A

PROJECT REPORT ON

"Student Management System using PYTHON"

THE PARTIAL FULFILLMENT OF REQUIREMENT FOR THE AWARD OF THE DEGREE

OF

MASTER OF SCIENCE

(DATA SCIENCE AND MACHINE LEARNING), (2023-2024)

Under the supervision of:

Dr. Gopal Singh

Associate Professor, DCSA

MDU, Rohtak

Submitted by:

Gaurav

MSc (DS&ML) – 4th Sem

Roll No -23322

DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS

MAHARSHI DAYANAND UNIVERSITY, ROHTAK- 124001

Telephone: +91-1262- 393203, 393202 (O)

E-Mail: hod.computerscience@mdurohtak.ac.in

DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS

MAHARSHI DAYANAND UNIVERSITY ROHTAK

(www.mdurohtak.ac.in)

(NAAC Accredited 'A+' Grade Haryana State University)

Dr. Preeti Gulia

HEAD OF DEPARTMENT

No. MDU/DCS/22/__

Date: _____

CERTIFICATE-cum-DECLARATION

I, Gaurav hereby declare that the work presented in the Project Report titled "Student Management System Application" and submitted as Industry Internship/Project-I as part of MSc (DS&ML) 4th Semester to Department of Computer Science & Applications, M. D. University, Rohtak is an authentic record of my work carried out during the 4th semester (Jan - June, 2024) under the supervision of **Dr. Gopal Singh**, Associate **Professor**, Department of Computer Science & Applications.

Further, I also undertake that the matter embodied in this Project Report is my own work and has not been submitted by me or by any other candidate for the award of any other degree anywhere else.

Student Name –Gaurav

Roll No. – 23322

MSc(DS&ML) 4th Sem

Countersigned by Internal Supervisor

Name: Dr. Gopal Singh

Designation: Associate Professor,

DCSA

Forwarded by:

(Head of Department - Dr. Preeti Gulia)

ACKNOWLEDGEMENT

Acknowledgement is not mere formality but a genuine opportunity to thank all those people without that active support in this project would not be able to be possible I am thankful to my guide "**Dr. Gopal Singh (Associate PROF.), DCSA, MDU, ROHTAK**" for his valuable time, exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by his time to time shall carry me a long way in the journey of life on which I am about to embark.

I also take this opportunity to express a deep sense of gratitude to All faculty members of **Department of Computer Science & Applications, M.D.U, Rohtak** for their cordial support, valuable information and guidance, which helped me in completing this task through various stages.

Finally, I am thankful to the almighty God who has given me the power, good sense and confidence to complete my project successfully. I also thank my parents who were a constant source of encouragement. Their moral was indispensable.

Signature of the student
Gaurav
MSc(DS&ML)
4th sem
Roll no. 23128
Maharshi Dayanand University, Rohtak

INDEX

Chapter 1

Introduction to Project

1.1 Overview

The **Student Management Application** is a desktop-based system developed in **Python** using the **Tkinter** library for the GUI and **SQLite** for the backend database. It is designed to efficiently manage student information such as personal details, student records, course , result , and certificate . The application provides a user-friendly interface for administrators, faculty, and students to interact with the system securely and effectively. All data is stored locally using a lightweight SQLite database, ensuring fast and reliable performance without the need for external servers.

◆ Python

Python is a high-level, versatile programming language known for its simplicity and readability. It is widely used for software development, web applications, data analysis, automation, and more. Python's large library support makes it ideal for building desktop applications.

◆ Tkinter

Tkinter is Python's built-in GUI (Graphical User Interface) library. It allows developers to create interactive desktop applications with windows, buttons, labels, entry fields, and more. It's lightweight and easy to integrate with other Python modules.

◆ SQLite

SQLite is a lightweight, serverless, and self-contained SQL database engine. It is used to store structured data like student records, attendance logs, and fee details. It does not require installation or configuration and stores data in a single file.

◆ Other Tools/Libraries (Optional)

- **Pillow**: For image handling in Tkinter GUI (e.g., profile pictures, sliders).
- **ttk (Themed Tkinter)**: For better-looking widgets like buttons, dropdowns, and tables.
- **datetime**: To manage dates and times (e.g., attendance tracking).
- **os**: To manage file paths or local files.
- **tkcalendar**: For adding calendar widgets to select dates easily.
- **customtkinter** (if used): An extended library to create modern-looking GUIs with dark/light mode options.

1.2 Objectives

1. **To develop a user-friendly application** that simplifies the management of student data including personal details, academic records, attendance, and fees.

2. **To implement a secure and efficient system** using **Python** and **SQLite** that stores all student-related data in a structured and easily retrievable manner.

3. **To design an interactive GUI** using **Tkinter**, allowing users (admins or faculty) to easily perform operations like add, update, delete, and search student records.

4. **To provide separate modules** for different functionalities such as student registration, attendance tracking, fee management, timetable scheduling, and faculty profiles.

5. **To ensure data accuracy and integrity** by performing proper validations and error handling across the application.

6. **To allow easy maintenance and upgrades** through modular code design and use of open-source tools.

7. **To enhance administrative efficiency** by replacing traditional paper-based systems with a fast, digital solution.

1.3 System Architecture

The architecture of the Student Management System follows a **three-tier structure**:

---

1. Presentation Layer (User Interface)

- **Technology Used**: Tkinter
- This is the front end of the application where users interact with the system.
- Provides windows, forms, buttons, and input fields for data entry and operations.
- Example: Student registration form, attendance entry screen, fee status viewer.

---

2. Application Logic Layer (Business Logic)

- **Technology Used**: Python
- Handles all internal processing and logic between the GUI and the database.
- Validates user inputs, processes user commands (e.g., add, delete, update), and communicates with the database.
- Example: Checking if a student ID already exists before adding a new record.

---

3. Data Layer (Database)

- **Technology Used**: SQLite
- Stores all the application data such as student records, attendance logs, fee details, and timetable.
- Uses SQL queries to insert, retrieve, update, or delete data.
- Example: A table called students storing fields like name, roll number, course, etc.

System Workflow

1. User interacts with GUI (Tkinter form).
2. Python functions process the request and apply business logic.
3. Data is fetched from or saved to SQLite database.
4. Output or result is shown back on the GUI.

1.4 Features and Functionality

- Add, view, update, and delete student records.
- Manage student result.
- Record and monitor course and their details.
- Search students by name, roll number, or class.
- Create and manage new user (LogIn/Register).
- User-friendly GUI built with Tkinter and data stored securely using SQLite.

1.5 Implementation Challenges

During the project development, several challenges were encountered, including:

- Designing a clean and responsive GUI layout using Tkinter for multiple modules.
- Managing data integrity while performing operations like update and delete in SQLite.
- Implementing input validation to prevent incorrect or duplicate entries.
- Ensuring smooth navigation between different sections (e.g., student, course).
- Handling error messages and exceptions to avoid application crashes.

Chapter 2

Requirement Specification

2.1 Hardware Specification

Memory	2 GB minimum, 4 GB recommended
Screen resolution	1280x1024 or larger
Application window size	1024x680 or larger
Internet connection	Required

For User

The Hardware requirements for this system are as follows:

1) 2nd-generation Core i5 (2GHz+), 3rd/4th-generation Core i5 processor, or equivalent

2) Memory (RAM) 1 GB or above

3) Hard Disk space 1 GB for the database and the client software

4) Cache Memory 128 KB or above

2.2 Software Requirements

For development

- **Python (v3.8 or above)** – Required for writing and executing the application code.
- **Tkinter (built-in with Python)** – Used to create the graphical user interface (GUI).
- **SQLite** – For local database management to store student records, attendance, and fees.

For User :

- **Windows 10 or higher / Linux OS** – The application can run on these platforms.
- **Python (v3.8 or above)** – Python must be installed to run the application, though an executable version can also be provided.

Chapter 3
Feasibility Analysis

1.Technical Feasibility

- **Technologies Used**: Python, Tkinter, SQLite are all open-source and widely supported. Python is ideal for rapid development of GUI-based applications, and Tkinter is lightweight, making it suitable for simple desktop applications. SQLite offers a simple, file-based database for storing student data.
- **System Requirements**: The application requires minimal system resources. Python 3.8+ and Tkinter are pre-installed on most systems, ensuring that the application can be run on a wide range of computers without the need for additional setup.
- **Scalability**: While SQLite is a lightweight database, it may not be suitable for very large-scale implementations. However, for a medium-sized school or institution, it's an ideal choice. If needed, the system can be scaled to a more robust database in the future.

2. Economic Feasibility

- **Development Costs**: The application is being developed using open-source tools (Python, Tkinter, SQLite), which makes it cost-effective. No licensing fees are required for the tools, and the project can be completed without high initial investment.
- **Maintenance Costs**: As the system is built using open-source technologies, ongoing maintenance and updates will be minimal. The system will only need updates when changes are required for functionality or when there's a need to scale the system.

3. Operational Feasibility

- **User Accessibility**: The system is designed to be easy to use for administrative staff, faculty, and students. The Tkinter GUI will provide an intuitive interface that does not require technical expertise to operate.
- **User Training**: Since Tkinter interfaces are simple, minimal training will be required for end-users to operate the system effectively. Basic training can be provided on how to manage student records, attendance, and fees.

4. Legal Feasibility

- **Compliance**: The application does not require any specific licenses or legal agreements to use Python, Tkinter, or SQLite. However, data protection measures (such as encryption) should be considered to ensure compliance with privacy laws (e.g., GDPR for institutions in the EU).

Chapter 4
Technologies Used

Technologies Used in the Student Management Application

The **Student Management Application** leverages several powerful technologies that are widely used in the software development world. These tools and technologies were chosen for their reliability, ease of use, and compatibility with each other. By using Python, Tkinter, and SQLite, we ensure that the application is both lightweight and easy to maintain, while also providing a smooth and intuitive user experience.

---

1. Python (v3.8 or above)

- **Introduction**: Python is a high-level, interpreted programming language that emphasizes code readability and simplicity. It is known for its versatility and is used for various types of programming, including web development, data analysis, machine learning, and desktop applications.
- **Advantages**:
  - **Easy to Learn**: Python has a simple and readable syntax, making it ideal for both beginners and experienced developers.
  - **Cross-Platform**: Python code can run on various operating systems (Windows, Linux, macOS).
  - **Extensive Libraries**: Python offers a vast collection of libraries and frameworks to extend its functionality (e.g., Tkinter, Pillow, SQLite).
  - **Rapid Development**: Python allows for quick prototyping, speeding up the development process.
- **Disadvantages**:
  - **Slower Execution**: Python is an interpreted language, which may be slower compared to compiled languages like C or Java.
  - **Memory Consumption**: Python's memory consumption is higher, which could be an issue for large-scale applications or systems with limited resources.

---

2. Tkinter (GUI Library)

- **Introduction**: Tkinter is the standard Python library for creating graphical user interfaces (GUIs). It is a wrapper around the Tk GUI toolkit, which allows developers to create cross-platform applications with a native look and feel.
- **Advantages**:
  - **Built-In with Python**: Tkinter is included in Python's standard library, so no additional installation is needed.
  - **Simple to Use**: It provides a straightforward way to create windows, buttons, labels, and other interactive elements.
  - **Cross-Platform Compatibility**: Applications built with Tkinter run on Windows, macOS, and Linux without modification.
- **Disadvantages**:

- **Limited Features**: Compared to other GUI frameworks like PyQt or Kivy, Tkinter offers fewer advanced features and customization options.
- **Outdated Look**: The default appearance of Tkinter-based applications can seem a bit old-fashioned compared to modern GUI frameworks.

---

3. SQLite (Database)

- **Introduction**: SQLite is a lightweight, serverless, and self-contained SQL database engine. It stores data in a single file, making it perfect for applications that require an embedded database system.
- **Advantages**:
  - **Serverless**: SQLite does not require a separate server or complex configurations, making it easy to set up and use.
  - **Lightweight**: SQLite is small in size and fast, making it ideal for small to medium-sized applications.
  - **Cross-Platform**: SQLite databases are portable and can be used across different operating systems without modification.
  - **No Licensing Fees**: SQLite is public domain software and does not require any licensing.
- **Disadvantages**:
  - **Limited Scalability**: While SQLite is ideal for small to medium applications, it may not scale well for very large datasets or high-concurrency applications.
  - **Lacks Advanced Features**: Some advanced features, like stored procedures and triggers, are not as robust in SQLite as in larger database systems like MySQL or PostgreSQL.

---

4. Python Libraries (Pillow, datetime, os)

- **Introduction**: Several additional Python libraries are used to enhance the functionality of the Student Management Application, making it more efficient and user-friendly.
  - **Pillow**: A library used for opening, manipulating, and saving many different image file formats. It is used for handling student profile pictures.
  - **datetime**: A built-in library for managing and manipulating dates and times. It is essential for handling attendance and fee payment due dates.
  - **os**: A standard library that provides a way to interact with the operating system, allowing for file management and system operations.
- **Advantages**:
  - **Ease of Use**: These libraries are easy to integrate into the project and are well-documented.
  - **Flexibility**: These libraries provide flexible ways to handle images, dates, and files.
  - **Reliability**: They are widely used and tested, ensuring stable functionality.

- **Disadvantages**:
  - **Limited to Basic Operations**: While the libraries provide the basic functionality needed, advanced features may require additional packages or third-party libraries.
  - **Performance**: For large datasets or complex tasks, these libraries may not perform as well as more specialized solutions.

These technologies were chosen to ensure the **Student Management Application** is simple to develop, easy to maintain, and provides a smooth user experience. They offer the right balance of simplicity, power, and scalability for the project's needs.

Chapter 5

Code and Screenshot - Frontend

5.1 Frontend

Entry point for all the components that will be render on our application. This file exists in Public folder of our -log_in.py file.

1. Application Theme & Colors

- **Primary Colors**: White (`#FFFFFF`), Light Gray (`#ECE8DD`)
- **Font**: Use **"Goudy old style "** or **"times new roman"**, bold headings

---

2. UI Structure

Here's a modular layout suggestion for your project:

A. Login Window

- University logo and welcome message
- **Username & Password** fields
- Eye icon to toggle password visibility
- Buttons: **Login**, **Register**, **Forgot Password**

B. Registration Window

- Fields: Name, Email, Phone, Password, Confirm Password, Security Question
- Validations: All fields required, confirm password , check box
- Button: **Register Now, LogIn**

C. Dashboard

- Menu bar with buttons:
  - Course
  - Student
  - Add Result
  - View Result
  - Log Out
  - Exit
- Department label
- Use images/icons with buttons (via `PIL.ImageTk`)

D. Student Management Window

- Form: Name, Roll No, Course, Year, Contact, Email
- Buttons: Add, Update, Delete, Clear , Search, Export
- Treeview to display all records in a table format

E. Course Management Window

- View course data .
- Export data.

- Add/Update/Delete/Clear buttons.

F. Add Student Result Window

- Student search bar
- Add marks field
- Add/Clear button

F. View Student Result Window

- Student search bar
- Delete/Clear/ Export/ Show Certificate buttons
- Table view for showing the student result.

F. Log Out Button

- For log out user.

F. Exit

- Exit/ Close the window.

---

3. Useful Widgets

- `LabelFrame`, `Treeview`, `Combobox`, `Messagebox`, `Canvas`
- `Progressbar` (for loading or processing animation)
- `Toplevel` (for pop-ups like "Reset Password" or "Add Record")

🖼️ 4. Visual Enhancements

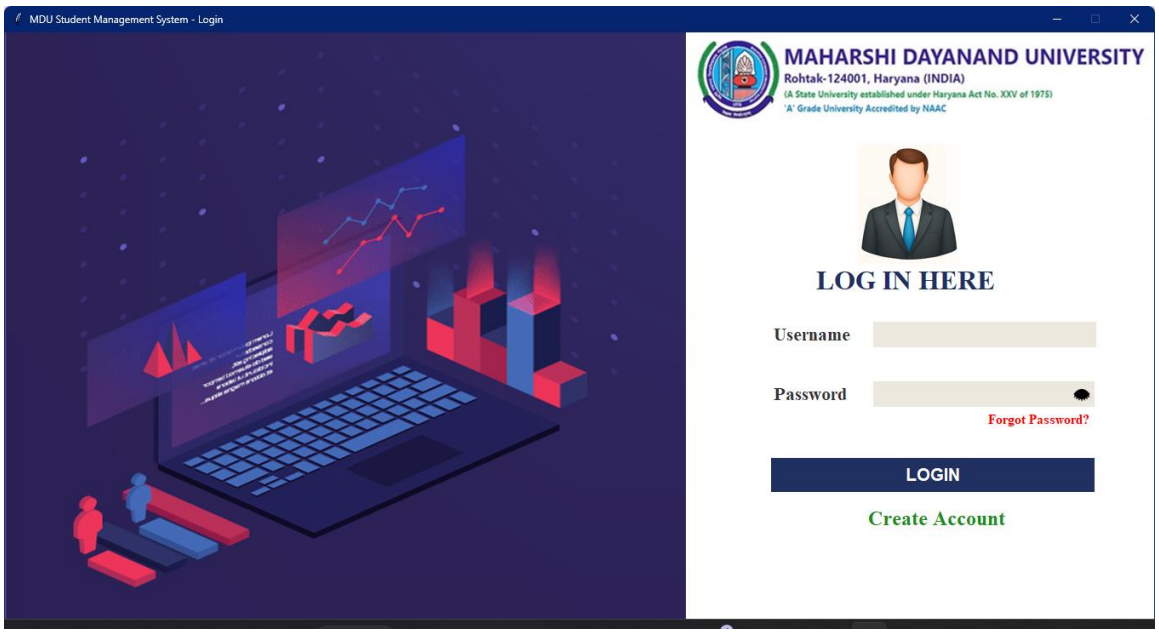
- Use `.png` or `.jpg` images for:
  - Add/Update/Delete/Clear /Export/ Show certificate buttons
  - Admin logo images
  - Institution branding/logo
- Custom button hover effects (use `bind()` for color change)
- Animated image slider (optional)

💡 Tip:

Use `ttk.Style()` to customize widget themes across the app for a professional appearance.

5.2 Components(pages) of the Application -

1. Login Page-



Code-

```
from tkinter import *
from PIL import Image, ImageTk
from tkinter import ttk, messagebox
import sqlite3
import os
import cv2

class Login:
    def __init__(self, root):
        self.root = root
        self.root.title("MDU Student Management System - Login")
        self.root.geometry("1350x700+0+0")
        self.root.resizable(False, False)
        self.root.config(bg="#CFEAF2")
```

```
main_bg = "img/bg2.png"
self.bg_img = Image.open(main_bg)
self.bg_img = self.bg_img.resize((1350, 700))  # Resize if needed
self.bg_img = ImageTk.PhotoImage(self.bg_img)
self.bg_label = Label(self.root, image=self.bg_img).place(
    x=0, y=0, relheight=1, relwidth=1
)

# ===== Login Frame =====
self.frame = Frame(self.root, bg="white", bd=0)
self.frame.place(x=800, y=0, width=600, height=700)

# Title
Label(
    self.frame,
    text="LOG IN HERE",
    font=("Times New Roman", 24, "bold"),
    fg="#203061",
    bg="white",
).place(x=150, y=270)

vc_pic1 = "img/mdu__logo.jpg"  # Replace with your new icon
img1 = Image.open(vc_pic1).resize((560, 103))
self.vc_img = ImageTk.PhotoImage(img1)
Label(self.frame, image=self.vc_img, bg="white").place(x=0, y=0)
# User Icon
admin_pic = "img/admin_log_in.png"  # Replace with your new icon
img = Image.open(admin_pic).resize((200 - 80, 220 - 80))
self.admin_img = ImageTk.PhotoImage(img)
Label(self.frame, image=self.admin_img, bg="white").place(x=200, y=130)

# Username Label & Entry
Label(
    self.frame,
    text="Username",
    font=("Times New Roman", 16, "bold"),
    fg="#333333",
    bg="white",
).place(x=100, y=300 + 40)
```

```
self.username_entry = Entry(
    self.frame,
    font=("Times New Roman", 15),
    bg="#ECE8DD",
    fg="#333333",
    bd=0,
    width=26,
)
self.username_entry.place(x=220, y=300 + 40, height=30)

# Password Label & Entry

Label(
    self.frame,
    text="Password",
    font=("Times New Roman", 16, "bold"),
    fg="#333333",
    bg="white",
).place(x=100, y=410)

self.password_entry = Entry(
    self.frame,
    font=("Times New Roman", 15),
    bg="#ECE8DD",
    fg="#333333",
    bd=0,
    show="*",
    width=25,
)
self.password_entry.place(x=220, y=410, height=30)

# Load Eye Images
eye_open = Image.open("img/open_eye.png").resize((20, 20))
eye_closed = Image.open("img/close_eye.png").resize((20, 20))
self.eye_open_icon = ImageTk.PhotoImage(eye_open)
self.eye_closed_icon = ImageTk.PhotoImage(eye_closed)

self.show_password = False
self.toggle_btn = Button(
```

```
    self.frame,
    image=self.eye_closed_icon,
    bg="#ECE8DD",
    bd=0,
    activebackground="#ECE8DD",
    cursor="hand2",
    command=self.toggle_password,
)
self.toggle_btn.place(x=450, y=410, height=30, width=30)

# Login Button
self.login_btn = Button(
    self.frame,
    text="LOGIN",
    font=("Arial", 16, "bold"),
    fg="white",
    bg="#203061",
    bd=0,
    cursor="hand2",
    command=self.log_in,
)
self.login_btn.place(x=100, y=500, width=380, height=40)



# Forgot Password
Button(
    self.frame,
    text="Forgot Password?",
    font=("Times New Roman", 12, "bold"),
    fg="red",
    bg="white",
    bd=0,
    cursor="hand2",
    command=self.forget_password,
).place(x=350, y=410 + 30)

# Create Account
Button(
```

```
            self.frame,
            text=" Create Account",  # ←
            font=("Times New Roman", 18, "bold"),
            fg="#228B22",
            bg="white",
            bd=0,
            cursor="hand2",
            command=self.register_window,
        ).place(x=200, y=550)

    def toggle_password(self):
        if self.show_password:
            self.password_entry.config(show="*")
            self.toggle_btn.config(image=self.eye_closed_icon)
            self.show_password = False
        else:
            self.password_entry.config(show="")
            self.toggle_btn.config(image=self.eye_open_icon)
            self.show_password = True

    def register_window(self):
        self.root.destroy()
        os.system("python register.py")

    def log_in(self):
        username = self.username_entry.get().strip()
        password = self.password_entry.get().strip()

        if not username or not password:
            messagebox.showerror("Error", "All fields are required", parent=self.root)
        elif len(password) < 8:
            messagebox.showerror(
                "Error", "Password must be at least 8 characters long", parent=self.root
            )
        else:
            try:
                con = sqlite3.connect("srms.db")
                cur = con.cursor()
                cur.execute(
```

```
            "SELECT * FROM employee WHERE email=? AND password=?",
            (username, password),
        )
        row = cur.fetchone()
        con.close()
        if row is None:
            messagebox.showerror(
                "Error", "Invalid username or password", parent=self.root
            )
        else:
            messagebox.showinfo(
                "Success", "Log In Successful", parent=self.root
            )
            self.show_processing()
    except Exception as e:
        messagebox.showerror("Error", f"Error: {e}", parent=self.root)

def show_processing(self):
    self.process_win = Toplevel(self.root)
    self.process_win.title("Processing...")
    self.process_win.geometry("350x150+500+300")
    self.process_win.config(bg="white")
    self.process_win.grab_set()

    Label(
        self.process_win,
        text="Please wait...",
        font=("Times New Roman", 15, "bold"),
        fg="#228B22",
        bg="white",
    ).pack(pady=10)

    progress = ttk.Progressbar(
        self.process_win, orient=HORIZONTAL, length=250, mode="determinate"
    )
    progress.pack(pady=10)
    progress.start(10)

    self.root.after(3000, self.open_dashboard)
```

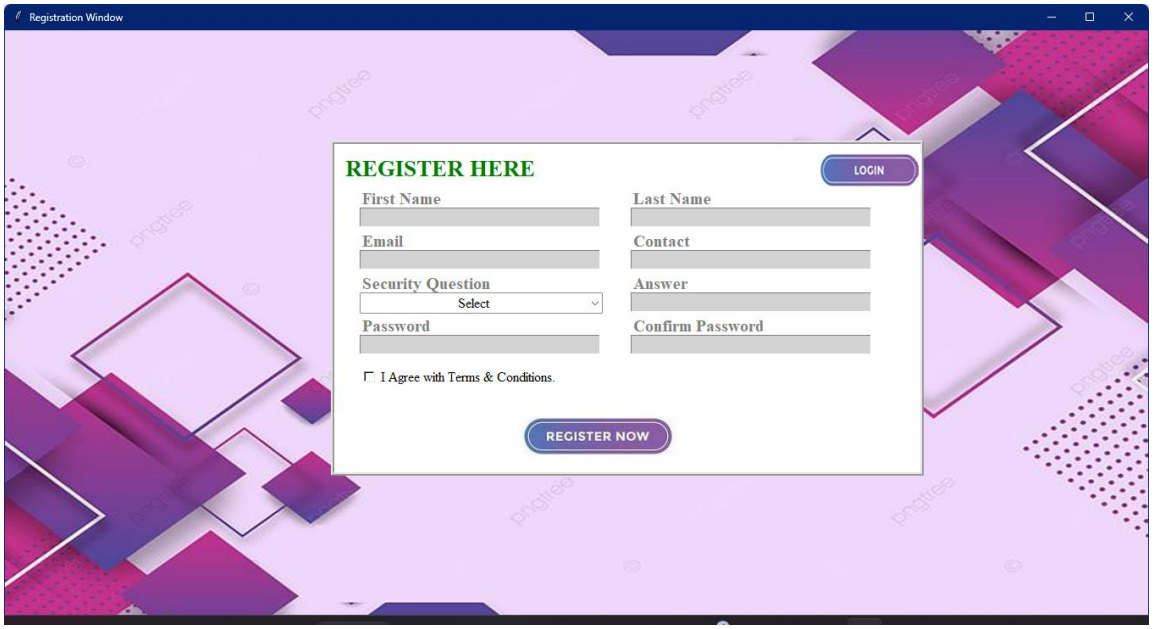
```
def open_dashboard(self):
    self.process_win.destroy()
    self.root.destroy()
    os.system("python dashboard.py")

def forget_password(self):
    # ------- Reset Window -------
    reset_win = Toplevel(self.root)
    reset_win.title("Reset Password")
    reset_win.geometry("400x350+500+200")
    reset_win.config(bg="white")
    reset_win.grab_set()



    Label(
        reset_win,
        text="Forget Password",
        font=("Times New Roman", 20, "bold"),
        fg="#228B22",
        bg="white",
    ).place(x=110, y=20)

    # Email
    Label(
        reset_win, text="Email", font=("Times New Roman", 15), fg="gray", bg="white"
    ).place(x=50, y=80)
    email_entry = Entry(
        reset_win, font=("Times New Roman", 15), bg="#ECE8DD", bd=0, width=30
    )
    email_entry.place(x=50, y=110, height=30)

    # New Password
    Label(
        reset_win,
        text="New Password",
        font=("Times New Roman", 15),
        fg="gray",
```

```
        bg="white",
    ).place(x=50, y=150)
    new_pass_entry = Entry(
        reset_win,
        font=("Times New Roman", 15),
        bg="#ECE8DD",
        bd=0,
        show="*",
        width=30,
)
    new_pass_entry.place(x=50, y=180, height=30)

    # Confirm Password
    Label(
        reset_win,
        text="Confirm Password",
        font=("Times New Roman", 15),
        fg="gray",
        bg="white",
    ).place(x=50, y=220)
    confirm_pass_entry = Entry(
        reset_win,
        font=("Times New Roman", 15),
        bg="#ECE8DD",
        bd=0,
        show="*",
        width=30,
    )
    confirm_pass_entry.place(x=50, y=250, height=30)

    # Reset Button
    Button(
        reset_win,
        text="Reset Password",
        font=("Arial", 15, "bold"),
        bg="#228B22",
        fg="white",
        bd=0,
        cursor="hand2",
```

```
        command=lambda: self.reset_password(
            email_entry, new_pass_entry, confirm_pass_entry, reset_win
        ),
    ).place(x=120, y=300, width=160, height=40)

def reset_password(self, email_entry, new_pass_entry, confirm_pass_entry, win):
    email = email_entry.get().strip()
    new_pass = new_pass_entry.get().strip()
    confirm_pass = confirm_pass_entry.get().strip()

    if not email or not new_pass or not confirm_pass:
        messagebox.showerror("Error", "All fields are required", parent=win)
    elif len(new_pass) < 8:
        messagebox.showerror(
            "Error", "Password must be at least 8 characters long", parent=win
        )
    elif new_pass != confirm_pass:
        messagebox.showerror("Error", "Passwords do not match", parent=win)
    else:
        try:
            con = sqlite3.connect("srms.db")
            cur = con.cursor()
            cur.execute("SELECT * FROM employee WHERE email=?", (email,))
            row = cur.fetchone()
            if row is None:
                messagebox.showerror("Error", "Email not found", parent=win)
            else:
                cur.execute(
                    "UPDATE employee SET password=? WHERE email=?",
                    (new_pass, email),
                )
                con.commit()
                messagebox.showinfo(
                    "Success", "Password reset successfully!", parent=win
                )
                win.destroy()
            con.close()
        except Exception as e:
            messagebox.showerror("Error", f"Error: {e}", parent=win)
```

```
if __name__ == "__main__":
    root = Tk()
    app = Login(root)
    root.mainloop()
```

2. Sign Up Page-



Code-

```
from tkinter import *
from PIL import Image, ImageTk
from tkinter import ttk
from tkinter import messagebox
import sqlite3
import os


class Register:
    def __init__(self, root):
```

```
self.root = root
self.root.title("Registration Window")
self.root.geometry("1350x700+0+0")
# =======Background Image ===============
main_bg = "img/bg_for_register.jpg"
self.bg_img = Image.open(main_bg)
self.bg_img = self.bg_img.resize((1350, 700))  # Resize if needed
self.bg_img = ImageTk.PhotoImage(self.bg_img)
self.bg_label = Label(self.root, image=self.bg_img).place(
   x=0, y=0, relheight=1, relwidth=1
)

# ======login frame==========
frame1 = Frame(self.root, bg="white", bd=4, relief=RIDGE)
frame1.place(x=385, y=130, height=395, width=700)

title = Label(
   frame1,
   text="REGISTER HERE",
   font=("times new roman", 20, "bold"),
   bg="white",
   fg="green",
).place(x=10, y=10)
# =========variables====================

# ======first name ,label,entry==============
first_name = Label(
   frame1,
   text="First Name",
   font=("times new roman", 15, "bold"),
   bg="white",
   fg="gray",
).place(x=30, y=50)
self.first_name_entry = Entry(
   frame1, bg="lightgray", font=("times new roman", 12), width=35
)
```

```
self.first_name_entry.place(x=30, y=75)

# ========last name, entry,lable,entry=========

last_name = Label(
   frame1,
   text="Last Name",
   font=("times new roman", 15, "bold"),
   bg="white",
   fg="gray",
).place(x=350, y=50)
self.last_name_entry = Entry(
   frame1, bg="lightgray", width=35, font=("times new roman", 12)
)
self.last_name_entry.place(x=350, y=75)

# ======email,label,entry===============
email = Label(
   frame1,
   text="Email",
   font=("times new roman", 15, "bold"),
   bg="white",
   fg="gray",
).place(x=30, y=100)
self.email_entry = Entry(
   frame1, bg="lightgray", font=("times new roman", 12), width=35
)
self.email_entry.place(x=30, y=125)

# ========contact, entry,lable,entry=========

contact = Label(
   frame1,
   text="Contact",
   font=("times new roman", 15, "bold"),
   bg="white",
```

```
    fg="gray",
).place(x=350, y=100)
self.contact_entry = Entry(
    frame1, bg="lightgray", width=35, font=("times new roman", 12)
)
self.contact_entry.place(x=350, y=125)

# ======question,label,entry===============
question = Label(
    frame1,
    text="Security Question",
    font=("times new roman", 15, "bold"),
    bg="white",
    fg="gray",
).place(x=30, y=150)
self.question_entry = ttk.Combobox(
    frame1,
    font=("times new roman", 12),
    width=33,
    state="readonly",
    justify=CENTER,
)
self.question_entry["values"] = (
    "Select",
    "Your Best Friend Name",
    "Your School Name",
    "Your Birth Place",
)
self.question_entry.current(0)
self.question_entry.place(x=30, y=175)

# ========answer, entry,lable,entry=========

answer = Label(
    frame1,
    text="Answer",
```

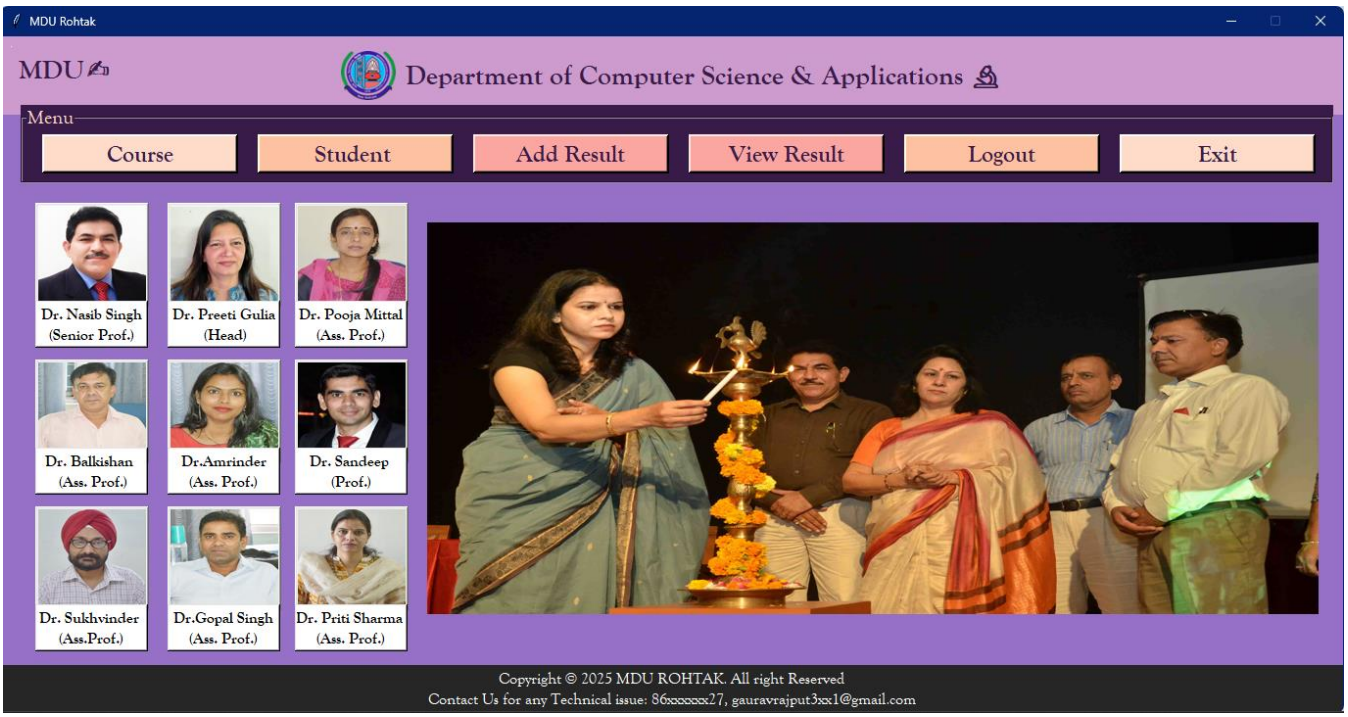
```
    font=("times new roman", 15, "bold"),
    bg="white",
    fg="gray",
).place(x=350, y=150)
self.answer_entry = Entry(
    frame1, bg="lightgray", width=35, font=("times new roman", 12)
)
self.answer_entry.place(x=350, y=175)

# ======password,label,entry===============
password = Label(
    frame1,

    text="Password",

    font=("times new roman", 15, "bold"),
    bg="white",
    fg="gray",
).place(x=30, y=200)
self.password_entry = Entry(
    frame1, show="*",bg="lightgray", font=("times new roman", 12), width=35
)
self.password_entry.place(x=30, y=225)

# ========confirm_pass, entry,lable,entry=========

confirm_pass = Label(
    frame1,
    text="Confirm Password",
    font=("times new roman", 15, "bold"),
    bg="white",
    fg="gray",
).place(x=350, y=200)
self.confirm_pass_entry = Entry(
    frame1, bg="lightgray", width=35, font=("times new roman", 12)
)
```

```
        self.confirm_pass_entry.place(x=350, y=225)

        # ==========checkbox for terms and conditions============
        self.var_term=IntVar()
        self.term = Checkbutton(
            frame1,
            text="I Agree with Terms & Conditions.",
            variable=self.var_term,
            onvalue=1,
            offvalue=0,
            bg="white",
            font=("times new roman", 12),
        ).place(x=30, y=260)

        # =======Register button ============================
        self.btn_img = ImageTk.PhotoImage(file="img/rg2.jpg")
        self.register_btn = Button(
            frame1,
            image=self.btn_img,
            bd=0,
            bg="white",
            cursor="hand2",
            command=self.register_data,
        ).place(x=220, y=320)

        self.login_btn = ImageTk.PhotoImage(file="img/login_btn.png")
        self.btn2 = Button(
            frame1, image=self.login_btn, bd=0, bg="white", cursor="hand2",
command=self.login_window
        ).place(x=570, y=10, height=40)

    def clear(self):
        self.first_name_entry.delete(0,END)
        self.last_name_entry.delete(0,END)
        self.email_entry.delete(0,END)
        self.contact_entry.delete(0,END)
        self.question_entry.current(0)
```

```
        self.answer_entry.delete(0,END)
        self.password_entry.delete(0,END)
        self.confirm_pass_entry.delete(0,END)


    def login_window(self):
        root.destroy()
        os.system("python log_in.py")


    def register_data(self):
        if self.first_name_entry.get()=="" or self.email_entry.get()=="" or
self.contact_entry.get()=="" or self.question_entry.get()=="Select" or
self.answer_entry.get()=="" or self.password_entry.get()==" " or
self.confirm_pass_entry.get()=="":
            messagebox.showerror("Error", "All fields required", parent=self.root)
        elif len(self.password_entry.get()) < 8:
            messagebox.showerror("Error", "Password must be at least 8 characters long",
parent=self.root)
        elif self.confirm_pass_entry.get() != self.password_entry.get():
            messagebox.showerror("Error", "Password and Confirm Password should be
same", parent=self.root)
        elif self.var_term.get()==0:
            messagebox.showerror("Error", "Please agree our term & conditions ")
        else:
            try:
                con = sqlite3.connect(database="srms.db")
                cur = con.cursor()
                cur.execute("select * from employee where email=?",(self.email_entry.get(),))
                row=cur.fetchone()
                if row!=None:

                    messagebox.showerror("Error", "User already exist, Please try with another
email !",parent=self.root)
                else:
                    cur.execute("insert into employee (
f_name,l_name,email,contact,question,answer,password) values(?,?,?,?,?,?,?)",
```

```
                (
                    self.first_name_entry.get(),
                    self.last_name_entry.get(),
                    self.email_entry.get(),
                    self.contact_entry.get(),
                    self.question_entry.get(),
                    self.answer_entry.get(),
                    self.password_entry.get()
                ))
            con.commit()
            con.close()
            messagebox.showinfo("Success", "Register Successfully")
            self.clear()
        except Exception as es:
            messagebox.showerror("Error",f"Error due to : {str(es)}",parent=self.root)

root = Tk()
obj = Register(root)
root.mainloop()
```

3. Dashboard window -



Code-

```
from tkinter import *
from PIL import Image, ImageTk
import tkinter as tk
from tkinter import ttk, messagebox
from itertools import cycle
from course import Course  # importing the course file for course window
from student import Student_cls  # importing the student file for student window
from result import result_cls  # importing the result file for result window
from view_result import view_result_cls  # importing the view result file for show
import os



class RMS:
    def __init__(self, root):
        self.root = root
        self.root.title("MDU Rohtak")
        self.root.geometry("1370x690+-10+0")
        self.root.config(bg="#9670c7")
        self.root.resizable(0, 0)
```

```
# ===icons===
self.logo_dash = ImageTk.PhotoImage(file="img/mdu_logo.png")


# ====title===
title = Label(
    self.root,
    text="Department of Computer Science & Applications 🔬",
    padx=10,
    compound=LEFT,
    image=self.logo_dash,
    font=("goudy old style", 21, "bold"),
    # bg="#FEC1A2",
    bg="#ce9bce",
    # fg="#6E3E70",
    fg="#391b49",
).place(x=0, y=0, relwidth=1, height=80)


lbl = Label(
    title,
    text="MDU ✍️",
```

```
    font=("goudy old style", 20, "bold"),
    bg="#ce9bce",
    fg="#391b49",
).place(x=15, y=15)
# =====menu+===
menu_frame = LabelFrame(
    self.root,
    text="Menu",
    font=("goudy old style", 15),
    bg="#391b49",
    fg="#FFDCC9",
)
menu_frame.place(x=20, y=70, width=1340, height=80)
# bg="#6E3E70"

# ===buttons===

course_btn = Button(
    menu_frame,
    text=" Course ",
    font=("goudy old style", 18, "bold"),
```

```
    bg="#FFDCC9",
    fg="#391b49",
    cursor="hand2",
    justify=CENTER,
    bd=3,
    relief=RAISED,
    #    activebackground="red",
    command=self.add_course,
).place(x=20, y=5, width=200, height=40)
student_btn = Button(
    menu_frame,
    text="Student ",
    font=("goudy old style", 18, "bold"),
    bg="#FEC1A2",
    fg="#391b49",
    cursor="hand2",
    bd=3,
    relief=RAISED,
    command=self.add_student,
).place(x=240, y=5, width=200, height=40)
```

```
result_btn = Button(
    menu_frame,
    text="Add Result",
    font=("goudy old style", 18, "bold"),
    bg="#FBA6A1",
    fg="#391b49",
    cursor="hand2",
    bd=3,
    relief=RAISED,
    command=self.add_result,
).place(x=460, y=5, width=200, height=40)
view_btn = Button(
    menu_frame,
    text="View Result",
    font=("goudy old style", 18, "bold"),
    bg="#FBA6A1",
    fg="#391b49",
    cursor="hand2",
    bd=3,
    relief=RAISED,
    command=self.show_result,
```

```
).place(x=680, y=5, width=200, height=40)
logout_btn = Button(
    menu_frame,
    text="Logout",
    font=("goudy old style", 18, "bold"),
    bg="#FEC1A2",
    fg="#391b49",
    cursor="hand2",
    bd=3,
    command=self.logout,
    relief=RAISED,
).place(x=900, y=5, width=200, height=40)
exit_btn = Button(
    menu_frame,
    text="Exit",
    font=("goudy old style", 18, "bold"),
    bg="#FFDCC9",
    fg="#391b49",
    cursor="hand2",
    bd=3,
    relief=RAISED,
```

```
        command=self.exit_btn,
    ).place(x=1120, y=5, width=200, height=40)

    # ===Footer===
    footer = Label(
        self.root,
        text="Copyright © 2025 MDU ROHTAK. All right Reserved\nContact Us for any Technical issue: 86xxxxxx27, gauravrajput3xx1@gmail.com",
        font=("goudy old style", 12),
        bg="#262626",
        fg="white",
    )
    footer.pack(side=BOTTOM, fill=X)

    # ================ Slider Section =================
    self.slider_images = [
        # "img/dept_1.jpg",
        "img/2nd.png",
        # "img/3rd.png",
        # "img/4th.png",
        "img/6th.png",
```

```
    # "img/dept_2.jpg",
    # "img/dept_3.jpg",
    # "img/dept_4.jpg",
    # "img/dept_6.jpg",
    "img/dept_7.jpg",
    # "img/dept_8.jpg",
    "img/dept_10.jpg",
    "img/activity_center.jpg",
    "img/final.jpg",
    "img/secratery.JPG",
    "img/library_wallpaper.jpg",
]  # List of images
self.current_image_index = 0  # Start with the first image


# Load and display the first image
self.bg_img = Image.open(self.slider_images[self.current_image_index])
self.bg_img = self.bg_img.resize((950, 400))  # Resize if needed
self.bg_img = ImageTk.PhotoImage(self.bg_img)
self.bg_label = Label(self.root, image=self.bg_img)
self.bg_label.place(x=435, y=190, height=400, width=910)
```

```
        # Start the slider
        self.update_slider()
        # -----------image section frame-------------------------
        img_frame = Frame(self.root, bd=2, relief=RIDGE).place(x=10, y=10)


        # ---------------------- right frame images section -------------------------
        self.gill_img = Image.open("img/gill.jpg")
        self.gill_img = self.gill_img.resize((110, 100))  # Resize if needed
        self.gill_img = ImageTk.PhotoImage(self.gill_img)


        self.gill_label1 = Button(
            self.root,
            image=self.gill_img,
            bd=2,
            relief=RAISED,
            command=lambda: self.show_profile_card(
                name="Dr. Nasib Sing Gill",
                image=self.gill_img,
                description="Designation : Professor, Department of Computer Science & Applications\n - Director, Centre for Distance and Online Education\n -Director, Digital Learning Centre\n -Nodal Officer, Academic Bank of Credits",
                extra_info="Email: nasibsgill@gmail.com, nasib.gill@mdurohtak.ac.in\nPhone:
```

```
+91-1262-293203\nMobile: +91-9050805136",
        ),
    )
    self.gill_label1.place(x=35, y=170)

    # self.gill_label1 = Label(self.root, image=self.gill_img, bd=2, relief=RAISED)
    # self.gill_label1.place(x=35, y=170)
    lbl_gill = Label(
        img_frame,
        text="Dr. Nasib Singh\n(Senior Prof.)",
        font=("goudy old style", 12, "bold"),
        bg="white",
        bd=2,
        relief=RAISED,
    ).place(x=35, y=270, width=115)
    # ----------------------------------------------------------------------

    self.gulliya_img = Image.open("img/gulliya.jpeg")
    self.gulliya_img = self.gulliya_img.resize((110, 100))  # Resize if needed
    self.gulliya_img = ImageTk.PhotoImage(self.gulliya_img)
    self.gulliya_label = Button(
```

```
            self.root,
            image=self.gulliya_img,
            bd=2,
            relief=RAISED,
            command=lambda: self.show_profile_card(
                name="Dr. Preeti Guliya",
                image=self.gulliya_img,
                description="Designation : Head of Department\n -Professor, Department of Computer  Science & Applications\n -Deputy Director, Centre for Distance and Online Education\n ",
                extra_info="Email: mdurohtak.ac.in\nPhone: +91-98xxxxxxx1\nRoom No: 101, CS Block",
            ),
        )
        self.gulliya_label.place(x=170, y=170)
        lbl_gulliya = Label(
            img_frame,
            text="Dr. Preeti Gulia\n (Head)",
            font=("goudy old style", 12, "bold"),
            bg="white",
            bd=2,
            relief=RAISED,
```

```
        ).place(x=170, y=270, width=115)

        # ---------------------------------------------------------------------

        self.pooja_img = Image.open("img/pooja.jpeg")

        self.pooja_img = self.pooja_img.resize((110, 100))  # Resize if needed

        self.pooja_img = ImageTk.PhotoImage(self.pooja_img)

        self.pooja_label = Button(

            self.root,

            image=self.pooja_img,

            bd=2,

            relief=RAISED,

            command=lambda: self.show_profile_card(

                name="Dr. Pooja Mittal",

                image=self.pooja_img,

                description="Designation : Assistant Professor, Department of Computer Science
& Applications",

                extra_info="Phone: +91-9468110000\nRoom No: 102, CS Block",

            ),

        )

        self.pooja_label.place(x=300, y=170)

        lbl_pooja = Label(

            img_frame,
```

```
        text="Dr. Pooja Mittal\n(Ass. Prof.)",
        font=("goudy old style", 12, "bold"),
        bg="white",
        bd=2,
        relief=RAISED,
    ).place(x=300, y=270, width=115)
    # ---------------------------------------------------------------------


    self.balu_img = Image.open("img/balu.jpg")
    self.balu_img = self.balu_img.resize((110, 100))  # Resize if needed
    self.balu_img = ImageTk.PhotoImage(self.balu_img)
    self.balu_label1 = Button(
        self.root,
        image=self.balu_img,
        bd=2,
        relief=RAISED,
        command=lambda: self.show_profile_card(
            name="Dr. Balkishan",
            image=self.balu_img,
            description="Designation : Assistant Professor, Department of Computer Science
& Applications",
```

```
            extra_info="Email: mdurohtak.ac.in\nPhone: +91-98xxxxxx1\nRoom No: 103, CS Block",
        ),
    )
    self.balu_label1.place(x=35, y=330)
    lbl_balu = Label(
        img_frame,
        text="Dr. Balkishan \n (Ass. Prof.)",
        font=("goudy old style", 12, "bold"),
        bg="white",
        bd=2,
        relief=RAISED,
    ).place(x=35, y=420, width=115)
    # ---------------------------------------------------------------------------
    self.amrinder_img = Image.open("img/amrindar.jpeg")
    self.amrinder_img = self.amrinder_img.resize((110, 100))  # Resize if needed
    self.amrinder_img = ImageTk.PhotoImage(self.amrinder_img)
    self.amrinder_label = Button(
        self.root,
        image=self.amrinder_img,
        bd=2,
```

```
            relief=RAISED,
            command=lambda: self.show_profile_card(
                name="Dr.Amrindar Kaur",
                image=self.amrinder_img,
                description="Designation : Assistant Professor, Department of Computer Science & Applications",
                extra_info="Email: mdurohtak.ac.in\nPhone: +91-98xxxxxxx1\nRoom No: 105, CS Block",
            ),
        )
        self.amrinder_label.place(x=170, y=330)
        lbl_amrinder = Label(
            img_frame,
            text="Dr.Amrinder\n(Ass. Prof.)",
            font=("goudy old style", 12, "bold"),
            bg="white",
            bd=2,
            relief=RAISED,
        ).place(x=170, y=420, width=115)
        # ---------------------------------------------------------------------
        self.sandeep_img = Image.open("img/sandeep.jpeg")
        self.sandeep_img = self.sandeep_img.resize((110, 100))  # Resize if needed
```

```
        self.sandeep_img = ImageTk.PhotoImage(self.sandeep_img)
        self.sandeep_label = Button(
            self.root,
            image=self.sandeep_img,
            bd=2,
            relief=RAISED,
            command=lambda: self.show_profile_card(
                name="Dr.Sandeep Dalal ",
                image=self.sandeep_img,
                description="Designation : Assistant Professor, Department of Computer Science
& Applications.",
                extra_info="Email: mdurohtak.ac.in\nPhone: +91-98xxxxxx1\nRoom No: 104,
CS Block",
            ),
        )
        self.sandeep_label.place(x=300, y=330)
        lbl_sandeep = Label(
            img_frame,
            text="Dr. Sandeep \n(Prof.)",
            font=("goudy old style", 12, "bold"),
            bg="white",
            bd=2,
```

```
        relief=RAISED,
    ).place(x=300, y=420, width=115)
    # ------------------------------------------------------------------------
    self.sukhi_img = Image.open("img/sukhvinder.jpg")
    self.sukhi_img = self.sukhi_img.resize((110, 100))  # Resize if needed
    self.sukhi_img = ImageTk.PhotoImage(self.sukhi_img)
    self.sukhi_label = Button(
        self.root,
        image=self.sukhi_img,
        bd=2,
        relief=RAISED,
        command=lambda: self.show_profile_card(
            name="Dr.Sukhvinder Singh Deora",
            image=self.sukhi_img,
            description="Designation : Assistant Professor, Department of Computer Science
& Applications",
            extra_info="Email: nasib.singh@mdurohtak.ac.in\nPhone: +91-
98xxxxxxx1\nRoom No: 101, CS Block",
        ),
    )
    self.sukhi_label.place(x=35, y=480)
    lbl_sukhi = Label(
```

```
        img_frame,
        text="Dr. Sukhvinder \n(Ass.Prof.)",
        font=("goudy old style", 12, "bold"),
        bg="white",
        bd=2,
        relief=RAISED,
    ).place(x=35, y=580, width=115)
    # ---------------------------------------------------------------------------
    self.gopal_img = Image.open("img/gopal.jpg")
    self.gopal_img = self.gopal_img.resize((110, 100))  # Resize if needed
    self.gopal_img = ImageTk.PhotoImage(self.gopal_img)
    self.gopal_label = Button(
        self.root,
        image=self.gopal_img,
        bd=2,
        relief=RAISED,
        command=lambda: self.show_profile_card(
            name="Dr.Gopal Singh ",
            image=self.gopal_img,
            description="Designation : Assistant Professor, Department of Computer Science
& Applications",
```

```
            extra_info="Email: nasib.singh@mdurohtak.ac.in\nPhone: +91-1262-393205
\nRoom No: 101, CS Block",

        ),

    )

    self.gopal_label.place(x=170, y=480)

    lbl_gopal = Label(

        img_frame,

        text="Dr.Gopal Singh\n(Ass. Prof.)",

        font=("goudy old style", 12, "bold"),

        bg="white",

        bd=2,

        relief=RAISED,

    ).place(x=170, y=580, width=115)

    # ---------------------------------------------------------------------------

    self.priti_img = Image.open("img/priti.jpg")

    self.priti_img = self.priti_img.resize((110, 100))  # Resize if needed

    self.priti_img = ImageTk.PhotoImage(self.priti_img)

    self.priti_label = Button(

        self.root,

        image=self.priti_img,

        bd=2,
```

```
            relief=RAISED,

            command=lambda: self.show_profile_card(

                name="Dr.Preeti Sharma ",

                image=self.priti_img,

                description="Senior Professor, specializing in Artificial Intelligence and expert in
Data Mining.",

                extra_info="Email: nasib.singh@mdurohtak.ac.in\nPhone: +91-
98xxxxxxx1\nRoom No: 101, CS Block",

            ),

        )

        self.priti_label.place(x=300, y=480)

        lbl_priti = Label(

            img_frame,

            text="Dr. Priti Sharma \n(Ass. Prof.)",

            font=("goudy old style", 12, "bold"),

            bg="white",

            bd=2,

            relief=RAISED,

        ).place(x=300, y=580, width=115)



    # ================== Slider Function ==================
```

```
def update_slider(self):
    """Function to update slider image every 3 seconds."""
    self.current_image_index = (self.current_image_index + 1) % len(
        self.slider_images
    )  # Loop images
    new_image = Image.open(self.slider_images[self.current_image_index]).resize(
        (910, 400)
    )
    self.bg_img = ImageTk.PhotoImage(new_image)
    self.bg_label.config(image=self.bg_img)  # Update the image in the label
    self.root.after(
        3000, self.update_slider
    )  # Call function every 3 seconds (3000ms)


# =================ADD WINDOWS FOR DIFFRENT BUTTONS ================
def add_course(self):
    self.new_window = Toplevel(self.root)
    self.new_obj = Course(self.new_window)


def add_student(self):
    self.new_window = Toplevel(self.root)
```

```
    self.new_obj = Student_cls(self.new_window)


def add_result(self):
    self.new_window = Toplevel(self.root)
    self.new_obj = result_cls(self.new_window)


def show_result(self):
    self.new_window = Toplevel(self.root)
    self.new_obj = view_result_cls(self.new_window)


def export_results(self):
    self.new_window = Toplevel(self.root)
    self.new_obj = view_result_cls(self.new_window)


def exit_btn(self):
    result = messagebox.askyesno("Confirm", " Are you sure want to exit ? ")
    if result:
        root.destroy()
    else:
        pass
```

```
def logout(self):
    op = messagebox.askyesno(
        "Confirm", "Do you really want to logout?", parent=self.root
    )
    if op == True:
        self.root.destroy()
        os.system("python log_in.py")


def show_description(self, desc):
    messagebox.showinfo("Faculty Info", desc)


def show_profile_card(self, name, image, description, extra_info):
    card = Toplevel(self.root)
    card.title(name)
    card.geometry("370x470+600+100")
    card.resizable(False, False)
    card.configure(bg="#fba6a1")  # Light Coral background

    # Card frame with soft rounded look
    card_frame = Frame(
        card,
```

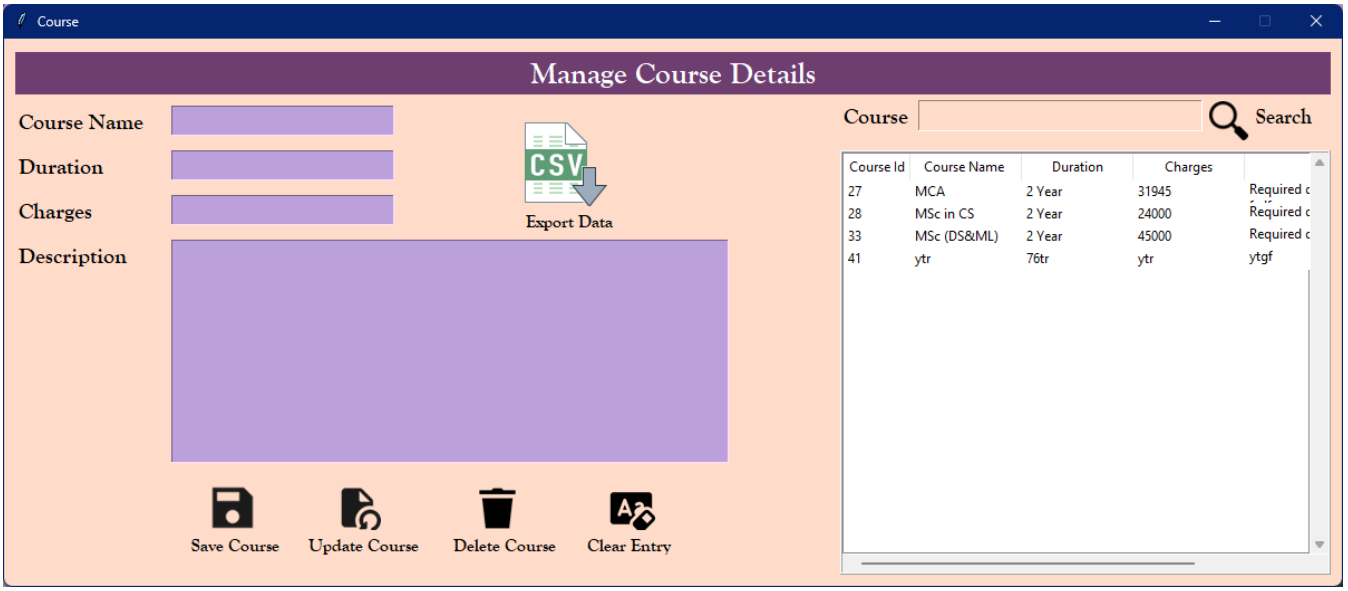
```
    bg="#ffffff",
    bd=0,
    highlightbackground="#bc83a4",
    highlightthickness=2,
)
card_frame.place(relx=0.5, rely=0.5, anchor="center", width=340, height=440)

# Image
img_label = Label(card_frame, image=image, bg="#ffffff")
img_label.image = image
img_label.pack(pady=(15, 8))

# Name
name_label = Label(
    card_frame,
    text=name,
    font=("Poppins", 16, "bold"),
    bg="#ffffff",
    fg="#6e3e70",
)
name_label.pack(pady=(0, 6))
```

```
# Divider line
Frame(card_frame, bg="#bc83a4", height=2).pack(fill="x", padx=40)

# Description
desc_label = Label(
    card_frame,
    text=description,
    wraplength=300,
    font=("Segoe UI", 11),
    bg="#ffffff",
    fg="#3c3c3c",
    justify=LEFT,
)
desc_label.pack(pady=(12, 6), padx=15)

# Extra Info
extra_label = Label(
    card_frame,
    text=extra_info,
    wraplength=300,
```

```
    font=("Calibri", 10, "italic"),
    bg="#ffffff",
    fg="#6e3e70",
    justify=LEFT,
)
extra_label.pack(pady=(0, 16), padx=15)

# Close button
Button(
    card_frame,
    text="Close",
    command=card.destroy,
    font=("Helvetica", 10, "bold"),
    bg="#6e3e70",
    fg="white",
    activebackground="#bc83a4",
    activeforeground="white",
    relief=FLAT,
    padx=18,
    pady=5,
    cursor="hand2",
```

```
        ).pack()

# ============== Run the Application ==============

if __name__ == "__main__":

    root = Tk()

    obj = RMS(root)

    root.mainloop()
```

4. Course Window -



Code-

```
from tkinter import *
from PIL import Image, ImageTk
```

```
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
import csv
from tkinter import filedialog


class Course:
    def __init__(self, root):
        self.root = root
        self.root.title("Course")
        self.root.geometry("1200x490+80+80")
        self.root.config(bg="#FFDCC9")
        self.root.focus_force()
        self.root.resizable(0, 0)

        # ====title===
        title = Label(
            self.root,
            text="Manage Course Details",
            padx=10,
            compound=LEFT,
            # image=self.logo_dash,
            font=("goudy old style", 20, "bold"),
            bg="#6E3E70",
            # fg="#FFDCC9",
            fg="white",

        ).place(x=10, y=12, width=1180, height=38)
        # ==========variable for widget entry==========
        self.var_course = StringVar()
        self.var_duration = StringVar()
        self.var_charges = StringVar()
        # ========widget===============
        lbl_course_name = Label(
            self.root,
```

```
    text="Course Name",
    font=("goudy old style", 15, "bold"),
    bg="#FFDCC9",
    # fg="#fec1a2"
).place(x=10, y=60)
lbl_duration = Label(
    self.root, text="Duration", font=("goudy old style", 15, "bold"), bg="#FFDCC9"
).place(x=10, y=100)
lbl_charges = Label(
    self.root, text="Charges", font=("goudy old style", 15, "bold"), bg="#FFDCC9"
).place(x=10, y=140)
lbl_description = Label(
    self.root,
    text="Description",
    font=("goudy old style", 15, "bold"),
    bg="#FFDCC9",
).place(x=10, y=180)
# ==========Entry fields ========
self.entry_course_name = Entry(
    self.root,
    textvariable=self.var_course,
    font=("goudy old style", 15, "bold"),
    bg="#bca0dc",
)
self.entry_course_name.place(x=150, y=60, width=200)
entry_duration = Entry(
    self.root,
    textvariable=self.var_duration,
    font=("goudy old style", 15, "bold"),
    bg="#bca0dc",
).place(x=150, y=100, width=200)
entry_charges = Entry(
    self.root,
    textvariable=self.var_charges,
    font=("goudy old style", 15, "bold"),
    bg="#bca0dc",
```

```
        ).place(x=150, y=140, width=200)
        self.entry_description = Text(
            self.root, font=("goudy old style", 15, "bold"), bg="#bca0dc"
        )
        self.entry_description.place(x=150, y=180, width=500, height=200)
        # =================BUTTONs====================

        # --------------------Export file button-------------------------
        self.login_btn = ImageTk.PhotoImage(file="img/export.png")
        btn2 = Button(
            self.root,
            image=self.login_btn,
            bd=0,
            bg="#FFDCC9",
            cursor="hand2",
            command=self.export_data,
        ).place(x=460, y=70)

        lbl_export = Label(self.root, text="Export Data",font=("goudy old style",12,"bold"),
bg="#FFDCC9").place(
            x=465, y=150
        )

        # ====================add button ================================
        self.save_btn = ImageTk.PhotoImage(file="img/save3.png")
        self.btn1 = Button(
            self.root,
            image=self.save_btn,
            bd=0,
            bg="#FFDCC9",
            cursor="hand2",
            command=self.add,
        )
        self.btn1.place(x=180, y=400, height=40)
        lbl_save = Label(
            self.root,
            text="Save Course",
```

```
    bg="#FFDCC9",
    font=("goudy old style", 12, "bold"),
).place(x=165, y=440)

self.update_btn = ImageTk.PhotoImage(file="img/update3.png")
self.btn2 = Button(
    self.root,
    image=self.update_btn,
    bd=0,
    bg="#FFDCC9",
    cursor="hand2",
    command=self.update,
)
self.btn2.place(x=300, y=400, height=40)
lbl_update = Label(
    self.root,
    text="Update Course",
    bg="#FFDCC9",
    font=("goudy old style", 12, "bold"),
).place(x=270, y=440)

self.delete_btn = ImageTk.PhotoImage(file="img/delete.png")
self.btn3 = Button(
    self.root,
    image=self.delete_btn,
    bd=0,
    bg="#FFDCC9",
    cursor="hand2",
    command=self.delete,
)
self.btn3.place(x=420, y=400, height=40)
lbl_delete = Label(
    self.root,
    text="Delete Course",
    bg="#FFDCC9",
    font=("goudy old style", 12, "bold"),
```

```
).place(x=400, y=440)


self.clear_btn = ImageTk.PhotoImage(file="img/clear.png")
self.btn4 = Button(
   self.root,
   image=self.clear_btn,
   bd=0,
   bg="#FFDCC9",
   cursor="hand2",
   command=self.clear,
)
self.btn4.place(x=540, y=400, height=40)
lbl_clear = Label(
   self.root,
   text="Clear Entry",
   bg="#FFDCC9",
   font=("goudy old style", 12, "bold"),
).place(x=520, y=440)

# ==========Search panal==================================
self.var_search = StringVar()
lbl_search = Label(
   self.root, text="Course", font=("goudy old style", 15, "bold"),bg="#FFDCC9"
).place(x=750, y=55)
entry_search = Entry(
   self.root,
   textvariable=self.var_search,
   font=("goudy old style", 15, "bold"),
   bg="#FFDCC9",  # bca0dc
)
entry_search.place(x=820, y=55, width=255, height=28)

self.search_btn = ImageTk.PhotoImage(file="img/search.png")
self.btn5 = Button(
   self.root,
```

```
    image=self.search_btn,
    bd=0,
    bg="#FFDCC9",
    cursor="hand2",
    command=self.search,
)
self.btn5.place(x=1080, y=55)
lbl_clear = Label(
    self.root, text="Search", bg="#FFDCC9",font=("goudy old style", 14, "bold")
).place(x=1120, y=55)

self.search_frame = Frame(self.root, bd=2, relief=RIDGE)
self.search_frame.place(x=750, y=100, height=380, width=440)

# =====Tree view for table==================
scrolly = Scrollbar(self.search_frame, orient=HORIZONTAL)
scrollx = Scrollbar(self.search_frame, orient=VERTICAL)

self.course_table = ttk.Treeview(
    self.search_frame,
    columns=("id", "name", "duration", "charges", "description"),
    xscrollcommand=scrolly.set,
    yscrollcommand=scrollx.set,
)
scrolly.pack(side=BOTTOM, fill=X)
scrollx.pack(side=RIGHT, fill=Y)
scrollx.config(command=self.course_table.yview)
scrolly.config(command=self.course_table.xview)

self.course_table.heading("id", text="Course Id")
self.course_table.heading("name", text="Course Name ")
self.course_table.heading("duration", text="Duration")
self.course_table.heading("charges", text="Charges")
self.course_table.heading("description", text="Description")
self.course_table["show"] = "headings"
self.course_table.column("id", width=60)
```

```
            self.course_table.column("name", width=100)
            self.course_table.column("duration", width=100)
            self.course_table.column("charges", width=100)
            self.course_table.column("description", width=200)
            self.course_table.pack(fill=BOTH, expand=1)
            self.course_table.bind("<ButtonRelease-1>", self.get_data)
            self.show()
            # ============================focus on data for filling entry =====

        def get_data(self, ev):
            self.entry_course_name.config(state="readonly")
            r = self.course_table.focus()
            content = self.course_table.item(r)
            row = content["values"]
            self.var_course.set(row[1])
            self.var_duration.set(row[2])
            self.var_charges.set(row[3])
            self.entry_description.delete("1.0", END)
            self.entry_description.insert(END, row[4])

            # ========================add function for save
button=======================

        def add(self):
            con = sqlite3.connect(database="srms.db")
            cur = con.cursor()
            try:
                if self.var_course.get() == "":
                    messagebox.showerror(
                        "Error", "Course Name should not be empty", parent=self.root
                    )
                else:
                    cur.execute(
                        "select * from course where name=?", (self.var_course.get(),)
                    )
                    row = cur.fetchone()
                    if row != None:
```

```
            messagebox.showerror(
                "Error", "Course Already Exists", parent=self.root
            )
        else:
            cur.execute(
                "insert into course (name,duration,charges,description) values(?,?,?,?)",
                (
                    self.var_course.get(),
                    self.var_duration.get(),
                    self.var_charges.get(),
                    self.entry_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)}")

# =============Show function for showing the data in the table==============
def show(self):
    con = sqlite3.connect(database="srms.db")
    cur = con.cursor()
    try:
        cur.execute("SELECT * FROM course")
        rows = cur.fetchall()
        self.course_table.delete(
            *self.course_table.get_children()
        )  # Clear all items

        for row in rows:
            self.course_table.insert("", END, values=row)

    except Exception as ex:
```

```
                messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
            finally:
                con.close()  # Close connection to avoid database lock issues

        #
================================================================================
========
        # ================================UPDATE FUNCTION FOR UPDATE DATA
====================
        def update(self):
            con = sqlite3.connect(database="srms.db")
            cur = con.cursor()
            try:
                if self.var_course.get() == "":
                    messagebox.showerror(
                        "Error", "Select course from list ", parent=self.root
                    )
                else:
                    cur.execute(
                        "select * from course where name=?", (self.var_course.get(),)
                    )
                    row = cur.fetchone()
                    if row == None:
                        messagebox.showerror(
                            "Error", "Select course from list ", parent=self.root
                        )
                    else:
                        cur.execute(
                            "update course set duration=?,charges=?, description=? where name=?",
                            (
                                self.var_duration.get(),
                                self.var_charges.get(),
                                self.entry_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)}")

            # ====================Clear Function for clear button to clear
entry==================
            def clear(self):

                self.var_course.set(""),
                self.var_duration.set(""),
                self.var_charges.set(""),
                self.entry_description.delete("1.0", END),
                self.var_search.set("")
                self.entry_course_name.config(state=NORMAL)
                # self.show()

            # ==================Delete function for delete button
=======================================
            def delete(self):
                con = sqlite3.connect(database="srms.db")
                cur = con.cursor()
                try:
                    if self.var_course.get() == "":
                        messagebox.showerror(
                            "Error", "Select course from list", parent=self.root
                        )
                    else:
                        cur.execute(
                            "SELECT * FROM course WHERE name=?", (self.var_course.get(),)
                        )
                        row = cur.fetchone()
                        if row is None:
                            messagebox.showerror(
                                "Error", "Please select course from the list", parent=self.root
```

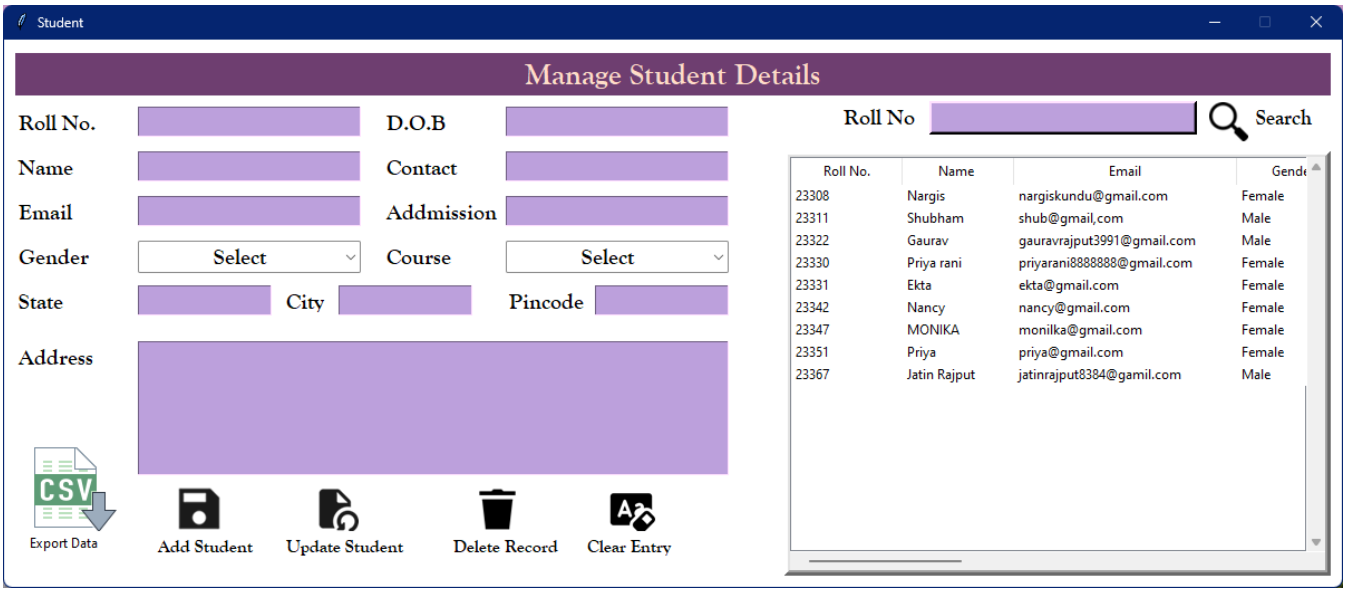
```
                    )
                else:
                    op = messagebox.askyesno(
                        "Confirm", "Do you really want to delete?", parent=self.root
                    )
                    if op:
                        cur.execute(
                            "DELETE FROM course WHERE name=?", (self.var_course.get(),)
                        )
                        con.commit()
                        messagebox.showinfo(
                            "Success", "Course deleted successfully", parent=self.root
                        )
                        self.clear()
                        self.show()  # Correct indentation
            except Exception as ex:
                messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)

        # ====================== Search function for search
button====================
        def search(self):
            con = sqlite3.connect(database="srms.db")
            cur = con.cursor()
            try:
                cur.execute(
                    f"SELECT * FROM course where name LIKE '%{self.var_search.get()}%'"
                )
                rows = cur.fetchall()
                self.course_table.delete(
                    *self.course_table.get_children()
                )  # Clear all items

                for row in rows:
                    self.course_table.insert("", END, values=row)

            except Exception as ex:
                messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
```

```
    finally:
        con.close()

# ======================================================
def export_data(self):
    con = sqlite3.connect(database="srms.db")
    cur = con.cursor()
    try:
        cur.execute("SELECT * FROM course")
        rows = cur.fetchall()

        if not rows:
            messagebox.showwarning(
                "No Data", "No data found to export", parent=self.root
            )
            return

        file_path = filedialog.asksaveasfilename(
            defaultextension=".csv",
            filetypes=[("CSV files", "*.csv")],
            title="Save file",
        )

        if file_path:
            with open(file_path, mode="w", newline="", encoding="utf-8") as file:
                writer = csv.writer(file)
                writer.writerow(
                    [
                        "Course Id",
                        "Course Name",
                        "Duration",
                        "Charges",
                        "Description",
                    ]
                )  # Column headers
                writer.writerows(rows)  # Writing data rows
```

```
            messagebox.showinfo(
                "Success",
                f"Data successfully exported to {file_path}",
                parent=self.root,
            )

    except Exception as ex:
        messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
    finally:
        con.close()

# --------------------------processing bar ----------------------
def show_processing(self):
    """Show a processing animation before opening the dashboard"""
    self.process_win = Toplevel(self.root)
    self.process_win.title("Processing...")
    self.process_win.geometry("350x150+500+300")
    self.process_win.config(bg="white")
    self.process_win.grab_set()

    Label(
        self.process_win,
        text="Please wait...",
        font=("times new roman", 15, "bold"),
        bg="white",
        fg="green",
    ).pack(pady=10)

    progress = ttk.Progressbar(
        self.process_win, orient=HORIZONTAL, length=250, mode="determinate"
    )
    progress.pack(pady=20)
    progress.start(10)

    # Simulate processing time before opening the dashboard
```

```
        self.root.after(3000, self.root)


# ================================================================
if __name__ == "__main__":
    root = Tk()
    obj = Course(root)
    root.mainloop()
```

5. Student Window -



Code-

```
 from tkinter import *
from PIL import Image, ImageTk
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
from tkinter import filedialog
import csv
```

```
class Student_cls:
    def __init__(self, root):
        self.root = root
        self.root.title("Student")
        self.root.geometry("1200x490+80+80")
        self.root.config(bg="white")
        self.root.focus_force()
        self.root.resizable(0, 0)

        # ====title====================

        title = Label(
            self.root,
            text="Manage Student Details",
            padx=10,
            compound=LEFT,
            # image=self.logo_dash,
            font=("goudy old style", 20, "bold"),
            bg="#6E3E70",
            fg="#FFDCC9",
        ).place(x=10, y=12, width=1180, height=38)

        # ==========variable for widget
entry=================================================

        self.var_rollno = StringVar()
        self.var_name = StringVar()
        self.var_email = StringVar()
        self.var_gender = StringVar()
        self.var_state = StringVar()
        self.var_city = StringVar()
        self.var_dob = StringVar()
        self.var_contact = StringVar()
        self.var_course = StringVar()
        self.var_a_date = StringVar()
        self.var_pincode = StringVar()
        self.var_addmission = StringVar()
        #
========widget=============================================================
==================
```

```
# --------------left Column lables----------------
lbl_rollno = Label(
   self.root,
   text="Roll No. ",
   font=("goudy old style", 15, "bold"),
   bg="white",
).place(x=10, y=60)
lbl_name = Label(
   self.root, text="Name", font=("goudy old style", 15, "bold"), bg="white"
).place(x=10, y=100)
lbl_email = Label(
   self.root, text="Email", font=("goudy old style", 15, "bold"), bg="white"
).place(x=10, y=140)
lbl_gender = Label(
   self.root, text="Gender", font=("goudy old style", 15, "bold"), bg="white"
).place(x=10, y=180)
lbl_state = Label(
   self.root, text="State", font=("goudy old style", 15, "bold"), bg="white"
).place(x=10, y=220)
lbl_address = Label(
   self.root,
   text="Address",
   font=("goudy old style", 15, "bold"),
   bg="white",
).place(x=10, y=270)
# -------------right Column lables ----------------

lbl_dob = Label(
   self.root,
   text="D.O.B ",
   font=("goudy old style", 15, "bold"),
   bg="white",
).place(x=340, y=60)

lbl_contact = Label(
   self.root, text="Contact", font=("goudy old style", 15, "bold"), bg="white"
).place(x=340, y=100)

lbl_addmission = Label(
   self.root,
```

```
            text="Addmission",
            font=("goudy old style", 15, "bold"),
            bg="white",
        ).place(x=340, y=140)

        lbl_Course = Label(
            self.root, text="Course", font=("goudy old style", 15, "bold"), bg="white"
        ).place(x=340, y=180)

        lbl_city = Label(
            self.root, text="City", font=("goudy old style", 15, "bold"), bg="white"
        ).place(x=250, y=220)

        lbl_pincode = Label(
            self.root, text="Pincode", font=("goudy old style", 15, "bold"), bg="white"
        ).place(x=450, y=220)

        # ==========Entry fields
=============================================================
        self.course_list = []
        self.fetch_course()
        # ----------left entry----------------------
        self.entry_rollno = Entry(
            self.root,
            textvariable=self.var_rollno,
            font=("goudy old style", 15, "bold"),
            bg="#bca0dc",
            # bd=3,
            # relief=RAISED,
        )
        self.entry_rollno.place(x=120, y=60, width=200)
        self.entry_name = Entry(
            self.root,
            textvariable=self.var_name,
            font=("goudy old style", 15, "bold"),
            bg="#bca0dc",
        ).place(x=120, y=100, width=200)
        self.entry_email = Entry(
            self.root,
            textvariable=self.var_email,
            font=("goudy old style", 15, "bold"),
```

```
    bg="#bca0dc",
).place(x=120, y=140, width=200)

self.entry_gender = ttk.Combobox(
    self.root,
    textvariable=self.var_gender,
    font=("goudy old style", 15, "bold"),
    justify=CENTER,
    values=("Select", "Male", "Female", "Other"),
)
self.entry_gender.place(x=120, y=180, width=200)
self.entry_gender.current(0)

self.entry_state = Entry(
    self.root,
    textvariable=self.var_state,
    font=("goudy old style", 15, "bold"),
    bg="#bca0dc",
).place(x=120, y=220, width=120)

self.entry_address = Text(
    self.root, font=("goudy old style", 15, "bold"), bg="#bca0dc"
)
self.entry_address.place(x=120, y=270, width=530, height=120)

# -------------Right Entry----------------------------

self.entry_dob = Entry(
    self.root,
    textvariable=self.var_dob,
    font=("goudy old style", 15, "bold"),
    bg="#bca0dc",
)
self.entry_dob.place(x=450, y=60, width=200)

self.entry_contact = Entry(
    self.root,
    textvariable=self.var_contact,
    font=("goudy old style", 15, "bold"),
    bg="#bca0dc",
).place(x=450, y=100, width=200)
```

```
        self.entry_addmission = Entry(
            self.root,
            textvariable=self.var_addmission,
            font=("goudy old style", 15, "bold"),
            bg="#bca0dc",
        ).place(x=450, y=140, width=200)

        self.entry_course = ttk.Combobox(
            self.root,
            textvariable=self.var_course,
            font=("goudy old style", 15, "bold"),
            justify=CENTER,
            values=(self.course_list),
        )
        self.entry_course.place(x=450, y=180, width=200)
        self.entry_course.set("Select")

        self.entry_city = Entry(
            self.root,
            textvariable=self.var_city,
            font=("goudy old style", 15, "bold"),
            bg="#bca0dc",
        ).place(x=300, y=220, width=120)

        self.entry_pincode = Entry(
            self.root,
            textvariable=self.var_pincode,
            font=("goudy old style", 15, "bold"),
            bg="#bca0dc",
        ).place(x=530, y=220, width=120)

        #
==================BUTTONs==========================================
===============


        self.save_btn = ImageTk.PhotoImage(file="img/save3.png")
        self.btn1 = Button(
            self.root,
            image=self.save_btn,
```

```
    bd=0,
    bg="white",
    cursor="hand2",
    command=self.add,
)
self.btn1.place(x=150, y=400, height=40)
lbl_save = Label(
    self.root,
    text="Add Student",
    bg="white",
    font=("goudy old style", 12, "bold"),
).place(x=135, y=440)


self.update_btn = ImageTk.PhotoImage(file="img/update3.png")
self.btn2 = Button(
    self.root,
    image=self.update_btn,
    bd=0,
    bg="white",
    cursor="hand2",
    command=self.update,
)
self.btn2.place(x=280, y=400, height=40)
lbl_update = Label(
    self.root,
    text="Update Student",
    bg="white",
    font=("goudy old style", 12, "bold"),
).place(x=250, y=440)


self.delete_btn = ImageTk.PhotoImage(file="img/delete.png")
self.btn3 = Button(
    self.root,
    image=self.delete_btn,
    bd=0,
    bg="white",
    cursor="hand2",
    command=self.delete,
)
```

```
self.btn3.place(x=420, y=400, height=40)
lbl_delete = Label(
   self.root,
   text="Delete Record",
   bg="white",
   font=("goudy old style", 12, "bold"),
).place(x=400, y=440)


self.clear_btn = ImageTk.PhotoImage(file="img/clear.png")
self.btn4 = Button(
   self.root,
   image=self.clear_btn,
   bd=0,
   bg="white",
   cursor="hand2",
   command=self.clear,
)
self.btn4.place(x=540, y=400, height=40)
lbl_clear = Label(
   self.root,
   text="Clear Entry",
   bg="white",
   font=("goudy old style", 12, "bold"),
).place(x=520, y=440)

self.export_btn = ImageTk.PhotoImage(file="img/export.png")
btn6 = Button(
   self.root,
   image=self.export_btn,
   bd=0,
   bg="white",
   cursor="hand2",
   command=self.export_data,
).place(x=20, y=360)

lbl_export = Label(self.root, text="Export Data", bg="white").place(x=20, y=440)

# ==========Search panal=======================================

self.var_search = StringVar()
```

```
lbl_search = Label(
   self.root, text="Roll No", font=("goudy old style", 15, "bold"), bg="white"
)
lbl_search.place(x=750, y=55)

entry_search = Entry(
   self.root,
   textvariable=self.var_search,
   font=("goudy old style", 15, "bold"),
   bg="#bca0dc",
   bd=3,
   relief=RAISED,
)
entry_search.place(x=830, y=55, width=240, height=30)

# search_btn = Button(
#    self.root,
#    text="Search",
#    font=("goudy old style", 15, "bold"),
#    bg="#F39c12",
#    fg="white",
#    cursor="hand2",
#    bd=3,
#    relief=RAISED,
#    command=self.search,
# ).place(x=1080, y=55, width=110, height=30)

self.search_btn = ImageTk.PhotoImage(file="img/search.png")
self.btn5 = Button(
   self.root,
   image=self.search_btn,
   bd=0,
   bg="white",
   cursor="hand2",
   command=self.search,
)
self.btn5.place(x=1080, y=55)
lbl_clear = Label(
   self.root, text="Search", bg="white", font=("goudy old style", 14, "bold")
).place(x=1120, y=55)
```

```
        self.search_frame = Frame(self.root, bd=5, relief=RAISED)
        self.search_frame.place(x=700, y=100, height=380, width=490)

        # =====Tree view for
    ==========================================================================

        scrolly = Scrollbar(self.search_frame, orient=HORIZONTAL)
        scrollx = Scrollbar(self.search_frame, orient=VERTICAL)

        self.rollno_table = ttk.Treeview(
            self.search_frame,
            columns=(
                "rollno",
                "name",
                "email",
                "gender",
                "state",
                "dob",
                "contact",
                "addmission",
                "course",
                "address",
                "city",
                "pincode",
            ),
            xscrollcommand=scrolly.set,
            yscrollcommand=scrollx.set,
        )
        scrolly.pack(side=BOTTOM, fill=X)
        scrollx.pack(side=RIGHT, fill=Y)
        scrollx.config(command=self.rollno_table.yview)
        scrolly.config(command=self.rollno_table.xview)

        self.rollno_table.heading("rollno", text="Roll No.")
        self.rollno_table.heading("name", text="Name ")
        self.rollno_table.heading("email", text="Email")
        self.rollno_table.heading("gender", text="Gender")
        self.rollno_table.heading("state", text="State")
        self.rollno_table.heading("dob", text="DOB")
        self.rollno_table.heading("contact", text="Contact")
        self.rollno_table.heading("addmission", text="Addmission")
```

```
        self.rollno_table.heading("course", text="Course")
        self.rollno_table.heading("address", text="Address")
        self.rollno_table.heading("city", text="City")
        self.rollno_table.heading("pincode", text="Pincode")
        self.rollno_table["show"] = "headings"
        self.rollno_table.column("rollno", width=100)
        self.rollno_table.column("name", width=100)
        self.rollno_table.column("email", width=200)
        self.rollno_table.column("gender", width=100)
        self.rollno_table.column("state", width=100)
        self.rollno_table.column("dob", width=100)
        self.rollno_table.column("contact", width=100)
        self.rollno_table.column("addmission", width=100)
        self.rollno_table.column("course", width=200)
        self.rollno_table.column("city", width=100)
        self.rollno_table.column("pincode", width=100)
        self.rollno_table.column("address", width=200)
        self.rollno_table.pack(fill=BOTH, expand=1)
        self.rollno_table.bind("<ButtonRelease-1>", self.get_data)
        self.show()

    # ============================focus on data for filling entry
==================

    def get_data(self, ev):
        self.entry_rollno.config(state="readonly")
        r = self.rollno_table.focus()
        content = self.rollno_table.item(r)
        row = content["values"]

        if row:
            self.var_rollno.set(row[0])  # Corrected index
            self.var_name.set(row[1])
            self.var_email.set(row[2])
            self.var_gender.set(row[3])
            self.var_state.set(row[4])
            self.var_dob.set(row[5])
            self.var_contact.set(row[6])
            self.var_addmission.set(row[7])
            self.var_course.set(row[8])
            self.entry_address.delete("1.0", END)
```

```
            self.entry_address.insert(END, row[9])
            self.var_city.set(row[10])
            self.var_pincode.set(row[11])

        # ========================add function for save
button====================

        def add(self):
            con = sqlite3.connect(database="srms.db")
            cur = con.cursor()
            try:
                if self.var_rollno.get() == "":
                    messagebox.showerror(
                        "Error", "Roll No. should not be empty", parent=self.root
                    )
                else:
                    cur.execute(
                        "select * from student where rollno=?", (self.var_rollno.get(),)
                    )
                    row = cur.fetchone()
                    if row != None:
                        messagebox.showerror(
                            "Error", "Roll No. Already Exists", parent=self.root
                        )
                    else:
                        cur.execute(
                            "insert into student(rollno,name ,email  ,gender  ,state  ,dob  ,contact
,addmission  ,course  ,address  ,city  ,   pincode  ) values(?,?,?,?,?,?,?,?,?,?,?,?)",
                            (
                                self.var_rollno.get(),
                                self.var_name.get(),
                                self.var_email.get(),
                                self.var_gender.get(),
                                self.var_state.get(),
                                self.var_dob.get(),
                                self.var_contact.get(),
                                self.var_addmission.get(),
                                self.var_course.get(),
                                self.entry_address.get("1.0", END),
                                self.var_city.get(),
                                self.var_pincode.get(),
```

```
                ),
            )
            con.commit()
            messagebox.showinfo(
                "Success", "Student Added Successfully", parent=self.root
            )
            self.show()
        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}")

    # =============Show function for showing the data in the
table=====================================

    def show(self):
        con = sqlite3.connect(database="srms.db")
        cur = con.cursor()
        try:
            cur.execute("SELECT * FROM student")
            rows = cur.fetchall()
            self.rollno_table.delete(
                *self.rollno_table.get_children()
            )  # Clear all items

            for row in rows:
                self.rollno_table.insert("", END, values=row)

        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
        finally:
            con.close()  # Close connection to avoid database lock issues

    #
=========================================================================
==============================

    # =================UPDATE FUNCTION FOR UPDATE DATA
====================
    def update(self):
        con = sqlite3.connect(database="srms.db")
        cur = con.cursor()
        try:
```

```
        if self.var_rollno.get() == "":
            messagebox.showerror(
                "Error", "Select Student from list ", parent=self.root
            )
        else:
            cur.execute(
                "select * from student where rollno=?", (self.var_rollno.get(),)
            )
            row = cur.fetchone()
            if row == None:
                messagebox.showerror(
                    "Error", "Select Student from list ", parent=self.root
                )
            else:
                cur.execute(
                    "UPDATE student SET name=?, email=?, gender=?, state=?, dob=?,
contact=?, addmission=?, course=?, address=?, city=?, pincode=? WHERE rollno=?",
                    (
                        self.var_name.get(),
                        self.var_email.get(),
                        self.var_gender.get(),
                        self.var_state.get(),
                        self.var_dob.get(),
                        self.var_contact.get(),
                        self.var_addmission.get(),
                        self.var_course.get(),
                        self.entry_address.get("1.0", END),
                        self.var_city.get(),
                        self.var_pincode.get(),
                        self.var_rollno.get(),
                    ),
                )
            con.commit()
            messagebox.showinfo(
                "Success", "Student updated Successfully", parent=self.root
            )
            self.show()
    except Exception as ex:
        messagebox.showerror("Error", f"Error due to {str(ex)}")

# ====================Clear Function for clear button to clear
```

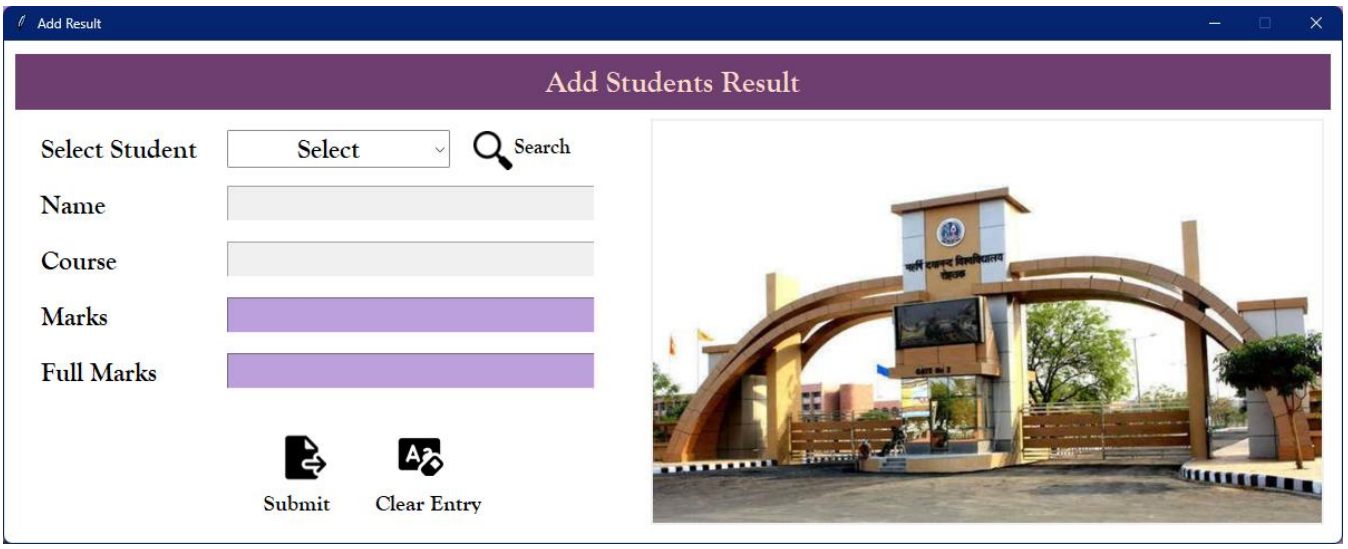
```
entry==============
   def clear(self):

      self.var_rollno.set(""),
      self.var_name.set(""),
      self.var_email.set(""),
      self.var_gender.set("select"),
      self.var_dob.set(""),
      self.var_city.set(""),
      self.var_state.set(""),
      self.var_addmission.set(""),
      self.var_contact.set(""),
      self.var_pincode.set(""),
      self.var_course.set("select"),
      self.entry_address.delete("1.0", END),
      self.var_search.set("")
      self.entry_rollno.config(state=NORMAL)
      # self.show()

   # ==================Delete function for delete button
=======================
   def delete(self):
      con = sqlite3.connect(database="srms.db")
      cur = con.cursor()
      try:
         if self.var_rollno.get() == "":
            messagebox.showerror(
               "Error", "Select rollno from list", parent=self.root
            )
         else:
            cur.execute(
               "SELECT * FROM student WHERE rollno=?", (self.var_rollno.get(),)
            )
            row = cur.fetchone()
            if row is None:
               messagebox.showerror(
                  "Error", "Please select rollno from the list", parent=self.root
               )
            else:
               op = messagebox.askyesno(
                  "Confirm", "Do you really want to delete?", parent=self.root
```

```
                )
                if op:
                    cur.execute(
                        "DELETE FROM student WHERE rollno=?",
                        (self.var_rollno.get(),),
                    )
                    con.commit()
                    messagebox.showinfo(
                        "Success", "rollno deleted successfully", parent=self.root
                    )
                    self.clear()
                    self.show()  # Correct indentation
        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)

    # ---------------------------fetch_course-----------------------------------------

    def fetch_course(self):
        con = sqlite3.connect(database="srms.db")
        cur = con.cursor()
        try:
            cur.execute("select name from course")
            rows = cur.fetchall()
            if len(rows) > 0:
                for row in rows:
                    self.course_list.append(row[0])
        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)

    # ====================== Search function for search
button=============================================

    def search(self):
        con = sqlite3.connect(database="srms.db")
        cur = con.cursor()
        try:

            if self.var_search.get() == "":
                messagebox.showerror(
                    "Error", "Roll No. should not be empty", parent=self.root
                )
```

```
            else:
                cur.execute(
                    "SELECT * FROM student WHERE rollno=?", (self.var_search.get(),)
                )
                row = cur.fetchone()
                if row is None:
                    messagebox.showerror("Error", "No record found", parent=self.root)
                else:
                    self.rollno_table.delete(
                        *self.rollno_table.get_children()
                    )  # Clear previous data
                    self.rollno_table.insert("", END, values=row)
        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
        finally:
            con.close()

        # ========================================================

    def export_data(self):
        con = sqlite3.connect(database="srms.db")
        cur = con.cursor()
        try:
            cur.execute("SELECT * FROM student")
            rows = cur.fetchall()

            if not rows:
                messagebox.showwarning(
                    "No Data", "No data found to export", parent=self.root
                )
                return

            file_path = filedialog.asksaveasfilename(
                defaultextension=".csv",
                filetypes=[("CSV files", "*.csv")],
                title="Save file",
            )

            if file_path:
                with open(file_path, mode="w", newline="", encoding="utf-8") as file:
                    writer = csv.writer(file)
```

```
                writer.writerow(
                    [
                        "Roll No",
                        "Name",
                        "Email",
                        "Gender",
                        "State",
                        "DOB",
                        "Contact",
                        "Addmission",
                        "Course",
                        "Address",
                        "City",
                        "Pincode",
                    ]
                )  # Column headers
                writer.writerows(rows)  # Writing data rows

            messagebox.showinfo(
                "Success",
                f"Data successfully exported to {file_path}",
                parent=self.root,
            )

        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
        finally:
            con.close()


#
================================================================================
=====================
if __name__ == "__main__":
    root = Tk()
    obj = Student_cls(root)
    root.mainloop()
```

6. Add Result Window -



Code-

```
from tkinter import *

from PIL import Image, ImageTk

import tkinter as tk

from tkinter import ttk, messagebox

import sqlite3
```

```
import csv
class result_cls:
    def __init__(self, root):
        self.root = root
        self.root.title("Add Result ")
        self.root.geometry("1200x450+80+80")
        self.root.config(bg="white")
        self.root.focus_force()
        self.root.resizable(0, 0)
        # ====title===
        title = Label(
            self.root,
            text="Add Students Result",
            padx=30,
            compound=LEFT,
            # image=self.logo_dash,
            font=("goudy old style", 20, "bold"),
            bg="#6E3E70",
            fg="#FFDCC9",
        ).place(x=10, y=12, width=1180, height=50)
        # -----------variables----------------------------------------------
```

```
self.var_rollno = StringVar()

self.var_name = StringVar()

self.var_course = StringVar()

self.var_marks = StringVar()

self.var_fullmarks = StringVar()

self.rollno_list = []

self.fetch_rollno()


# ----------------select student , entry, search button----------------------------------

lbl_rollno = Label(

    self.root,

    text="Select Student",

    font=("goudy old style", 18, "bold"),

    bg="white",

).place(x=30, y=80)

self.entry_rollno = ttk.Combobox(

    self.root,

    textvariable=self.var_rollno,

    font=("goudy old style", 18, "bold"),

    justify=CENTER,

    values=self.rollno_list,
```

```
        )
        self.entry_rollno.place(x=200, y=80, width=200)
        self.entry_rollno.set("Select")
        self.search_btn = ImageTk.PhotoImage(file="img/search.png")
        self.btn5 = Button(self.root, image=self.search_btn,bd=0,bg="white",cursor='hand2',
            command=self.search,
        )
        self.btn5.place(x=420, y=80)
        lbl_clear=Label(self.root, text="Search", bg="white",font=("goudy old style", 14,
"bold")).place(x=455, y=80)
        # ----------------------name , entry, ------------------------------
        lbl_name = Label(
            self.root, text="Name", font=("goudy old style", 18, "bold"), bg="white"
        ).place(x=30, y=130)
        entry_name = Entry(
            self.root,
            textvariable=self.var_name,
            font=("goudy old style", 18, "bold"),
            state="readonly",
            bg="#bca0dc",
        ).place(x=200, y=130, width=330)
```

```
# -----------------course--entrty-------------------------
lbl_course = Label(
    self.root,
    text="Course",
    font=("goudy old style", 18, "bold"),
    bg="white",
).place(x=30, y=180)

entry_course = Entry(
    self.root,
    textvariable=self.var_course,
    font=("goudy old style", 18, "bold"),
    state="readonly",
    bg="#bca0dc",
).place(x=200, y=180, width=330)

# -------------marks--entry-----------------------------------
lbl_marks = Label(
    self.root,
    text="Marks",
    font=("goudy old style", 18, "bold"),
```

```
    bg="white",
).place(x=30, y=230)
entry_marks = Entry(
    self.root,
    textvariable=self.var_marks,
    font=("goudy old style", 18, "bold"),
    bg="#bca0dc",
).place(x=200, y=230, width=330)
# ------------obtain--marks-----------------------------
lbl_full_marks = Label(
    self.root,
    text="Full Marks",
    font=("goudy old style", 18, "bold"),
    bg="white",
).place(x=30, y=280)
entry_full_marks = Entry(
    self.root,
    textvariable=self.var_fullmarks,
    font=("goudy old style", 18, "bold"),
    bg="#bca0dc",
).place(x=200, y=280, width=330)
```

```
        # ------------save--clear--button-------------------

        self.submit_btn = ImageTk.PhotoImage(file="img/submit.png")

        self.btn1 = Button(self.root, image=self.submit_btn,bd=0,bg="white",cursor='hand2',

           command=self.add_result,

        )

        self.btn1.place(x=250, y=350)

        lbl_clear=Label(self.root, text="Submit", bg="white",font=("goudy old style", 15,
"bold")).place(x=230,y=400)



        self.clear_btn = ImageTk.PhotoImage(file="img/clear.png")

        self.btn4 = Button(self.root, image=self.clear_btn,bd=0,bg="white",cursor='hand2',

           command=self.clear,

        )

        self.btn4.place(x=350, y=350, height=40)

        lbl_clear=Label(self.root, text="Clear Entry", bg="white",font=("goudy old style", 15,
"bold")).place(x=330,y=400)



        # ----------background--image----------------------------------------



        self.bg_img = Image.open("img\MDU_Result.jpg")

        self.bg_img = self.bg_img.resize((600, 360))  # Resize if needed

        self.bg_img = ImageTk.PhotoImage(self.bg_img)
```

```
    self.bg_label = Label(self.root, image=self.bg_img)
    self.bg_label.place(x=580, y=70)

# ---------fetch--data--function ----------------
def fetch_rollno(self):
    con = sqlite3.connect(database="srms.db")
    cur = con.cursor()
    try:
        cur.execute("select rollno from student")
        rows = cur.fetchall()
        if len(rows) > 0:
            for row in rows:
                self.rollno_list.append(row[0])
    except Exception as ex:
        messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)

# ---------search--function----------------------
def search(self):
    con = sqlite3.connect(database="srms.db")
    cur = con.cursor()
```

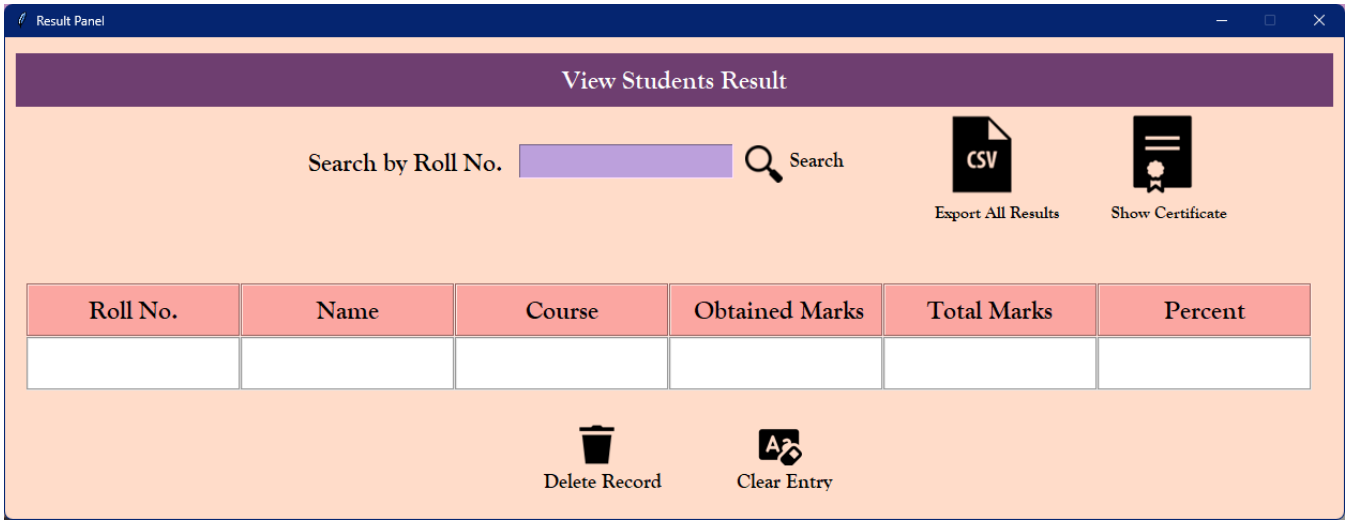
```
    try:


        cur.execute(
            "SELECT  name, course from student WHERE rollno=?",
            (self.var_rollno.get(),),
        )
        row = cur.fetchone()
        if row != None:
            self.var_name.set(row[0])
            self.var_course.set(row[1])
        else:
            messagebox.showerror("Error", "No record found", parent=self.root)


    except Exception as ex:
        messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
    finally:
        con.close()


# ------------add-result----------------------------
def add_result(self):
    con = sqlite3.connect(database="srms.db")
```

```
cur = con.cursor()
try:
    if self.var_name.get() == "":
        messagebox.showerror(
            "Error", " Name should not be empty", parent=self.root
        )
    else:
        cur.execute(
            "select * from result where rollno=? and course=?",
            (self.var_rollno.get(), (self.var_course.get())),
        )
        row = cur.fetchone()
        if row != None:
            messagebox.showerror(
                "Error", "Result Already Exists", parent=self.root
            )
        else:
            per = round((int(self.var_marks.get()) * 100) / int(self.var_fullmarks.get()), 2
            )
            cur.execute(
                "insert into result (rollno,name,course,marks_ob,full_marks,per)
```

```
values(?,?,?,?,?,?)",
                    (
                        self.var_rollno.get(),
                        self.var_name.get(),
                        self.var_course.get(),
                        self.var_marks.get(),
                        self.var_fullmarks.get(),
                        str(per),
                    ),
                )
                con.commit()
                messagebox.showinfo(
                    "Success", "Result Added Successfully", parent=self.root
                )

        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}")

    # -----------clear---function--------
    def clear(self):
        self.var_rollno.set("select"),
```

```
        self.var_name.set(""),
        self.var_course.set(""),
        self.var_marks.set(""),
        self.var_fullmarks.set(""),
    #---------------------------------------------------------
    # ------------------------------
    if __name__ == "__main__":
       root = Tk()
       obj = result_cls(root)
       root.mainloop()
```

7. Recent Feedback Page –



Code-

```
from tkinter import *
from PIL import Image, ImageTk
```

```
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
import csv
from tkinter import filedialog

class view_result_cls:
    def __init__(self, root):
        self.root = root
        self.root.title(" Result Panel ")
        self.root.geometry("1250x450+80+80")
        self.root.config(bg="#FFDCC9")
        self.root.focus_force()
        self.root.resizable(0, 0)

        # ====title===
        title = Label(
            self.root,
            text="View Students Result",
            padx=30,
            compound=LEFT,
            # image=self.logo_dash,
            font=("goudy old style", 18, "bold"),
            bg="#6E3E70",
            fg="white",
            # fg="#FBA6A1",
        ).place(x=10, y=15, width=1230, height=50)
        # ---------------variables----------------------
        self.var_search = StringVar()
        self.var_id = ""
        # -----------serach_by_rollno_--------------------------
        lbl_search = Label(
            self.root,
            text="Search by Roll No.",
            font=("goudy old style", 18, "bold"),
            bg="#FFDCC9",
        ).place(x=280, y=100)

        entry_search = Entry(
            self.root,
            textvariable=self.var_search,
```

```
            font=("goudy old style", 18, "bold"),
            #   state="readonly",
            bg="#bca0dc",
            fg='black'
        ).place(x=480, y=100, width=200)

        # --------------button---------------------
        self.certificate_btn = ImageTk.PhotoImage(file="img/certficate_icon.png")  # you can design
a small certificate icon
        btn_certificate = Button(
    self.root,
    image=self.certificate_btn,
    bd=0,
    bg="#FFDCC9",
    cursor="hand2",
    command=self.show_certificate,
)
        btn_certificate.place(x=1050, y=70)

        lbl_certificate = Label(
    self.root,
    text="Show Certificate",
    font=("goudy old style", 12, "bold"),
    bg="#FFDCC9",
).place(x=1030, y=150)



        self.search_btn = ImageTk.PhotoImage(file="img/search.png")
        self.btn5 = Button(self.root, image=self.search_btn,bd=0,bg="#FFDCC9",cursor='hand2',
            command=self.search,
        )
        self.btn5.place(x=690, y=100)
        lbl_clear=Label(self.root, text="Search", bg="#FFDCC9",font=("goudy old style", 14,
"bold")).place(x=730, y=100)

        self.clear_btn = ImageTk.PhotoImage(file="img/clear.png")
        self.btn4 = Button(self.root, image=self.clear_btn,bd=0,bg="#FFDCC9",cursor='hand2',
            command=self.clear,
        )
        self.btn4.place(x=700, y=360, height=40)
```

```
    lbl_clear=Label(self.root, text="Clear Entry", bg="#FFDCC9",font=("goudy old style", 14,
"bold")).place(x=680,y=400)



    self.delete_btn = ImageTk.PhotoImage(file="img/delete.png")
    self.btn3 = Button(self.root, image=self.delete_btn,bd=0,bg="#FFDCC9",cursor='hand2',
       command=self.delete,
    )
    self.btn3.place(x=530, y=360, height=40)
    lbl_delete=Label(self.root, text="Delete Record", bg="#FFDCC9",font=("goudy old style", 14,
"bold")).place(x=500,y=400)



    self.login_btn = ImageTk.PhotoImage(file="img/csv.png")
    btn2 = Button(
       self.root,
       image=self.login_btn,
       bd=0,
       bg="#FFDCC9",
       cursor="hand2",
       command=self.export_results,
    ).place(x=880, y=70)

    lbl_export = Label(self.root, text="Export All Results",font=("goudy old style",12,"bold"),
bg="#FFDCC9").place(
       x=865, y=150
    )

    # --------------result_lables---------------------------------
    lbl_rollno = Label(
       self.root,
       text="Roll No.",
       font=("goudy old style", 18, "bold"),
       bg="#fba6a1",
       bd=2,
       relief=GROOVE,
    ).place(x=20, y=230, height=50, width=200)

    lbl_name = Label(
       self.root,
```

```
    text="Name",
    font=("goudy old style", 18, "bold"),
    bg="#fba6a1",
    bd=2,
    relief=GROOVE,
).place(x=220, y=230, height=50, width=200)

lbl_course = Label(
    self.root,
    text="Course",
    font=("goudy old style", 18, "bold"),
    bg="#fba6a1",
    bd=2,
    relief=GROOVE,
).place(x=420, y=230, height=50, width=200)

lbl_marks = Label(
    self.root,
    text=" Obtained Marks",
    font=("goudy old style", 18, "bold"),
    bg="#fba6a1",
    bd=2,
    relief=GROOVE,
).place(x=620, y=230, height=50, width=200)

lbl_full_marks = Label(
    self.root,
    text="Total Marks",
    font=("goudy old style", 18, "bold"),
    bg="#fba6a1",
    bd=2,
    relief=GROOVE,
).place(x=820, y=230, height=50, width=200)

lbl_per = Label(
    self.root,
    text="Percent",
    font=("goudy old style", 18, "bold"),
    bg="#fba6a1",
    bd=2,
    relief=GROOVE,
```

```
).place(x=1020, y=230, height=50, width=200)

# -------------------lower labels ----------------------
self.rollno = Label(
    self.root,
    font=("goudy old style", 18, "bold"),
    bg="white",
    bd=2,
    relief=GROOVE,
)
self.rollno.place(x=20, y=280, height=50, width=200)

self.name = Label(
    self.root,
    font=("goudy old style", 18, "bold"),
    bg="white",
    bd=2,
    relief=GROOVE,
)
self.name.place(x=220, y=280, height=50, width=200)

self.course = Label(
    self.root,
    font=("goudy old style", 18, "bold"),
    bg="white",
    bd=2,
    relief=GROOVE,
)
self.course.place(x=420, y=280, height=50, width=200)

self.marks = Label(
    self.root,
    font=("goudy old style", 18, "bold"),
    bg="white",
    bd=2,
    relief=GROOVE,
)
self.marks.place(x=620, y=280, height=50, width=200)

self.full_marks = Label(
    self.root,
```

```
            font=("goudy old style", 18, "bold"),
            bg="white",
            bd=2,
            relief=GROOVE,
        )
        self.full_marks.place(x=820, y=280, height=50, width=200)

        self.per = Label(
            self.root,
            font=("goudy old style", 18, "bold"),
            bg="white",
            bd=2,
            relief=GROOVE,
        )
        self.per.place(x=1020, y=280, height=50, width=200)
    # ---------------------------------
    def show_certificate(self):
        if self.rollno.cget("text") == "":
            messagebox.showerror("Error", "No result selected to show Certificate", parent=self.root)
            return

        self.cert_window = Toplevel(self.root)   # <- Made it self.cert_window for access inside other
functions
        self.cert_window.title("Student Result Certificate")
        self.cert_window.geometry("600x600+400+50")
        self.cert_window.config(bg="white")
        self.cert_window.resizable(0, 0)

        title = Label(self.cert_window, text="Certificate of Achievement", font=("times new roman",
24, "bold"), bg="white", fg="#6E3E70")
        title.pack(pady=20)

        line = Label(self.cert_window, text="-----------------------------------------------------",
font=("times new roman", 18), bg="white", fg="#6E3E70")
        line.pack()

        self.lbl_name = Label(self.cert_window, text=f"Presented to: {self.name.cget('text')}",
font=("times new roman", 20, "bold"), bg="white")
        self.lbl_name.pack(pady=20)

        self.lbl_course = Label(self.cert_window, text=f"For successfully completing the course:
```

```
{self.course.cget('text')}", font=("times new roman", 16), bg="white")
        self.lbl_course.pack(pady=10)

        self.lbl_marks = Label(self.cert_window, text=f"Marks Obtained: {self.marks.cget('text')} /
{self.full_marks.cget('text')}", font=("times new roman", 16), bg="white")
        self.lbl_marks.pack(pady=10)

        self.lbl_per = Label(self.cert_window, text=f"Percentage: {self.per.cget('text')}", font=("times
new roman", 16), bg="white")
        self.lbl_per.pack(pady=10)

        self.lbl_rollno = Label(self.cert_window, text=f"Roll Number: {self.rollno.cget('text')}",
font=("times new roman", 16), bg="white")
        self.lbl_rollno.pack(pady=10)

        footer = Label(self.cert_window, text="Congratulations on your achievement!", font=("times
new roman", 18, "italic"), bg="white", fg="#6E3E70")
        footer.pack(pady=30)

        # Buttons Frame
        btn_frame = Frame(self.cert_window, bg="white")
        btn_frame.pack(pady=10)

        print_btn = Button(btn_frame, text="Print", command=self.print_certificate, font=("goudy
old style", 14, "bold"), bg="#27ae60", fg="white", cursor="hand2")
        print_btn.grid(row=0, column=0, padx=10)

        close_btn = Button(btn_frame, text="Close", command=self.cert_window.destroy,
font=("goudy old style", 14, "bold"), bg="#e74c3c", fg="white", cursor="hand2")
        close_btn.grid(row=0, column=1, padx=10)
    # ------------------------------------------------


    def print_certificate(self):
        try:
            from reportlab.lib.pagesizes import A4
            from reportlab.pdfgen import canvas
            from reportlab.lib.units import inch
            import tempfile
            import os
            from tkinter.filedialog import asksaveasfilename
```

```
        # Create temporary PDF file
        temp_pdf = tempfile.NamedTemporaryFile(delete=False, suffix=".pdf")
        temp_pdf_path = temp_pdf.name
        temp_pdf.close()

        c = canvas.Canvas(temp_pdf_path, pagesize=A4)
        width, height = A4

        # ---------- Draw Border ----------
        c.setStrokeColorRGB(0.8, 0.5, 0.2)  # Golden color
        c.setLineWidth(8)
        c.rect(30, 30, width-60, height-60)

        # ---------- Add Logo ----------
        try:
            c.drawImage("img/university_logo.png", width/2 - 50, 670, width=100, height=100,
preserveAspectRatio=True)
        except:
            pass  # In case logo not found, ignore.

        # ---------- Title ----------
        c.setFont("Helvetica-Bold", 30)
        c.setFillColorRGB(0.2, 0.3, 0.7)
        c.drawCentredString(width/2, height - 170 -40, "Certificate of Achievement")  # Reduced y
by 20

        # ---------- Decorative Line ----------
        c.setLineWidth(1)
        c.line(100, height-180 -40, width-100, height-180 -40)  # Reduced y by 20

        # ---------- Recipient Name ----------
        c.setFont("Helvetica", 18)
        c.setFillColorRGB(0,0,0)
        c.drawCentredString(width/2, height - 230 -40, "This is proudly presented to")  # Reduced
y by 20

        c.setFont("Helvetica-Bold", 24)
        c.drawCentredString(width/2, height - 270 -40, self.name.cget("text"))  # Reduced y by 20

        # ---------- Course Name ----------
```

```
        c.setFont("Helvetica", 18)
        c.drawCentredString(width/2, height - 310 -40, f"For completing the course")  # Reduced y
by 20

        c.setFont("Helvetica-Bold", 20)
        c.drawCentredString(width/2, height - 340 -40, self.course.cget("text"))  # Reduced y by 20



       # ---------- Performance Paragraph ----------
        # Prepare the values for the paragraph
        n=self.name.cget("text")
        roll = self.rollno.cget("text")
        course = self.course.cget("text")
        marks = self.marks.cget("text")
        full = self.full_marks.cget("text")
        percent = self.per.cget("text")


        lines = [
  f"Mr./Ms. {n} has successfully completed the {course}",
  f"course with excellence.They achieved a score of {marks} out of {full},",
  f"showcasing their hard work and determination with an impressive",
  f"overall percentage of {percent}%. They have proven their dedication.",
   "Throughout the duration of the course, they demonstrated",
  " outstanding, participation a strong grasp of the subject matter,  ",
  "and a sincere commitment to academic excellence."
]


        # Create a text object for the paragraph
        text = c.beginText(70, height - 400 - 40)  # 40 is your earlier vertical shift
        text.setFont("Times-Roman", 18)  # Italic font for emphasis
        text.setFillColorRGB(0.2,0.4,0.8)  # Soft blue color for the text
        text.setLeading(22)  # Line spacing

        # Add each line to the paragraph
        for line in lines:
            text.textLine(line)

        # Draw the text object onto the PDF
```

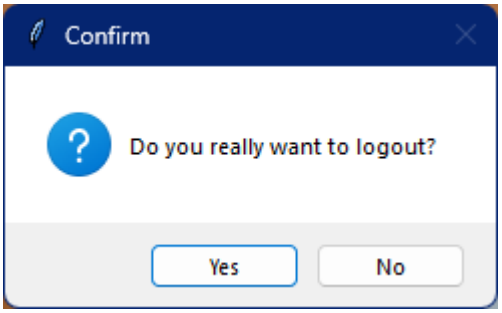
```
        c.drawText(text)

        # Optional: Add bold emphasis to specific parts of the paragraph
        c.setFont("Helvetica-Bold", 14)  # Switch to bold font for specific lines or words
        c.setFillColorRGB(0.8, 0.2, 0.2)  # Bold red color to highlight key points

        # Draw specific bold lines or words, such as the marks or percentage
        c.drawString(100, height - 470-130, f"Marks: {marks}/{full}")
        c.drawString(width - 200 , height - 470 -130, f"Percentage: {percent}%")

        # Return to original style after bold
        c.setFont("Helvetica-Oblique", 14)
        c.setFillColorRGB(0.2, 0.4, 0.8)  # Reset to original color

        # ---------- Badge Image ----------
        try:
            c.drawImage("img/medal2.png", 50, 718, width=100, height=100,
preserveAspectRatio=True)  # Keep original position
        except:
            pass

        # ---------- Signature ----------
        try:
            c.drawImage("img/signature2.png", width-200, 80, width=150, height=50,
preserveAspectRatio=True)  # Keep original position
        except:
            pass

        # Signature Text
        c.setFont("Helvetica-Oblique", 12)

        c.drawCentredString(width-130, 80, " Head of Dept.")  # Keep original position
        c.drawCentredString(width-130, 60, " Authorized Signature")  # Keep original position

        c.save()

        # ---------- Save PDF ----------
        save_path = asksaveasfilename(defaultextension=".pdf", filetypes=[("PDF files", "*.pdf")])

        if save_path:
            os.rename(temp_pdf_path, save_path)  # Save the file to the chosen path
```

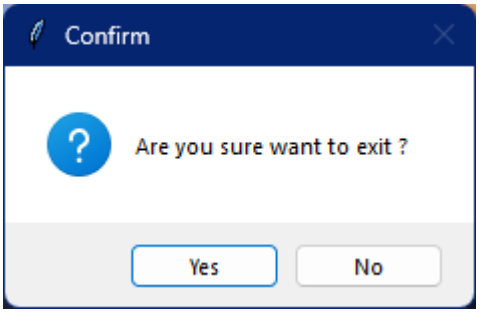
```
            messagebox.showinfo("Success", f"Certificate saved as {save_path}",
parent=self.cert_window)
        else:
            os.remove(temp_pdf_path)  # Delete the temporary file if user cancels the save dialog

    except Exception as ex:
        messagebox.showerror("Error", f"Error during saving the certificate: {str(ex)}",
parent=self.cert_window)


  # -------------serarch function------------------------
  def search(self):
    con = sqlite3.connect(database="srms.db")
    cur = con.cursor()
    try:
      if self.var_search.get() == "":
        messagebox.showerror("Error", "Roll no. should be required",parent=self.root)
      else:
        cur.execute(
          "SELECT  * from result WHERE rollno=?", (self.var_search.get(),)
        )
        row = cur.fetchone()
        if row != None:
          self.var_id = row[0]
          self.rollno.config(text=row[1])
          self.name.config(text=row[2])
          self.course.config(text=row[3])
          self.marks.config(text=row[4])
          self.full_marks.config(text=row[5])
          self.per.config(text=row[6])
        else:
          messagebox.showerror("Error", "No record found", parent=self.root)

    except Exception as ex:
      messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
    finally:
      con.close()

    # ----------clear function -------------

  def clear(self):
```

```
    self.var_id = ""
    self.rollno.config(text=" ")
    self.name.config(text="")
    self.course.config(text="")
    self.marks.config(text="")
    self.full_marks.config(text="")
    self.per.config(text="")
    self.var_search.set("")

# ------------delete funtion ----------------------

def delete(self):
    con = sqlite3.connect(database="srms.db")
    cur = con.cursor()
    try:
        if self.var_id == "":
            messagebox.showerror("Error", "Select record first", parent=self.root)
        else:
            cur.execute("SELECT * FROM result WHERE rid=?", (self.var_id,))
            row = cur.fetchone()
            if row is None:
                messagebox.showerror(
                    "Error", "Invalid student result", parent=self.root
                )
            else:
                op = messagebox.askyesno(
                    "Confirm", "Do you really want to delete?", parent=self.root
                )
                if op:
                    cur.execute("DELETE FROM result WHERE rid=?", (self.var_id,))
                    con.commit()
                    messagebox.showinfo(
                        "Success", "Result deleted successfully", parent=self.root
                    )
                    self.clear()
                    # self.show()# Correct indentation
    except Exception as ex:
        messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)

def export_results(self):
    con = sqlite3.connect(database="srms.db")
```

```
        cur = con.cursor()
        try:
            cur.execute("SELECT * FROM result")
            rows = cur.fetchall()

            if not rows:
                messagebox.showwarning(
                "No Data", "No results found to export", parent=self.root
            )
                return

            file_path = filedialog.asksaveasfilename(
            defaultextension=".csv",
            filetypes=[("CSV files", "*.csv")],
            title="Save file",
        )

            if file_path:
                with open(file_path, mode="w", newline="", encoding="utf-8") as file:
                    writer = csv.writer(file)
                    # Write column headers dynamically
                    headers = [description[0] for description in cur.description]
                    writer.writerow(headers)
                    writer.writerows(rows)

                messagebox.showinfo(
                "Success",
                f"Results successfully exported to {file_path}",
                parent=self.root,
            )

        except Exception as ex:
            messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
        finally:
            con.close()


if __name__ == "__main__":
    root = Tk()
    obj = view_result_cls(root)
    root.mainloop()
```

8. Log Out and Exit message -





Code-

```
    logout_btn = Button(
    menu_frame,
    text="Logout",
    font=("goudy old style", 18, "bold"),
    bg="#FEC1A2",
    fg="#391b49",
    cursor="hand2",
    bd=3,
    command=self.logout,
    relief=RAISED,
).place(x=900, y=5, width=200, height=40)
exit_btn = Button(
    menu_frame,
    text="Exit",
    font=("goudy old style", 18, "bold"),
    bg="#FFDCC9",
    fg="#391b49",
    cursor="hand2",
    bd=3,
    relief=RAISED,
    command=self.exit_btn,
).place(x=1120, y=5, width=200, height=40)

   def logout(self):
op = messagebox.askyesno(
    "Confirm", "Do you really want to logout?", parent=self.root
)
if op == True:
```

```
    self.root.destroy()
    os.system("python log_in.py")

   def exit_btn(self):
result = messagebox.askyesno("Confirm", " Are you sure want to exit ? ")
if result:
    root.destroy()
else:
    pass
```

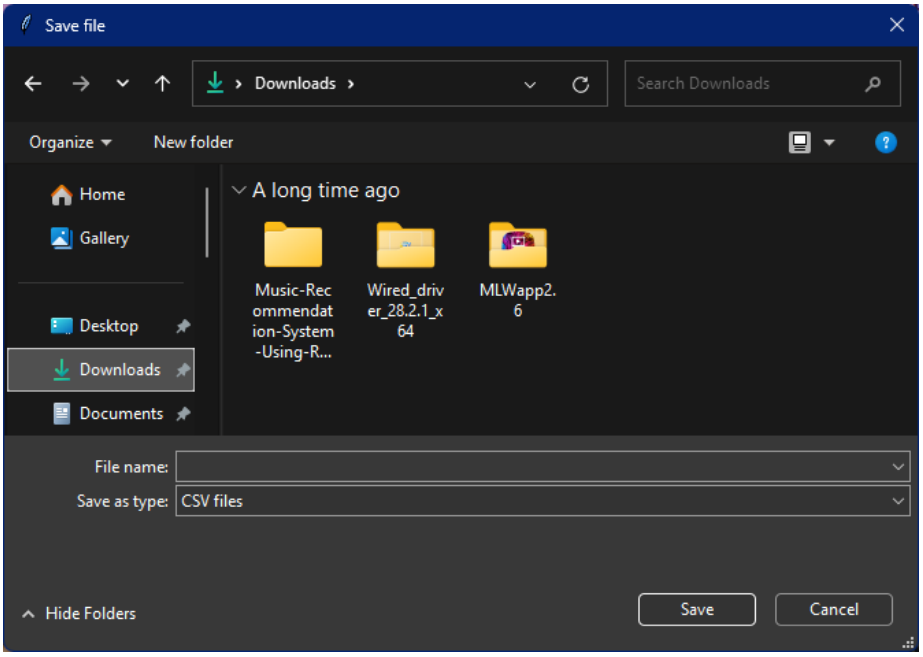
9. Information window which is embedded in dashboard window -

Code-

```
    def show_description(self, desc):
        messagebox.showinfo("Faculty Info", desc)

    def show_profile_card(self, name, image, description, extra_info):
        card = Toplevel(self.root)
        card.title(name)
        card.geometry("370x470+600+100")
        card.resizable(False, False)
        card.configure(bg="#fba6a1")  # Light Coral background

        # Card frame with soft rounded look
        card_frame = Frame(
            card,
            bg="#ffffff",
            bd=0,
            highlightbackground="#bc83a4",
            highlightthickness=2,
        )
        card_frame.place(relx=0.5, rely=0.5, anchor="center", width=340, height=440)

        # Image
        img_label = Label(card_frame, image=image, bg="#ffffff")
        img_label.image = image
        img_label.pack(pady=(15, 8))

        # Name
        name_label = Label(
            card_frame,
            text=name,
            font=("Poppins", 16, "bold"),
            bg="#ffffff",
            fg="#6e3e70",
        )
        name_label.pack(pady=(0, 6))

        # Divider line
        Frame(card_frame, bg="#bc83a4", height=2).pack(fill="x", padx=40)

        # Description
        desc_label = Label(
```

```
    card_frame,
    text=description,
    wraplength=300,
    font=("Segoe UI", 11),
    bg="#ffffff",
    fg="#3c3c3c",
    justify=LEFT,
)
desc_label.pack(pady=(12, 6), padx=15)

# Extra Info
extra_label = Label(
    card_frame,
    text=extra_info,
    wraplength=300,
    font=("Calibri", 10, "italic"),
    bg="#ffffff",
    fg="#6e3e70",
    justify=LEFT,
)
extra_label.pack(pady=(0, 16), padx=15)

# Close button
Button(
    card_frame,
    text="Close",
    command=card.destroy,
    font=("Helvetica", 10, "bold"),
    bg="#6e3e70",
    fg="white",
    activebackground="#bc83a4",
    activeforeground="white",
    relief=FLAT,
    padx=18,
    pady=5,
    cursor="hand2",
).pack()
```

10. Export Data -



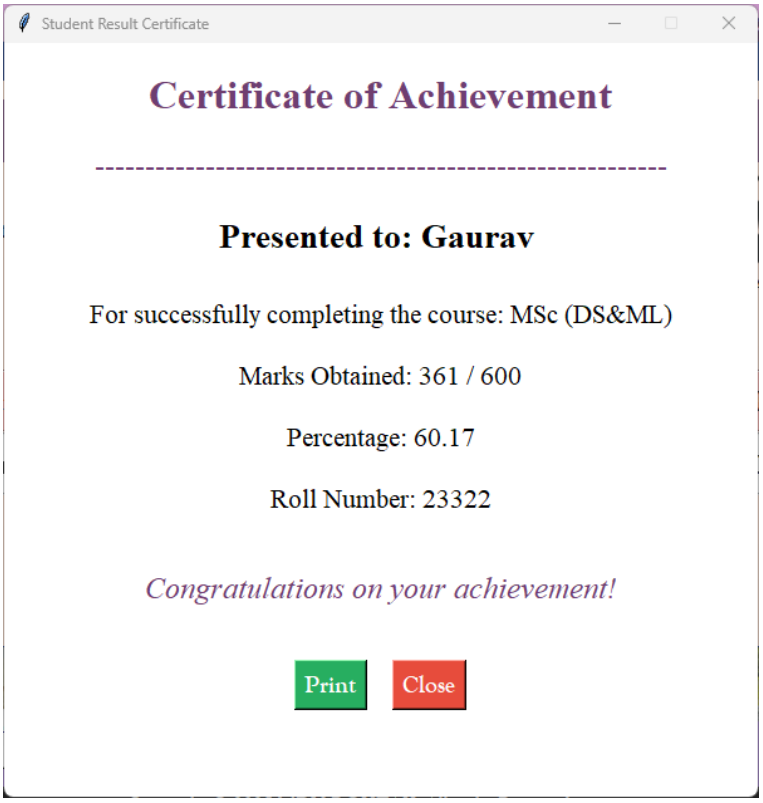
Code-

```
def export_data(self):
    con = sqlite3.connect(database="srms.db")
    cur = con.cursor()
    try:
        cur.execute("SELECT * FROM course")
        rows = cur.fetchall()
        if not rows:
            messagebox.showwarning(
                "No Data", "No data found to export", parent=self.root
            )
```

```
    Return
file_path = filedialog.asksaveasfilename(
    defaultextension=".csv",
    filetypes=[("CSV files", "*.csv")],
    title="Save file",
)
if file_path:
    with open(file_path, mode="w", newline="", encoding="utf-8") as file:
        writer = csv.writer(file)
        writer.writerow(
            [
                "Course Id",
                "Course Name",
                "Duration",
                "Charges",
                "Description",
            ]
        )  # Column headers
        writer.writerows(rows)  # Writing data rows
```

```
        messagebox.showinfo(
            "Success",
            f"Data successfully exported to {file_path}",
            parent=self.root,
        )
except Exception as ex:
    messagebox.showerror("Error", f"Error due to {str(ex)}", parent=self.root)
finally:
    con.close()
```

11. Certificate Window -



Code-

```
def show_certificate(self):
        if self.rollno.cget("text") == "":
            messagebox.showerror("Error", "No result selected to show Certificate",
    parent=self.root)
            return

        self.cert_window = Toplevel(self.root)   # <- Made it self.cert_window for access
    inside other functions
        self.cert_window.title("Student Result Certificate")
        self.cert_window.geometry("600x600+400+50")
        self.cert_window.config(bg="white")
        self.cert_window.resizable(0, 0)

        title = Label(self.cert_window, text="Certificate of Achievement", font=("times
    new roman", 24, "bold"), bg="white", fg="#6E3E70")
        title.pack(pady=20)

        line = Label(self.cert_window, text="------------------------------------------------------
    ", font=("times new roman", 18), bg="white", fg="#6E3E70")
        line.pack()

        self.lbl_name = Label(self.cert_window, text=f"Presented to:
    {self.name.cget('text')}", font=("times new roman", 20, "bold"), bg="white")
        self.lbl_name.pack(pady=20)

        self.lbl_course = Label(self.cert_window, text=f"For successfully completing the
    course: {self.course.cget('text')}", font=("times new roman", 16), bg="white")
        self.lbl_course.pack(pady=10)

        self.lbl_marks = Label(self.cert_window, text=f"Marks Obtained:
    {self.marks.cget('text')} / {self.full_marks.cget('text')}", font=("times new roman",
    16), bg="white")
        self.lbl_marks.pack(pady=10)

        self.lbl_per = Label(self.cert_window, text=f"Percentage: {self.per.cget('text')}",
    font=("times new roman", 16), bg="white")
        self.lbl_per.pack(pady=10)

        self.lbl_rollno = Label(self.cert_window, text=f"Roll Number:
```

```
{self.rollno.cget('text')}", font=("times new roman", 16), bg="white")
        self.lbl_rollno.pack(pady=10)

        footer = Label(self.cert_window, text="Congratulations on your achievement!",
font=("times new roman", 18, "italic"), bg="white", fg="#6E3E70")
        footer.pack(pady=30)

        # Buttons Frame
        btn_frame = Frame(self.cert_window, bg="white")
        btn_frame.pack(pady=10)

        print_btn = Button(btn_frame, text="Print", command=self.print_certificate,
font=("goudy old style", 14, "bold"), bg="#27ae60", fg="white", cursor="hand2")
        print_btn.grid(row=0, column=0, padx=10)

        close_btn = Button(btn_frame, text="Close",
command=self.cert_window.destroy, font=("goudy old style", 14, "bold"),
bg="#e74c3c", fg="white", cursor="hand2")
        close_btn.grid(row=0, column=1, padx=10)
    # ------------------------------------------------
```

Chapter 6
Backend

6.1 🎓 Student Result Management System — Backend (Python + SQLite3)

Introduction

This backend is built using **Python and SQLite3**, providing data handling and business logic for your **Tkinter-based Student Result Management System (SRMS)**. The system manages:

- Courses
- Student Records
- Exam Results
- Employee (Admin/Faculty) Accounts

This backend ensures all data is securely stored and manipulated in a **local SQLite database (`srms.db`)**, making it suitable for desktop applications.

Database Tables Overview

Table	Purpose
`course`	Stores course name, duration, charges, and description
`student`	Stores student details (name, contact, course, etc.)
`result`	Stores student result data
`employee`	Stores admin/faculty login credentials & security Q/A

6.2 Schemas for DataBase

Database Schemas

1. course Table – Stores course-related details

sql

CopyEdit

```
CREATE TABLE IF NOT EXISTS course (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    duration TEXT,
    charges TEXT,
    description TEXT
```

);

- id: Auto-incrementing unique course ID.
- name: Name of the course (e.g., B.Sc, MCA).
- duration: Duration of the course (e.g., "3 Years").
- charges: Course fees.
- description: Additional details.

---

2. student Table – Stores student personal and academic info

sql

CopyEdit

```
CREATE TABLE IF NOT EXISTS student (
    rollno INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    email TEXT,
    gender TEXT,
    state TEXT,
    dob TEXT,
    contact TEXT,
    addmission TEXT,
    course TEXT,
    address TEXT,
    city TEXT,
```

pincode TEXT

);

- rollno: Auto-incrementing student roll number.
- name, email, gender, dob, contact: Personal info.
- state, city, address, pincode: Location info.
- addmission: Admission date (as string).
- course: Course name linked to the course table.

---

3. result Table – Stores student exam results

sql

CopyEdit

CREATE TABLE IF NOT EXISTS result (

rid INTEGER PRIMARY KEY AUTOINCREMENT,

rollno TEXT,

name TEXT,

course TEXT,

marks_ob TEXT,

full_marks TEXT,

per TEXT

);

- rid: Auto-incrementing result ID.
- rollno, name: Student identification.

- course: Course name.
- marks_ob: Marks obtained.
- full_marks: Total marks.
  - per: Percentage.

4. employee Table – Stores login credentials and admin/faculty info

sql

CopyEdit

```
CREATE TABLE IF NOT EXISTS employee (
   eid INTEGER PRIMARY KEY AUTOINCREMENT,
   f_name TEXT,
   l_name TEXT,
   email TEXT,
   contact TEXT,
   question TEXT,
   answer TEXT,
   password TEXT
);
```

- eid: Auto-incrementing employee ID.
- f_name, l_name: Full name of the employee.
- email, contact: Login and communication details.
- question, answer: Security question and answer (for password recovery).
- password: Login password (stored in plaintext – **you should hash this in real apps**).

Chapter 7

Overview of System Architecture

System Architecture Overview

🔷 1. Presentation Layer (Frontend)

Technology: Python Tkinter GUI framework

Purpose: Provides the graphical interface for:

Login/Register

Managing students, courses, results, and employees

Displaying records and search functionality

Features:

Form-based data input

Buttons, labels, dropdowns

Interactive image sliders or category cards

Theme/dark mode toggle (if implemented)

Validation and error messages

🔷 **2. Application Layer (Backend Logic)**

Technology: Core Python scripts

Role:

Handles all business logic and operations

Processes form inputs and interacts with the database

Performs CRUD operations (Create, Read, Update, Delete)

Modules/Functions:

create_db() – initializes the SQLite DB and tables

Modules for:

Course Management

Student Management

Result Entry and Calculation

Employee Authentication (Login/Forgot Password)

🔷 **3. Data Layer (Database)**

Technology: SQLite3 (file-based database)

Database File: srms.db

Tables:

course – Course info

student – Student data

result – Exam scores and performance

employee – Login and user credentials

Interaction:

Accessed using Python's sqlite3 library

Each form operation triggers a corresponding SQL query

🔁 **Data Flow Summary**

User interacts with the GUI (Presentation Layer)

GUI calls Backend Logic to process the input

Backend interacts with SQLite database to:

Fetch/store/update data

Output or result is shown back to user via GUI

🧩 **Optional Extensions (Scalable Ideas)**

Add Login roles (Admin vs. Faculty)

Secure passwords using hashing (e.g., bcrypt)

Generate PDF reports or mark sheets

Add data validation and error logging

Use Object-Oriented Programming (OOP) for better structure

Chapter 8

Testing

8.1 Testing Plan (Numbered Bullet Points)

1. Unit Testing

1.1. Test `create_db()` function – Ensure all tables are created without errors.
1.2. Test student insertion – Add a student with valid data and verify in DB.
1.3. Test result entry – Enter result for an existing student and check table update.
1.4. Test student search – Search by name or roll number and confirm correct data is shown.
1.5. Test login system – Validate successful and failed login scenarios.

---

2. Integration Testing

2.1. Register new employee and login – Confirm login works after registration.
2.2. Add a course and enroll a student – Ensure course appears in student's record.
2.3. Enter result and view report – Verify marks, full marks, and percentage calculations.

---

3. GUI Testing (Manual)

3.1. Check button functionality – Add, Update, Delete, Search.
3.2. Validate entry fields – Ensure required fields are not left blank.
3.3. Test dropdowns, radio buttons – Make sure inputs are selectable and saved.
3.4. Test layout on different screen sizes – Interface should remain usable.
3.5. Test dark/light theme toggle – If implemented, check correct application.

---

4. Negative Testing

4.1. Leave required fields empty – System should prompt for input.
4.2. Input invalid email – Should trigger validation error.
4.3. Attempt duplicate roll number – Should show warning or block input.
4.4. Enter result for non-existing student – System should alert or restrict entry.

---

5. Database Testing

5.1. Directly query `student`, `result`, `course`, and `employee` tables.
5.2. Verify entries are correctly inserted, updated, or deleted as expected.
5.3. Test foreign key relationships (e.g., result tied to valid roll number).
5.4. Backup and restore the database to ensure reliability.

8.2 Cost effectiveness

- **Open-source technologies** – Utilizes free tools like Python, Tkinter, and SQLite, eliminating software licensing costs.

- **No need for high-end hardware** – The system runs efficiently on standard computers without additional infrastructure investment.

- **Low maintenance requirements** – Easy to manage and update without specialized technical staff.

- **Offline functionality** – Operates without internet dependency, saving on hosting or cloud service expenses.

- **All-in-one solution** – Integrates course, student, result, and employee management, reducing the need for multiple systems.

- **Scalable architecture** – New features can be added over time without complete system redesign or major costs.

- **No recurring subscription fees** – One-time setup with long-term usability and minimal operational costs.

- **Educational and training use** – Ideal for academic institutions as a teaching tool without extra investment.

- **Minimal third-party reliance** – Reduces costs associated with external plugins, APIs, or frameworks.

8.1 Customer Satisfaction

In any business, the ultimate goal is to give the best customer satisfaction. Yes, customer satisfaction is very important. Software testing improves the user experience of an

Happy customers mean more revenue for a business. One of the reasons why software testing is necessary is to provide the best user experience.

8.2 Security

The **Student Result Management System (SRMS)** employs various security measures to protect sensitive data and ensure safe operation. It uses data encryption to safeguard user information, including passwords, and protects against unauthorized access with a secure login system and role-based access control. To prevent SQL injection attacks, parameterized queries are implemented. The system also supports optional two-factor authentication for an added layer of security. User passwords are securely hashed before storage, ensuring protection even if the database is compromised. The application ensures a secure connection to the SQLite database, encrypting data in transit, and performs regular backups to facilitate recovery in case of any data loss or breach. These security protocols work together to provide a safe and reliable environment for managing student data.

8.3 Product Quality

The **Student Result Management System (SRMS)** ensures high product quality by adhering to best practices in software development, including thorough testing and validation processes. The system undergoes comprehensive unit testing, integration testing, and user

acceptance testing to identify and resolve any bugs or inconsistencies before deployment. It follows a modular design approach, making it easy to maintain and update. The user interface is intuitive and responsive, ensuring an optimal experience across various devices. The system is also optimized for performance, ensuring fast load times even with large volumes of data. Additionally, regular updates and improvements are incorporated based on user feedback to maintain and enhance the overall product quality*Principles of Software Testing:*

Testing of software consist of some principles that play a vital role while testing the project.The Principles of Software Testing are as follows:

8.3.1 Testing shows presence of defects
8.3.2 Exhaustive testing is impossible
8.3.3 Early testing
8.3.4 Defect clustering

Chapter 9

Conclusion and Future Scope

Conclusion:

The **Student Result Management System (SRMS)** offers an efficient and streamlined solution for managing student data, including their academic records, courses, and personal information. The system's user-friendly interface, combined with a secure backend, provides both students and administrators with a reliable tool for tracking academic progress, attendance, and other relevant information. The integration of features like login, result management, and course handling contributes to a seamless experience for all users. The use of **Tkinter** for the frontend and **SQLite** for the database ensures that the system is lightweight, cost-effective, and easily scalable for small to medium-sized institutions.

Future Scope:

1. **Mobile Application Integration**: Future versions of the system can be developed into a mobile application for easy access by students and faculty on the go.
2. **Cloud Integration**: Migrating the system to cloud platforms like AWS or Google Cloud could help in enhancing scalability and ensure data availability across various devices.
3. **Advanced Analytics**: Incorporating advanced data analytics and machine learning algorithms for predicting student performance and offering personalized recommendations for improvement could further enhance the system's capabilities.
4. **Multi-language Support**: Introducing support for multiple languages would make the system more inclusive for institutions with diverse linguistic backgrounds.
5. **Real-time Updates and Notifications**: Implementing real-time notifications for students regarding their results, assignments, and upcoming exams can improve communication.
6. **Enhanced Security Features**: Advanced security measures, such as two-factor authentication (2FA) and data encryption, could be added to improve system security.
7. **Integration with External Platforms**: The system could be integrated with Learning Management Systems (LMS) and other external platforms to streamline workflows and improve overall efficiency.
8. **AI-based Chatbot Support**: Introducing AI-powered chatbots to answer common student queries about results, courses, and other academic details would enhance the user experience.

References

Here are some potential references for your project, based on common sources related to software development, system architecture, database design, and Tkinter applications:

1. **Tkinter Documentation:**

Python Software Foundation. (n.d.). Tkinter — Python interface to Tcl/Tk. Retrieved from https://docs.python.org/3/library/tkinter.html

2. **SQLite Documentation:**

Hipp, D. R. (2000). SQLite Database System: Relational Database Management System. Retrieved from https://www.sqlite.org/

3. **Python Programming Language:**

Python Software Foundation. (n.d.). Python 3 documentation. Retrieved from https://docs.python.org/3/

4. **Designing Databases:**

Elmasri, R., & Navathe, S. B. (2015). Fundamentals of Database Systems (7th ed.). Pearson Education.

5. **User Interface Design Principles:**

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems

(pp. 249–256).

6. **System Architecture:**

Bass, L., Clements, P., & Kazman, R. (2003). Software Architecture in Practice (2nd ed.). Addison-Wesley.

7. **Python and Tkinter for GUI Development:**

Grayson, P. (2000). Python and Tkinter Programming. Manning Publications.

8. **Database Management Systems:**

Korth, H. F., & Silberschatz, A. (2006). Database System Concepts (6th ed.). McGraw-Hill.

9. **Security Measures for Web Applications:**

Schneier, B. (2015). Secrets and Lies: Digital Security in a Networked World. Wiley.

These references will help in understanding the technologies and methodologies used in your system, as well as offer in-depth knowledge of relevant tools and techniques.