Student Management system

Submitted By

Student Name	Student ID
Md. Mohiuddin Saker	0242310005101134
Imtiaz Ahamed	0242310005101280
Pm Tasmin Hossain Togor	0242310005101974

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **C312: Subject Name in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY

Dhaka, Bangladesh

November 12, 2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Name of the course teacher**, **course teacher's Designation**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

_____

Course Teacher's Name
Designation
Department of Computer Science and Engineering Daffodil International University

Submitted by

<table>
<tr><td colspan="2">_____
Student Name Student
ID:
Dept. of CSE, DIU</td></tr>
<tr><td>_____
Student Name Student
ID:
Dept. of CSE, DIU</td><td>_____
Student Name Student
ID:
Dept. of CSE, DIU</td></tr>
<tr><td>_____
Student Name Student
ID:
Dept. of CSE, DIU</td><td>_____
Student Name Student
ID:
Dept. of CSE, DIU</td></tr>
</table>

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:.

Table 1: Course Outcome Statements

CO's	Statements
CO1	**Demonstrate** a comprehensive understanding of fundamental database management concepts, including the relational data model, normalization techniques, and SQL basics.
CO2	**Design, implement** and optimize relational databases with their corresponding dependencies, incorporating advanced SQL queries, indexing techniques and query optimization strategies.
CO3	Understand and **Analyze** security measures, distributed database architectures and emerging trends in database management, demonstrating an understanding of the broader context, tools and challenges in the field.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C2, A2	KP3, KP4	EP1, EP2, EP4
CO2	PO3	C3, A3	KP5	EP1, EP2, EP7
CO3	PO5	C4, A4	KP6	EP1, EP4, EP5

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Chapter 1

Introduction

1.1 Introduction

The **Student Management System** is a console-based application developed using Python. It helps manage student data efficiently by performing operations such as adding, viewing, updating, and deleting student records.

1.2 Motivation

In educational institutions, maintaining student records manually is often time-consuming and prone to errors. The motivation behind developing the **Student Management System** was to create an efficient, reliable, and user-friendly tool that could automate this process.

1.3 Objectives

To create a simple and user-friendly interface for managing student records.

To simulate a real-world application involving CRUD operations (Create, Read, Update, Delete).

1.4 Feasibility Study

The project is technically feasible as it uses Python, a well-documented and widely supported programming language. The system only requires a basic development environment (like VS Code or any IDE), and no special hardware or software is needed.

1.5 Gap Analysis

- 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.

1.6 Project Outcome

- A fully functional console-based system capable of **adding, viewing, updating, searching, and deleting** student records.
- A simple, user-friendly interface that can be operated without any technical expertise.
- Enhancement of practical knowledge in **Python programming**, particularly with **file handling**, **dictionaries**, and **modular code structure**.

Chapter 2

Proposed Methodology/Architecture

2.1 Requirement Analysis & Design Specification

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

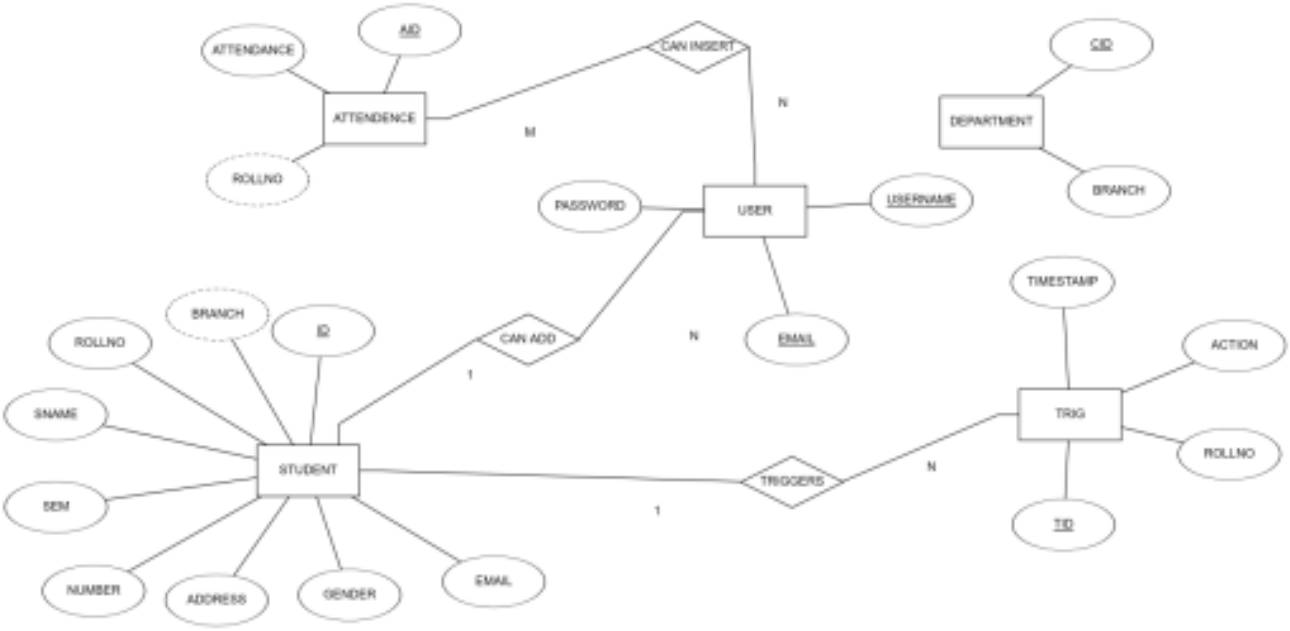
· 5400 RPM hard drive

· 1366 × 768 or higher-resolution display

· DVD-ROM drive

2.1.1 Overview

E-R DIAGRAM:



SCHEMA DIAGRAM:

2.1.2 Proposed Methodology/ System Design

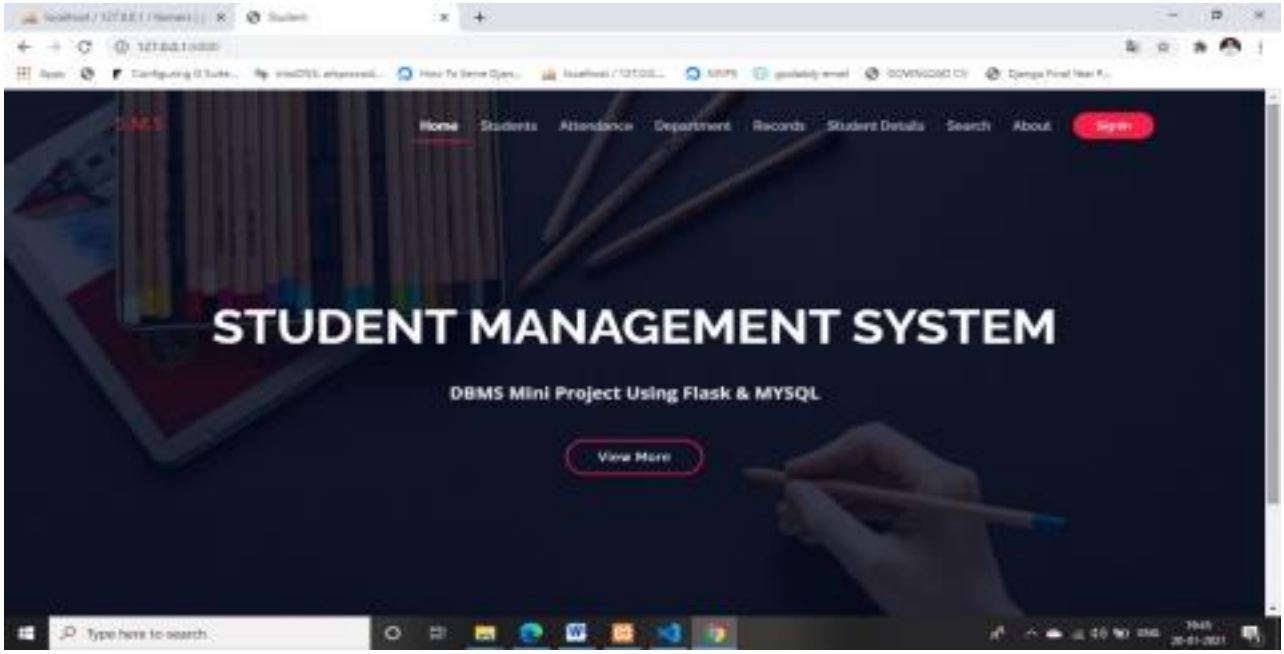
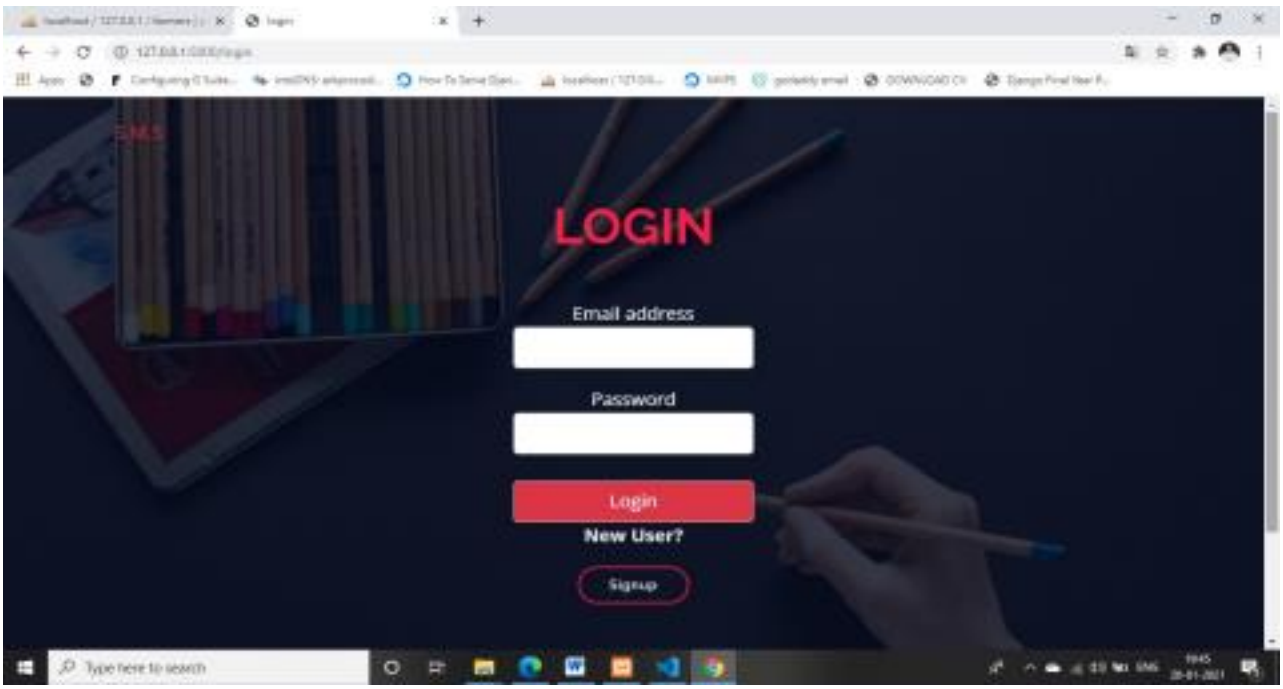
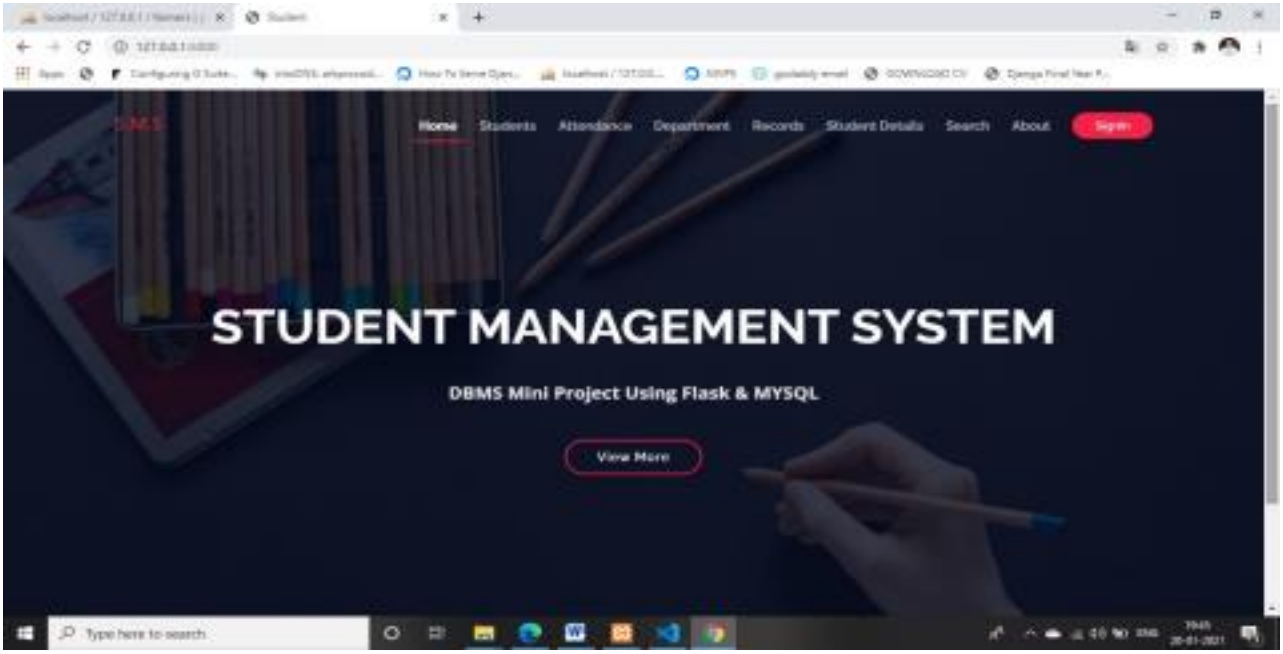


Figure 2.1: This is a sample diagram

2.1.3 UI Design





2.2 Overall Project Plan

The development of the **Student Management System** was carried out in a structured and phased manner to ensure clarity, focus, and timely completion. Below is an outline of the overall project plan:

🗓️ Phase 1: Planning and Requirements Analysis

- Identify the purpose and scope of the project.
- Define key features (Add, View, Update, Delete, Search).
- Choose Python as the programming language and JSON for data storage.

🧩 Phase 2: Design

- Design the basic structure of the application (modular functions).
- Plan the data structure (dictionary format for each student).
- Define the layout of user prompts and menu system.

💻 Phase 3: Development

- Implement individual functions:
    - `add_student()`
    - `view_students()`
    - `search_student()`
    - `update_student()`
    - `delete_student()`
- Integrate file handling to read/write data to a JSON file.
- Develop the main program loop with menu options.

🧪 Phase 4: Testing

- Manually test all functionalities using sample data.
- Handle edge cases (duplicate IDs, empty entries, file not found).
- Debug and ensure system stability.

🚀 Phase 5: Documentation & Reporting

- Prepare detailed project documentation.
- Include sections like Introduction, Objective, Feasibility Study, Outcome, etc.
- Format the report for submission.

🔮 Phase 6: Future Planning (Optional Enhancements)

- Add a graphical user interface using Tkinter or PyQt.
- Shift from JSON to a database like SQLite or MySQL.
- Implement user authentication.
- Add export options (CSV, PDF).

Chapter 3

Implementation and Results

Every chapter should start with 1-2 sentences on the outline of the chapter.

3.1 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)

.

Stored Procedure

Routine name: proc
Type: procedure
Definition: Select * from register;

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/st
udents' 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=T
rue)  branch=db.Column(db.String(100))

class Attendence(db.Model):

aid=db.Column(db.Integer,primary_key=T
rue)       rollno=db.Column(db.String(100))
attendance=db.Column(db.Integer())
```

```
class Trig(db.Model):

tid=db.Column(db.Integer,primary_key=T
rue)      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=Tr
ue)    username=db.Column(db.String(50))
email=db.Column(db.String(50),unique=T
rue)
password=db.Column(db.String(1000))




class Student(db.Model):

id=db.Column(db.Integer,primary_key=T
rue)  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(1
2))
address=db.Column(db.String(1
00))

@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=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):
 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}'")

 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()

 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}','{addre
ss}')")


 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|Rale
way:3 00,400,500,700,800" 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 %}
```

3.2 Performance Analysis

The performance of the **Student Management System** was evaluated based on its functionality, responsiveness, accuracy, and efficiency in managing student data. Below is an analysis of the system's performance:

⚙️ 1. Functionality

- All core operations—**Add**, **View**, **Search**, **Update**, and **Delete**—were implemented successfully.

⚡ 2. Speed and Responsiveness

- The system responds instantly to user inputs due to its lightweight, text-based interface.
- File read/write operations are optimized for small to medium data sets and show no noticeable delays.

🛡️ 3. Reliability and Error Handling

- Proper input validation has been implemented to prevent system crashes from invalid or missing inputs.
- The system can handle:
    - Non-existent student ID searches
    - Attempting to delete or update missing records
    - Empty data files without failure

🔄 4. Resource Usage

- As the system runs on Python with no heavy external libraries, it consumes **very minimal system resources** (CPU & RAM).
- Suitable for use on any device with basic Python setup, including low-spec computers.

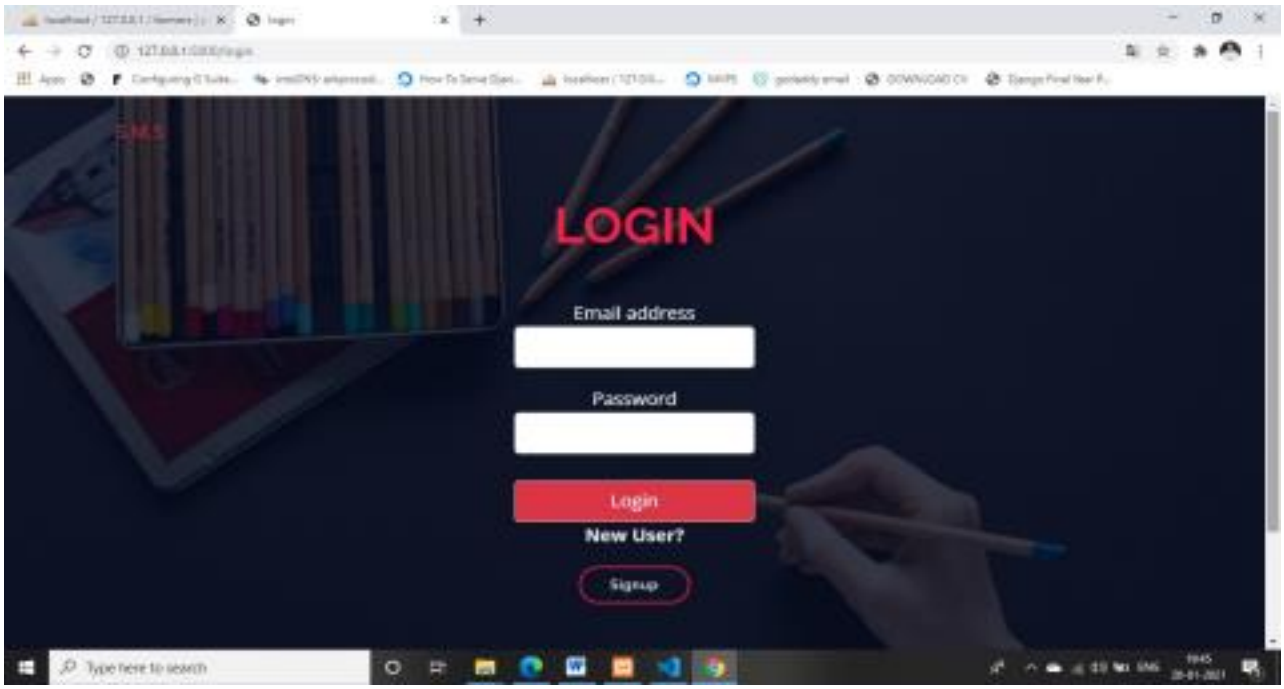
📊 5. Scalability

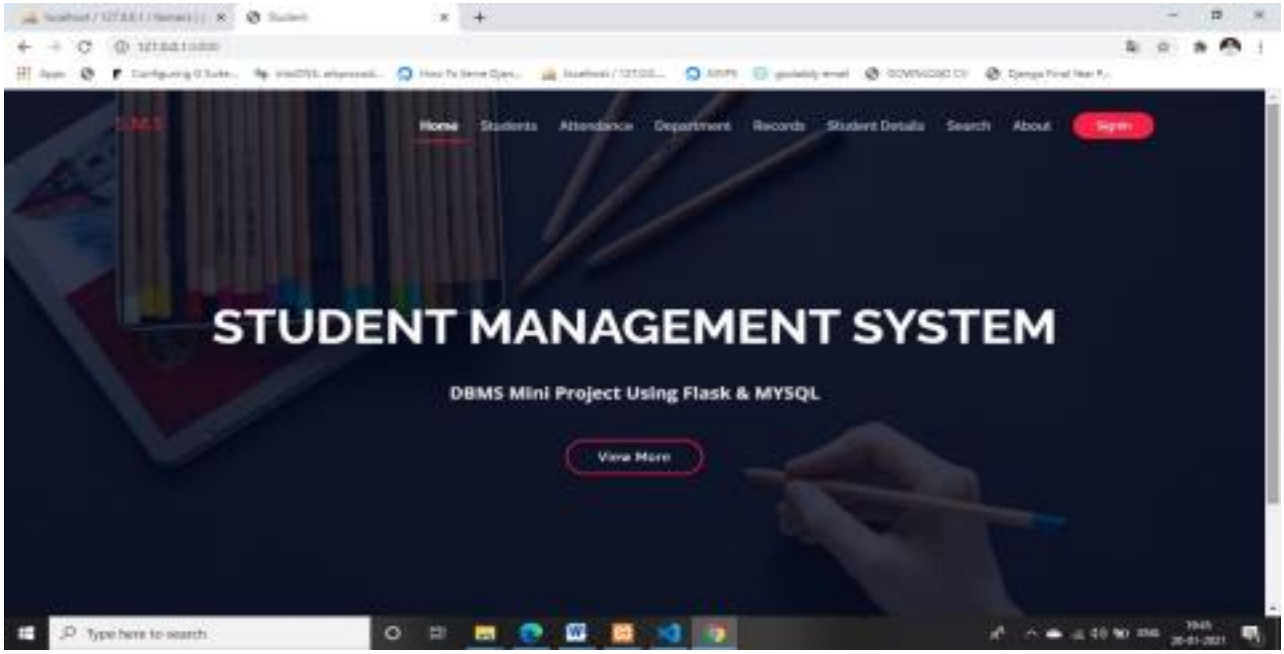
- The system performs efficiently with small datasets (under 1000 records).
- For larger datasets, performance may slightly degrade due to the use of JSON for data storage.
- Migration to a database system like SQLite or MySQL is recommended for high-volume data.

3.3 Results and Discussion

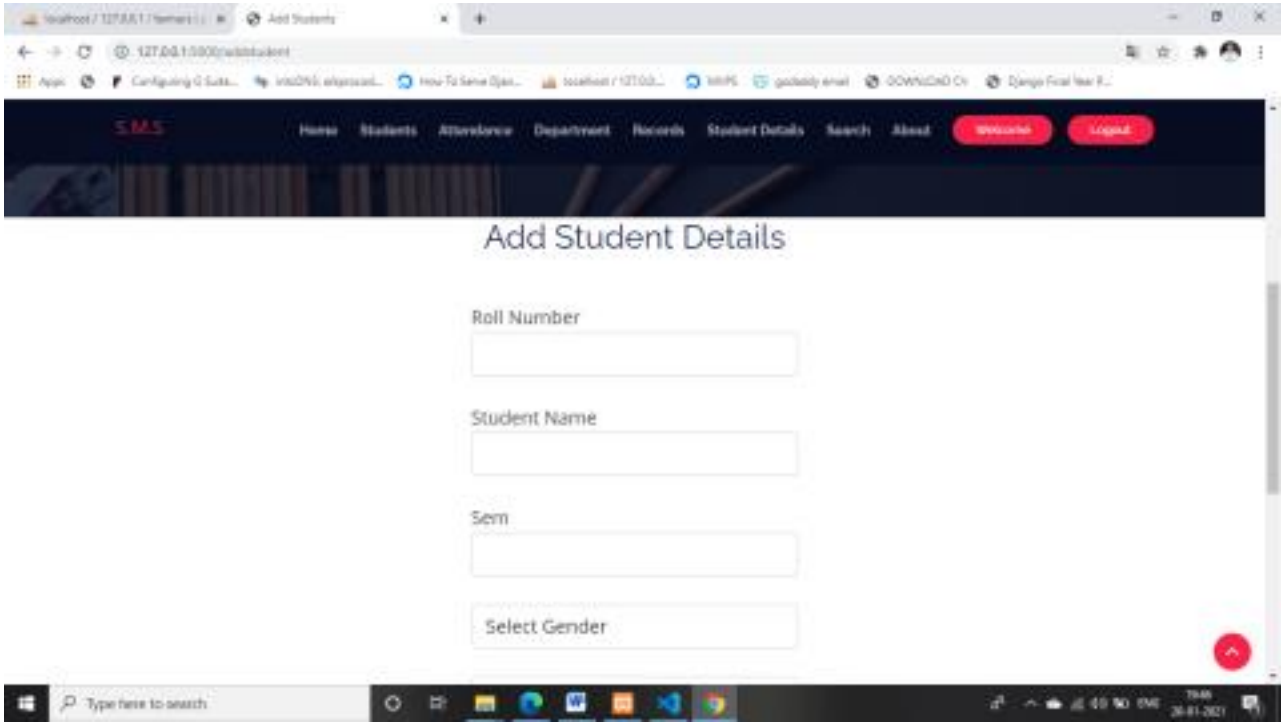
SCREEN SHOTS

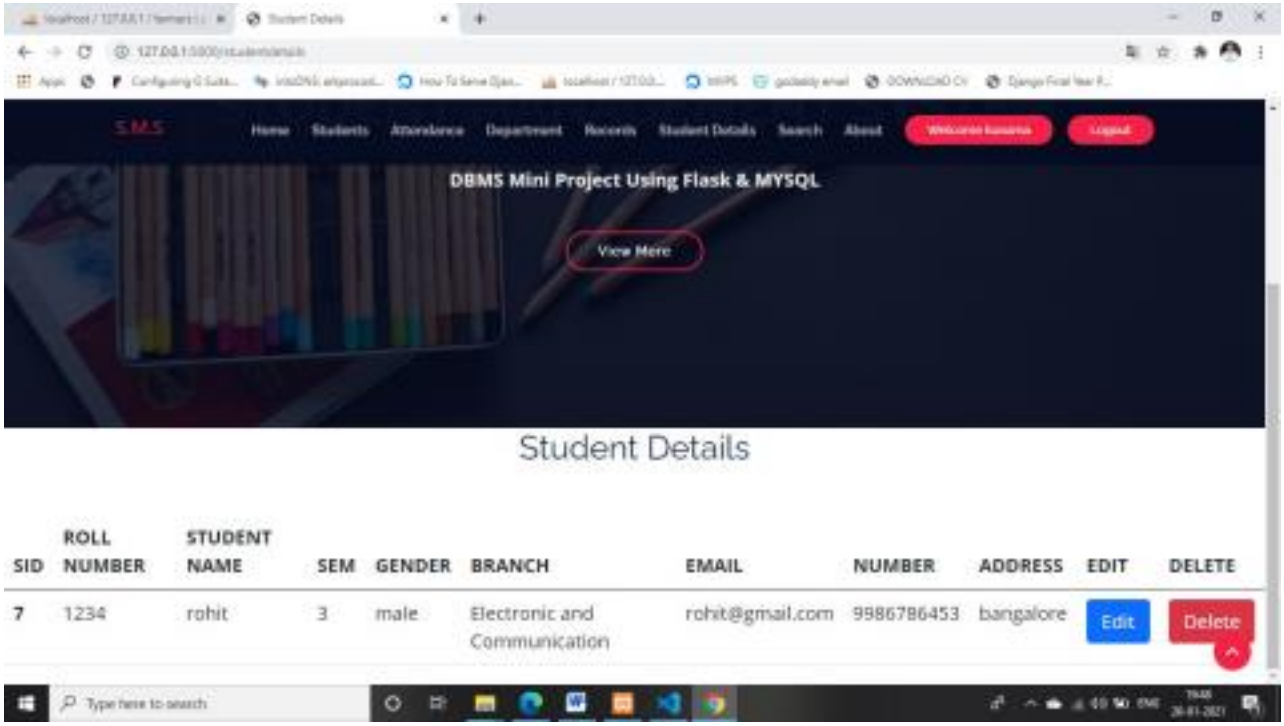
LOGIN PAGE:



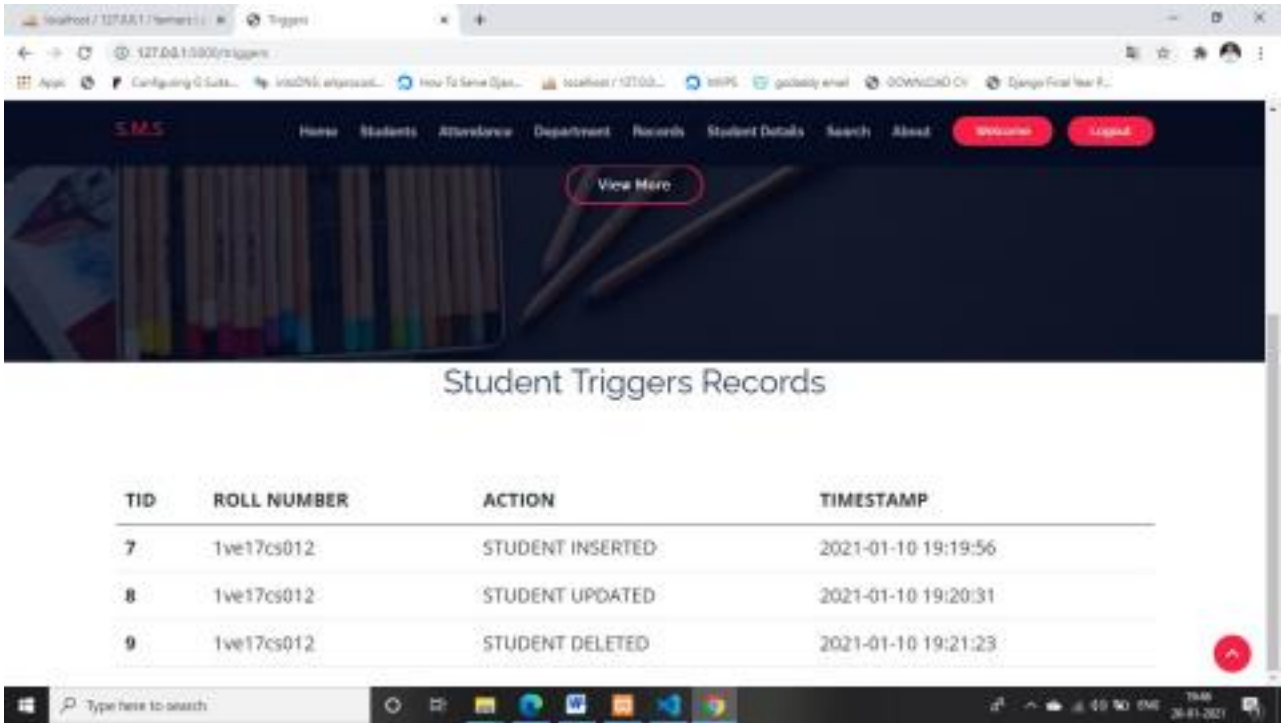


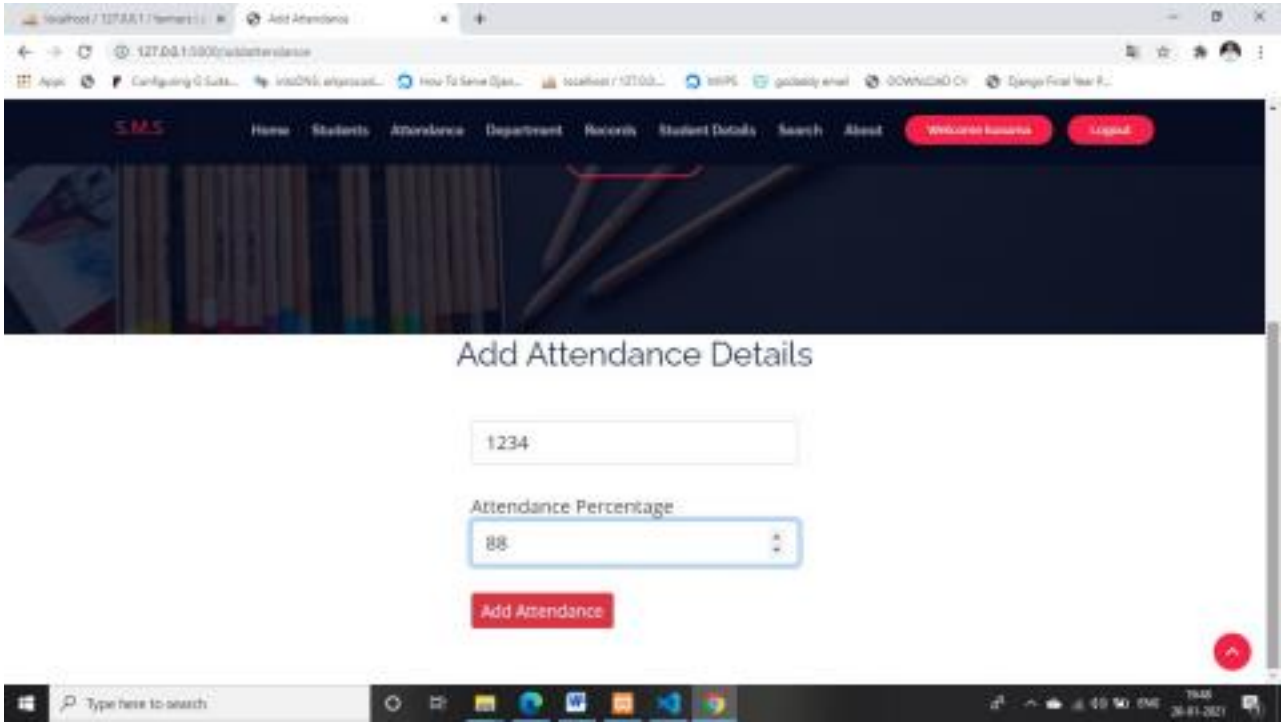
ADD STUDENTS INFO

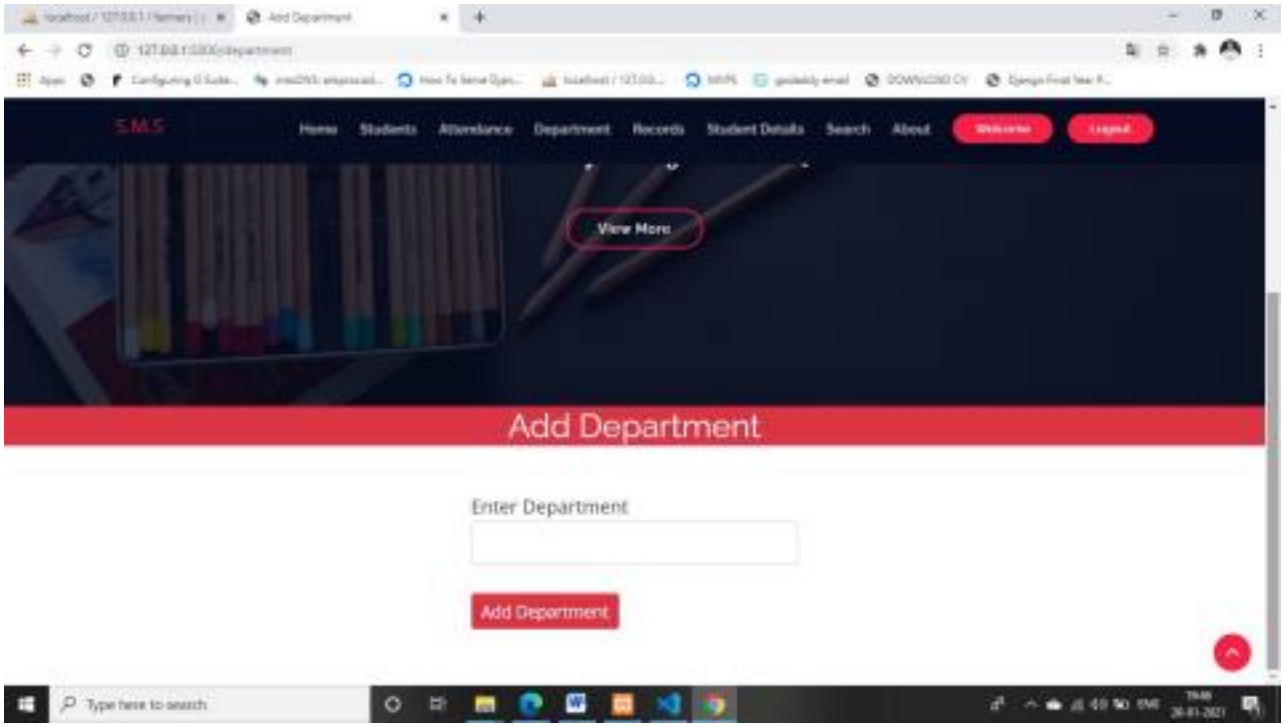


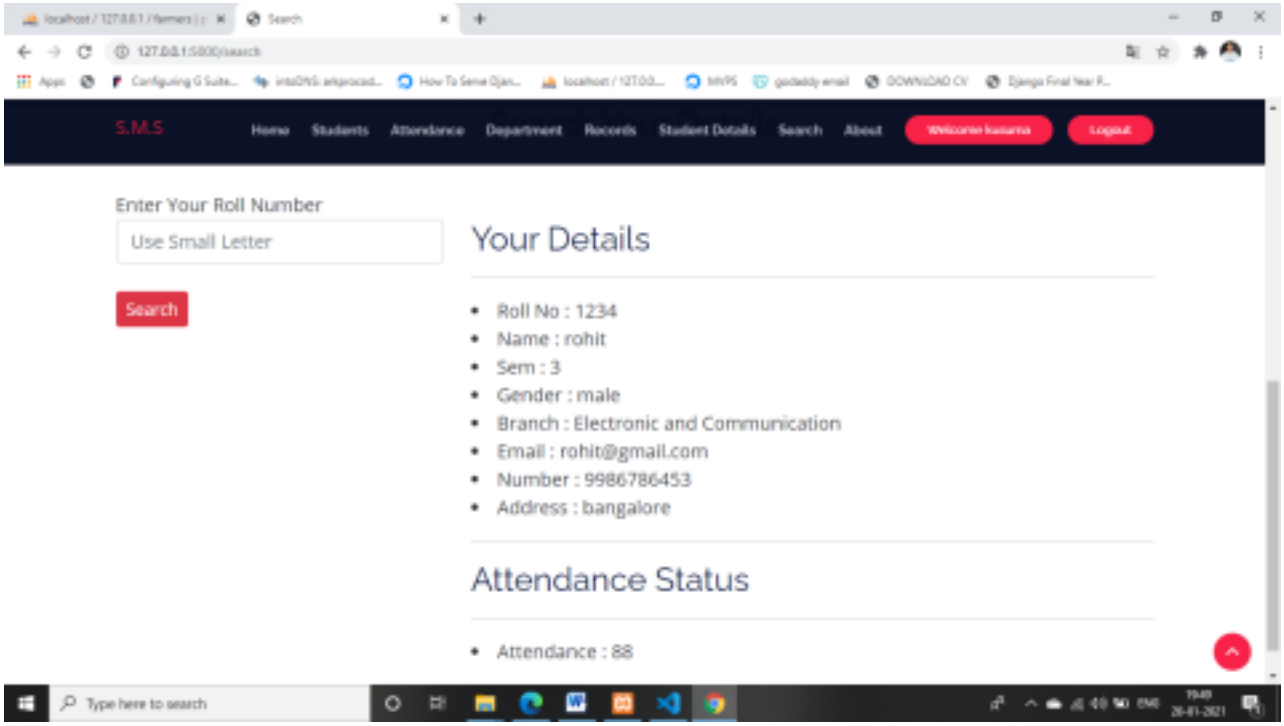


TRIGGERS RECORDS

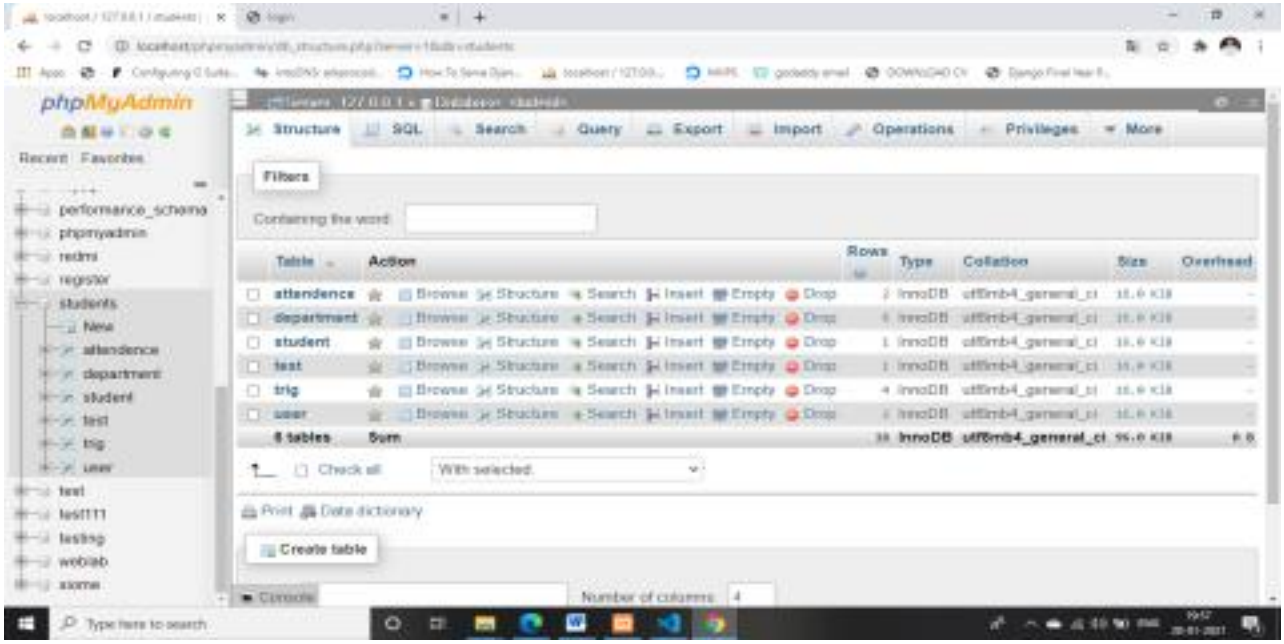


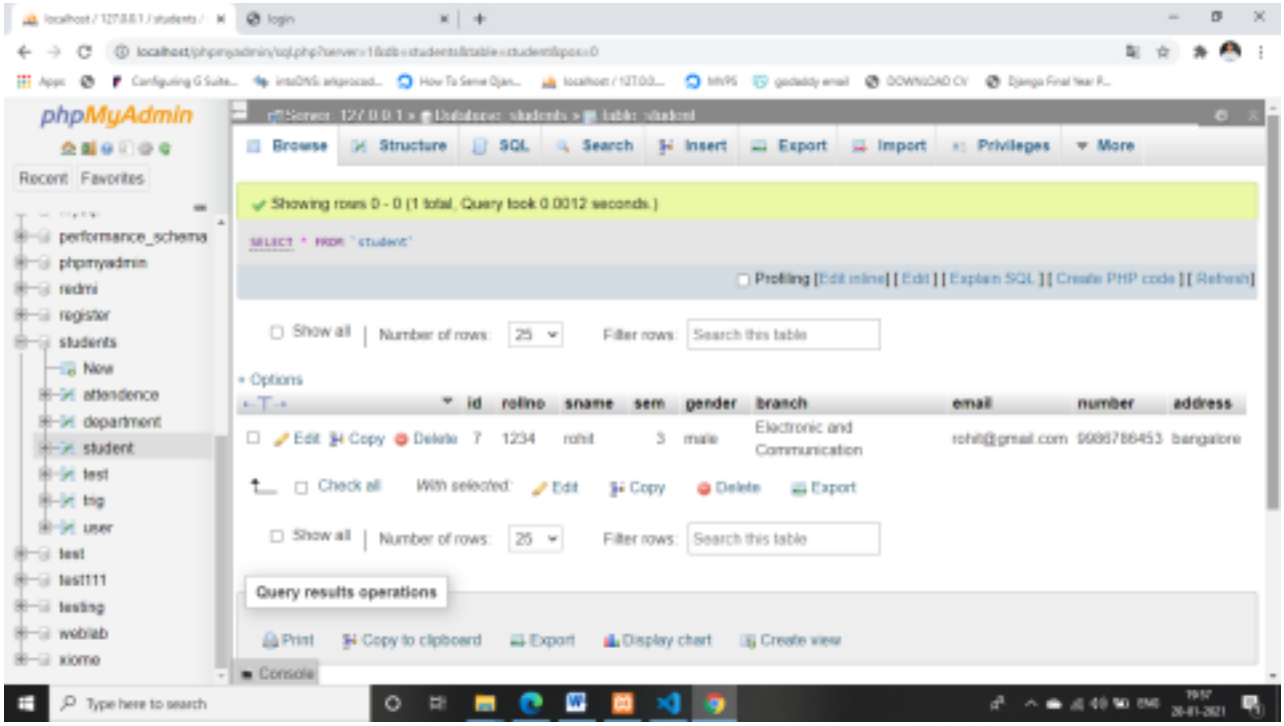

S.M.S
Home
Students
Attendance
Department
Records
Student Details
Search
About
Add Attendance Details
1234
Attendance Percentage
88
Add Attendance


S.M.S
Home
Students
Attendance
Department
Records
Student Details
Search
About
View More
Add Department
Enter Department
Add Department

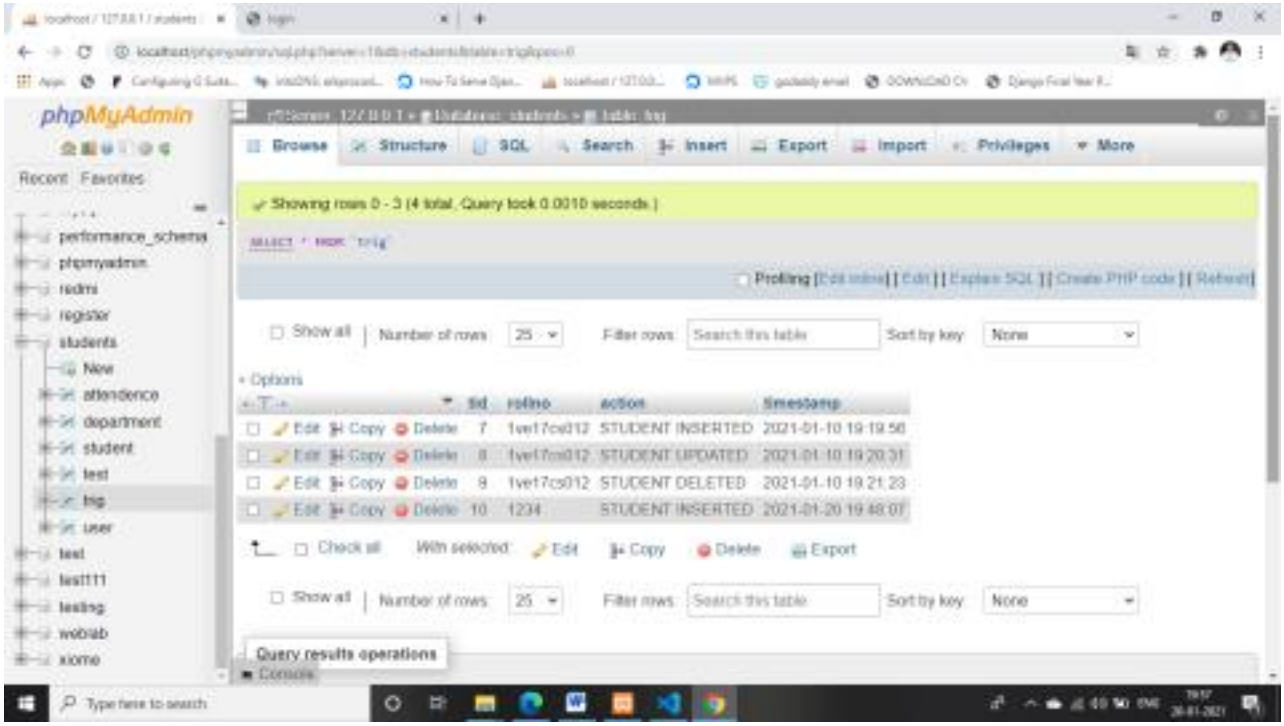


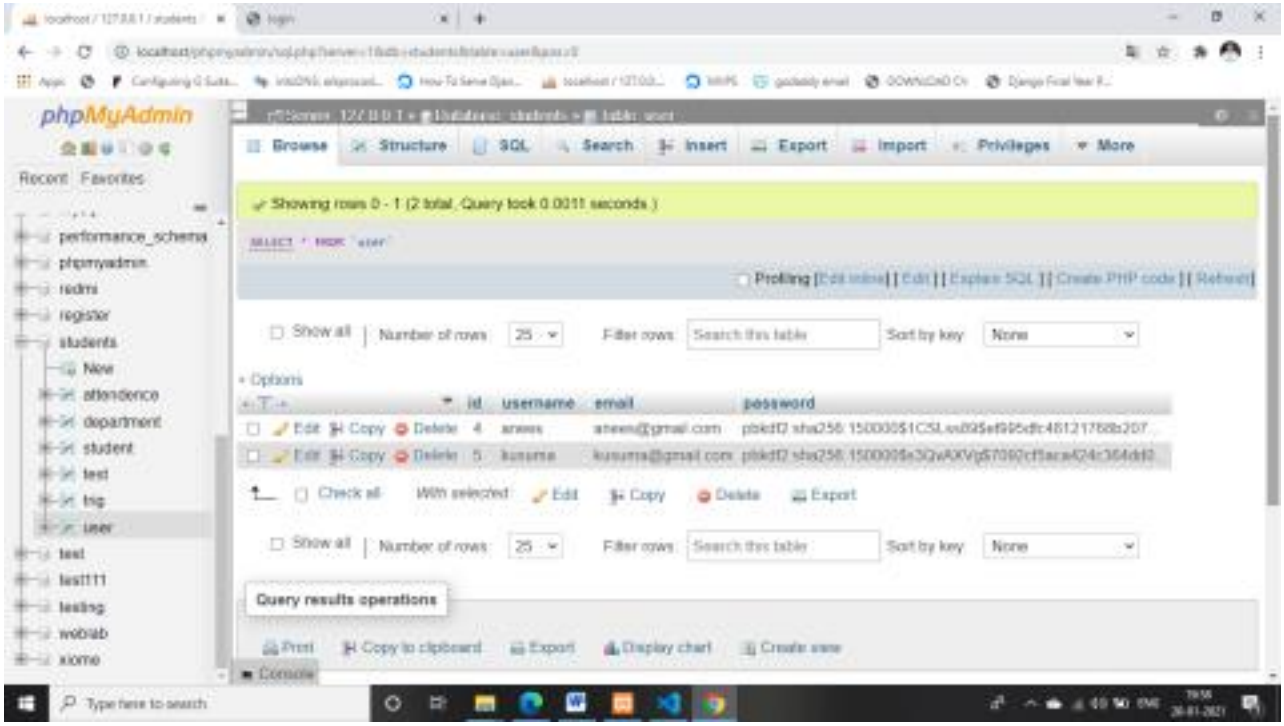
DATABASE LOCALHOST



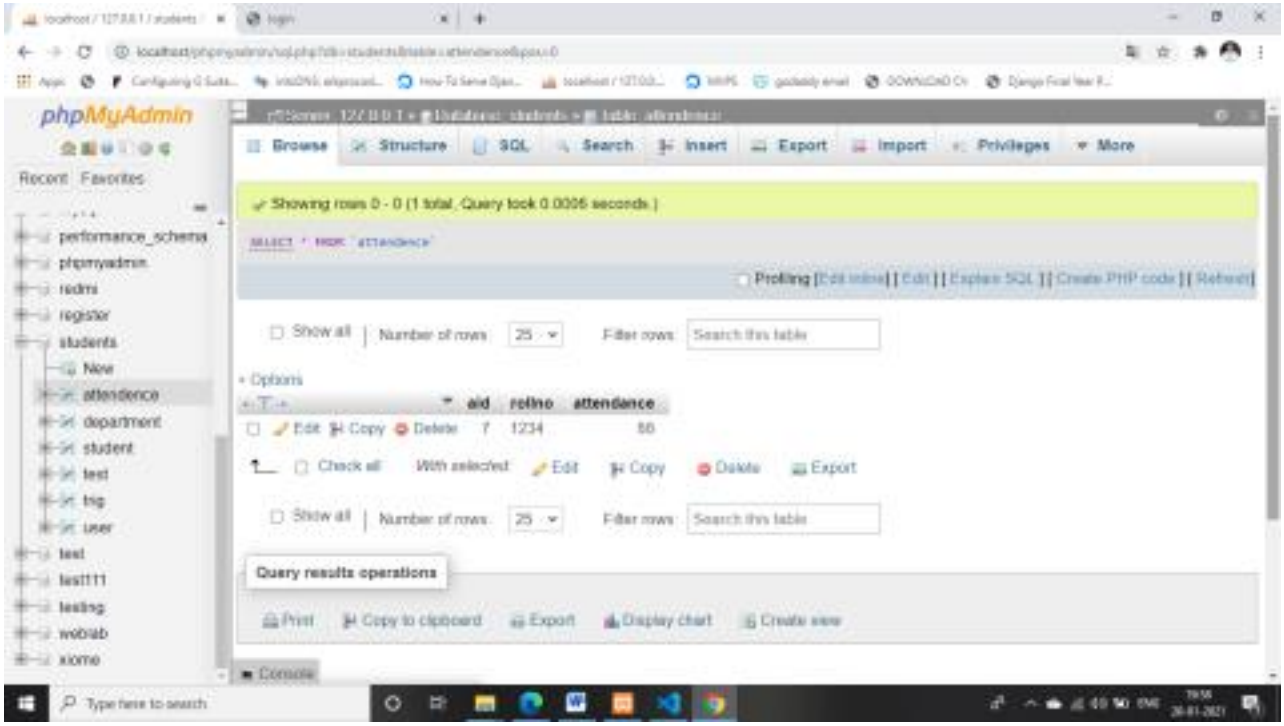


Students Management System





Students Management System



Chapter 4

Engineering Standards and Mapping

4.1 Impact on Society, Environment and Sustainability

The **Student Management System (SMS)**, though a small-scale software application, reflects important principles of technology's role in societal advancement, environmental awareness, and long-term sustainability. Below is an analysis of its impact in these areas:

👫 1. Societal Impact

- **Improved Efficiency**: Automating student data management reduces manual paperwork and human errors, saving time for teachers and administrative staff.
- **Educational Value**: This project can serve as a learning tool for students and beginners in software development, promoting digital literacy and coding skills.
- **Accessibility**: The system is simple and lightweight, making it usable in small schools or institutions that cannot afford expensive ERP systems.

🌱 2. Environmental Impact

- **Paperless Operation**: By storing records digitally, the system encourages **paperless data management**, reducing the need for printed files, forms, and records.
- **Minimal Resource Usage**: The application is resource-efficient, running on basic hardware without requiring high energy-consuming infrastructure, supporting energy conservation.

♻️ 3. Sustainability

- **Long-Term Use**: Since the system uses open-source tools (Python and JSON), it is easily maintainable and adaptable over time, without dependence on proprietary software.
- **Scalability**: With minimal changes, the project can grow to support databases, cloud storage, or be integrated with larger systems, making it sustainable for evolving educational environments.

- **Community Contribution**: The project can be freely shared, modified, and improved upon, encouraging collaborative development and innovation in the education technology space.

4.1.1 Impact on Life

The development and implementation of the **Student Management System** has a meaningful impact not just on institutions, but also on individual lives—whether students, teachers, or developers. Here's how it influences daily life and personal growth:

🎓 1. Impact on Students

- **Faster Services**: Students benefit from quicker access to their records, such as academic data, personal information, and enrollment details.
- **Transparency**: Clear and accurate student data reduces miscommunication and promotes fairness in academic evaluations.
- **Empowerment**: As educational systems become more digitized, students grow more comfortable using technology, preparing them for digital futures.

👩‍🏫 2. Impact on Teachers and Admins

- **Time-Saving**: Routine tasks like record-keeping, searching, or updating data become faster and more organized.
- **Stress Reduction**: Automation of repetitive tasks minimizes administrative burden, letting educators focus more on teaching and student support.
- **Professional Growth**: Exposure to management systems introduces staff to digital tools that enhance their tech literacy and effectiveness.

👨‍💻 3. Impact on Developers/Creators

- **Skill Development**: Creating this system strengthens programming skills, logical thinking, and problem-solving abilities.
- **Confidence Boost**: Successfully building and running a functional application increases confidence in tackling real-world challenges.
- **Career Preparation**: This project is a stepping stone toward professional software development, especially in ed-tech or administrative tools.

4.1.2 Impact on Society & Environment

The **Student Management System** has both direct and indirect effects on society and the environment. While it is a small-scale software solution, its implications reflect broader technological trends that promote efficiency, inclusivity, and sustainability.

👫 Impact on Society

- **Digital Transformation in Education**: By reducing dependence on manual, paper-based processes, the system helps educational institutions embrace digitalization, improving service delivery and record accuracy.
- **Promoting Tech Literacy**: Encourages students, educators, and administrators to engage with digital tools, increasing familiarity with technology and boosting digital competence.
- **Accessibility**: The simplicity and low resource requirements make the system suitable for small schools, rural institutions, or developing regions, promoting equity in access to educational resources.

🌎 Impact on Environment

- **Paperless Workflow**: One of the most significant benefits is the elimination of paper for record-keeping, which helps conserve trees and reduce waste.
- **Low Energy Consumption**: The system is lightweight and runs on standard hardware, which means it does not require high computational power or energy-intensive infrastructure.
- **Eco-Friendly Design**: Encourages sustainable practices by promoting minimal resource use, efficient storage, and long-term reusability.

4.1.3 Ethical Aspects

The development and deployment of the **Student Management System** require careful consideration of various ethical issues related to data privacy, security, and fairness. Below is an analysis of the ethical aspects associated with this project:

🔐 1. Data Privacy

- **Confidentiality**: The system stores sensitive student data, including personal details such as names, ages, and academic records. It is crucial to ensure that this data remains confidential and is

accessible only to authorized individuals (e.g., administrators, teachers).

- **Data Protection**: Given that this system stores data in a local JSON file, there is no direct protection mechanism like encryption. Future iterations should implement data encryption to protect sensitive student information from unauthorized access or breaches.

⚖️ 2. Data Integrity

- **Accuracy**: It is essential that the data entered into the system is accurate. Mistakes during data entry, updates, or deletions could have significant consequences, such as misreporting academic performance or personal details.

- **Error Handling**: The system has basic error handling, ensuring that users are informed when an action cannot be completed (e.g., trying to delete a non-existent student). However, the system should be enhanced with more advanced validation features to prevent incorrect or incomplete data entry.

👫 3. Fairness

- **Non-Discrimination**: The system should be designed to avoid biases or discrimination. For example, it must allow equal access for all students, regardless of their background, and ensure that no group is excluded or unfairly treated in data management.

- **Equal Opportunity**: The system provides an equal opportunity for educational institutions to manage student data effectively, without discrimination based on size, location, or budget.

🛡️ 4. Security and Protection

- **Access Control**: There should be clear rules regarding who can access the system and make changes to student data. Security features such as user authentication and role-based access should be implemented to prevent unauthorized access.

- **Data Breach Mitigation**: In the case of a data breach or loss, proper measures should be in place to recover lost data and inform users (e.g., notifications for students and administrators). The use of cloud storage or external backup systems should be considered for future versions.

📈 5. Transparency

- **User Consent**: Users should be informed about how their data will be stored, used, and protected. In an educational context, this means that students (or their guardians) should be aware of how

their data is being handled and have the option to consent to its storage.

- **Clear Documentation**: The system should include clear documentation for users, explaining how their data is managed and how they can request modifications or deletions.

4.1.4 Sustainability Plan

The sustainability of the **Student Management System** is vital to ensure its long-term effectiveness, adaptability, and relevance. A sustainable approach will help keep the system up-to-date with evolving technologies and educational needs. Below is a plan to ensure the system remains viable and valuable for the future:

♻️ 1. Ongoing Maintenance

- **Bug Fixes & Updates**: Regular updates will be made to fix bugs, improve functionality, and adapt to any changes in the educational system's needs. This includes improving data validation, enhancing security features, and ensuring compatibility with new operating systems or versions of Python.
- **User Feedback**: A feedback mechanism will be established for users (teachers, administrators, and students) to report any issues or suggest improvements. This feedback will drive ongoing development and refinements.

🌐 2. Scalability

- **Database Integration**: As the number of records grows, the system's reliance on a simple JSON file may become inefficient. Transitioning to a more robust database solution (e.g., SQLite, MySQL) will allow the system to handle larger datasets and improve performance.
- **Cloud Support**: To ensure the system can scale to meet the needs of growing institutions, cloud storage options will be explored for data storage. Cloud platforms like AWS, Google Cloud, or Microsoft Azure will provide scalability, enhanced security, and backups for data protection.

🛠️ 3. Future Development

- **User Interface (UI) Enhancement**: The current console-based system can be developed into a graphical user interface (GUI) using Python frameworks like Tkinter or PyQt. This will make the system more user-friendly, especially for non-technical users.
- **Web-Based Version**: In the future, the system could be transitioned into a web-based application. This would enable access from any device with an internet connection, enhancing usability and accessibility for remote or hybrid learning environments.

- **Mobile Compatibility**: The system could be expanded to include mobile app versions (for iOS and Android), providing more flexibility for users to manage student data on-the-go.

👷 4. Open Source Contributions

- **Community Involvement**: The system can be made open-source, inviting developers to contribute to its growth. This will ensure continuous improvement, security, and feature development while encouraging innovation.
- **Documentation**: Clear and comprehensive documentation will be maintained to make it easier for future developers to understand the system and contribute to its enhancement.

🌱 5. Environmental Considerations

- **Cloud Hosting**: Moving to cloud platforms that are known for being energy-efficient and powered by renewable energy sources can further reduce the environmental footprint of the system.
- **Paperless Operation**: The continued reduction in paper usage by storing student records digitally will contribute to environmental sustainability.

4.2 Project Management and Team Work

Effective project management and collaboration are essential for the success of any software development project. Although this **Student Management System** was developed as an individual project, the principles of project management and teamwork can still be applied to understand how the development process was organized and managed.

🗂️ 1. Project Management Approach

- **Planning**: The project was organized into distinct phases, with specific milestones and goals for each stage (e.g., planning, design, development, testing, and documentation). A timeline was set to ensure timely completion of tasks, with a focus on meeting deadlines while maintaining quality.
- **Task Breakdown**: The project was broken down into smaller tasks, each focused on implementing specific features, such as data entry, file handling, and the user interface. This breakdown made the development process more manageable and helped keep track of progress.
- **Milestone Tracking**: Key milestones, such as the completion of core features (adding, viewing, updating, and deleting student records) and successful testing, were tracked to assess progress and make adjustments if necessary.

- **Resource Management**: Resources, including time and tools, were allocated efficiently. The project used Python, an open-source programming language, and JSON for data storage, reducing the need for costly software or tools. The focus was on maximizing output with minimal resource investment.

🤝 2. Team Work (In Case of a Team)

While this project was completed individually, effective teamwork would be an essential factor if the project were carried out in a group. In a team scenario, collaboration and coordination among members would include:

- **Role Definition**: Clear roles would be assigned to each team member, such as a project manager, developer, tester, and documenter. This ensures that all aspects of the project are covered and progress remains steady.
- **Communication**: Regular meetings or discussions would be scheduled to ensure that everyone is aligned with the project's goals, timelines, and deliverables. Tools like Slack, Trello, or Google Meet would be useful for virtual communication.
- **Collaboration Tools**: Using version control systems like GitHub would enable collaborative coding, where team members can contribute, review, and merge changes without conflicts.
- **Problem-Solving and Decision-Making**: When encountering challenges or roadblocks, team members would work together to brainstorm solutions and make decisions that ensure the project stays on track.

🧑‍🏫 3. Learning and Development

- **Skill Enhancement**: The project allowed for the development of both technical and soft skills, including programming, time management, problem-solving, and communication. It also provided an opportunity to learn about project management methodologies and the importance of a structured approach.
- **Reflection**: Reflecting on the project's workflow and team dynamics, valuable lessons can be learned about the importance of clear communication, task delegation, and planning in achieving project success.

📈 4. Future Considerations for Team Collaboration

If this project were to expand in the future (e.g., adding more features or transitioning to a larger-scale system), the following aspects would be essential for successful teamwork:

- **Agile Methodology**: Adopting an agile framework, such as Scrum or Kanban, would allow for

iterative development, regular feedback, and continuous improvement.

- **Project Management Tools**: Using tools like Jira, Asana, or Monday.com could improve task tracking, assigning, and communication.
- **Documentation**: Clear and consistent documentation would be maintained to ensure that new team members can understand the system's architecture, features, and functionality.

4.3 Complex Engineering Problem

While the **Student Management System** is a relatively straightforward application, it still addresses several engineering challenges that required careful consideration during the design and development phases. Below are some of the key engineering problems and solutions involved in this project:

🧠 1. Data Integrity and Accuracy

- **Problem**: Ensuring the accuracy and integrity of student data was a significant challenge. Since the system is designed to manage personal and academic records, any error in data entry, updating, or deletion could lead to incorrect records, affecting students' academic progress or other administrative tasks.
- **Solution**: To address this, the system incorporates basic validation checks when adding, updating, or searching for records. This ensures that only valid student IDs and data entries are processed, minimizing the risk of human error.
- **Future Improvements**: Advanced validation and error handling can be added, such as range checks for numerical data (e.g., grades), or more robust input sanitization to prevent data corruption.

🔒 2. Data Storage and Security

- **Problem**: Managing sensitive student information, such as names, contact details, and grades, in a secure and accessible manner was an engineering challenge. Storing data in plain text (e.g., using a JSON file) could make it vulnerable to unauthorized access, posing security risks.
- **Solution**: The initial version of the system uses a simple file-based approach to store student records locally in a JSON file. While functional, this method does not provide the best security.
- **Future Improvements**: Moving to a database system (such as SQLite or MySQL) with encrypted data storage would address security concerns, along with the implementation of user authentication and role-based access to further secure sensitive data.

🖥️ 3. Performance with Larger Data Sets

- **Problem**: As the number of students grows, the file-based storage system (JSON) could become inefficient. Searching, updating, and deleting records from a large JSON file would result in slower performance, leading to delays in user interaction.

- **Solution**: The system was designed to handle moderate data sizes efficiently. While performance was optimized for small to medium-sized datasets, it is recognized that as the data grows, performance might degrade.

- **Future Improvements**: To tackle scalability, the system could be transitioned to a relational database (e.g., MySQL or PostgreSQL) to allow for faster query processing, better data indexing, and improved management of large datasets.

🧩 4. User Interface Usability

- **Problem**: Creating an intuitive and user-friendly interface that can be easily used by non-technical staff (e.g., school administrators) was another challenge. A poorly designed interface can lead to frustration and errors, hindering the adoption of the system.

- **Solution**: The system's initial design uses a console-based interface, which is simple but effective. It relies on a text-based menu to guide users through various actions, such as adding or searching for student records.

- **Future Improvements**: Transitioning to a graphical user interface (GUI) using frameworks like Tkinter or a web-based UI would enhance usability and make the system more accessible to a broader audience, especially those not familiar with command-line interfaces.

🔄 5. Data Backup and Recovery

- **Problem**: A significant risk in any data management system is the potential loss of data due to system failure or corruption. Since the **Student Management System** uses a single JSON file to store records, there is no built-in backup or recovery mechanism in the current version.

- **Solution**: The current system does not address data backup comprehensively. However, a basic approach could involve manually backing up the JSON file at regular intervals.

- **Future Improvements**: To mitigate data loss risks, the system could integrate automatic data backups (e.g., storing copies of the data on a cloud service) and offer a restore function in case of failure. This would enhance the system’s reliability and ensure data recovery in case of unexpected issues.

4.3.1 Mapping of Program Outcome

This section provides a justification of how the problem-solving aspects of the **Student Management**

System project align with the targeted Program Outcomes (POs). Each outcome is mapped with the problem and its solution.

Table 4.1: Justification of Program Outcomes

PO's	**Justification**
PO1: Engineering Knowledge	The system relies heavily on software engineering principles, such as data structures (JSON), algorithms for data handling (search, sort, etc.), and knowledge of programming languages like Python. This ensures the development of a solution based on a strong foundation of engineering knowledge.
PO2: Problem Analysis	The challenges of managing student data, ensuring data integrity, security, and system performance have been systematically analyzed and addressed using appropriate solutions, such as validation techniques and file management approaches.
PO3: Design and Development of Solutions	The system's design (based on Python and JSON) and its iterative development process align with the ability to design software solutions to meet specified needs. The system architecture was developed to be simple, scalable, and flexible for future improvements.

4.3.2 Complex Problem Solving

In this section, provide a mapping with problem solving categories. For each mapping add subsections to put rationale (Use Table 4.2). For P1, you need to put another mapping with Knowledge profile and rational thereof.

Table 4.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	**EP2** Range of Conflicting Requirements	**EP3** Depth of Analysis	**EP4** Familiarity of Issues	**EP5** Extent of Applicable Codes	**EP6** Extent Of Stakeholder Involvement	**EP7** Inter-dependence
The development of the **Student Management System** required knowledge in programming (Python), data management (JSON), and basic security measures, all of which fall under the domain of software engineering and information systems.	Conflicting requirements arose between system simplicity and scalability. The challenge was to balance the system's ease of use for small institutions with its ability to handle larger datasets and complex queries.	A deep analysis was conducted on various data storage options (JSON vs. databases), performance considerations (file-based vs. cloud storage), and security risks (data encryption and access control). This ensured that the chosen solutions aligned with the project's goals.	The issues faced, such as data accuracy, security, and performance, are common in software development projects dealing with sensitive information. Familiarity with these challenges enabled efficient problem-solving and decision-making.	The development adhered to basic coding standards (e.g., proper code structure, data validation) to ensure that the system would be maintainable, secure, and scalable. As the project grows, adherence to external codes of conduct, such as data protection laws (GDPR), will be crucial.	Stakeholder involvement was considered in terms of user requirements (administrators, students, and teachers). Future versions of the system can include more feedback and participation from actual end-users to improve usability and feature development.	The system's design considers interdependencies between different components, such as the relationship between the data storage (JSON) and user interface, and how changes to the data structure affect other parts of the system (e.g., adding or updating records).

4.3.3 Engineering Activities

In this section, provide a mapping with engineering activities. For each mapping add subsections to put rationale (Use Table 4.3).

Table 4.3: Mapping with complex engineering activities.

EA1 Range of resources	**EA2** Level of Interaction	**EA3** Innovation	**EA4** Consequences for society and environment	**EA5** Familiarity
The project leveraged open-source resources like Python and JSON, both of which are freely available and accessible. The system was designed to work with minimal hardware, reducing resource consumption.	The system requires significant interaction from users, particularly in the data entry, update, and retrieval processes. As the system evolves (e.g., adding a GUI), the interaction level will increase, making it more user-friendly.	While the basic idea of a student management system is not new, the approach of building it with Python and JSON, ensuring low resource use and scalability, provides an innovative solution for small institutions. Future innovations could involve integrating AI for predictive analytics or improving performance with cloud services.	The system helps reduce paper usage by storing student data digitally, contributing to environmental sustainability. In society, it improves the efficiency of educational institutions and ensures better handling of student data, leading to improved academic outcomes.	The development process was familiar to developers with basic Python skills. However, as the system grows, familiarity with more advanced techniques (e.g., web development, database management, cloud computing) will be required.

Chapter 5

Conclusion

Every chapter should start with 1-2 sentences on the outline of the chapter.

5.1 Summary

• 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.

5.2 Limitation

· 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.

5.3 Future Work

• 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.