

**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.** **Mandaliya Yashkumar and Mr. Kaushal Muniwala** has satisfactorily completed the course of Activity – “A2” prescribed by the JAIN(Deemed-to-be-University) for the semester “2nd” M.Sc. – CS & IT degree course in the year 2024 - 2026.

Date: “1th April 2025”

Signature of Student Head of the Department Signature of Faculty In charge



PROGRAM: MASTER OF SCIENCE IN COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

[M.Sc. - CS & IT]

Mini Project

**Student Management System**

Semester - 2

Submitted To: Submitted By:

Dr M N Nachappa Ashish Khadela (24MSRCI012)

Prof. Haripriya V. Kaushal Muniwala (24MSRCI013)

Prof. Raghavendra R. Yash Mandaliya (24MSRCI028)

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

# Abstract

1. Project Proposal

Purpose:

The Student Management System (SMS) is a software application designed to manage student records efficiently. It allows administrators and teachers to store, modify, and retrieve student information such as personal details, academic records and attendance record. The system enhances efficiency, reduces paperwork, and ensures data accuracy.

Objectives:

* + Store and manage student details securely.
  + Provide an easy-to-use interface for administrators.
  + Allow CRUD (Create, Read, Update, Delete) operations on student data.
  + Implement an attendance tracking system to record student presence.
  + Ensure data integrity using MySQL as the backend database.

1. Features to be Implemented
2. User Authentication:
   * Admin login with credentials (Email, Password).
3. Student Record Management:
   * Add, update, delete student information (name, age, gender, class, contact details).
4. Course Management:
   * Assign courses to students.
5. Grade Management:
   * Store, update, and display student grades.
6. Search and Filtering:
   * Search students by roll number.
7. Attendance Management:
   * Teachers can record student attendance (Present or Absent)
   * Attendance history can be retrieved and displayed.
8. Report Generation:
   * Generate student reports (performance, Grades and course details).
9. GUI Interface:
   * Tkinter for user-friendly forms and tables.
10. Features to be Implemented

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)

# 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:

* + Frontend: Tkinter (Python)
  + Backend: Python
  + Database: MySQL (MySQL Workbench)

# Introduction

Background & Motivation

Managing student records manually can be time-consuming and error-prone. This system automates record-keeping and ensures efficient data handling.

Problem Statement

The system aims to address challenges in managing student details, results, and attendance efficiently in an educational institution.

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

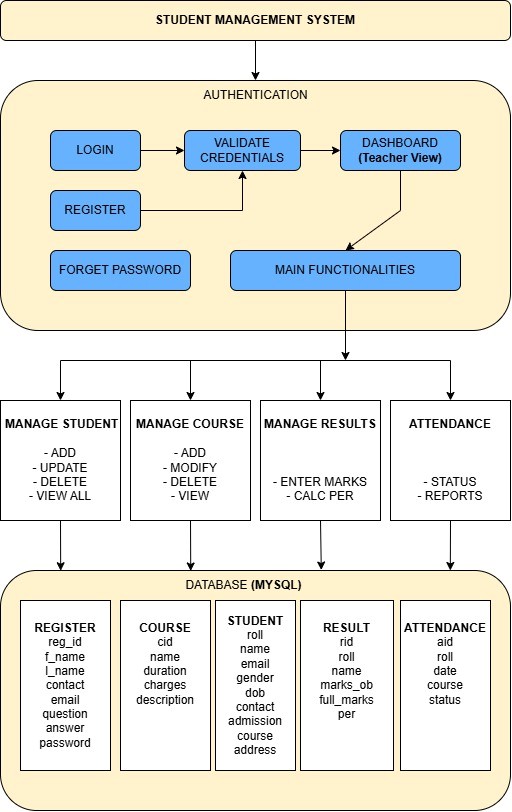
Other Tools: MySQL (MySQL Workbench), VS Code

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

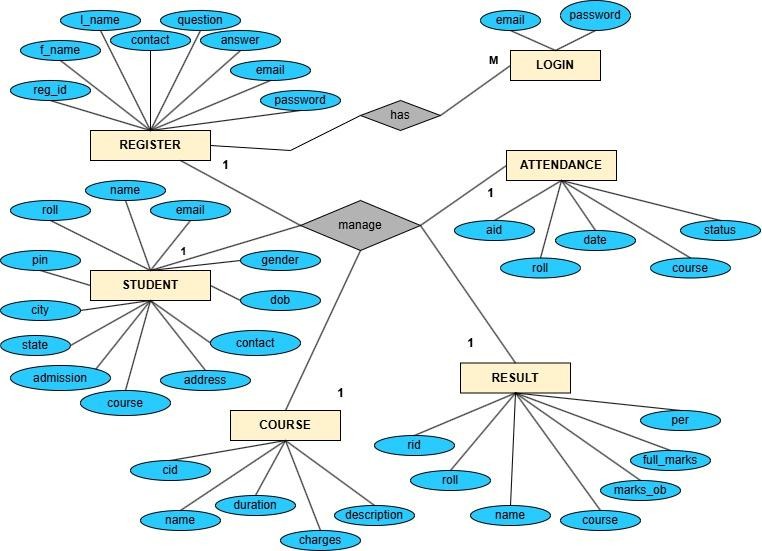
OS: Windows

# System Design

1. Block Diagram



1. Database Design
   * ER Diagram:



* + Tables:

Manage Register

|  |  |  |
| --- | --- | --- |
| Table Name | Field Name | Data Type |
| register | reg\_id | INT |
|  | f\_name | VARCHAR(50) |
|  | l\_name | VARCHAR(50) |
|  | contact | BIGINT |
|  | email | VARCHAR(50) |
|  | question | VARCHAR(50) |
|  | answer | VARCHAR(50) |
|  | password | VARCHAR(50) |

Manage Course

|  |  |  |
| --- | --- | --- |
| Table Name | Field Name | Data Type |
| course | cid | INT |
|  | name | VARCHAR(50) |
|  | duration | VARCHAR(50) |
|  | charges | INT |
|  | description | VARCHAR(100) |

Manage Student

|  |  |  |
| --- | --- | --- |
| Table Name | Field Name | Data Type |
| student | roll | INT |
|  | name | VARCHAR(50) |
|  | email | VARCHAR(50) |
|  | gender | VARCHAR(10) |
|  | dob | DATE |
|  | contact | BIGINT |
|  | admission | DATE |
|  | course | VARCHAR(50) |
|  | state | VARCHAR(50) |

|  |  |  |
| --- | --- | --- |
|  | city | VARCHAR(50) |
|  | pin | INT |
|  | address | VARCHAR(100) |

Manage Result

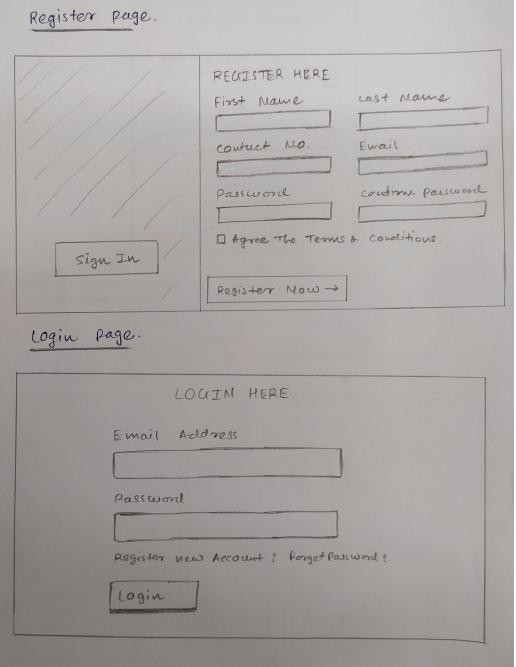
|  |  |  |
| --- | --- | --- |
| Table Name | Field Name | Data Type |
| result | rid | INT |
|  | roll | INT |
|  | name | VARCHAR(50) |
|  | course | VARCHAR(50) |
|  | marks\_ob | INT |
|  | full\_marks | INT |
|  | per | VARCHAR(50) |

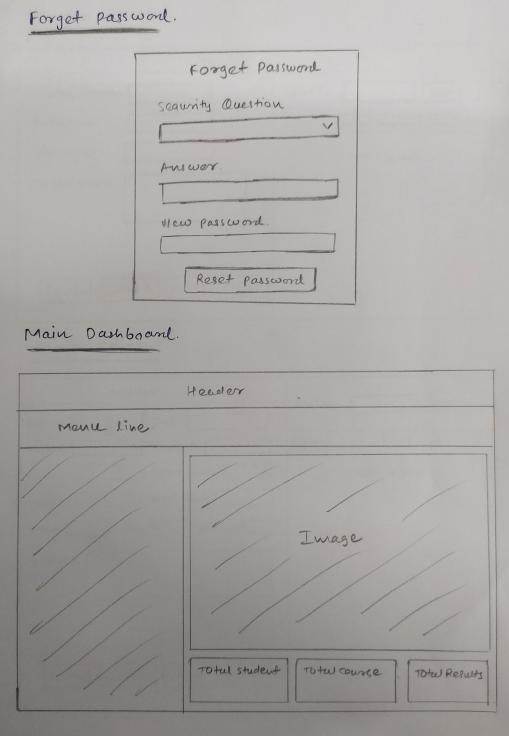
Manage Attendance

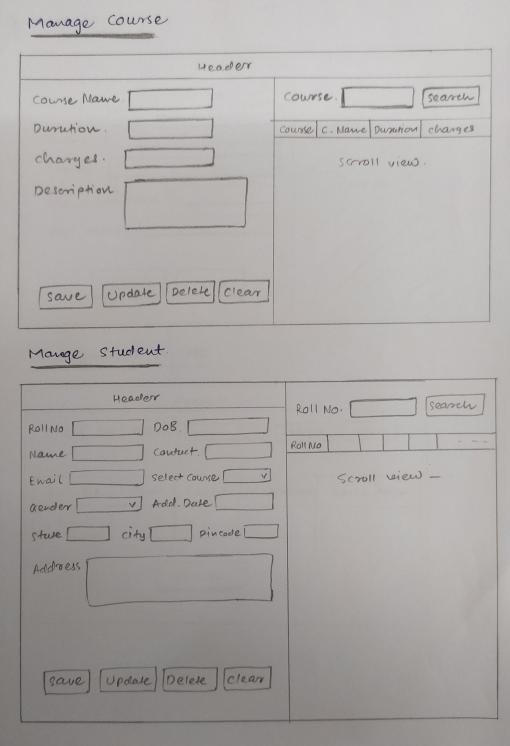
|  |  |  |
| --- | --- | --- |
| Table Name | Field Name | Data Type |
| attendance | aid | INT |
|  | roll | INT |
|  | date | DATE |
|  | course | VARCHAR(50) |
|  | status | ENUM(‘P’, ‘A’) |

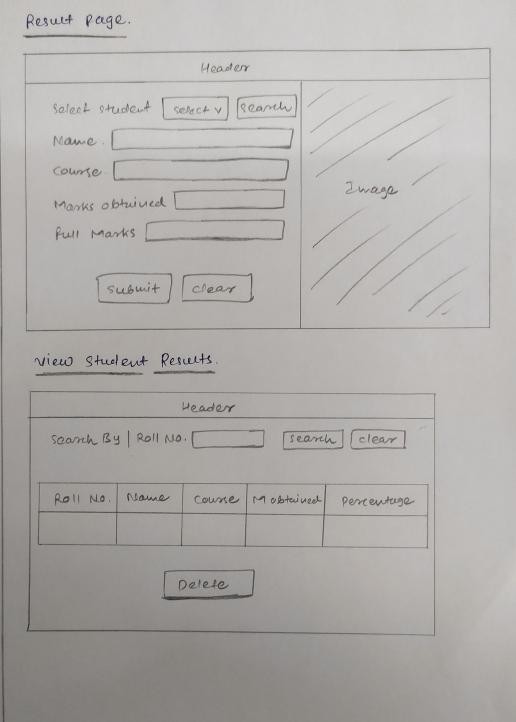
1. User Interface Design

* Wireframes

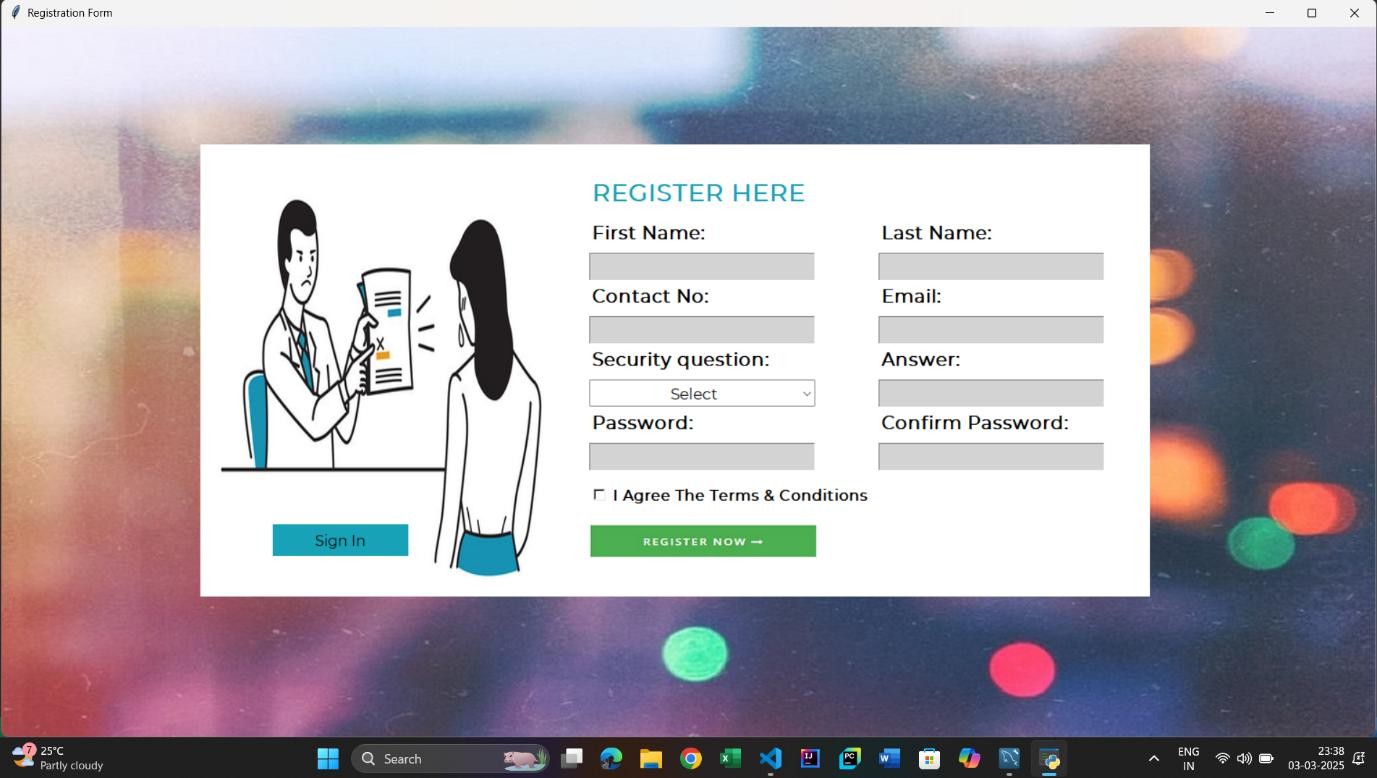




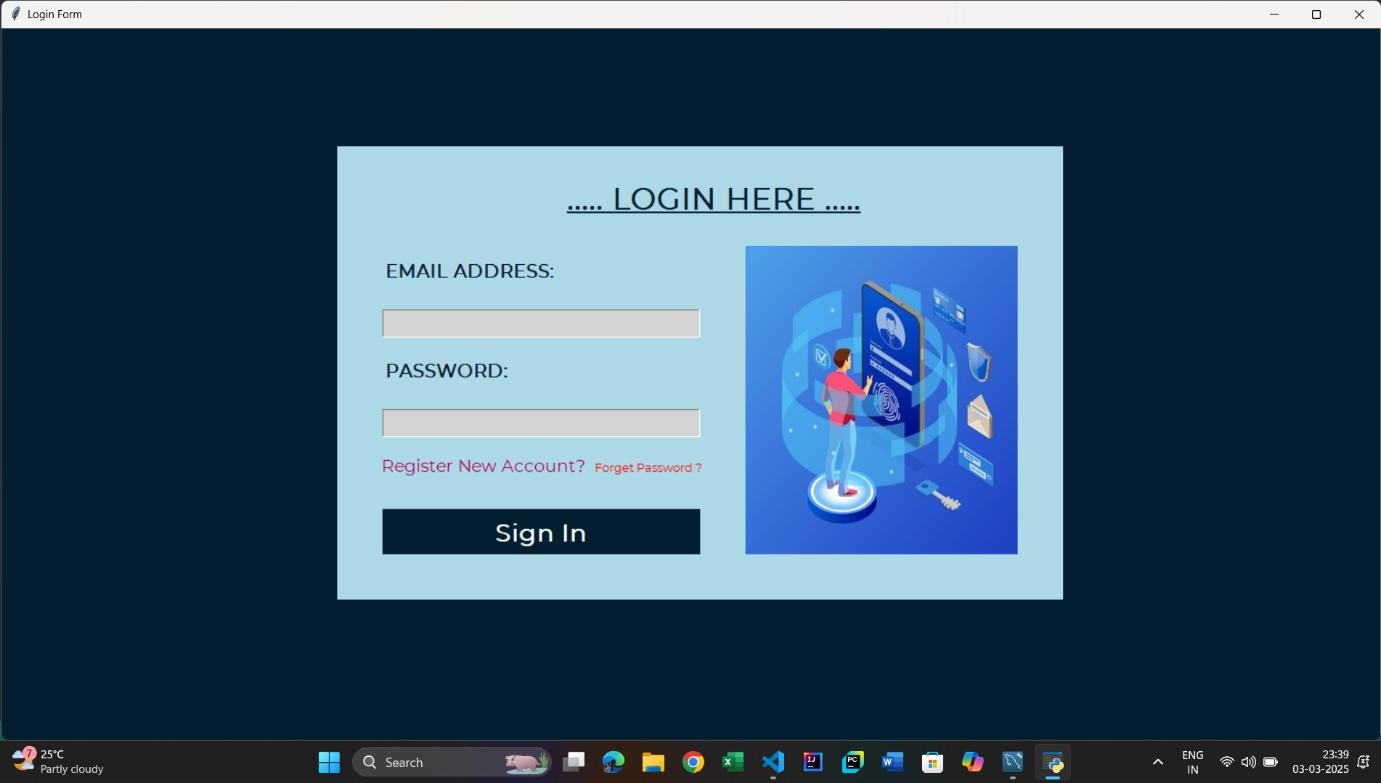




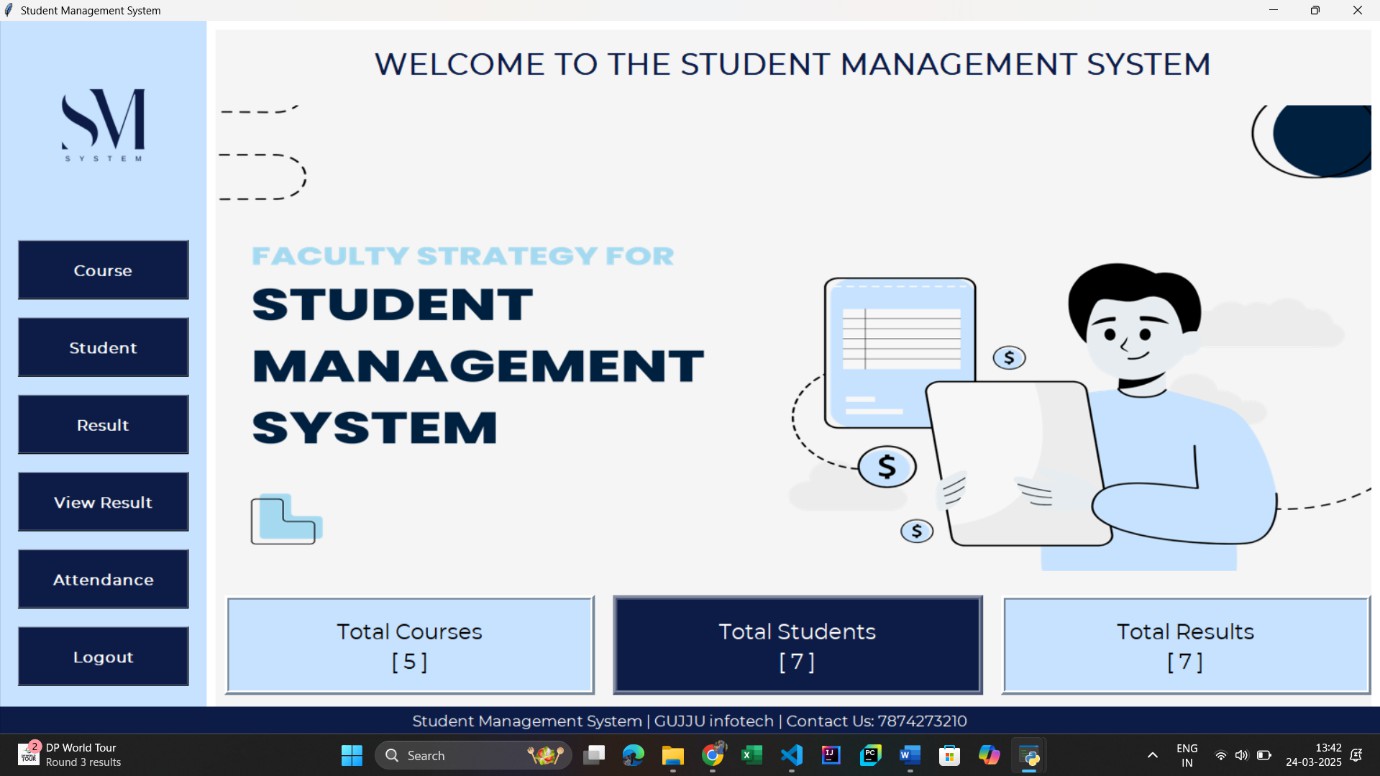
* Screenshots
* Register Form



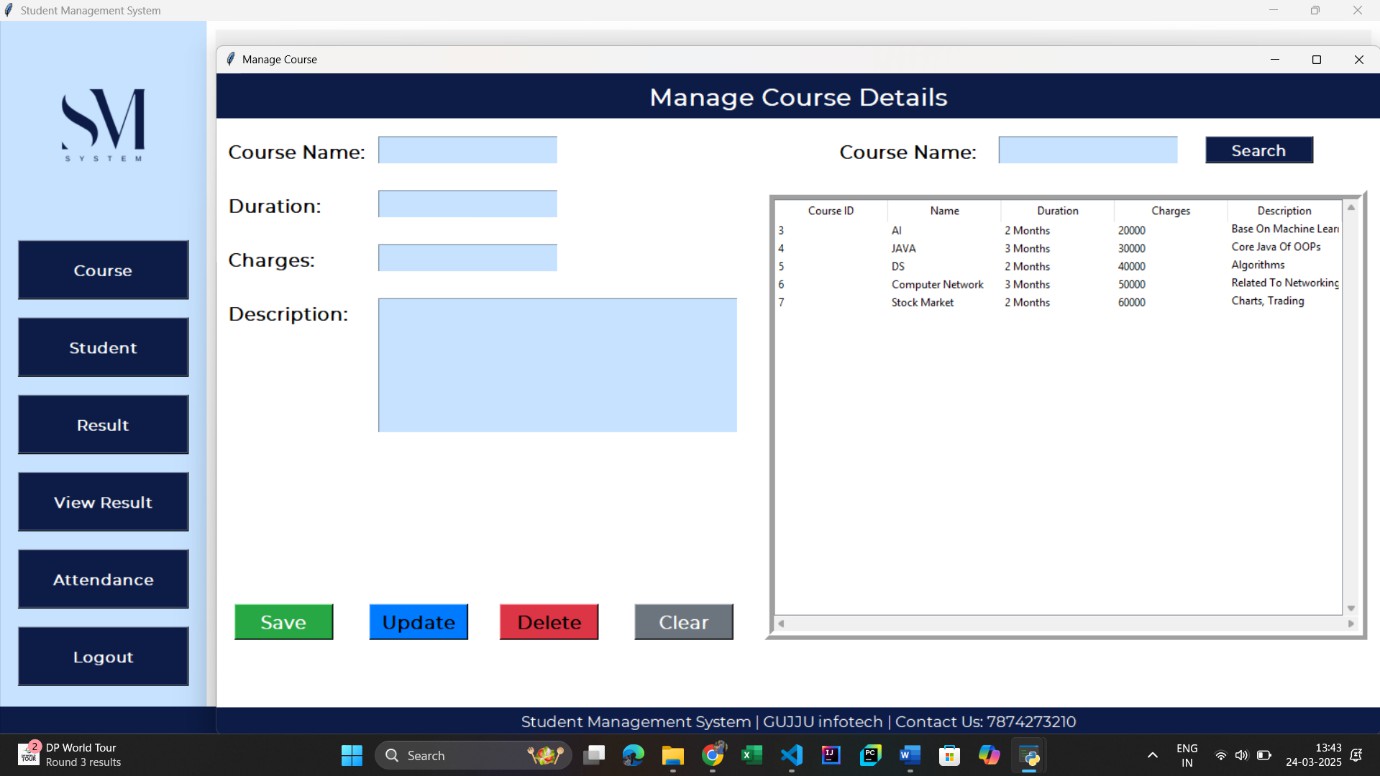
* Login Form



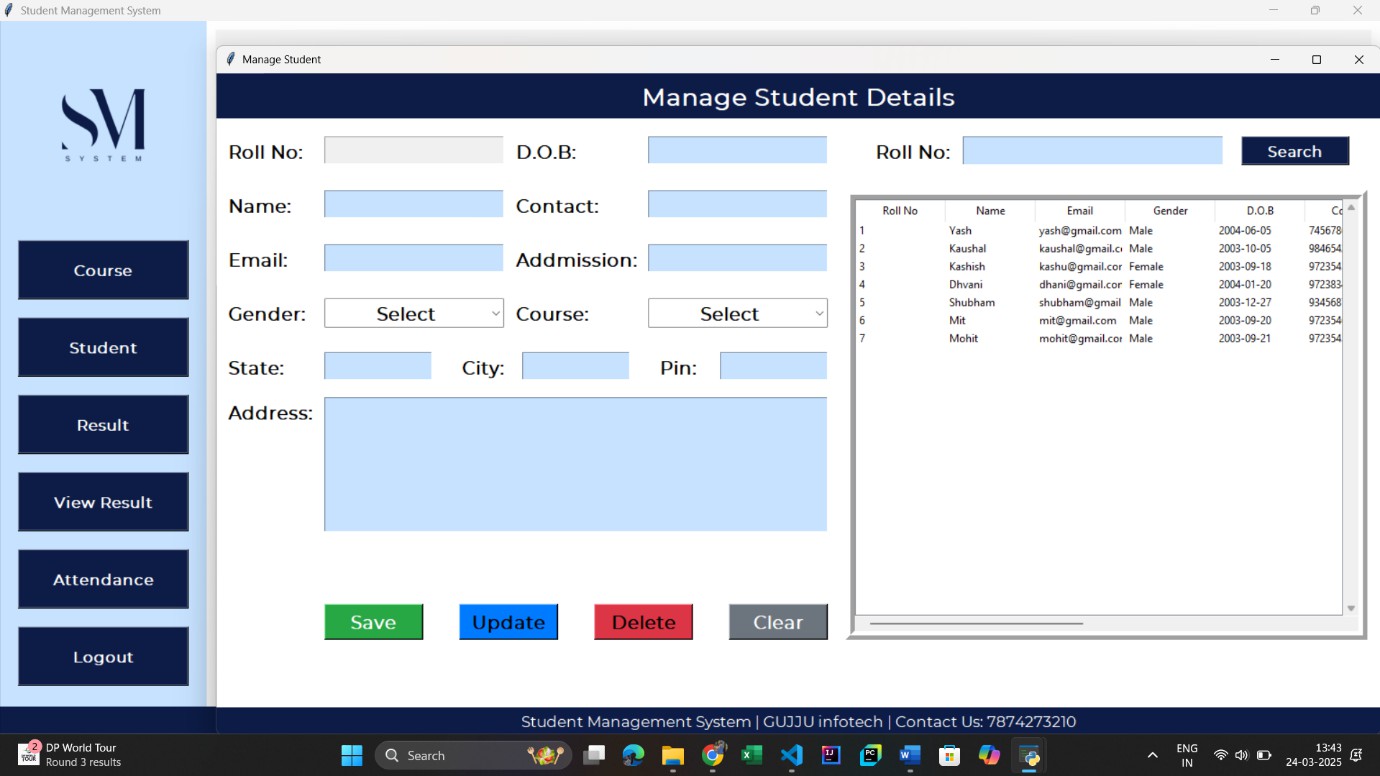
* Dash-Board Form



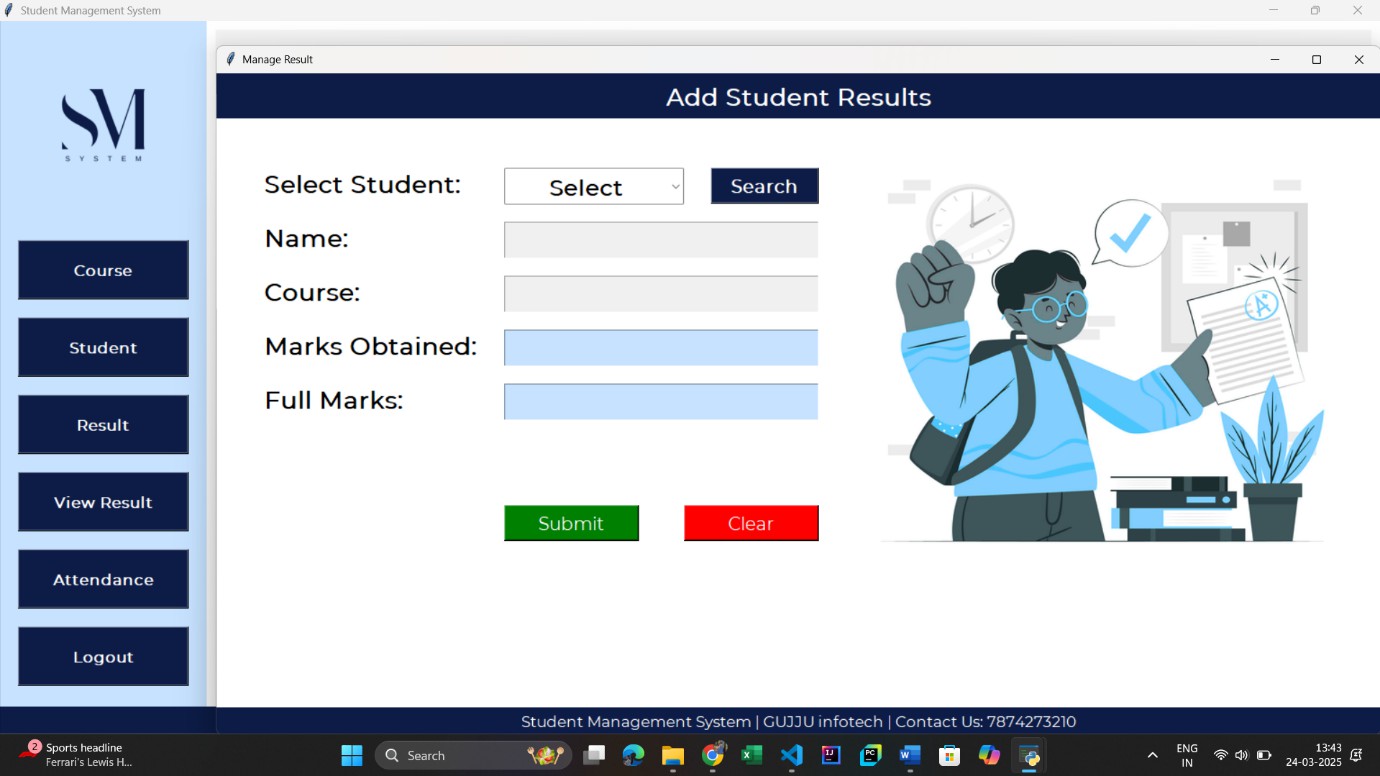
* Course Form



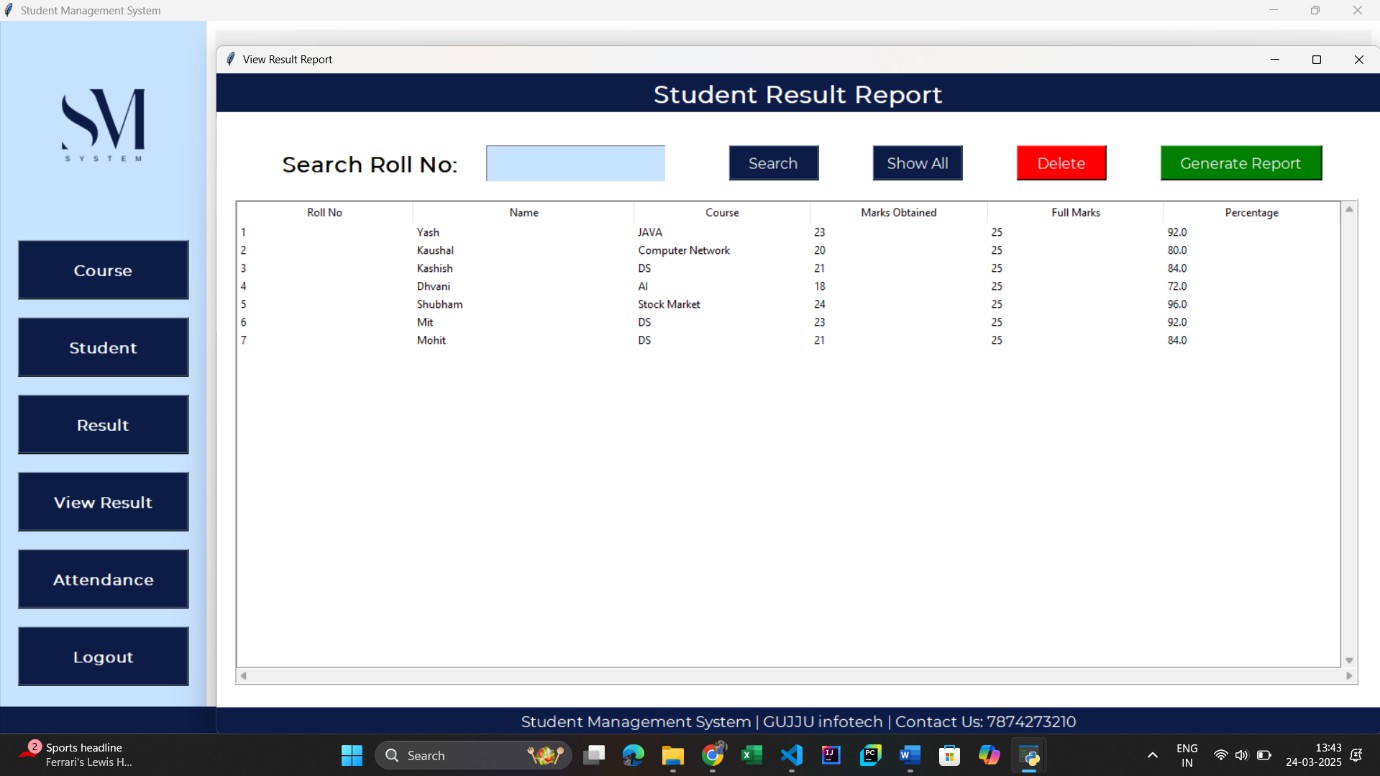
* Student Form



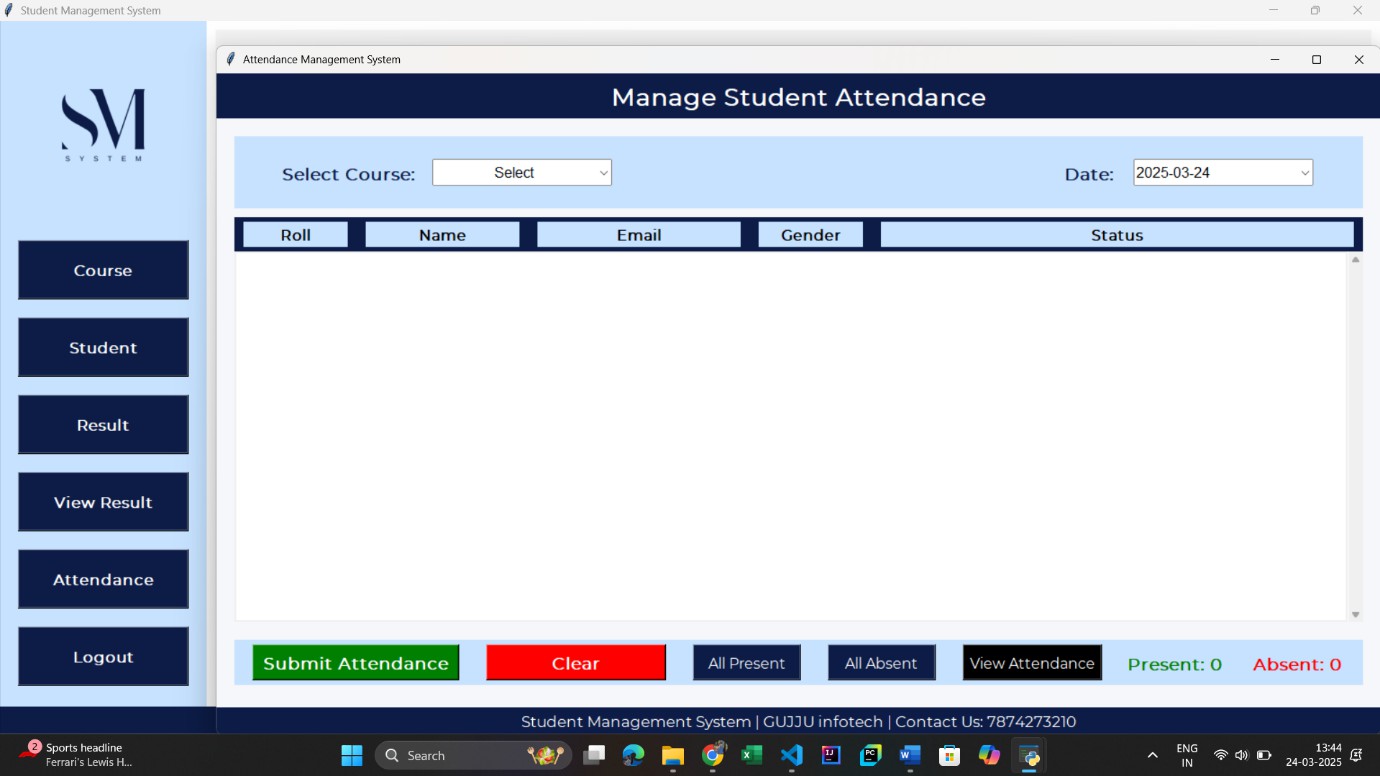
* Result Form



* View Result Form



* Attendance Form



# 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()

1. 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\_pasword = Entry(Frame1, font=("montserrat", 15), bg="lightgray") self.txt\_pasword.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)

1. 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)

1. 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)

1. 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="#ffcbd1", 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\_addmission = 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)

1. 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)

1. 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)

1. 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\_pasword.get() == "" or self.txt\_cpassword.get() == ""):

messagebox.showerror("Error", "All Fields Are Required", parent=self.root) elif self.txt\_pasword.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\_pasword.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)

1. 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)

1. 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.

# 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>