



**PROGRAM: MASTER OF SCIENCE IN
COMPUTER SCIENCE AND INFORMATION
TECHNOLOGY**

[M.Sc. - CS & IT]

Mini Project

Student Management System

Semester - 2

Submitted To:

Dr M N Nachappa

Prof. Haripriya V.

Prof. Raghavendra R.

Submitted By:

Ashish Khadela (24MSRCI012)

Kaushal Muniwala (24MSRCI013)

Yash Mandaliya (24MSRCI028)

**JAIN**
DEEMED-TO-BE UNIVERSITYSCHOOL OF
COMPUTER
SCIENCE AND IT**Department of Computer Science & Information Technology****Programme: Master of Science in Computer Science & Information Technology****[MSc-CS&IT]***Certificate*

This is to certify that **Mr. Ashish Khadela, Mr. Kaushal Muniwala and Mr. Yash Mandaliya** has satisfactorily completed the course of **Activity – 2** prescribed by the JAIN(Deemed-to-be-University) for the **semester 2 M.Sc. – CS & IT** degree course in the year **2024 - 2026**.

Date: 01/04/2025

v. good
11/04/25

1. Ashish
2. Yash
3. Kaushal

Signature of Student

1. _____
2. 11/04/25
3. 11/04/25

Signature of Faculty In charge

11/04/25
Head of the Department

Student Management System

- **Title Page**

Project Title: Student Management System

Project Type: Mini Project

Course: Advance Database Management System
Human-Computer Interaction
Python Programming

Submitted By: Ashish Khadela (24MSRCI012)
Kaushal Muniwala (24MSRCI013)
Yash Mandaliya (24MSRCI028)

Date Of Submission: 01-04-2025

• **Table Of Contents**

- ❖ Abstract
- ❖ Introduction
- ❖ Objectives
- ❖ Technologies Used
- ❖ System Design
 - **Block Diagram**
 - **Database Design**
 - **User Interface Design**
- ❖ Implementation
- ❖ Testing And Results
- ❖ Conclusion
- ❖ Future Enhancements
- ❖ References

• Abstract

The Student Management System (SMS) is a desktop-based application developed using Python (Tkinter) for the frontend and MySQL for the backend. The system provides functionalities such as student registration, course management, attendance tracking, result management, and user authentication (login, register, forgot password).

The project follows CRUD (Create, Read, Update, Delete) operations for database interactions and ensures a user-friendly interface with Tkinter GUI. The system helps educational institutions automate student data management efficiently.

- **Dashboard** for quick navigation
- **Manage Course** for adding, updating, and deleting courses
- **Manage Students** for student record management
- **Manage Results** for storing and retrieving student grades
- **View Results** for student performance analysis
- **Manage Attendance** for tracking student participation
- **Authentication System** with Login, Register, and Forgot Password

Key Technologies Used:

- **Programming Language:** Python
- **Integrated Development Environment (IDE):** VS Code
- **Database Management System (DBMS):** MySQL (using MySQL Workbench)
- **Human-Computer Interaction (HCI) Framework:** Tkinter (for GUI development)

• Introduction

Background

In educational institutions, managing student records, course enrollment, attendance, and academic performance is a critical yet time-consuming task. Traditionally, these processes are handled manually using paper-based records or spreadsheets, which often lead to inefficiencies, data inconsistency, and security issues.

To overcome these challenges, the **Student Management System (SMS)** has been developed as a **desktop-based application** using **Python (Tkinter) for the frontend** and **MySQL for the backend**. This system provides a structured approach to handling student-related operations, ensuring that data is securely stored, easily accessible, and efficiently managed.

The system supports essential functionalities such as **student registration, course management, attendance tracking, result management, and user authentication**. Additionally, it follows **CRUD (Create, Read, Update, Delete) operations** to facilitate seamless interactions with the database, allowing faculty and administrators to manage student information with ease.

Benefits of the Student Management System

1. **Automation & Efficiency**
 - Eliminates manual record-keeping, reducing human errors and workload.
 - Enables quick **data retrieval, updates, and deletion**.
2. **Data Security & Integrity**
 - MySQL database ensures **secure storage with access control mechanisms**.
 - Prevents **unauthorized modifications** with authentication protocols.
3. **Organized Academic Tracking**
 - Faculty can **analyze student performance** efficiently through the **View Results** module.
 - Attendance and grade records are easily accessible for reporting and decision-making.
4. **Scalability & Future Enhancements**
 - The system can be **expanded** to support more students, courses, and features.
 - Future upgrades may include **data analytics, cloud integration, and AI-based recommendations**.

Motivation

The motivation behind developing this **Student Management System** arises from the need to:

1. **Automate Student Data Management**

- Manual data entry and record-keeping are prone to errors and inefficiencies.
- The system provides a **structured, digital approach** to store and retrieve student records effortlessly.

2. **Enhance Institutional Efficiency**

- Automating **course management, attendance tracking, and result management** reduces administrative workload.
- A **dashboard** provides quick access to essential features, making navigation easier.

3. **Improve Data Accuracy & Security**

- Using **MySQL as the database** ensures data integrity and prevents unauthorized modifications.
- The authentication system with **Login, Register, and Forgot Password** enhances security.

4. **Simplify Course & Student Management**

- Faculty members can **add, update, and delete courses and student records** efficiently.
- Student performance can be analyzed through the **View Results** module.

5. **Streamline Attendance & Result Management**

- Tracking **student participation** is simplified through the **Manage Attendance** feature.
- Storing and retrieving **grades and performance records** ensures organized academic tracking.

6. **Ensure a User-Friendly Experience**

- **Tkinter GUI** provides an interactive and visually appealing interface.
- The **CRUD-based approach** makes operations intuitive for faculty and administrators.

Problem Statement

Educational institutions face challenges in efficiently managing student-related data, including enrollment, attendance, grades, and personal information. Manual processes or outdated systems often lead to errors, delays, and difficulties in accessing up-to-date records. Additionally, administrators, teachers, and students lack a centralized platform to streamline communication, track academic progress, and generate reports. This inefficiency hampers decision-making, increases administrative workload, and affects the overall educational experience. There is a need for an automated, user-friendly Student Management System to organize data, improve operational efficiency, and provide real-time access to stakeholders.

Educational institutions require an **automated system** to handle:

- **Student registration**
- **Course allocation**
- **Attendance tracking**
- **Result management**

Relevance of DBMS & HCI

- **DBMS (MySQL)** ensures structured data storage.
- **HCI (Tkinter GUI)** provides an intuitive user experience.

Report Overview

This document outlines system design, implementation, testing, and potential improvements.

• Objectives

1. To develop a **Tkinter-based GUI** for easy interaction.
 2. To implement **MySQL database** for storing student records.
 3. To provide **secure authentication** (login, register, forgot password).
 4. To enable **CRUD operations** for students, courses, attendance, and results.
 5. To ensure **user-friendly navigation** (HCI principles).
-

• Technologies Used

Programming Language : Python

DBMS : MySQL

User Interface Tools : Tkinter

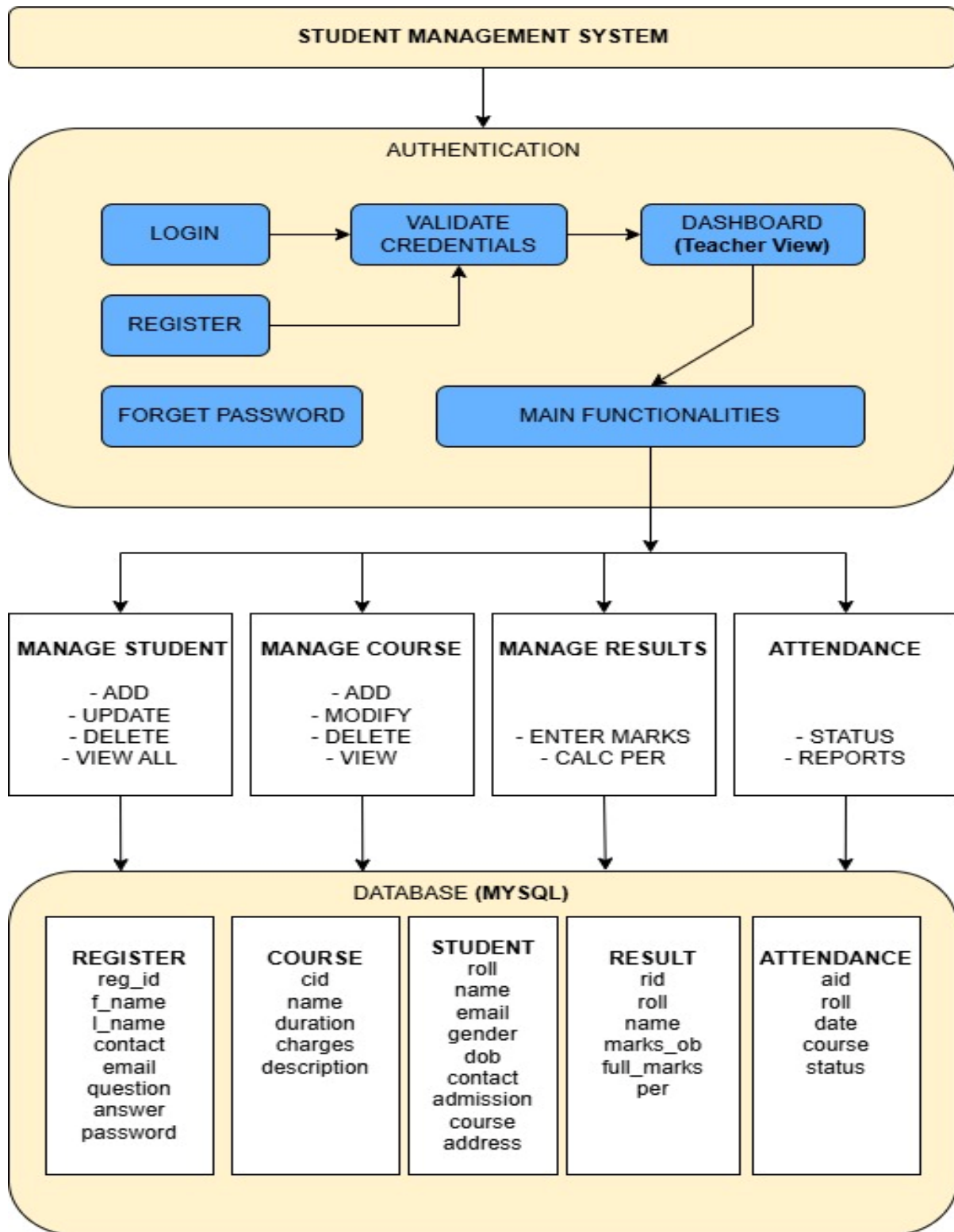
Other Tools : MySQL Workbench (For ADBMS)
VS Code (For Python)
Balsamiq (For HCI)

Libraries : mysql-connector-
python, tkinter,
messagebox, ttk

OS : Windows

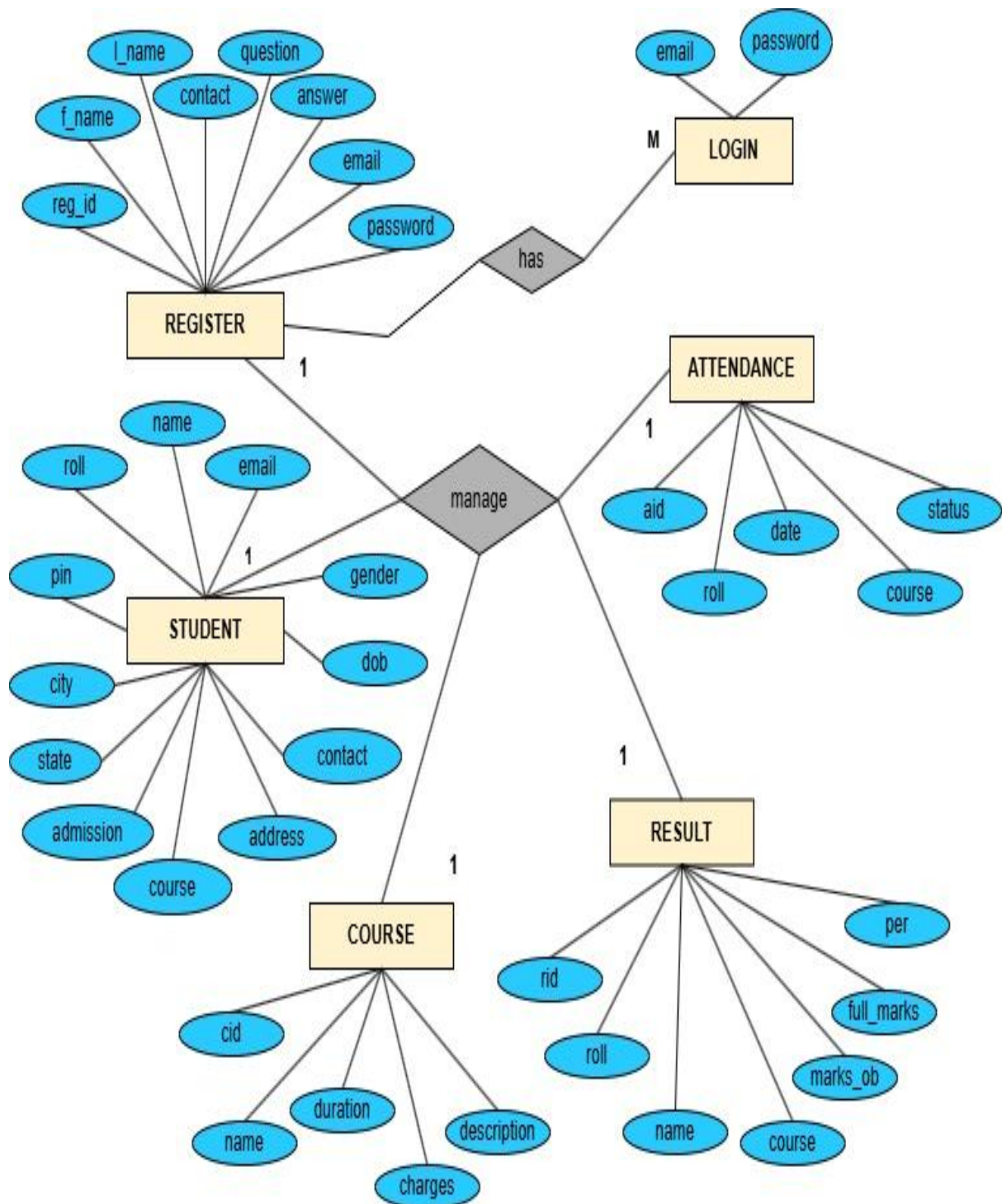
• System Design

1. Block Diagram



2. Database Design

• ER Diagram:



- **Tables:**

Manage Register

Column name	Data Type	Constraints	Description
reg_id_ID	INT(Primary Key, Auto Increment)	UNIQUE, NOT NULL	Unique identifier for faculty
F_name	VARCHAR(100)	NOT NULL	First name
L_name	VARCHAR(100)	NOT NULL	Last name
email	VARCHAR(100)	UNIQUE, NOT NULL	Faculty email for login
Contact	BIGINT	NOT NULL	Contact no of faculty
Question	VARCHAR(50)	NOT NULL	Question by system used later
Answer	VARCHAR(50)	NOT NULL	Used later at the time of forgot password
password	VARCHAR(255)	NOT NULL	Hashed password for authentication

Manage Course

Column name	Data Type	Constraints	Description
Course_ID	INT(Primary Key, Auto Increment)	UNIQUE, NOT NULL	Unique identifier for Courses
Course_name	VARCHAR(100)	NOT NULL	Course name
Duration	VARCHAR(50)	NOT NULL	Duration of course
Charges	INT	NOT NULL	Charges of particular course
Description	VARCHAR(100)	NOT NULL	Description of course

Manage Result

Column name	Data Type	Constraints	Description
Result_ID	INT(Primary Key, Auto Increment)	UNIQUE, NOT NULL	Unique identifier for Result
Course_ID	INT	FOREIGN KEY REFERENCES	Course ID
Student_id	INT	FOREIGN KEY REFERENCES	student ID
marks	INT	CHECK(marks BETWEEN 0 AND 100)	Marks obtained (0-100)
Full_marks	INT	CHECK(marks BETWEEN 0 AND 100)	Marks obtained (0-100)
per	VARCHAR(50)	NOT NULL	Percentage obtain

Manage Student

Column name	Data Type	Constraints	Description
Student_ID	INT(Primary Key, Auto Increment)	UNIQUE, NOT NULL	Unique identifier for student
name	VARCHAR(100)	NOT NULL	Student name
email	VARCHAR(100)	UNIQUE, NOT NULL	Student email for login
Gender	VARCHAR(20)	NOT NULL	Hashed password for authentication
Course_ID	INT	FOREIGN KEY REFERENCES	Foreign key of course table
State	VARCHAR(50)	NOT NULL	State of student
City	VARCHAR(50)	NOT NULL	City of student
Pin code	INT	NOT NULL	Pin code of student
Address	VARCHAR(250)	NOT NULL	Address of student
DOB	Date	NOT NULL	Date of birth of student
Contact NO	BIG INT	UNIQUE ,NOT NULL	Contact details of student
Admission Date	Date	NOT NULL	Date of admission of student

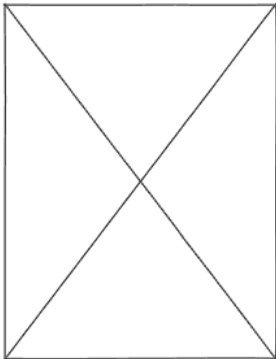
Manage Attendance

Column name	Data Type	Constraints	Description
Attendance_id	INT(Primary Key, Auto Increment)	UNIQUE, NOT NULL	Unique identifier for Result
Course_ID	INT	FOREIGN KEY REFERENCES	Course ID
Enrollment_id	INT	FOREIGN KEY REFERENCES	Enrollment ID
Date	DATE	NOT NULL	Attendance date
status	VARCHAR(10)	CHECK(status IN ('Present', 'Absent', 'Late'))	Attendance status

3. User Interface Design

- Wireframes

REGISTER FORM



REGISTER HERE..

First Name :

Last Name :

Contact No :

Email :

Security Que :

Answer :

Password :

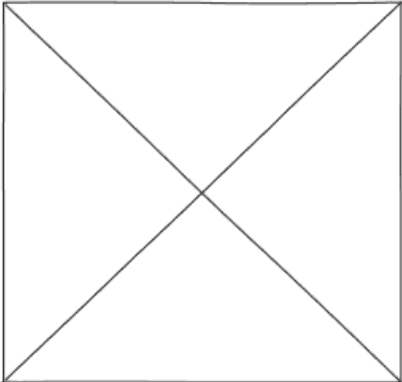
Confirm Pass :

LOGIN FORM

... LOGIN HERE ...

Email Add :

Password :



FORGET PASSWORD

... FORGET PASSWORD ...

Security Question :

ComboBox

Answer :

Password :

Reset Password

STUDENT DASHBOARD

WELCOME TO THE STUDENT MANAGEMENT SYSTEM

SMS

COURSE

STUDENT

RESULT

VIEW RESULT

ATTENDANCE

LOGOUT

TOTAL COURSE

TOTAL STUDENT

TOTAL RESULTS

FOOTER

STUDENT MANAGEMENT SYSTEM

[illegible]

MANAGE STUDENT DETAILS

Manage Student Details

Roll No :

DOB :

Roll No :

Search

Name :

Contact :

Email :

Addmission :

Gender :

Course :

Address :

SAVE

UPDATE

DELETE

CLEAR

FOOTER

RESULT MANAGEMENT

Add Student Result

Select Student :

Select

▼

Search

Name :

Course :

Marks Ob :

Full Marks :

SUBMIT

CLEAR

FOOTER

VIEW RESULT

Student Result Report

Security Question :

ComboBox

▼

Search

Show All

Delete

Generate Report

FOOTER

- Screenshots
- Register Form

Register - Student Management System

The image shows a student registration interface. On the left, a large smartphone displays a registration form with fields for Username, Email, Password, and a Sign Up button. A student character is interacting with the phone. On the right, a separate registration form is shown with the following fields:

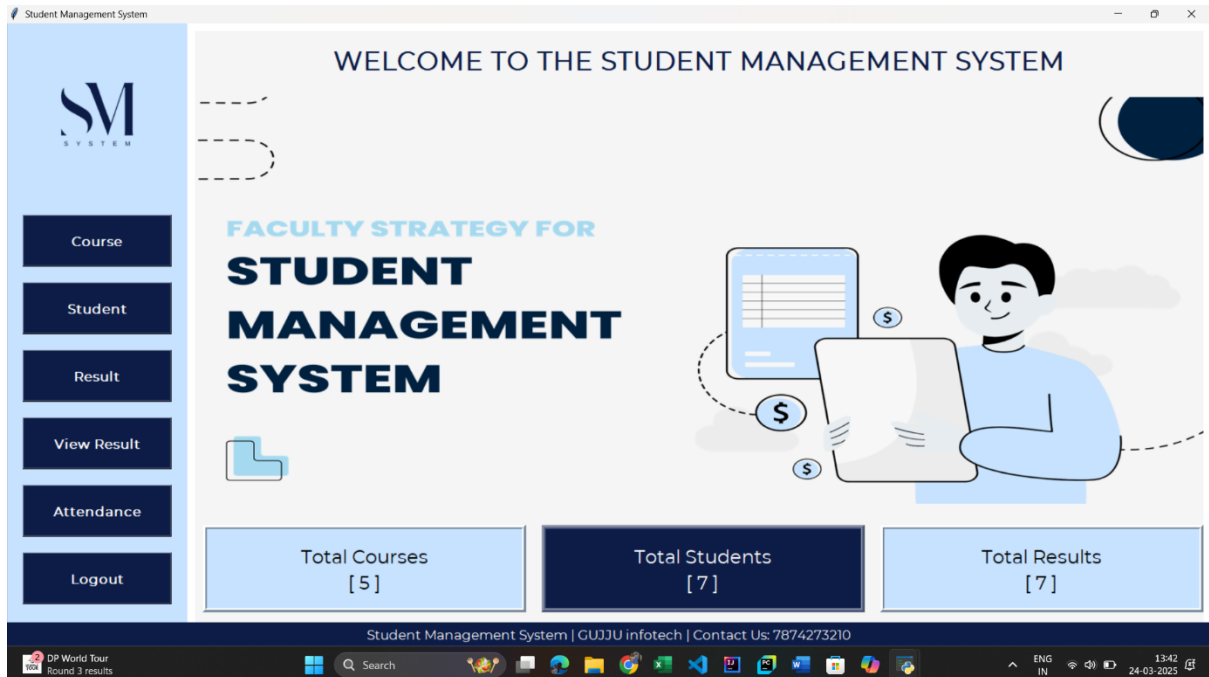
- First Name: [Text Input]
- Last Name: [Text Input]
- Contact No: [Text Input]
- Email: [Text Input]
- Security Question: [Dropdown Menu]
- Answer: [Text Input]
- Password: [Text Input]
- Confirm Password: [Text Input]

Below the form, there is a checkbox labeled "I Agree to the Terms & Conditions" and a "Register!!" button.

- **Login Form**

The image shows a login page for a 'Student Management System'. The page is framed by a light blue border decorated with various school supplies like pencils, crayons, a paint palette, a globe, and a pair of scissors. The central white area contains the login form. At the top, the title 'Login to Your Account' is underlined in blue. Below it, there are two input fields: 'Email Address:' and 'Password:'. Under the password field, there are two links: 'New User? Register Here' in blue and 'Forgot Password?' in red. A large blue 'Sign In' button is at the bottom. To the right of the form is an illustration of a man and a woman in blue uniforms standing next to a large blue server rack. The man is holding a tablet, and the woman is pointing at the server. The overall design is clean and modern, with a focus on education.

- Dash-Board Form



The dashboard features a sidebar with navigation links: Course, Student, Result, View Result, Attendance, and Logout. The main content area includes a welcome message, a faculty strategy graphic, and three summary cards: Total Courses [5], Total Students [7], and Total Results [7].

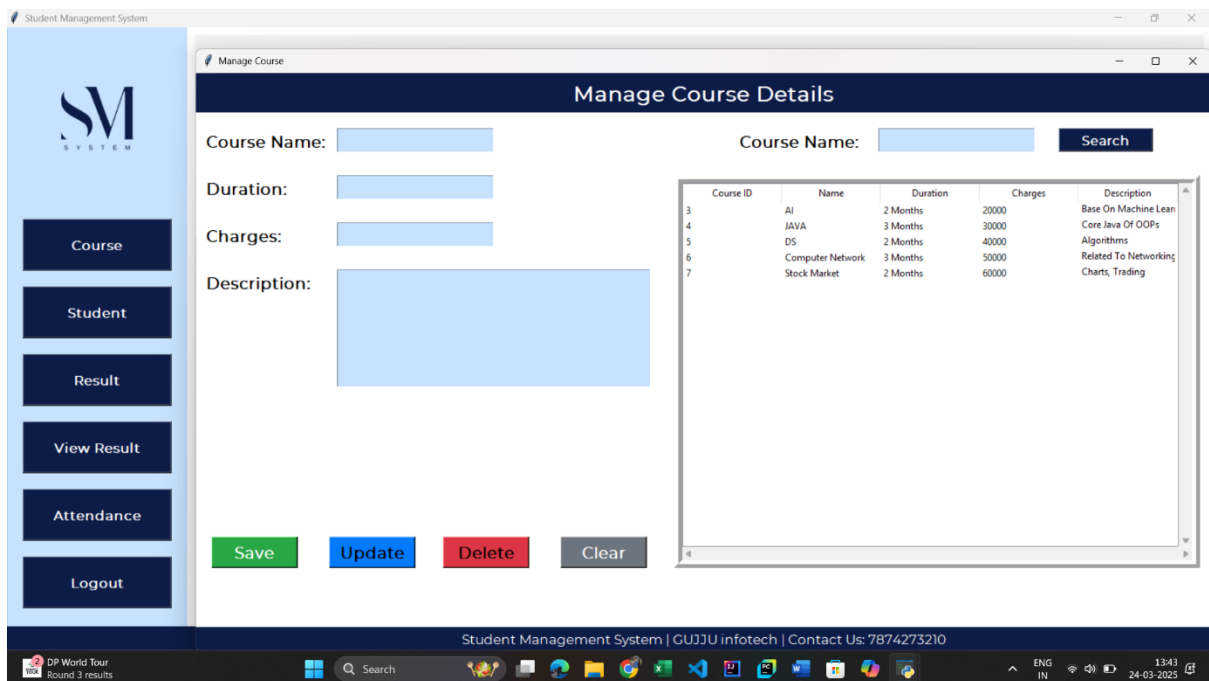
WELCOME TO THE STUDENT MANAGEMENT SYSTEM

FACULTY STRATEGY FOR
**STUDENT
MANAGEMENT
SYSTEM**

Total Courses [5] Total Students [7] Total Results [7]

Student Management System | GUJJU infotech | Contact Us: 7874273210

- Course Form



The 'Manage Course Details' form includes input fields for Course Name, Duration, Charges, and Description. It also features a search bar and a table listing existing courses.

Manage Course Details

Course Name: Course Name: Search

Duration:

Charges:

Description:

Course ID	Name	Duration	Charges	Description
3	AI	2 Months	20000	Base On Machine Learn
4	JAVA	3 Months	30000	Core Java Of OOPs
5	DS	2 Months	40000	Algorithms
6	Computer Network	3 Months	50000	Related To Networking
7	Stock Market	2 Months	60000	Charts, Trading

Save Update Delete Clear

Student Management System | GUJJU infotech | Contact Us: 7874273210

• Student Form

Manage Student Details

Roll No: D.O.B: Roll No: **Search**

Name: Contact:

Email: Admission:

Gender: Course:

State: City: Pin:

Address:

Save **Update** **Delete** **Clear**

Roll No	Name	Email	Gender	D.O.B	Cc
1	Yash	yash@gmail.com	Male	2004-06-05	745678
2	Kaushal	kaushal@gmail.co	Male	2003-10-05	984654
3	Kashish	kashu@gmail.cor	Female	2003-09-18	972354
4	Dhvani	dhani@gmail.cor	Female	2004-01-20	972383
5	Shubham	shubham@gmail	Male	2003-12-27	934568
6	Mit	mit@gmail.com	Male	2003-09-20	972354
7	Mohit	mohit@gmail.cor	Male	2003-09-21	972354

Student Management System | GUJJU infotech | Contact Us: 7874273210

• Result Form

Add Student Results

Select Student: **Search**

Name:

Course:

Marks Obtained:

Full Marks:

Submit **Clear**

Student Management System | GUJJU infotech | Contact Us: 7874273210

- View Result Form

The screenshot shows the 'View Result Report' window of the Student Management System. On the left is a sidebar with the 'SM SYSTEM' logo and navigation buttons: Course, Student, Result, View Result, Attendance, and Logout. The main window has a title bar 'View Result Report' and a header 'Student Result Report'. Below the header is a search bar with the text 'Search Roll No:' and a 'Search' button. To the right of the search bar are buttons for 'Show All', 'Delete', and 'Generate Report'. The main area contains a table with the following data:

Roll No	Name	Course	Marks Obtained	Full Marks	Percentage
1	Yash	JAVA	23	25	92.0
2	Kaushal	Computer Network	20	25	80.0
3	Kashish	DS	21	25	84.0
4	Dhvani	AI	18	25	72.0
5	Shubham	Stock Market	24	25	96.0
6	Mit	DS	23	25	92.0
7	Mohit	DS	21	25	84.0

The footer of the window displays 'Student Management System | GUJJU infotech | Contact Us: 7874273210' and a Windows taskbar at the bottom with the date '24-03-2025' and time '13:43'.

- Attendance Form

The screenshot shows the 'Manage Student Attendance' window of the Student Management System. On the left is the same sidebar as the previous form. The main window has a title bar 'Attendance Management System' and a header 'Manage Student Attendance'. Below the header is a form with 'Select Course:' and a dropdown menu, and a 'Date:' field with the value '2025-03-24'. Below this is a table with the following columns: Roll, Name, Email, Gender, and Status. At the bottom of the window are buttons for 'Submit Attendance', 'Clear', 'All Present', 'All Absent', and 'View Attendance'. To the right of these buttons are status indicators: 'Present: 0' and 'Absent: 0'. The footer of the window displays 'Student Management System | GUJJU infotech | Contact Us: 7874273210' and a Windows taskbar at the bottom with the date '24-03-2025' and time '13:44'.

• Implementation

Database Interaction

```
import mysql.connector

def create_db():

    try:

        # Connect to MySQL

        con = mysql.connector.connect(

            host="localhost", # Change this if MySQL is running on another server

            user="root",      # Your MySQL username

            password="Ashish@0629",    # Your MySQL password

        )

        cur = con.cursor()

        # Create database if not exists

        cur.execute("CREATE DATABASE IF NOT EXISTS sms")

        con.commit()

        # Connect to the newly created database

        con.database = "sms"

        # Create course table

        cur.execute("""

        CREATE TABLE IF NOT EXISTS course(

        cid INT AUTO_INCREMENT PRIMARY KEY,

        name VARCHAR(255) UNIQUE, # Ensure course names are unique

        duration VARCHAR(100),

        charges VARCHAR(50),

        description TEXT

        )

        """)

        con.commit()

        # Create student table with foreign key

        cur.execute("""

        CREATE TABLE IF NOT EXISTS student(

        roll INT AUTO_INCREMENT PRIMARY KEY,

        name VARCHAR(255),
```

```
email VARCHAR(255),
gender VARCHAR(10),
dob DATE,
contact VARCHAR(15),
admission DATE,
course VARCHAR(255),
state VARCHAR(100),
city VARCHAR(100),
pin VARCHAR(10),
address TEXT,
CONSTRAINT fk_course FOREIGN KEY (course) REFERENCES course(name) ON DELETE
CASCADE
)
)
con.commit()

# Create result table with foreign key
cur.execute("""
CREATE TABLE IF NOT EXISTS result(
rid INT AUTO_INCREMENT PRIMARY KEY,
roll INT,
name VARCHAR(255),
course VARCHAR(255),
marks_ob INT,
full_marks INT,
per FLOAT,
CONSTRAINT fk_student FOREIGN KEY (roll) REFERENCES student(roll) ON DELETE
CASCADE
)
)
con.commit()

# Create register table
cur.execute("""
CREATE TABLE IF NOT EXISTS register (
reg_id INT AUTO_INCREMENT PRIMARY KEY,
f_name VARCHAR(255) NOT NULL,
```

```

l_name VARCHAR(255) NOT NULL,
contact VARCHAR(15),
email VARCHAR(255) UNIQUE NOT NULL,
question VARCHAR(255),
answer TEXT,
password VARCHAR(255) NOT NULL -- Ensure password is always provided
)
)
con.commit()

# Create attendance table with foreign key to course and student
cur.execute("""
CREATE TABLE IF NOT EXISTS attendance (
aid INT AUTO_INCREMENT PRIMARY KEY,
roll INT,
date DATE,
course INT,
status ENUM('P', 'A'),

CONSTRAINT fk_attendance_student FOREIGN KEY (roll) REFERENCES student(roll) ON
DELETE CASCADE,

CONSTRAINT fk_attendance_course FOREIGN KEY (course) REFERENCES course(cid) ON
DELETE CASCADE,

CONSTRAINT unique_attendance UNIQUE (roll, date, course)
)
)
con.commit()

print("Database and Tables Created Successfully!")

con.close()

except mysql.connector.Error as err:
print(f"Error: {err}")

# Run the function to create the database and tables
create_db()

```

User Interface Development

1. Setting Up the Main Dash-Board Window

```
from tkinter import *  
  
from PIL import Image, ImageTk  
  
from course import CourseClass  
  
from student import studentClass  
  
from result import resultClass  
  
from report import ReportClass  
  
from attendance import AttendanceClass  
  
from tkinter import messagebox  
  
import os  
  
import mysql.connector  
  
class SMS:  
  
    def __init__(self, root):  
        self.root = root  
  
        self.root.title("Student Management System")  
  
        self.root.geometry("1520x785+0+0")  
  
        self.root.config(bg="white")  
  
# Run Application  
  
if __name__ == "__main__":  
    root = Tk()  
    obj = SMS(root)  
    print("Dashboard is running...")  
    root.mainloop()
```

2. Creating a Register Page

```
# Register Frame  
  
Frame1 = Frame(self.root, bg="white")  
  
Frame1.place(x=600, y=130, width=670, height=500)  
  
title = Label(Frame1, text="REGISTER HERE", font=("montserrat", 20, "bold"), bg="white",  
fg="#17A2B8").place(x=50, y=30)  
  
# Form Fields  
  
f_name = Label(Frame1, text="First Name:", font=("montserrat", 15, "bold"), bg="white",  
fg="black").place(x=50, y=80)  
  
self.txt_fname = Entry(Frame1, font=("montserrat", 15), bg="lightgray")
```

```

self.txt_fname.place(x=50, y=120, width=250)

l_name = Label(Frame1, text="Last Name:", font=("montserrat", 15, "bold"), bg="white",
fg="black").place(x=370, y=80)

self.txt_lname = Entry(Frame1, font=("montserrat", 15), bg="lightgray")

self.txt_lname.place(x=370, y=120, width=250)

contact = Label(Frame1, text="Contact No:", font=("montserrat", 15, "bold"), bg="white",
fg="black").place(x=50, y=150)

self.txt_contact = Entry(Frame1, font=("montserrat", 15), bg="lightgray")

self.txt_contact.place(x=50, y=190, width=250)

email = Label(Frame1, text="Email:", font=("montserrat", 15, "bold"), bg="white",
fg="black").place(x=370, y=150)

self.txt_email = Entry(Frame1, font=("montserrat", 15), bg="lightgray")

self.txt_email.place(x=370, y=190, width=250)

question = Label(Frame1, text="Security question:", font=("montserrat", 15, "bold"), bg="white",
fg="black").place(x=50, y=220)

self.cmd_quest = ttk.Combobox(Frame1, font=("montserrat", 13), state='readonly', justify=CENTER)

self.cmd_quest['values'] = ("Select", "Your First Pet Name", "Your Birth Place", "Your Best Friend
Name")

self.cmd_quest.place(x=50, y=260, width=250)

self.cmd_quest.current(0)

answer = Label(Frame1, text="Answer:", font=("montserrat", 15, "bold"), bg="white",
fg="black").place(x=370, y=220)

self.txt_answer = Entry(Frame1, font=("montserrat", 15), bg="lightgray")

self.txt_answer.place(x=370, y=260, width=250)

password = Label(Frame1, text="Password:", font=("montserrat", 15, "bold"), bg="white",
fg="black").place(x=50, y=290)

self.txt_password = Entry(Frame1, font=("montserrat", 15), bg="lightgray")

self.txt_password.place(x=50, y=330, width=250)

cpassword = Label(Frame1, text="Confirm Password:", font=("montserrat", 15, "bold"), bg="white",
fg="black").place(x=370, y=290)

self.txt_cpassword = Entry(Frame1, font=("montserrat", 15), bg="lightgray")

self.txt_cpassword.place(x=370, y=330, width=250)

self.var_chk = IntVar()

chk = Checkbutton(Frame1, text="I Agree The Terms & Conditions", variable=self.var_chk,
onvalue=1, offvalue=0, bg="white", font=("montserrat", 12, 'bold')).place(x=50, y=370)

self.btn_img = ImageTk.PhotoImage(file="D:/Ashish
khadela/Master/SEM_2/PYTHON/Mini_Project/SMS/Code/img/register.png")

```

```
btn_register = Button(Frame1, image=self.btn_img, bd=0, cursor="hand2",
command=self.register_data).place(x=50, y=420)
```

```
btn_login = Button(self.root, text="Sign In", font=("montserrat", 12), bd=0, cursor="hand2",
command=self.login_window, bg="#17A2B8", fg="black", activebackground="#138496",
activeforeground="white")
```

```
btn_login.place(x=300, y=550, width=150)
```

3. Creating a Login Page

```
login_frame = Frame(self.root, bg="lightblue")
```

```
login_frame.place(x=370, y=130, width=800, height=500)
```

```
title = Label(login_frame, text="..... LOGIN HERE .....", font=("montserrat", 25, "bold", "underline"),
bg="lightblue", fg="#021e2f").place(x=250, y=30)
```

Labels & Entry Fields

```
Label(login_frame, text="EMAIL ADDRESS:", font=("montserrat", 15, "bold"), bg="lightblue",
fg="#021e2f").place(x=50, y=120)
```

```
self.txt_email = Entry(login_frame, font=("montserrat", 15), bg="lightgray")
```

```
self.txt_email.place(x=50, y=180, width=350)
```

```
Label(login_frame, text="PASSWORD:", font=("montserrat", 15, "bold"), bg="lightblue",
fg="#021e2f").place(x=50, y=230)
```

```
self.txt_password = Entry(login_frame, font=("montserrat", 15), bg="lightgray", show="*")
```

```
self.txt_password.place(x=50, y=290, width=350)
```

```
btn_reg = Button(login_frame, cursor="hand2", command=self.register_window, text="Register New
Account?", font=("montserrat", 14), bg="lightblue", bd=0, fg="#B00857").place(x=42, y=330)
```

```
btn_forget = Button(login_frame, cursor="hand2", command=self.forget_password, text="Forget
Password ?", font=("montserrat", 10), bg="lightblue", bd=0, fg="red").place(x=280, y=340)
```

```
btn_login = Button(login_frame, text="Sign In", font=("montserrat", 20, "bold"), bd=0, cursor="hand2",
command=self.login, bg="#021e2f", fg="white", activebackground="black",
activeforeground="white").place(x=50, y=400, width=350, height=50)
```

4. Creating a Course Page

Title

```
title = Label(self.root, text="Manage Course Details", padx=10, compound=LEFT, font=("montserrat",
20, "bold"), bg="#0C1C47", fg="white")
```

```
title.place(x=0, y=0, relwidth=1, height=50)
```

Widgets

```
lbl_courseName = Label(self.root, text="Course Name:", font=("montserrat", 15, 'bold'),
bg="white").place(x=10, y=70)
```

```
lbl_duration = Label(self.root, text="Duration:", font=("montserrat", 15, 'bold'),
bg="white").place(x=10, y=130)
```



```
lbl_charges = Label(self.root, text="Charges:", font=("montserrat", 15, 'bold'), bg="white").place(x=10, y=190)
```

```
lbl_description = Label(self.root, text="Description:", font=("montserrat", 15, 'bold'), bg="white").place(x=10, y=250)
```

Entry Fields

```
self.txt_courseName = Entry(self.root, textvariable=self.var_course, font=("montserrat", 15, 'bold'), bg="#C6E2FF")
```

```
self.txt_courseName.place(x=180, y=70, width=200)
```

```
txt_duration = Entry(self.root, textvariable=self.var_duration, font=("montserrat", 15, 'bold'), bg="#C6E2FF").place(x=180, y=130, width=200)
```

```
txt_charges = Entry(self.root, textvariable=self.var_charges, font=("montserrat", 15, 'bold'), bg="#C6E2FF").place(x=180, y=190, width=200)
```

```
self.txt_description = Text(self.root, font=("montserrat", 15, 'bold'), bg="#C6E2FF")
```

```
self.txt_description.place(x=180, y=250, width=400, height=150)
```

Buttons

```
self.btn_add = Button(self.root, text="Save", font=("montserrat", 15, 'bold'), bg="#28A745", fg="white", cursor="hand2", command=self.add)
```

```
self.btn_add.place(x=20, y=590, width=110, height=40)
```

```
self.btn_update = Button(self.root, text="Update", font=("montserrat", 15, 'bold'), bg="#007BFF", fg="black", cursor="hand2", command=self.update)
```

```
self.btn_update.place(x=170, y=590, width=110, height=40)
```

```
self.btn_delete = Button(self.root, text="Delete", font=("montserrat", 15, 'bold'), bg="#DC3545", fg="black", cursor="hand2", command=self.delete)
```

```
self.btn_delete.place(x=315, y=590, width=110, height=40)
```

```
self.btn_clear = Button(self.root, text="Clear", font=("montserrat", 15, 'bold'), bg="#6C757D", fg="white", cursor="hand2", command=self.clear)
```

```
self.btn_clear.place(x=465, y=590, width=110, height=40)
```

5. Creating a Student Page

Title

```
title = Label(self.root, text="Manage Student Details", padx=10, compound=LEFT, font=("montserrat", 20, "bold"), bg="#0C1C47", fg="white")
```

```
title.place(x=0, y=0, relwidth=1, height=50)
```

```
self.var_city = StringVar()
```

```
self.var_pin = StringVar()
```

column1

```
lbl_roll = Label(self.root, text="Roll No:", font=("montserrat", 15, 'bold'), bg="white").place(x=10, y=70)
```

Roll No Entry (Disabled)

```
self.txt_roll = Entry(self.root, textvariable=self.var_roll, font=("montserrat", 15, 'bold'), bg="#ffcb1",
state="readonly")
```

```
self.txt_roll.place(x=120, y=70, width=200)
```

```
lbl_name = Label(self.root, text="Name:", font=("montserrat", 15, 'bold'), bg="white").place(x=10,
y=130)
```

```
lbl_email = Label(self.root, text="Email:", font=("montserrat", 15, 'bold'), bg="white").place(x=10,
y=190)
```

```
lbl_gender = Label(self.root, text="Gender:", font=("montserrat", 15, 'bold'), bg="white").place(x=10,
y=250)
```

```
lbl_state = Label(self.root, text="State:", font=("montserrat", 15, 'bold'), bg="white").place(x=10,
y=310)
```

```
txt_state = Entry(self.root, textvariable=self.var_state, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=120, y=310, width=120)
```

```
lbl_city = Label(self.root, text="City:", font=("montserrat", 15, 'bold'), bg="white").place(x=270,
y=310)
```

```
txt_city = Entry(self.root, textvariable=self.var_city, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=340, y=310, width=120)
```

```
lbl_pin = Label(self.root, text="Pin:", font=("montserrat", 15, 'bold'), bg="white").place(x=490, y=310)
```

```
txt_pin = Entry(self.root, textvariable=self.var_pin, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=560, y=310, width=120)
```

```
lbl_address = Label(self.root, text="Address:", font=("montserrat", 15, 'bold'), bg="white").place(x=10,
y=360)
```

Entry Fields

```
txt_name = Entry(self.root, textvariable=self.var_name, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=120, y=130, width=200)
```

```
txt_email = Entry(self.root, textvariable=self.var_email, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=120, y=190, width=200)
```

```
self.txt_gender = ttk.Combobox(self.root, textvariable=self.var_gender, values=("Select", "Male",
"Female", "Other"), font=("montserrat", 15, 'bold'), state='readonly', justify=CENTER)
```

```
self.txt_gender.place(x=120, y=250, width=200)
```

```
self.txt_gender.current(0)
```

Widgets

Column2

```
lbl_dob = Label(self.root, text="D.O.B:", font=("montserrat", 15, 'bold'), bg="white").place(x=330,
y=70)
```

```
lbl_contact = Label(self.root, text="Contact:", font=("montserrat", 15, 'bold'),
bg="white").place(x=330, y=130)
```

```
lbl_admission = Label(self.root, text="Addmission:", font=("montserrat", 15, 'bold'),
bg="white").place(x=330, y=190)
```

```
lbl_course = Label(self.root, text="Course:", font=("montserrat", 15, 'bold'), bg="white").place(x=330,
y=250)
```

Entry Fields

```
self.course_list = []
```

```
# Fuction_call to update the list
```

```
txt_dob = Entry(self.root, textvariable=self.var_dob, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=480, y=70, width=200)
```

```
txt_contact = Entry(self.root, textvariable=self.var_contact, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=480, y=130, width=200)
```

```
txt_addmision = Entry(self.root, textvariable=self.var_a_date, font=("montserrat", 15, 'bold'),
bg="#C6E2FF").place(x=480, y=190, width=200)
```

```
self.txt_course = ttk.Combobox(self.root, textvariable=self.var_course, values=self.course_list,
font=("montserrat", 15, 'bold'), state='readonly', justify=CENTER)
```

```
self.txt_course.place(x=480, y=250, width=200)
```

```
self.txt_course.set("Select")
```

```
self.fetch_course()
```

Text Address

```
self.txt_address = Text(self.root, font=("montserrat", 15, 'bold'), bg="#C6E2FF")
```

```
self.txt_address.place(x=120, y=360, width=560, height=150)
```

Buttons

```
self.btn_add = Button(self.root, text="Save", font=("montserrat", 15, 'bold'), bg="#28A745",
fg="white", cursor="hand2", command=self.add)
```

```
self.btn_add.place(x=120, y=590, width=110, height=40)
```

```
self.btn_update = Button(self.root, text="Update", font=("montserrat", 15, 'bold'), bg="#007BFF",
fg="black", cursor="hand2", command=self.update)
```

```
self.btn_update.place(x=270, y=590, width=110, height=40)
```

```
self.btn_delete = Button(self.root, text="Delete", font=("montserrat", 15, 'bold'), bg="#DC3545",
fg="black", cursor="hand2", command=self.delete)
```

```
self.btn_delete.place(x=420, y=590, width=110, height=40)
```

```
self.btn_clear = Button(self.root, text="Clear", font=("montserrat", 15, 'bold'), bg="#6C757D",
fg="white", cursor="hand2", command=self.clear)
```

```
self.btn_clear.place(x=570, y=590, width=110, height=40)
```

6. Creating a Result Page

Title

```
title = Label(self.root, text="Add Student Results", padx=10, compound=LEFT, font=("montserrat",
20, "bold"), bg="#0C1C47", fg="white")
```

```
title.place(x=0, y=0, relwidth=1, height=50)
```

```
lbl_select = Label(self.root, text="Select Student:", font=("montserrat", 20, "bold"),
bg="white").place(x=50, y=100)
```

```

lbl_name = Label(self.root, text="Name:", font=("montserrat", 20, "bold"), bg="white").place(x=50,
y=160)

lbl_course = Label(self.root, text="Course:", font=("montserrat", 20, "bold"), bg="white").place(x=50,
y=220)

lbl_marks = Label(self.root, text="Marks Obtained:", font=("montserrat", 20, "bold"),
bg="white").place(x=50, y=280)

lbl_full_marks = Label(self.root, text="Full Marks:", font=("montserrat", 20, "bold"),
bg="white").place(x=50, y=340)

self.txt_student = ttk.Combobox(self.root, textvariable=self.var_roll, values=self.roll_list,
font=("montserrat", 19, 'bold'), state='readonly', justify=CENTER)

self.txt_student.place(x=320, y=105, width=200)

self.txt_student.set("Select")

btn_search = Button(self.root, text="Search", font=("montserrat", 15, 'bold'), bg="#0C1C47",
fg="white", cursor="hand2", command=self.search).place(x=550, y=105, width=120, height=40)

txt_name = Entry(self.root, textvariable=self.var_name, font=("montserrat", 20, 'bold'),
bg="#C6E2FF", state="readonly").place(x=320, y=165, width=350)

txt_course = Entry(self.root, textvariable=self.var_course, font=("montserrat", 20, 'bold'),
bg="#C6E2FF", state="readonly").place(x=320, y=225, width=350)

txt_marks = Entry(self.root, textvariable=self.var_marks, font=("montserrat", 20, 'bold'),
bg="#C6E2FF").place(x=320, y=285, width=350)

txt_full_marks = Entry(self.root, textvariable=self.var_full_marks, font=("montserrat", 20, 'bold'),
bg="#C6E2FF").place(x=320, y=345, width=350)

# Button

btn_add = Button(self.root, text="Submit", font=("montserrat", 15), bg="green", fg="white",
activebackground="green", cursor="hand2", command=self.add).place(x=320, y=480, width=150,
height=40)

btn_clear = Button(self.root, text="Clear", font=("montserrat", 15), bg="red", fg="white",
activebackground="red", cursor="hand2", command=self.clear).place(x=520, y=480, width=150,
height=40)

```

7. Creating a Attendance Page

Title

```

title = Label(self.root, text="Manage Student Attendance", padx=10, compound=LEFT,
font=("montserrat", 20, "bold"), bg="#0C1C47", fg="white")

title.place(x=0, y=0, relwidth=1, height=50)

```

Main Frame for Inputs

```

input_frame = Frame(self.root, bg="#C6E2FF", relief=FLAT, bd=0)

input_frame.place(x=20, y=70, width=1255, height=80)

```

```

lbl_select_course = Label(input_frame, text="Select Course:", font=("montserrat", 14, "bold"),
bg="#C6E2FF", fg="#0C1C47")

```

```

lbl_select_course.place(x=50, y=25)

self.var_course = StringVar()

self.course_list = []

self.txt_course = ttk.Combobox(input_frame, textvariable=self.var_course, values=self.course_list,
font=("Helvetica", 12), state='readonly', justify=CENTER)

self.txt_course.place(x=220, y=25, width=200, height=30)

self.txt_course.set("Select")

self.txt_course.bind("<<ComboboxSelected>>", self.update_student_list)

lbl_date = Label(input_frame, text="Date:", font=("montserrat", 14, "bold"), bg="#C6E2FF",
fg="#0C1C47")

lbl_date.place(x=500, y=25)

self.date = StringVar()

self.cal = DateEntry(input_frame, textvariable=self.date, date_pattern='yyyy-mm-dd',
font=("Helvetica", 12), bg="FFFFFF", fg="333333", borderwidth=1)

self.cal.place(x=580, y=25, width=200, height=30)

self.btn_csv = Button(input_frame, text="Attendance Report", font=("montserrat", 12),
bg="#0C1C47", fg="white", activebackground="white", command=self.export_to_csv)

self.btn_csv.place(x=1050, y=25, width=180, height=35)

self.btn_csv.bind("<Enter>", lambda e: self.btn_csv.config(bg="black"))

self.btn_csv.bind("<Leave>", lambda e: self.btn_csv.config(bg="#0C1C47"))

# Frame for Student List with Table-Like Structure

self.student_frame = Frame(self.root, bd=0, relief=FLAT, bg="FFFFFF")

self.student_frame.place(x=20, y=160, width=1255, height=450)

# Header for Student List

header_frame = Frame(self.student_frame, bg="#0C1C47", bd=0, relief=FLAT)

header_frame.pack(fill=X)

Label(header_frame, text="Roll", font=("montserrat", 12, "bold"), bg="#C6E2FF", fg="black",
width=10).pack(side=LEFT, padx=10, pady=5)

Label(header_frame, text="Name", font=("montserrat", 12, "bold"), bg="#C6E2FF", fg="black",
width=15).pack(side=LEFT, padx=10, pady=5)

Label(header_frame, text="Email", font=("montserrat", 12, "bold"), bg="#C6E2FF", fg="black",
width=20).pack(side=LEFT, padx=10, pady=5)

Label(header_frame, text="Gender", font=("montserrat", 12, "bold"), bg="#C6E2FF", fg="black",
width=10).pack(side=LEFT, padx=10, pady=5)

```

Code Snippet

1. Login Authentication

```
def login(self):  
    if self.txt_email.get() == "" or self.txt_password.get() == "":  
        messagebox.showerror("Error", "All fields are required", parent=self.root)  
    else:  
        try:  
            # Using MySQL Connector  
            con = mysql.connector.connect(  
                host="localhost",  
                user="root",  
                password="Ashish@0629",  
                database="sms"  
            )  
            cur = con.cursor(dictionary=True)  
            # Check if the email and password exist in the register table  
            cur.execute("SELECT * FROM register WHERE email=%s AND  
password=%s", (self.txt_email.get(), self.txt_password.get()))  
            row = cur.fetchone()  
            if row is None:  
                messagebox.showerror("Error", "Invalid EMAIL & PASSWORD",  
parent=self.root)  
            else:  
                messagebox.showinfo("Success", "Welcome!", parent=self.root)  
                self.open_dashboard() # Call function to open dashboard  
                con.close() # Close the connection  
        except mysql.connector.Error as es:  
            messagebox.showerror("Error", f"Database Error: {str(es)}",  
parent=self.root)
```

2. Register Feature

def register_data(self):

```
if (self.txt_fname.get() == "" or self.txt_contact.get() == "" or self.txt_email.get() == "" or
self.cmd_quest.get() == "Select" or self.txt_answer.get() == "" or self.txt_password.get() == ""
or self.txt_cpassword.get() == ""):
```

```
    messagebox.showerror("Error", "All Fields Are Required", parent=self.root)
```

```
elif self.txt_password.get() != self.txt_cpassword.get():
```

```
    messagebox.showerror("Error", "Password & Confirm Password Should Be Same",
parent=self.root)
```

```
elif self.var_chk.get() == 0:
```

```
    messagebox.showerror("Error", "Please Agree to Our Terms & Conditions", parent=self.root)
```

```
else:
```

try:

```
    con = mysql.connector.connect(
```

```
        host="localhost",
```

```
        user="root",
```

```
        password="Ashish@0629",
```

```
        database="sms"
```

```
    )
```

```
    cur = con.cursor()
```

```
    cur.execute("SELECT * FROM register WHERE email=%s",
(self.txt_email.get(),))
```

```
    row = cur.fetchone()
```

```
    if row is not None:
```

```
        messagebox.showerror("Error", "Email already registered! Try with another
email.", parent=self.root)
```

```
    else:
```

```
        cur.execute("""
```

```
INSERT INTO register (f_name, l_name, contact, email, question, answer,
password) VALUES (%s, %s, %s, %s, %s, %s, %s)
```

```
""", (
```

```
self.txt_fname.get(),
```

```
self.txt_lname.get(),
```

```
self.txt_contact.get(),
```

```
self.txt_email.get(),
```

```
self.cmd_quest.get(),
```



```

        self.txt_answer.get(),
        self.txt_password.get()
    ))
    con.commit()

    messagebox.showinfo("Success", "Registration Successful!",
        parent=self.root)

    self.clear()

    self.login_window()

    cur.close()

    con.close()

except mysql.connector.Error as err:
    messagebox.showerror("Error", f"Error due to: {str(err)}", parent=self.root)

```

3. Forget Password Feature

```
def forget_password(self):
```

```

    if self.txt_email.get() == "":
        messagebox.showerror("Error", "Please enter the email address to reset your password",
            parent=self.root)
    else:
        try:
            # Using MySQL Connector
            con = mysql.connector.connect(
                host="localhost",
                user="root",
                password="Ashish@0629",
                database="sms"
            )
            cur = con.cursor(dictionary=True)

            # Check if the email exists in the register table
            cur.execute("SELECT * FROM register WHERE email=%s",
                (self.txt_email.get(),))
            row = cur.fetchone()

            if row is None:
                messagebox.showerror("Error", "Please enter the valid email address to reset
                    your password", parent=self.root)
            else:

```

```

        con.close() # Close the connection

    except mysql.connector.Error as es:

        messagebox.showerror("Error", f"Database Error: {str(es)}",
                             parent=self.root)

```

4. ADD, UPDATE, DELETE, SEARCH, SHOW Feature

def add(self):

```

    con = mysql.connector.connect(host="localhost", user="root", password="Ashish@0629",
    database="sms")

    cur = con.cursor()

    try:

        if self.var_course.get() == "":

            messagebox.showerror("Error", "Course Name should be required", parent=self.root)

        else:

            cur.execute("SELECT * FROM course WHERE name=%s", (self.var_course.get(),))

            row = cur.fetchone()

            if row is not None:

                messagebox.showerror("Error", "Course name already present", parent=self.root)

            else:

                cur.execute("INSERT INTO course (name, duration, charges, description) VALUES
                (%s, %s, %s, %s)", (

                    self.var_course.get(),

                    self.var_duration.get(),

                    self.var_charges.get(),

                    self.txt_description.get("1.0", END)

                ))

                con.commit()

                messagebox.showinfo("Success", "Course Added Successfully", parent=self.root)

                self.show()

    except Exception as ex:

        messagebox.showerror("Error", f"Error due to {str(ex)}")

    finally:

        con.close()

```

def update(self):

```
con = mysql.connector.connect(host="localhost", user="root", password="Ashish@0629",
database="sms")

cur = con.cursor()

try:

    if self.var_course.get() == "":
        messagebox.showerror("Error", "Course Name should be required", parent=self.root)
    else:
        cur.execute("SELECT * FROM course WHERE name=%s", (self.var_course.get(),))
        row = cur.fetchone()

        if row is None:
            messagebox.showerror("Error", "Select Course from list", parent=self.root)
        else:
            cur.execute("UPDATE course SET duration=%s, charges=%s, description=%s
WHERE name=%s", (
                self.var_duration.get(),
                self.var_charges.get(),
                self.txt_description.get("1.0", END),
                self.var_course.get()
            ))
            con.commit()

            messagebox.showinfo("Success", "Course Updated Successfully", parent=self.root)

            self.show()

except Exception as ex:

    messagebox.showerror("Error", f"Error due to {str(ex)}")

finally:

    con.close()
```

def delete(self):

```
con = mysql.connector.connect(host="localhost", user="root", password="Ashish@0629",
database="sms")

cur = con.cursor()

try:

    if self.var_course.get() == "":
        messagebox.showerror("Error", "Course Name should be required", parent=self.root)
    else:
```

```
cur.execute("SELECT * FROM course WHERE name=%s", (self.var_course.get(),))

row = cur.fetchone()

if row is None:

    messagebox.showerror("Error", "Please select course from the list first",
parent=self.root)

else:

    op = messagebox.askyesno("Confirm", "Do you really want to delete?",
parent=self.root)

    if op:

        cur.execute("DELETE FROM course WHERE name=%s", (self.var_course.get(),))

        con.commit()

        messagebox.showinfo("Delete", "Course Deleted Successfully", parent=self.root)

        self.clear()

except Exception as ex:

    messagebox.showerror("Error", f"Error due to {str(ex)}")

finally:

    con.close()
```

def search(self):

```
con = mysql.connector.connect(host="localhost", user="root", password="Ashish@0629",
database="sms")

cur = con.cursor()

try:

    cur.execute(f"SELECT * FROM course WHERE name LIKE
    '%{self.var_search.get()}%'")

    rows = cur.fetchall()

    self.CourseTable.delete(*self.CourseTable.get_children())

    for row in rows:

        self.CourseTable.insert("", END, values=row)

except Exception as ex:

    messagebox.showerror("Error", f"Error due to {str(ex)}")

finally:

    con.close()
```

def show(self):

```
con = mysql.connector.connect(host="localhost", user="root", password="Ashish@0629",  
database="sms")
```

```
cur = con.cursor()
```

try:

```
cur.execute("SELECT * FROM course")
```

```
rows = cur.fetchall()
```

```
self.CourseTable.delete(*self.CourseTable.get_children())
```

```
for row in rows:
```

```
self.CourseTable.insert("", END, values=row)
```

except Exception as ex:

```
messagebox.showerror("Error", f"Error due to {str(ex)}")
```

finally:

```
con.close()
```

• Testing and Results

Testing

- **Unit Testing:** Individual functions tested.
- **Integration Testing:** Verified seamless database interaction.
- **System Testing:** Complete workflow validated.

Results

- System successfully stores and retrieves student data.
 - CRUD operation in all module
-

• Conclusion

Achievements

- Fully functional Student Management System.
- Implemented CRUD operations with MySQL.

Limitations

- Currently a standalone desktop application.

Skills Gained

- Python GUI development
 - Database management with MySQL
 - Software design principles
-

• Future Enhancements

- Convert to a web-based system using Flask/Django.
 - Role-based access for users.
 - Export results in Excel/PDF format.
 - Integration with an attendance tracking system.
-

• References

- **Python Official Documentation:**
<https://docs.python.org/3/>
 - **MySQL Documentation:**
<https://dev.mysql.com/doc/>
 - **Tkinter Documentation:**
<https://docs.python.org/3/library/tkinter.html>
-