MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

SCHOOL MANAGEMENT SYSTEM

Barri Harshith - 54 - 230905474

Harsha NK - 58- 230905498

Abstract

The aim of this project is to design and implement a Student Database Management System (DBMS) that efficiently manages and maintains student-related data for an academic institution. The system provides a user-friendly interface for administrators to handle student records, including personal details, academic performance, department allocation, and attendance tracking.

Built using Flask (a Python web framework) and MySQL, the project emphasizes the integration of backend database management with a functional and interactive web interface. Key features include CRUD operations on student data, department management, user authentication, and the implementation of database triggers to log changes in student records.

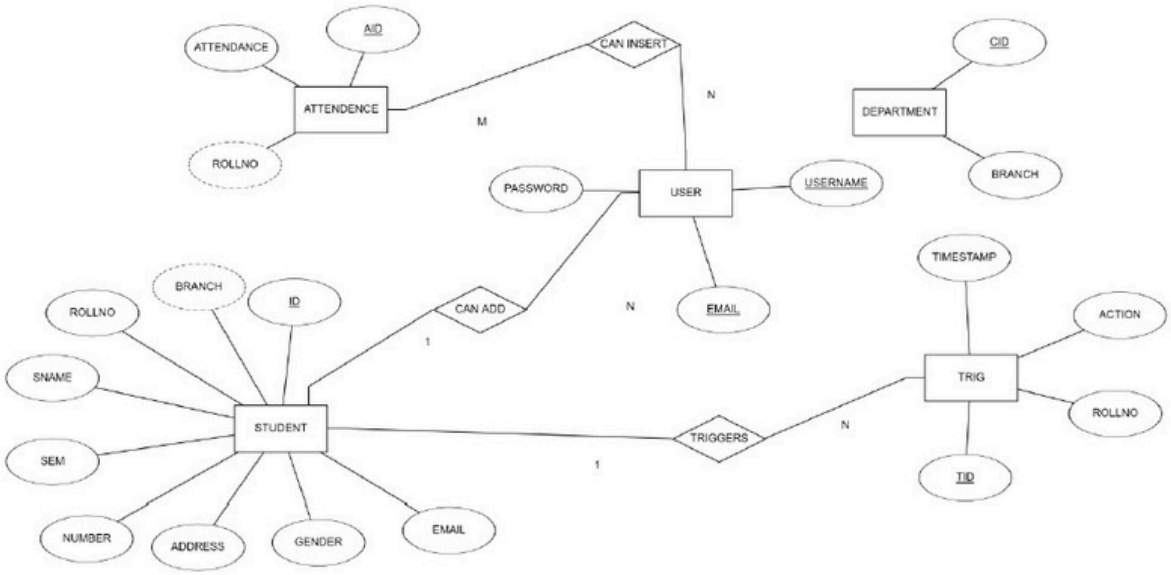
The system also supports the execution of basic and complex SQL queries, ensuring robust data retrieval and manipulation. Additionally, the use of triggers enables tracking modifications in a secure and auditable manner. This project demonstrates a practical understanding of database design principles, normalization, entity-relationship modeling, and real-time application of SQL-based operations in a web environment.

Problem Statement

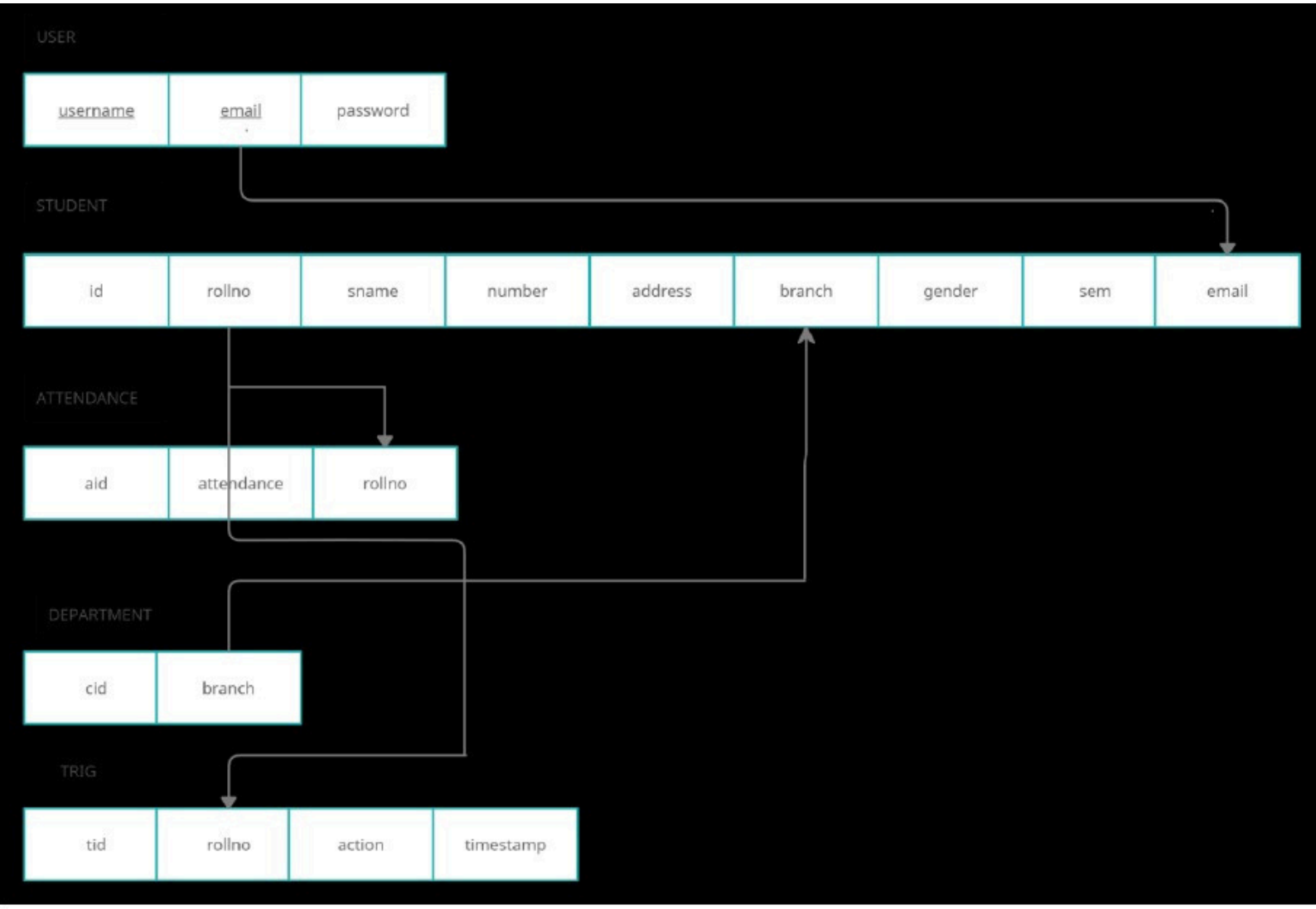
In academic institutions, managing student data efficiently is a challenging task due to the growing number of students, departments, and administrative operations. Traditional methods, such as paper-based records or disconnected spreadsheets, are inefficient, error-prone, and difficult to scale. This project addresses the need for a centralized, reliable, and scalable Student Database Management System (DBMS) that automates data handling, reduces manual effort, and provides real-time access to information.
The goal is to create a system that can:

- In this project, MySQL serves as the core database engine, providing structured data storage, retrieval, and manipulation capabilities. The MySQL database is responsible for:
- Storing Student Information: Tables store structured data such as names, roll numbers, contact details, department, marks, and attendance.
- Department Management: Includes a separate department table with relations to student records, allowing efficient categorization and filtering.
- User Authentication: A login system backed by MySQL ensures secure access for administrators.
- CRUD Operations: MySQL allows Create, Read, Update, and Delete operations on student and department records through SQL queries integrated with Flask.
- Triggers: A database trigger automatically logs updates made to the student table into a separate student_log table, ensuring traceability of change
- Foreign Key Constraints: Enforces referential integrity between related tables (e.g., students and departments).
- Efficient Query Execution: Ensures optimized performance for data search and manipulation operations.

ER Diagram


ATTENDANCE
AID
ATTENDENCE
ROLLNO
CAN INSERT
M
N
DEPARTMENT
CID
BRANCH
PASSWORD
USER
USERNAME
EMAIL
CAN ADD
1
N
BRANCH
ROLLNO
ID
SNAME
SEM
STUDENT
NUMBER
ADDRESS
GENDER
EMAIL
TRIGGERS
1
N
TIMESTAMP
ACTION
TRIG
ROLLNO
TID

Schema Diagram



DDL For Table Creation

- Attendance:

```
CREATE TABLE `attendence` (
  `aid` int(11) NOT NULL,
  `rollno` varchar(20) NOT NULL,
  `attendance` int(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `attendence` (`aid`, `rollno`, `attendance`) VALUES
(6, '1ve17cs012', 98);
```

- Department:

```
CREATE TABLE `department` (
  `cid` int(11) NOT NULL,
  `branch` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `department` (`cid`, `branch`) VALUES
(2, 'Information Science'),
(3, 'Electronic and Communication'),
(4, 'Electrical & Electronic'),
(5, 'Civil '),
(7, 'computer science'),
(8, 'IOT');
```

- Student:

```
CREATE TABLE `student` (
  `id` int(11) NOT NULL,
  `rollno` varchar(20) NOT NULL,
  `sname` varchar(50) NOT NULL,
  `sem` int(20) NOT NULL,
  `gender` varchar(50) NOT NULL,
  `branch` varchar(50) NOT NULL,
  `email` varchar(50) NOT NULL,
  `number` varchar(12) NOT NULL,
  `address` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

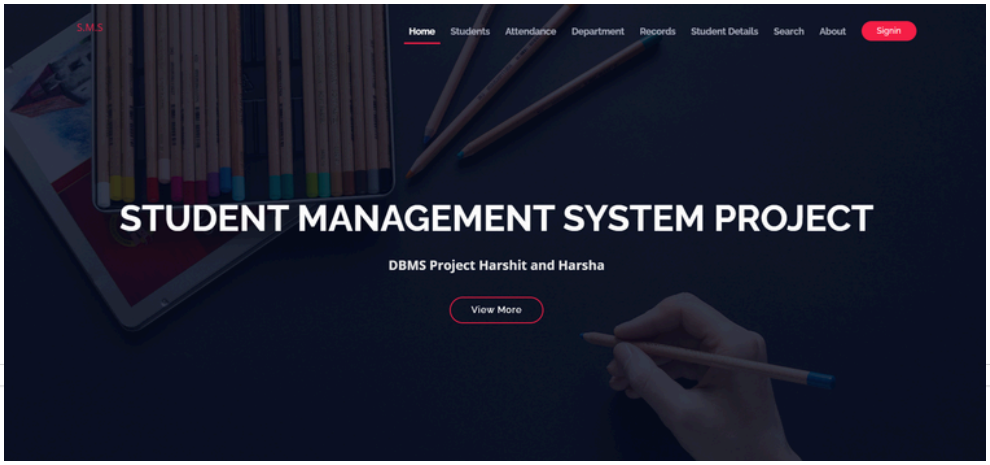
- Triggers:

```
DELIMITER $$
CREATE TRIGGER `DELETE` BEFORE DELETE ON `student` FOR EACH ROW INSERT INTO trig VALUES(null,OLD.rollno,'STUDENT DELETED',NOW())
$$
DELIMITER ;
DELIMITER $$
CREATE TRIGGER `Insert` AFTER INSERT ON `student` FOR EACH ROW INSERT INTO trig VALUES(null,NEW.rollno,'STUDENT INSERTED',NOW())
$$
DELIMITER ;
DELIMITER $$
CREATE TRIGGER `UPDATE` AFTER UPDATE ON `student` FOR EACH ROW INSERT INTO trig VALUES(null,NEW.rollno,'STUDENT UPDATED',NOW())
$$
DELIMITER ;
```

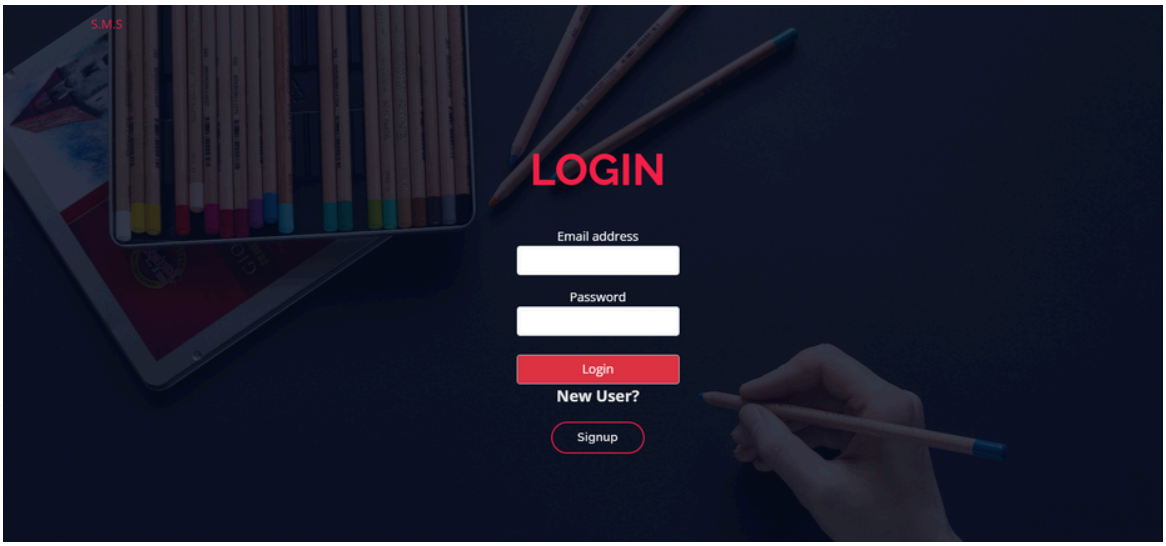
Result

The user interface (UI) of the Student Database Management System was built using a combination of HTML, CSS, Flask, and PHP. HTML was used to structure the web pages, while CSS provided styling to ensure a clean and responsive layout. Flask, a lightweight Python web framework, served as the backend to handle routing, form submissions, and interaction with the MySQL database. PHP was also integrated for certain server-side operations, particularly to handle dynamic content and session management. Together, these technologies created an intuitive and interactive UI, allowing users to add, update, and view student records seamlessly.

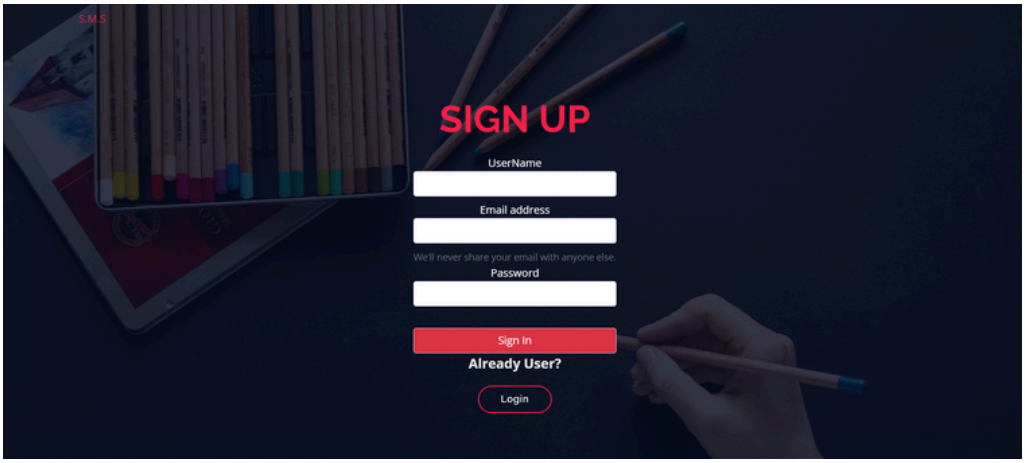
• Welcome Page:



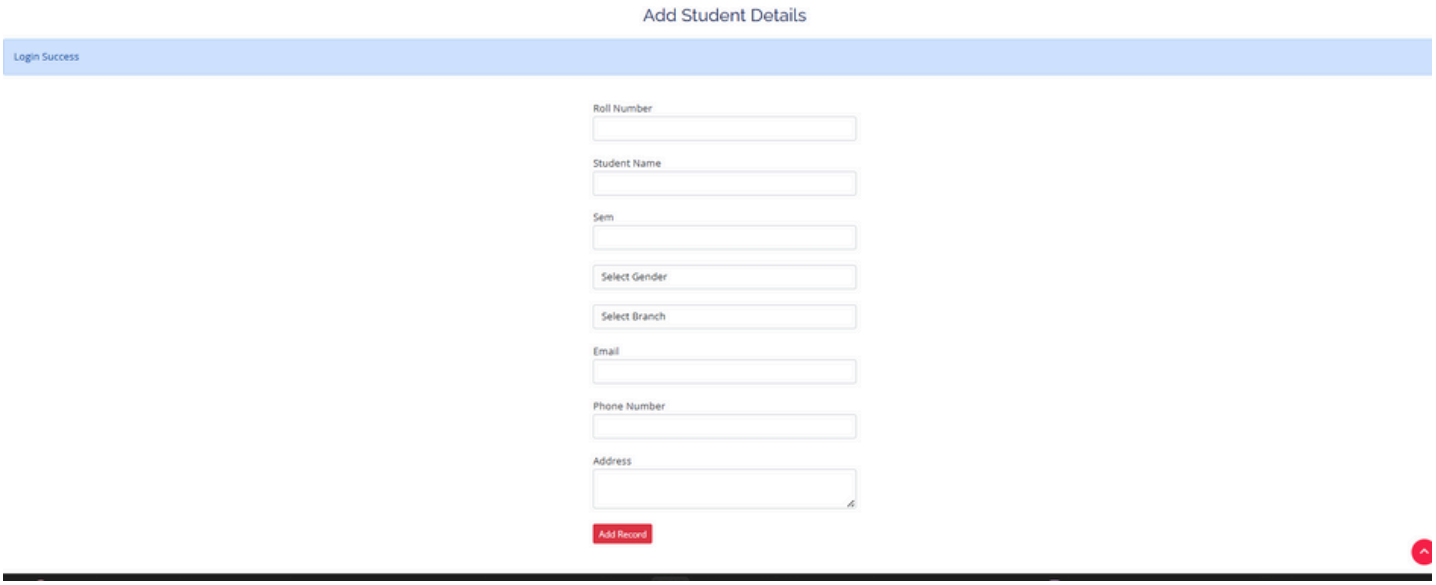
- Login Page:



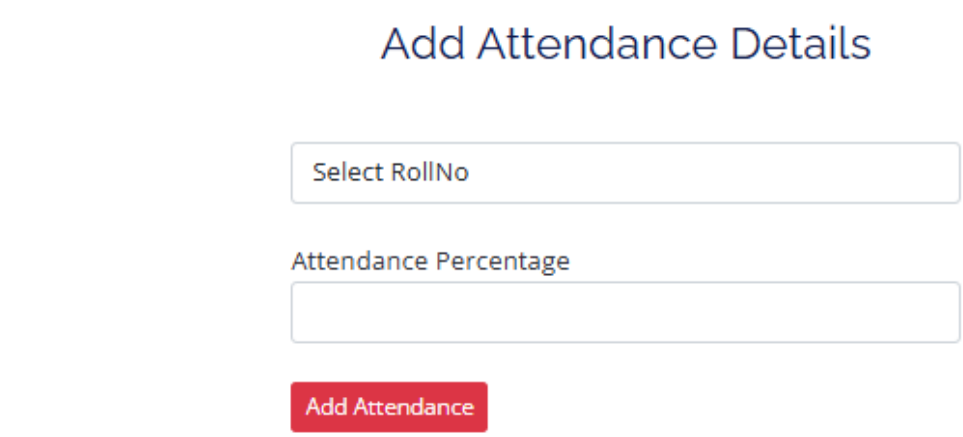
- Sign-Up Page:



- Add Student Record:



- Add Attendance:



- Trigger Record:

Student Triggers Records

TID	ROLL NUMBER	ACTION	TIMESTAMP
7	1ve17cs012	STUDENT INSERTED	2025-05-10 19:19:56
8	1ve17cs012	STUDENT UPDATED	2025-06-15 19:20:31
9	1ve17cs012	STUDENT DELETED	2025-12-29 00:00:00
10	12	STUDENT INSERTED	2025-04-02 09:18:12

Main 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='harshith'

# 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@localh
ost/databas_table_name'
app.config['SQLALCHEMY_DATABASE_URI']='mysql://root:@localhost/studentdbms'
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))
```

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

class Attendence(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))
```

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

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

@app.route('/triggers')
def triggers():
   # query=db.engine.execute(f"SELECT * FROM trig")
   query=Trig.query.all()
   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")
   query=Student.query.all()
   if request.method=="POST":
      rollno=request.form.get('rollno')
      attend=request.form.get('attend')
      print(attend,rollno)
```

```
 atte=Attendence(rollno=rollno,attendance=attend)
 db.session.add(atte)
 db.session.commit()
 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=Attendence.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):
   post=Student.query.filter_by(id=id).first()
   db.session.delete(post)
   db.session.commit()
   # 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")
   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='{ema
il}',number='{num}',address='{address}'")

post=Student.query.filter_by(id=id).first()
      post.rollno=rollno
      post.sname=sname
      post.sem=sem
      post.gender=gender
      post.branch=branch
      post.email=email
      post.number=num
      post.address=address
      db.session.commit()
      flash("Slot is Updates","success")
      return redirect('/studentdetails')
   dept=Department.query.all()
   posts=Student.query.filter_by(id=id).first()
   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=password)
        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()

        # if user and check_password_hash(user.password,password):
        if user and 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")
   dept=Department.query.all()
   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}')")

query=Student(rollno=rollno,sname=sname,sem=sem,gender=gender,branch=branch,email=
email,number=num,address=address)
      db.session.add(query)
      db.session.commit()

      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)
```