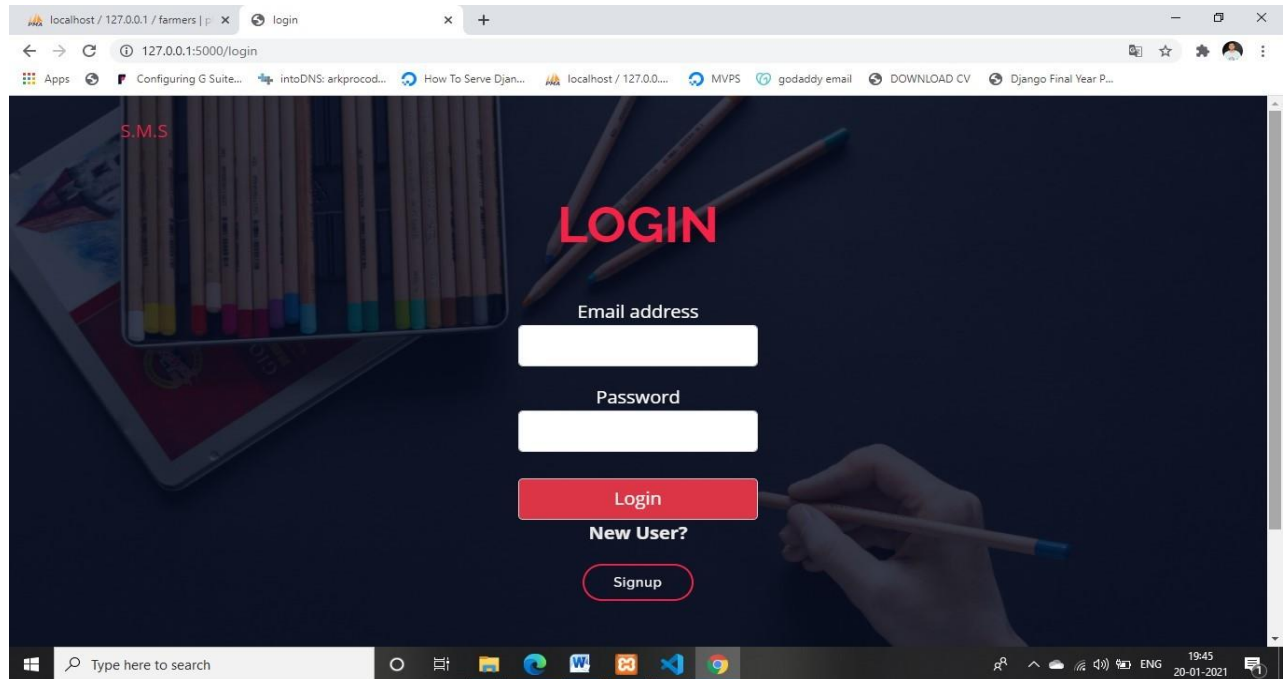
</div></div>

{% endblock body %}

## USER INTERFACE

### 4.1 SCREEN SHOTS

#### LOGIN PAGE:



## ADD STUDENTS INFO

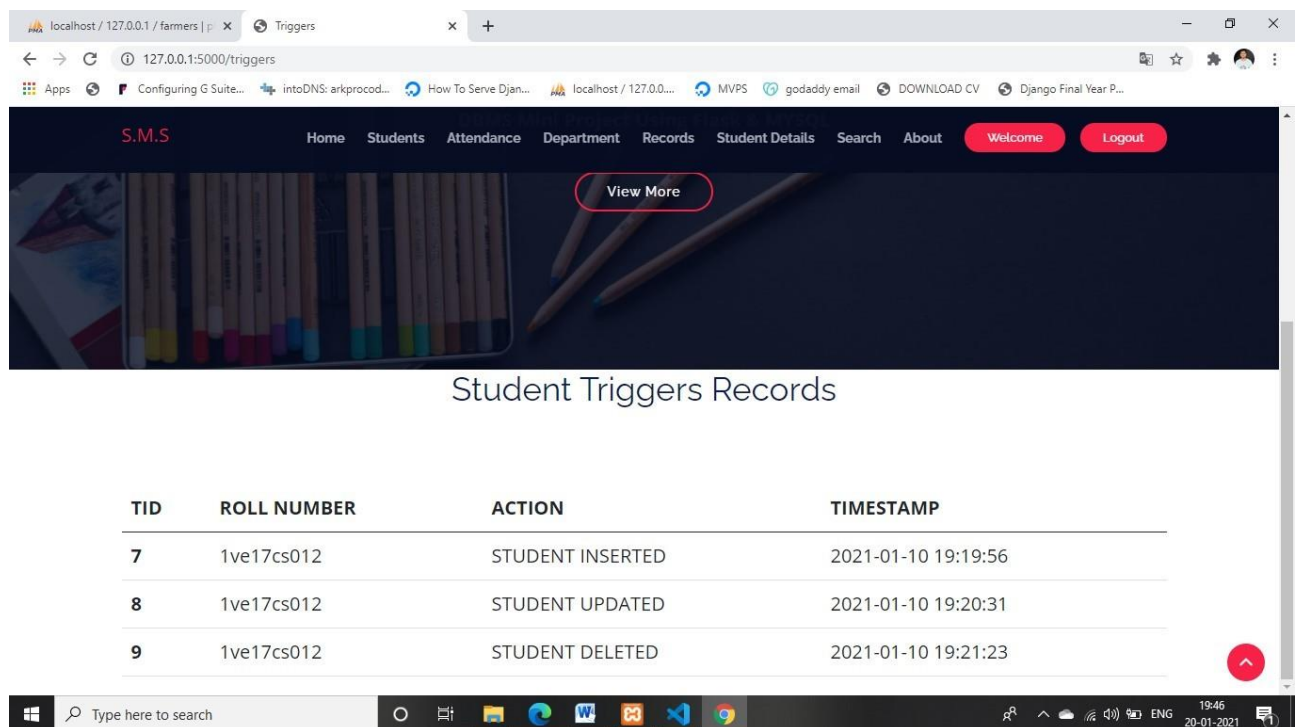
The screenshot shows a web browser window with the URL `127.0.0.1:5000/addstudent`. The application header includes the logo 'S.M.S' and navigation links: Home, Students, Attendance, Department, Records, Student Details, Search, and About. There are 'Welcome' and 'Logout' buttons. The main content area is titled 'Add Student Details' and contains four input fields: 'Roll Number', 'Student Name', 'Sem', and 'Select Gender'. A red 'Add' button is visible at the bottom right of the form area. The Windows taskbar at the bottom shows the search bar and several open applications.

The screenshot shows the 'Student Details' page with a banner for 'DBMS Mini Project Using Flask & MYSQL' and a 'View More' button. Below the banner is a table listing student information. The table has columns for SID, ROLL NUMBER, STUDENT NAME, SEM, GENDER, BRANCH, EMAIL, NUMBER, ADDRESS, EDIT, and DELETE. One student record is displayed: SID 7, ROLL NUMBER 1234, STUDENT NAME rohit, SEM 3, GENDER male, BRANCH Electronic and Communication, EMAIL rohit@gmail.com, NUMBER 9986786453, and ADDRESS bangalore. The 'EDIT' and 'DELETE' buttons are highlighted with red circles.

SID	ROLL NUMBER	STUDENT NAME	SEM	GENDER	BRANCH	EMAIL	NUMBER	ADDRESS	EDIT	DELETE
7	1234	rohit	3	male	Electronic and Communication	rohit@gmail.com	9986786453	bangalore	Edit	Delete



## TRIGGERS RECORDS



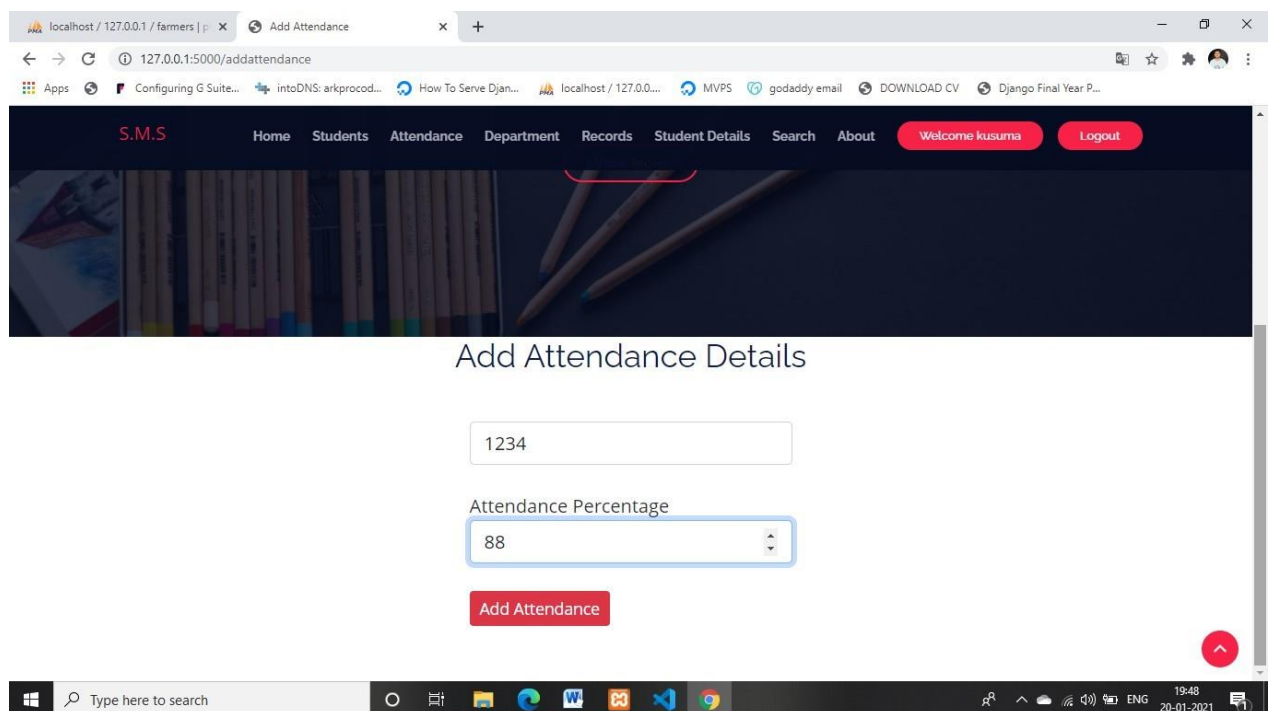
S.M.S.

Home Students Attendance Department Records Student Details Search About Welcome Logout

View More

### Student Triggers Records

TID	ROLL NUMBER	ACTION	TIMESTAMP
7	1ve17cs012	STUDENT INSERTED	2021-01-10 19:19:56
8	1ve17cs012	STUDENT UPDATED	2021-01-10 19:20:31
9	1ve17cs012	STUDENT DELETED	2021-01-10 19:21:23



S.M.S.

Home Students Attendance Department Records Student Details Search About Welcome kusuma Logout

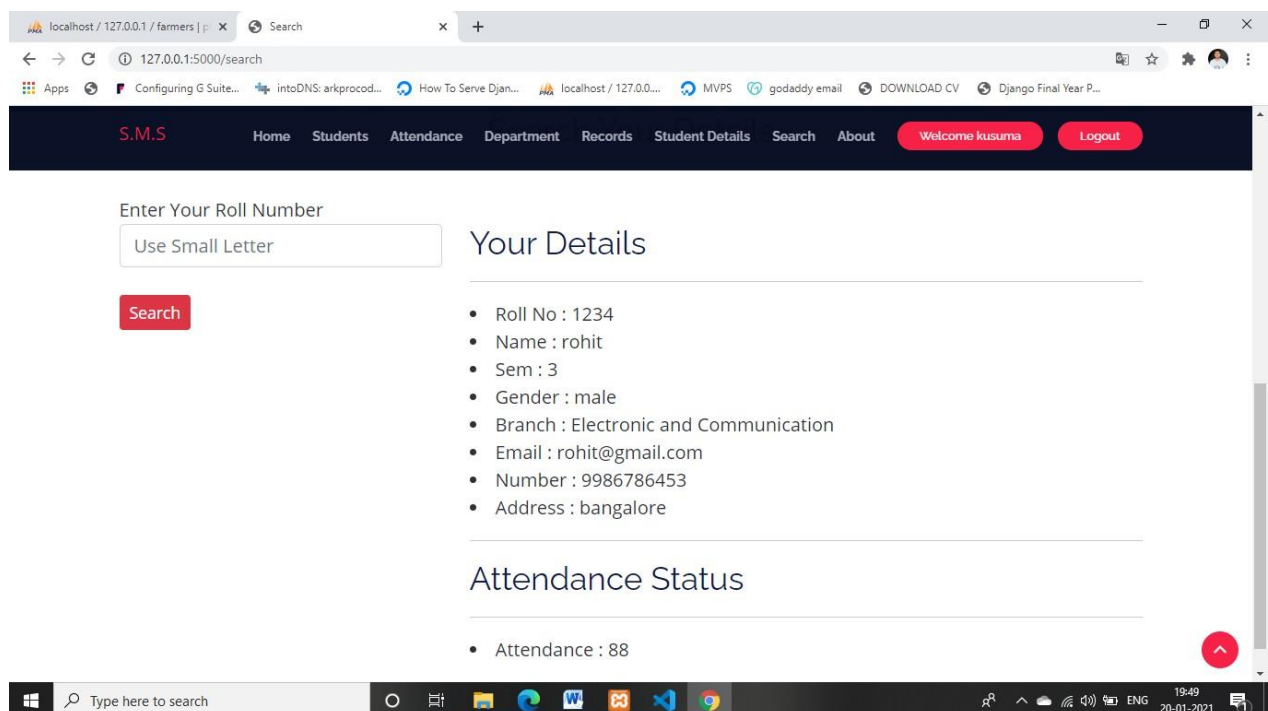
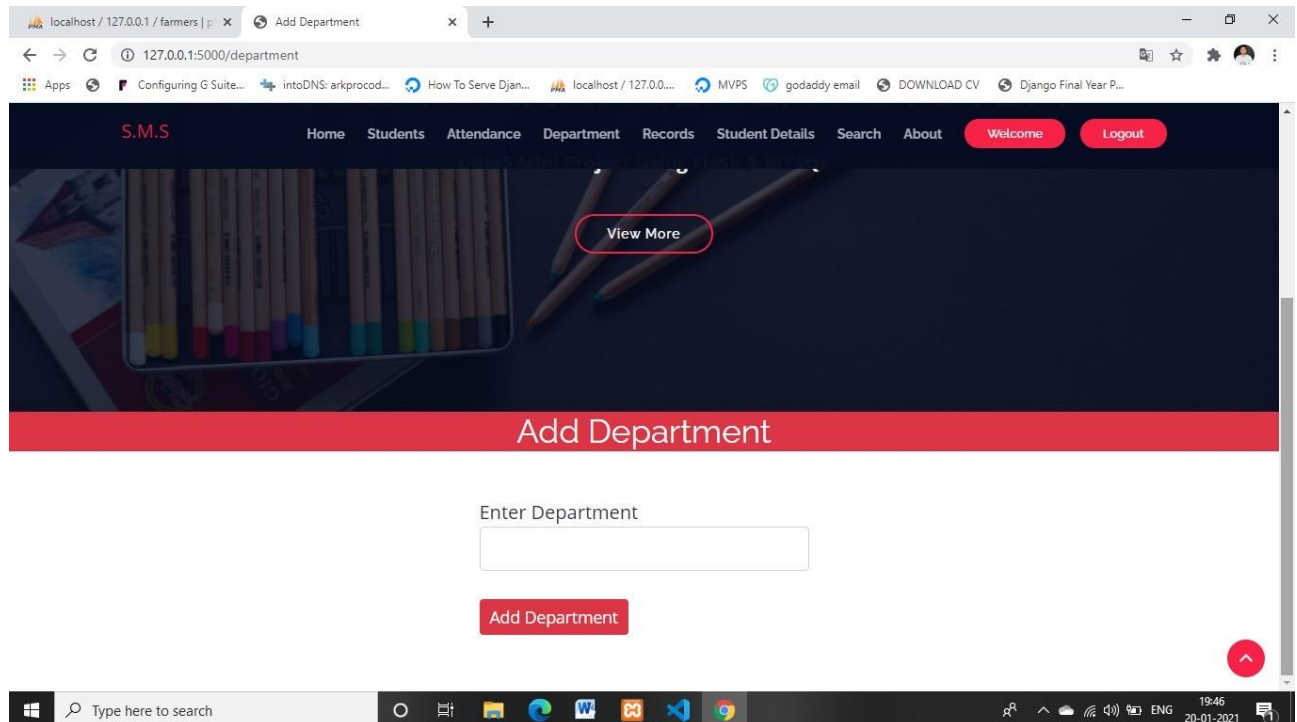
### Add Attendance Details

1234

Attendance Percentage

88

Add Attendance



## DATABASE LOCALHOST

The screenshot shows the phpMyAdmin interface for the 'students' database. The left sidebar lists the database structure, including tables like 'attendance', 'department', 'student', 'test', 'trig', and 'user'. The main panel displays the 'Structure' tab for the 'students' database, showing a list of tables with their respective actions (Browse, Structure, Search, Insert, Empty, Drop) and details (Rows, Type, Collation, Size, Overhead).

Table	Action	Rows	Type	Collation	Size	Overhead
attendance	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
department	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_general_ci	16.0 KiB	-
student	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
test	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
trig	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
user	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<b>6 tables</b>	<b>Sum</b>	<b>16</b>	<b>InnoDB</b>	<b>utf8mb4_general_ci</b>	<b>96.0 KiB</b>	<b>0 B</b>

The screenshot shows the phpMyAdmin interface for the 'student' table. The left sidebar lists the database structure, including tables like 'attendance', 'department', 'student', 'test', 'trig', and 'user'. The main panel displays the 'Structure' tab for the 'student' table, showing the table structure and data. The table has columns: id, rollno, sname, sem, gender, branch, email, number, and address.

id	rollno	sname	sem	gender	branch	email	number	address
7	1234	rohit	3	male	Electronic and Communication	rohit@gmail.com	9986786453	bangalore

## Students Management System

The screenshot shows the phpMyAdmin interface for the 'students' database. The 'trig' table is selected, and the SQL query 'SELECT \* FROM `trig`' is executed. The results show 4 rows of student activity logs.

	tid	rollno	action	timestamp
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	7	1ve17cs012	STUDENT INSERTED	2021-01-10 19:19:56
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	8	1ve17cs012	STUDENT UPDATED	2021-01-10 19:20:31
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	9	1ve17cs012	STUDENT DELETED	2021-01-10 19:21:23
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	10	1234	STUDENT INSERTED	2021-01-20 19:48:07

The screenshot shows the phpMyAdmin interface for the 'students' database. The 'user' table is selected, and the SQL query 'SELECT \* FROM `user`' is executed. The results show 2 rows of user information.

	id	username	email	password
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	4	anees	anees@gmail.com	pbkdf2-sha256:150000\$1CSLss89\$ef995dfc48121768b207...
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	5	kusuma	kusuma@gmail.com	pbkdf2-sha256:150000\$s3QvAXVg\$7092cf5aca424c364dd0...

## Students Management System

The screenshot displays the phpMyAdmin web interface in a browser. The address bar shows the URL: `localhost/phpmyadmin/sql.php?db=students&table=attendance&pos=0`. The interface is titled "Server: 127.0.0.1 » Database: students » Table: attendance".

On the left sidebar, the database structure is visible, including a tree for "students" with tables like "attendance", "department", "student", "test", "trig", "user", "test", "test111", "testing", "weblab", and "xiome".

The main panel shows the "attendance" table with the following details:

- Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)
- SQL query: `SELECT * FROM `attendance``
- Options: `aid`, `rollno`, `attendance`
- Table data (1 row):

aid	rollno	attendance
7	1234	88
- Actions: Edit, Copy, Delete, Check all, With selected, Edit, Copy, Delete, Export
- Query results operations: Print, Copy to clipboard, Export, Display chart, Create view

The bottom of the image shows the Windows taskbar with the search bar and various application icons. The system clock indicates 19:58 on 20-01-2021.

## CONCLUSION

STUDENT MANAGEMENT SYSTEM successfully implemented based on online data filling which helps us in administrating the data user for managing the tasks performed in students. The project successfully used various functionalities of Xampp and python flask and also create the fully functional database management system for online portals.

Using MySQL as the database is highly beneficial as it is free to download, popular and can be easily customized. The data stored in the MySQL database can easily be retrieved and manipulated according to the requirements with basic knowledge of SQL.

With the theoretical inclination of our syllabus it becomes very essential to take the utmost advantage of any opportunity of gaining practical experience that comes along. The building blocks of this Major Project “Students Management System” was one of these opportunities. It gave us the requisite practical knowledge to supplement the already taught theoretical concepts thus making us more competent as a computer engineer. The project from a personal point of view also helped us in understanding the following aspects of project development:

- The planning that goes into implementing a project.
- The importance of proper planning and an organized methodology.
- The key element of team spirit and co-ordination in a successful project.



## **FUTURE ENHANCEMENT**

- Enhanced database storage facility
- Enhanced user friendly GUI
- more advanced results systems
- online feedbacks forms

## REFERENCES

- <https://www.youtube.com>
- <https://www.google.com>
- <http://www.getbootstrap.com>





