



SHRI VILEPARLE KELAVANI MANDAL'S
SHRI BHAGUBHAI MAFATLAL POLYTECHNIC



Python Mini Project

Student Bio-Data RMS

Course: Programming in Python

Course Code: PRP198918

Roll No: T010

Name: Jainam Barbhaya

Table of Contents

| | |
|--|-----------|
| 1. Acknowledgement | 1 |
| 2. Problem Statement | 2 |
| 3. Features | 2 |
| 4. System Requirement Specification | 3 |
| 4.1 Functional Requirements | |
| 4.2. Non-Functional Requirements | |
| 4.3 System Requirements | |
| 4.3.1. Hardware Requirement | |
| 4.3.2. Software Requirement | |
| 5. System Design | 4 |
| 6. Literature Survey | 6 |
| 7. Gantt Chart | 7 |
| 8. Module Description | 7 |
| 9. System Implementation | 10 |
| 10.System Testing | 37 |
| 11.Results | 38 |
| 12.Future Scope | 44 |
| 13.Conclusion | 44 |
| 14.References | 44 |

1. Acknowledgement

I would like to sincerely thank my mentors, teachers, the open-source community, my friends and family, and the online forums and communities for their unwavering support throughout the development of this Python project. The availability of extensive resources and collaborative spirit have significantly contributed to the efficiency and effectiveness of this project. I am truly grateful for their belief in my abilities and the constant motivation they have provided. I would also like to express my appreciation to the project evaluators and reviewers for their valuable feedback and suggestions. This project would not have been possible without the combined efforts and contributions of these individuals and resources.

2. Problem Statement

Student Biodata Record Management System (RMS)

3. Features

The student bio data record management system is a system that allows admin to manage and maintain student records. The system includes following features:

- Login page: Admin can login into the system using their username and password.
- Add New student profile: Allows admin to insert a new record of a student by providing details such as SAP no, Roll no, Student Name, Student Image, contact, address, email id, etc.
- Delete any student profile: Allows admin to remove a student record by providing a mandatory removing reason.
- Search for record: Searching for record manually is a difficult task this function allows admin to search by providing a specific keywords.
- Profile Update: Admin can update a record by selecting an individual record and can by clicking on update function, this allows admin to change information such as contacts, emails, address and semester percentages.

4. System Requirements Specification

4.1. Functional Requirements

Student biodata Record Management System should be able to perform following functions:

- Allows Admin to login
- Insert Student Record
- Delete Student Record
- Search For Record
- Check for details about individual student
- Update student details

4.2. Non-Functional Requirements

Student biodata Record Management System should be able to fulfill following non-functional requirements:

- Security: The system must be secured and protected from unauthorized access.
- Performance: The system must be performant and should be able to handle large number of student records.
- Usability: The system must be easy to use and navigate.

4.3. System Requirements

The student bio-data record management system requires following hardware and software requirements:

4.3.1. Hardware Requirements:

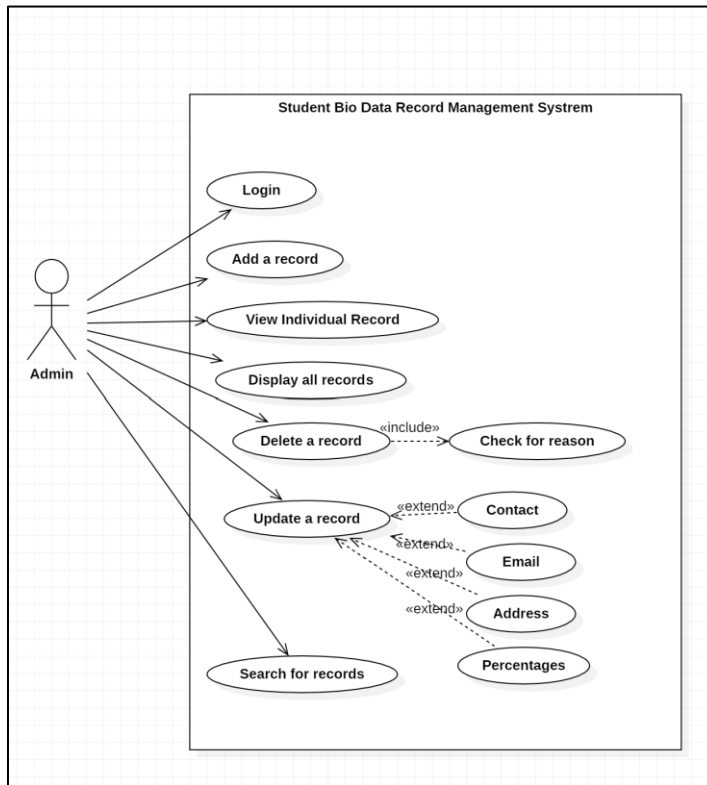
- An OS with minimum 2GB of RAM
- 1 GHz Processor
- At least 5GB of disk space.

4.3.2. Software Requirements:

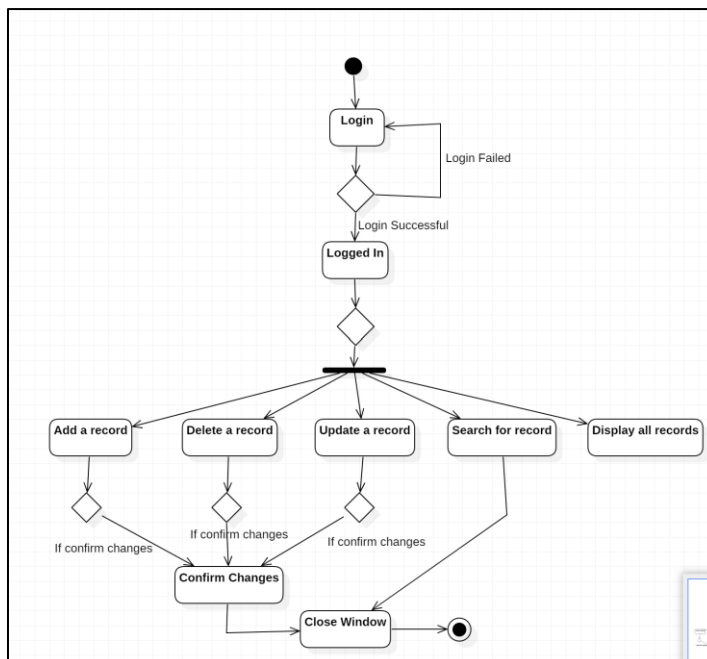
- VS Code or any other text editor
- Python 3.7 or later
- MySQL 5.7 or later
- XAMPP

5. System Design

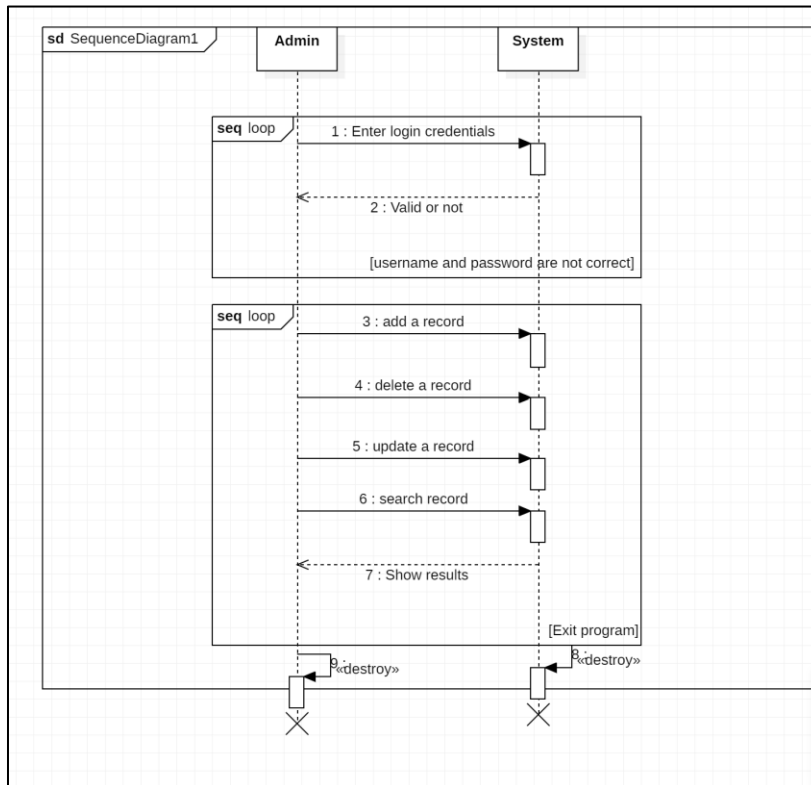
- Use Case Diagram



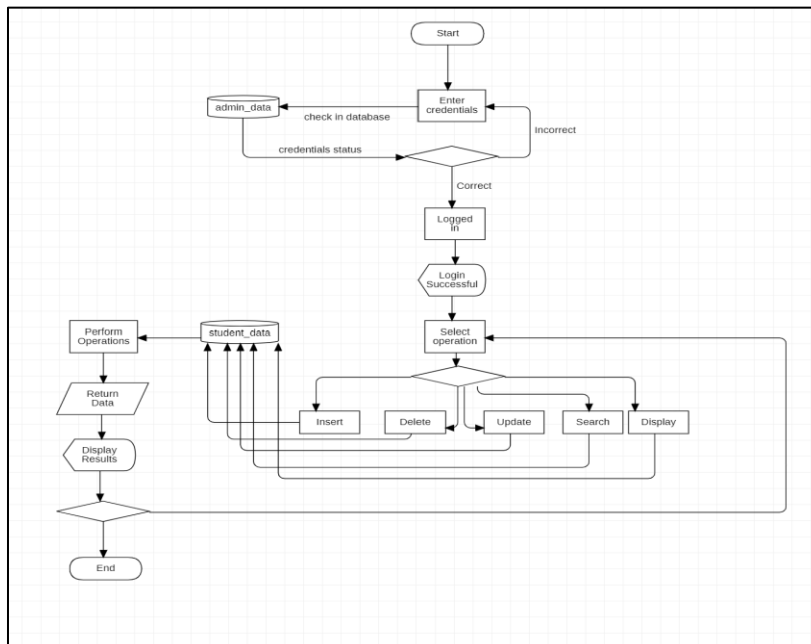
- Activity Diagram



- Sequence Diagram



- Flowchart



6. Literature Survey

Here is the literature survey of new technologies and tools learnt while building this project:

- **Python:**

Python is a general-purpose programming language that is known for its simplicity, readability, and flexibility. It is a popular choice for developing web applications, data science applications, and machine learning applications.

We used Python to develop the student bio data record management system. Python made it easy to write efficient and maintainable code. It also provided a wide range of libraries and tools that I could use to add functionality to the system.

Module used under Python 3.11 are:

1. tkinter
2. pytsx3
3. mysql.connector
4. PIL
5. datetime
6. tkcalendar
7. tkinter.filedialog
8. tkinter.messagebox

- **XAMPP Software:**

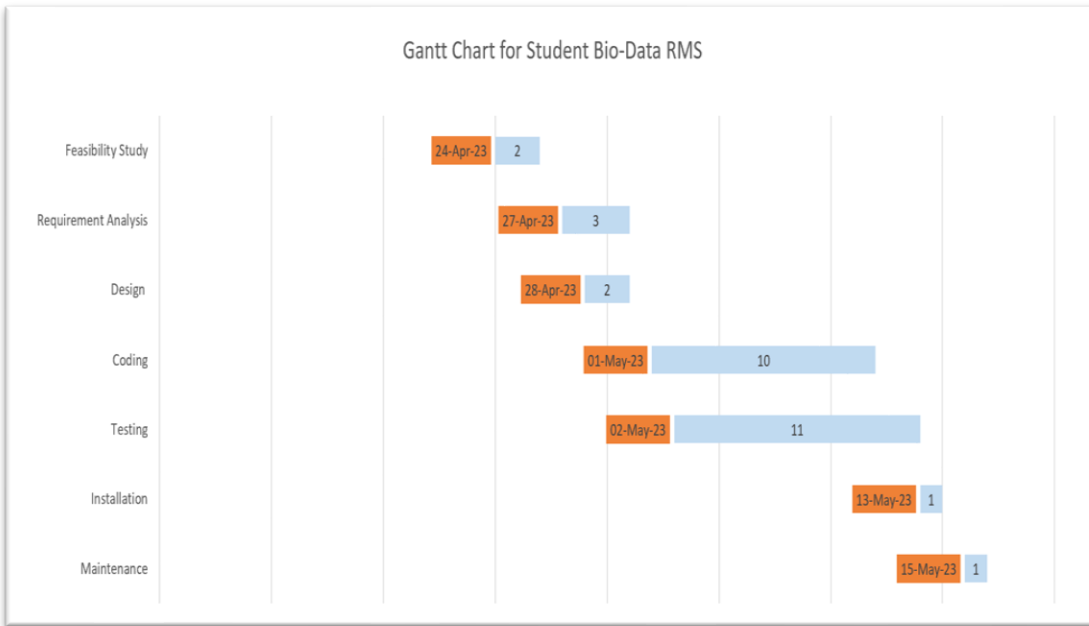
Xampp is a free and open-source cross-platform web server solution stack package developed by Apache Friends. It includes a complete set of tools for developing and running dynamic web applications, including Apache HTTP Server, MySQL database server, and PHP scripting language.

We used Xampp to host the student bio data record management system. Xampp made it easy to install and configure the necessary software, and it provided a convenient development environment for testing the system.

We created a database named “*python_mini_project*” with the following tables:

- **admin_data:** Used to store username and password of admin for login purposes.
- **student_data:** Used to store all the detailed record of student.

7. Gantt Chart



8. Module Description

1. admin.py

- **adminWindow():** **[1-05-2023]**
The adminWindow() function creates a login window for an admin panel. It includes input fields for username and password, a submit button to submit the form, and handles the login process. Upon successful login, it displays a success message and opens another window (the main admin panel). If the login fails, it displays an error message.
- **verifyAdmin():** **[1-05-2003]**
The verifyAdmin() function establishes a connection to a MySQL database named "python_mini_project" and a table named "admin_data". It verifies whether the provided admin username and password exist in the database. If the data is found, it returns True indicating a successful login; otherwise, it returns False. The function also handles potential errors that may occur during the database connection and ensures that the connection is closed properly.

2. main.py

- **window():** [1-05-2023]
This is a Python function that creates a GUI window using the Tkinter library, with a TreeView widget to display data in a tabular format. The window includes clickable labels that redirect to different screens where CRUD operations (insert, update, delete, sort, search, and display) can be performed.
- **database():** [1-05-2023]
The database() function establishes a connection to a MySQL database and initializes a cursor object. It handles potential errors during the connection process and sets up the global variables con and cur for database operations.
- **display():** [1-05-2023]
The display() function retrieves student records from a MySQL database and populates a treeview widget with the retrieved data. It also binds a double-click event to each record in the treeview to display the details of the selected record. Additionally, it shows a message box with the total number of records found.
- **detailDisplay():** [3-05-2023]
The detailDisplay(event) function retrieves student record details from a MySQL database and displays them in a Tkinter GUI window. It creates labels to represent different fields of the record and places them in the GUI window. Additionally, it retrieves an image from the database, crops and resizes it, and displays it in a canvas within the GUI window.
- **insert():** [4-05-2023]
The code defines a function insert() that creates a GUI window for inserting student biodata records. It includes various input fields and options for entering student information. The function also includes sub-functions for clearing the data and storing the data in a database. The GUI window includes labels, entry fields, comboboxes, buttons, and other components for capturing and displaying the student data.

- **delete():** [4-05-2023]
The delete() function creates a dialog box to delete a record from a database and asks for a reason for the deletion. It retrieves the selected item(s) from a tree view, prompts for a reason using a separate dialog box, and then deletes the selected record(s) from the database based on the provided reason
- **search():** [5-05-2023]
The search() function creates a search window with a label, text box, search button, and close button. It allows the user to enter keywords and performs a search query on a database, displaying the matching records in a tree view. It also provides an option for advanced search, which is currently under development.
- **update():** [7-05-2023]
This is a function called "update" that checks if a student record has been selected in a treeview. If there is a selection, it retrieves the SAP number from the selected item and passes it to a function called "updateDisplay" to display the details of the selected student. If there is no selection, it displays an information message asking the user to select a record.
- **updateDisplay():** [8-05-2023]
This function updates student details in a Tkinter GUI app. It fetches the record from a database, displays it on the GUI, and lets the user make updates. It includes functions to revert changes and update the database. The GUI has a fixed size and white background. The updated record is saved and a success message is shown.
- **help_option():** [9-05-2023]
This is a function called "help_option" that opens a help window in a graphical user interface. It provides descriptions for various features of a record management system, including insert, delete, search, sort, and display.
- **aboutUs():** [10-05-2023]
This is a function called "aboutUs" that creates an "About Us" window in a graphical user interface. It displays information about a person, including their name, contact details, department, SAP number, roll number, and semester. It also includes an image of the person.

9. System Implementation

1. admin.py

```
import tkinter as tk
from tkinter import *
from tkinter import ttk
import mysql.connector
import pytsx3
from PIL import Image, ImageTk
import main as m
from main import window

engine = pytsx3.init()

def verifyAdmin(username, password):
    """
    In this function i have established a connection to my database "python_mini_project" and table
    "admin_data"
    this function verifies weather the admin data is registered in database or not
    if data is there then success message is displayed else failure
    """

    try:
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="",
            database="python_mini_project"
        )
        print("Database Status: Connection Established Successfully")

        cursor = con.cursor()
        sql = "select * from admin_data where username=%s and password=%s"
        val = (username, password)
        cursor.execute(sql, val)
        result = cursor.fetchone()
        if result:
            print("Login Status: Login Success")
            return True
        else:
            print("Login Status: Login Failure")
            return False

    except:
        print("Database Status: There was an error connecting to the database")

    finally:
        if con:
            con.close()
```

```

print("Database Status: Connecton Revoked Successfully")
print()

def adminWindow():
'''
In this function i have created a login window for admin
which have input fields such as username and password as well as submit button to submit the
form
'''

# creating a window
admin = tk.Tk()
admin.title("SBMP Student Profile RMS: Admin Panel")
icon = PhotoImage(file = "Python Mini Project\\Images\\Project-Icon.png")
admin.iconphoto(False, icon)

# setting window on the center of the screen
window_width = 1000
window_height = 500

screen_width = admin.winfo_screenwidth()
screen_height = admin.winfo_screenheight()

x = (screen_width/2) - (window_width/2)
y = (screen_height/2) - (window_height/2)

admin.geometry('%dx%d+%d+%d' % (window_width, window_height, x, y))
admin.resizable(width=False, height=False)

admin.configure(bg='white')

# setting canvas frame - start
canvas = Canvas(admin, width=window_width, height=40, bg="blue")
canvas.create_text(470, 20, text="Admin Login", fill="white", font=("Helvetica", 12, "bold"))
canvas.pack()
# setting canvas frame - end

canvas = Canvas(width=600, height=145)
canvas.configure(bg="white")
canvas.place(x=350,y=150)
image = PhotoImage(file="Python Mini Project\\Images\\sbmp.png")
cropped_image = image.subsample(2, 2)
canvas.create_image(0, 0, anchor=NW, image=cropped_image)

# setting login fields --> Username & Password

username_label = tk.Label(admin, text="Username: ", bg="white",font=("Helvetica", 10, "bold"))
username_text = tk.Entry(admin, width=20, font=("Helvetica", 11), relief="groove", bd=2)
username_text.place(x=100,y=150)
username_label.place(x=20,y=150)

```

```
password_label = tk.Label(admin, text="Password: ", bg="white", font=("Helvetica", 10, "bold"))
password_text = tk.Entry(admin, show="*", width=20, font=("Helvetica", 11), relief="groove",
bd=2)
password_text.place(x=100,y=220)
password_label.place(x=20,y=220)
```

```
def login():
"""
this function displays the label on screen to notify the user weather it is a success or a failure
the result is retrned to "verifyAdmin()" and it checks wether the admin data exists or not
"""
result = verifyAdmin(username_text.get(), password_text.get())
if result:
status_label = tk.Label(admin, text="You have been successfully logged in!", bg="green",
fg="white")
status_label.place(x=50,y=90)
admin.update()
engine.say("Login Successful")
engine.runAndWait()
admin.after(500, admin.destroy)
m.window()

else:
status_label = tk.Label(admin, text="There was an error logging you in!", bg="red", fg="white")
status_label.place(x=60,y=90)
admin.update()
engine.say("Login failed please enter correct id or password")
engine.runAndWait()

# adding a submit button
submit_button = tk.Button(admin, text="Submit", command=login, width=30, bg="blue",
fg="white") # the command attribute will run the "login()" function
submit_button.place(x=40,y=285)

admin.mainloop()

# Calling the function
adminWindow()
```

2. main.py

```
import tkinter
from tkinter import *
from tkinter import ttk
import tkinter.messagebox as tkMessageBox
from tkinter.filedialog import askopenfilename
from tkinter import filedialog
from PIL import Image, ImageTk
```

```

import mysql.connector
import datetime
from tkcalendar import Calendar, DateEntry
import re
import io
from io import BytesIO
import tkinter.messagebox as mbox
from tkinter.font import Font
import pytsx3

engine = pytsx3.init()

def database():
    try:
        global con, cur
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="",
            database="python_mini_project"
        )
        print("Connection Established Successfully")

        cur = con.cursor()

    except:
        print("Database Status: There was an error")

def detailDisplay(event):
    database()
    detail_display = Tk()
    window_width = 700
    window_height = 700
    screen_width = detail_display.winfo_screenwidth()
    screen_height = detail_display.winfo_screenheight()
    x = (screen_width/2) - (window_width/2)
    y = (screen_height/2) - (window_height/2)
    detail_display.geometry('%dx%d+%d+%d' % (window_width, window_height, x, y-50))
    detail_display.resizable(width=False, height=False)
    detail_display.configure(bg='white')
    item = tree.selection()[0]
    sap_no = tree.item(item, "values")[3]
    cur.execute("SELECT * FROM student_data WHERE sap_no=%s", (sap_no,))
    record = cur.fetchone()

    if record:
        detail_display.title(f'{record[2]} Record Details')

    # Add labels to display the record details

```

```

roll_label = Label(detail_display, text="Roll No", font=("Helvetica", 10, "bold"), bg="white")
roll_label.place(x=370, y=50)
Label(detail_display, text=record[2], bg="white", font=("Helvetica", 10)).place(x=430,y=50)

sap_label = Label(detail_display, text="Sap No", font=("Helvetica", 10, "bold"), bg="white")
sap_label.place(x=150, y=50)
Label(detail_display, text=record[1],font=("Helvetica", 10), bg="white").place(x=210,y=50)

year_of_admission_label = Label(detail_display, text="Year of Admission: ", bg="white",
font=("Helvetica", 10, "bold"))
year_of_admission_label.place(x=20,y=120)
Label(detail_display, text=record[3],font=("Helvetica", 10), bg="white").place(x=150,y=120)

type_of_admission_label = Label(detail_display, text="Type of Admission: ", bg="white",
font=("Helvetica", 10, "bold"))
type_of_admission_label.place(x=20,y=155)
Label(detail_display, text=record[4], font=("Helvetica", 10), bg="white").place(x=150,y=155)

branch_label = Label(detail_display, text="Branch: ", bg="white", font=("Helvetica", 10, "bold"))
branch_label.place(x=20,y=210)
Label(detail_display, text=record[5], font=("Helvetica", 10), bg="white").place(x=150,y=210)

name_label = Label(detail_display, text="Student Name: ", bg="white", font=("Helvetica", 10,
"bold"))
name_label.place(x=20, y=245)
Label(detail_display, text=record[6], font=("Helvetica", 10), bg="white").place(x=150, y=245)

contact_label = Label(detail_display, text="Contact No.: ", bg="white", font=("Helvetica", 10,
"bold"))
contact_label.place(x=20,y=290)
Label(detail_display, text=record[8], font=("Helvetica", 10), bg="white").place(x=150,y=315)
Label(detail_display, text=record[7], font=("Helvetica", 10), bg="white").place(x=150,y=290)

email_label = Label(detail_display, text="Email: ", bg="white", font=("Helvetica", 10, "bold"))
email_label.place(x=20,y=360)
Label(detail_display, text=record[10], font=("Helvetica", 10), bg="white").place(x=150,y=390)
Label(detail_display, text=record[9], font=("Helvetica", 10), bg="white").place(x=150,y=360)

date_of_birth_label = Label(detail_display, text="Date of Birth:", bg="white", font=("Helvetica", 10,
"bold"))
date_of_birth_label.place(x=20,y=445)
Label(detail_display, text=record[11], font=("Helvetica", 10), bg="white").place(x=150, y=445)

address_label = Label(detail_display, text="Current Address: ", bg="white", font=("Helvetica", 10,
"bold"))
address_label.place(x=20, y=505)
addr = Label(detail_display, text=record[12], font=("Helvetica", 10), bg="white", width=20,
height=2, wraplength=160, anchor="w", justify="left")
addr.place(x=150, y=505)

```



```
achedemic_label = Label(detail_display, text="Achedemic Record: ", bg="white", font=("Helvetica",
10, "bold"))
achedemic_label.place(x=20, y=560)
```

```
l1 = Label(detail_display, text="Exam ", bg="white", font=("Helvetica", 10)).place(x=20, y=580)
l2 = Label(detail_display, text="Year of Passing", bg="white", font=("Helvetica", 10)).place(x=80,
y=580)
l3 = Label(detail_display, text="Percentage(%)", bg="white", font=("Helvetica",
10)).place(x=200,y=580)
l4 = Label(detail_display, text="SSC", bg="white", font=("Helvetica", 10)).place(x=20, y=620)
l5 = Label(detail_display, text="HSC", bg="white", font=("Helvetica", 10)).place(x=20, y=660)
```

```
Label(detail_display, text=record[13], font=("Helvetica", 10), bg="white").place(x=80,y=620)
Label(detail_display, text=record[14], font=("Helvetica", 10), bg="white").place(x=200,y=620)
Label(detail_display, text=record[15], font=("Helvetica", 10), bg="white").place(x=80,y=660)
Label(detail_display, text=record[16], font=("Helvetica", 10), bg="white").place(x=200,y=660)
```

```
sem_label = Label(detail_display, text="Semesters", bg="white", font=("Helvetica", 10,
"bold")).place(x=400, y=400)
year_label = Label(detail_display, text="Year", bg="white", font=("Helvetica", 10,
"bold")).place(x=500, y=400)
percent_label = Label(detail_display, text="Percentages(%)", bg="white", font=("Helvetica", 10,
"bold")).place(x=580, y=400)
```

```
sem1_label = Label(detail_display, text="Semester 1", bg="white", font=("Helvetica",
10)).place(x=400, y=440)
sem2_label = Label(detail_display, text="Semester 2", bg="white", font=("Helvetica",
10)).place(x=400, y=460)
sem3_label = Label(detail_display, text="Semester 3", bg="white", font=("Helvetica",
10)).place(x=400, y=480)
sem4_label = Label(detail_display, text="Semester 4", bg="white", font=("Helvetica",
10)).place(x=400, y=500)
sem5_label = Label(detail_display, text="Semester 5", bg="white", font=("Helvetica",
10)).place(x=400, y=520)
sem6_label = Label(detail_display, text="Semester 6", bg="white", font=("Helvetica",
10)).place(x=400, y=540)
```

```
Label(detail_display, text=record[18], font=("Helvetica", 10), bg="white").place(x=500, y=440)
Label(detail_display, text=record[19], font=("Helvetica", 10), bg="white").place(x=500, y=460)
Label(detail_display, text=record[20], font=("Helvetica", 10), bg="white").place(x=500, y=480)
Label(detail_display, text=record[21], font=("Helvetica", 10), bg="white").place(x=500, y=500)
Label(detail_display, text=record[22], font=("Helvetica", 10), bg="white").place(x=500, y=520)
Label(detail_display, text=record[23], font=("Helvetica", 10), bg="white").place(x=500, y=540)
Label(detail_display, text=record[24], font=("Helvetica", 10), bg="white").place(x=580, y=440)
Label(detail_display, text=record[25], font=("Helvetica", 10), bg="white").place(x=580, y=460)
Label(detail_display, text=record[26], font=("Helvetica", 10), bg="white").place(x=580, y=480)
Label(detail_display, text=record[27], font=("Helvetica", 10), bg="white").place(x=580, y=500)
Label(detail_display, text=record[28], font=("Helvetica", 10), bg="white").place(x=580, y=520)
Label(detail_display, text=record[29], font=("Helvetica", 10), bg="white").place(x=580, y=540)
```

```

image_data = record[30]

# Create an image from the retrieved data
image = Image.open(io.BytesIO(image_data))

# Calculate the aspect ratio of the image
image_width, image_height = image.size
aspect_ratio = image_width / image_height

# Determine the dimensions for cropping
canvas_width = 150
canvas_height = 150
if aspect_ratio > 1:
    # Landscape image, crop the width
    crop_width = image_height * aspect_ratio
    crop_height = image_height
else:
    # Portrait image or square image, crop the height
    crop_width = image_width
    crop_height = image_width / aspect_ratio
    crop_x = (image_width - crop_width) / 2
    crop_y = (image_height - crop_height) / 2

# Crop the image to desired dimensions
cropped_image = image.crop((crop_x, crop_y, crop_x + crop_width, crop_y + crop_height))

# Resize the cropped image to fit the canvas
resized_image = cropped_image.resize((canvas_width, canvas_height), Image.ANTIALIAS)

# Display the resized image in a canvas
image_upload = Canvas(detail_display, width=canvas_width, height=canvas_height)
image_upload.place(x=500, y=120)
image_object = ImageTk.PhotoImage(resized_image)
image_upload.image = image_object # Keep a reference to the image object
image_upload.create_image(int(canvas_width / 2), int(canvas_height / 2), image=image_object)
image_upload.update()

def display():
    database()
    count = 0
    tree.delete(*tree.get_children())
    cur.execute("ALTER TABLE student_data AUTO_INCREMENT = 1")
    cur.execute("select sr_no, stu_name, roll_no, sap_no, branch, present_year from student_data")
    fetch = cur.fetchall()
    for row in fetch:
        tree.insert("", "end", values=row)
    tree.bind("<Double-Button-1>", detailDisplay)
    count += 1

tkMessageBox.showinfo("Total Records", f"Total Records Found: {count}")

```

```

cur.close()
con.close()

def insert():
    """
    This function creates a GUI window for inserting student biodata records with various input fields
    and options.
    """

def clearData():
    image = canvas_image.delete(0, ALL)
    sap_no = sap_text.delete(0,END)
    roll_no = roll_text.delete(0,END)
    yoa = year_of_admission_option.delete(0,END)
    branch = branch_text.delete(0,END)
    stu_name = name_text.delete(0,END)
    contact1 = contact_text1.delete(0,END)
    contact2 = contact_text2.delete(0,END)
    email1 = email_text1.delete(0,END)
    email2 = email_text2.delete(0,END)
    address = address_text.delete("1.0",END)
    ssc_pass_year = tf1.delete(0,END)
    ssc_marks = tf2.delete(0,END)
    hsc_pass_year = tf3.delete(0,END)
    hsc_marks = tf4.delete(0,END)
    sem1 = year1_entry.delete(0,END)
    sem2 = year2_entry.delete(0,END)
    sem3 = year3_entry.delete(0,END)
    sem4 = year4_entry.delete(0,END)
    sem5 = year5_entry.delete(0,END)
    sem6 = year6_entry.delete(0,END)
    sem1_percent = percentage1_entry.delete(0,END)
    sem2_percent = percentage2_entry.delete(0,END)
    sem3_percent = percentage3_entry.delete(0,END)
    sem4_percent = percentage4_entry.delete(0,END)
    sem5_percent = percentage5_entry.delete(0,END)
    sem6_percent = percentage6_entry.delete(0,END)

def storeData():
    """
    The function connects to a database, inserts a record, commits the changes, and closes the
    database.
    """
    try:
        database()

        sap_no = sap_text.get()
        if not sap_no:
            tkMessageBox.showinfo("Alert", "Please Enter SAP No")
        return False

```

```

cur.execute("SELECT COUNT(*) FROM student_data WHERE sap_no = %s", (sap_no,))
count = cur.fetchone()[0]
if count > 0:
tkMessageBox.showinfo("Alert", "SAP No already exists")
return False

image_binary = open(filename, 'rb').read()
if not image_binary:
tkMessageBox.showinfo("Alert", "Please Select a Image")
return False

sap_no = sap_text.get()
if not sap_no:
tkMessageBox.showinfo("Alert", "Please Enter SAP No")
return False

roll_no = roll_text.get()
if not roll_no:
tkMessageBox.showinfo("Alert", "Please Enter Roll No")
return False

yoa = year_of_admission_option.get()
if not yoa:
tkMessageBox.showinfo("Alert", "Please Select Year of Admission")
return False

toa = selected_value.get()

branch = branch_text.get()
if not branch:
tkMessageBox.showinfo("Alert", "Please Select A Branch")
return False

stu_name = name_text.get()
if not stu_name:
tkMessageBox.showinfo("Alert", "Please Enter Stuent Name")
return False

contact1 = contact_text1.get()
contact2 = contact_text2.get()
if not contact1 and not contact2:
tkMessageBox.showinfo("Alert", "Please Enter Atleast 1 Contact Info.")
return False

email1 = email_text1.get()
email2 = email_text2.get()
if not email1 and not email2:
tkMessageBox.showinfo("Alert", "Please Enter Atleast 1 Email ID")

if validate_email_input:

```



```

con.close()
print("Database Closed Successfully")
print()

insert = Tk()
insert.title("Add Student")

# setting window on the center of the screen
window_width = 700
window_height = 700
screen_width = insert.winfo_screenwidth()
screen_height = insert.winfo_screenheight()
x = (screen_width/2) - (window_width/2)
y = (screen_height/2) - (window_height/2)
insert.geometry('%dx%d+%d+%d' % (window_width, window_height, x, y-50))
insert.resizable(width=False, height=False)
insert.configure(bg='white')

def choose_image():
    global canvas_image, image, filename
    canvas_image = Canvas(width=150, height=150)
    canvas_image.place(x=500,y=120)
    filename = filedialog.askopenfilename(title="Select file", filetypes=(("PNG files", "*.png"), ("JPEG files", "*.jpg"), ("JPG files", "*.jpg")))
    print("Selected file:", filename) # Add this line to print the filename
    # Load the image
    if filename:
        try:
            image = Image.open(filename)
        except Exception as e:
            print("Error loading image:", e) # Add this line to print any errors
        return
    image_width, image_height = image.size
    canvas_width, canvas_height = canvas_image.winfo_width(), canvas_image.winfo_height()
    # Calculate the scaling factor
    scale = min(canvas_width/image_width, canvas_height/image_height)
    new_width = int(image_width*scale)
    new_height = int(image_height*scale)
    # Resize the image
    resized_image = image.resize((new_width, new_height), Image.ANTIALIAS)
    image_object = ImageTk.PhotoImage(resized_image)
    # Add the image to the canvas
    canvas_image.delete("all") # Remove any previously added image
    canvas_image.create_image(canvas_width/2, canvas_height/2, image=image_object)
    canvas_image.image = image_object

# Creating a form
canvas_image_upload = Canvas(insert, width=150, height=150)
canvas_image_upload.place(x=500,y=120)

```

```

img_label = Label(insert, text="Upload Photo: ", bg="white", font=("Helvetica", 10, "bold"))
img_label.place(x=400, y=120)

# Create a button for selecting the image
upload_button = Button(insert, text="Choose Photo", command=choose_image)
upload_button.place(x=530, y=290)

sap_label = Label(insert, text="Sap No", font=("Helvetica", 10, "bold"), bg="white")
sap_label.place(x=50, y=50)
sap_text = Entry(insert, width=20, font=("Helvetica", 10), relief="groove", bd=2)
sap_text.place(x=110, y=50)

roll_label = Label(insert, text="Roll No", font=("Helvetica", 10, "bold"), bg="white")
roll_label.place(x=330, y=50)
roll_text = Entry(insert, width=20, font=("Helvetica", 10), relief="groove", bd=2)
roll_text.place(x=400, y=50)

current_year = datetime.datetime.now().year
year = [i for i in range(2005, current_year)]
year_of_admission_label = Label(insert, text="Year of Admission: ", bg="white", font=("Helvetica", 10, "bold"))
year_of_admission_option = ttk.Combobox(insert, width=20, font=("Helvetica", 10), values=year, state="readonly")
year_of_admission_option.place(x=150, y=120)
year_of_admission_label.place(x=20, y=120)

selected_value = StringVar(value="First Year")

def update_selection(selection):
    selected_value.set(selection)

type_of_admission_label = Label(insert, text="Type of Admission: ", bg="white", font=("Helvetica", 10, "bold"))
type_of_admission_option1 = Radiobutton(insert, text="First Year", variable=selected_value, value="First Year", bg="white", command=lambda: update_selection("First Year"))
type_of_admission_option2 = Radiobutton(insert, text="Direct Second Year", variable=selected_value, value="Direct Second Year", bg="white", command=lambda: update_selection("Direct Second Year"))
type_of_admission_option1.place(x=150, y=155)
type_of_admission_option2.place(x=150, y=176)
type_of_admission_label.place(x=20, y=155)

branch_options = ["Information Technology", "Computer Engineering", "Mechanical Engineering", "Civil Engineering", "Electrical Engineering", "Plastic Engineering", "Chemical Engineering", "Electronics & Telecommunication Engineering"]
branch_label = Label(insert, text="Branch: ", bg="white", font=("Helvetica", 10, "bold"))
branch_text = ttk.Combobox(insert, width=20, font=("Helvetica", 10), values=branch_options, state="readonly")

```

```

branch_text.place(x=150,y=210)
branch_label.place(x=20,y=210)

def validate_name_input(name):
    if name == "":
        return True
    name_regex = "^[A-Za-z\s]{1,}[\.]{0,1}[A-Za-z\s]{0,}$"
    if re.match(name_regex, name):
        return True
    else:
        return False
    validate_name = (insert.register(validate_name_input), '%P')
    name_label = Label(insert, text="Student Name: ", bg="white", font=("Helvetica", 10, "bold"))
    name_text = Entry(insert, width=23, font=("Helvetica", 10), relief="groove", bd=2, validate='all',
        validatecommand=validate_name)
    name_text.place(x=150, y=245)
    name_label.place(x=20, y=245)

def validate_contact_input(contact):
    contact_regex = r"\d{10}"
    if re.match(contact_regex, contact):
        return True
    else:
        return False
    validate_contact = (insert.register(validate_contact_input), '%P')
    contact_label = Label(insert, text="Contact No.: ", bg="white", font=("Helvetica", 10, "bold"))
    contact_text1 = Entry(insert, width=23, font=("Helvetica", 10), relief="groove", bd=2,
        validate='focusout', validatecommand=validate_contact)
    contact_text1.place(x=150,y=290)
    contact_text2 = Entry(insert, width=23, font=("Helvetica", 10), relief="groove", bd=2,
        validate='focusout', validatecommand=validate_contact)
    contact_text2.place(x=150,y=315)
    contact_label.place(x=20,y=290)

def validate_email_input(email):
    email_regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(email_regex, email):
        return True
    else:
        return False
    validate_email = (insert.register(validate_email_input), '%P')
    email_label = Label(insert, text="Email: ", bg="white", font=("Helvetica", 10, "bold"))
    email_text1 = Entry(insert, width=23, font=("Helvetica", 10), relief="groove", bd=2,
        validate='focusout', validatecommand=validate_email)
    email_text1.place(x=150,y=360)
    email_text2 = Entry(insert, width=23, font=("Helvetica", 10), relief="groove", bd=2,
        validate='focusout', validatecommand=validate_email)
    email_text2.place(x=150,y=390)
    email_label.place(x=20,y=360)

```



```
date_of_birth_label = Label(insert, text="Date of Birth:", bg="white", font=("Helvetica", 10, "bold"))
date_of_birth_label.place(x=20,y=445)
date_of_birth = DateEntry(insert, width=12, background='darkblue', foreground='white',
borderwidth=2, state="readonly", date_pattern="dd-mm-yyyy")
date_of_birth.place(x=150, y=445)
```

```
address_label = Label(insert, text="Current Address: ", bg="white", font=("Helvetica", 10, "bold"))
address_label.place(x=20, y=505)
address_text = Text(insert, width=23, font=("Helvetica", 10), relief="groove", bd=2, height=2,
wrap="word")
address_text.place(x=150, y=505)
```

```
achedemic_label = Label(insert, text="Achedemic Record: ", bg="white", font=("Helvetica", 10,
"bold"))
achedemic_label.place(x=20, y=560)
```

```
l1 = Label(insert, text="Exam ", bg="white", font=("Helvetica", 10)).place(x=20, y=580)
l2 = Label(insert, text="Year of Passing", bg="white", font=("Helvetica", 10)).place(x=80, y=580)
l3 = Label(insert, text="Percentage(%)", bg="white", font=("Helvetica", 10)).place(x=200,y=580)
l4 = Label(insert, text="SSC", bg="white", font=("Helvetica", 10)).place(x=20, y=620)
l5 = Label(insert, text="HSC", bg="white", font=("Helvetica", 10)).place(x=20, y=660)
```

```
tf1 = Entry(insert, width=10, font=("Helvetica", 10), relief="groove", bd=2)
tf1.place(x=80,y=620)
tf2 = Entry(insert, width=10, font=("Helvetica", 10), relief="groove", bd=2)
tf2.place(x=200,y=620)
tf3 = Entry(insert, width=10, font=("Helvetica", 10), relief="groove", bd=2)
tf3.place(x=80,y=660)
tf4 = Entry(insert, width=10, font=("Helvetica", 10), relief="groove", bd=2)
tf4.place(x=200,y=660)
```

```
sem_label = Label(insert, text="Semesters", bg="white", font=("Helvetica", 10, "bold")).place(x=350,
y=400)
```

```
sem1_label = Label(insert, text="Semester 1", bg="white", font=("Helvetica", 10)).place(x=350,
y=440)
sem2_label = Label(insert, text="Semester 2", bg="white", font=("Helvetica", 10)).place(x=350,
y=460)
sem3_label = Label(insert, text="Semester 3", bg="white", font=("Helvetica", 10)).place(x=350,
y=480)
sem4_label = Label(insert, text="Semester 4", bg="white", font=("Helvetica", 10)).place(x=350,
y=500)
sem5_label = Label(insert, text="Semester 5", bg="white", font=("Helvetica", 10)).place(x=350,
y=520)
sem6_label = Label(insert, text="Semester 6", bg="white", font=("Helvetica", 10)).place(x=350,
y=540)
```

```
year_label = Label(insert, text="Year, Month", bg="white", font=("Helvetica", 10,
"bold")).place(x=460, y=400)
```

```

year1_entry = Entry(insert, text="M1", bg="white", font=("Helvetica", 10), width=9, relief="groove",
bd=2)
year1_entry.place(x=470, y=440)
year2_entry = Entry(insert, text="M2", bg="white", font=("Helvetica", 10), width=9, relief="groove",
bd=2)
year2_entry.place(x=470, y=460)
year3_entry = Entry(insert, text="M3", bg="white", font=("Helvetica", 10), width=9, relief="groove",
bd=2)
year3_entry.place(x=470, y=480)
year4_entry = Entry(insert, text="M4", bg="white", font=("Helvetica", 10), width=9, relief="groove",
bd=2)
year4_entry.place(x=470, y=500)
year5_entry = Entry(insert, text="M5", bg="white", font=("Helvetica", 10), width=9, relief="groove",
bd=2)
year5_entry.place(x=470, y=520)
year6_entry = Entry(insert, text="M6", bg="white", font=("Helvetica", 10), width=9, relief="groove",
bd=2)
year6_entry.place(x=470, y=540)

percent_label = Label(insert, text="%", bg="white", font=("Helvetica", 10, "bold")).place(x=570,
y=400)
percentage1_entry = Entry(insert, text="Percentage 1", bg="white", font=("Helvetica", 10), width=7,
relief="groove", bd=2)
percentage1_entry.place(x=560, y=440)
percentage2_entry = Entry(insert, text="Percentage 2", bg="white", font=("Helvetica", 10), width=7,
relief="groove", bd=2)
percentage2_entry.place(x=560, y=460)
percentage3_entry = Entry(insert, text="Percentage 3", bg="white", font=("Helvetica", 10), width=7,
relief="groove", bd=2)
percentage3_entry.place(x=560, y=480)
percentage4_entry = Entry(insert, text="Percentage 4", bg="white", font=("Helvetica", 10), width=7,
relief="groove", bd=2)
percentage4_entry.place(x=560, y=500)
percentage5_entry = Entry(insert, text="Percentage 5", bg="white", font=("Helvetica", 10), width=7,
relief="groove", bd=2)
percentage5_entry.place(x=560, y=520)
percentage6_entry = Entry(insert, text="Percentage 6", bg="white", font=("Helvetica", 10), width=7,
relief="groove", bd=2)
percentage6_entry.place(x=560, y=540)

submit_record = Button(insert, text="Submit", command=storeData, width=10, bg="blue",
fg="white", font=("Helvetica", 10, "bold"))
submit_record.place(x=400, y=600)

clear_record = Button(insert, text="Clear", command=clearData, width=10, bg="blue", fg="white",
font=("Helvetica", 10, "bold"))
clear_record.place(x=500, y=600)

insert.mainloop()

```

```

def delete():
    """
    This function creates a frame to delete a record from the database and asks for a reason.
    """
    def delete_student():
        database()
        reason = delete_text.get('1.0', 'end-1c')
        if not reason:
            tkMessageBox.showinfo("Alert", "Please fill the mandatory field")
            return False

        selected_items = tree.selection() # Get the selected items (rows)
        for item in selected_items:
            sap_id = tree.item(item)['values'][3] # Retrieve the Sap Id (index 3) from the item
            cur.execute("DELETE FROM student_data WHERE sap_no=%s", (sap_id,))

            con.commit()
            display()
            tkMessageBox.showinfo("", f"{sap_id} Deleted")

    def delete_reason():
        global delete_text
        delete_reason = Tk()
        delete_reason.geometry("300x200")
        delete_reason.resizable(False, False)
        delete_reason.configure(background="#1338BE")
        delete_label = Label(delete_reason, text="Please Enter A Reason", font=("Helvetica", 11, "bold"),
            bg="#1338BE", fg="white").place(x=70, y=30)
        delete_text = Text(delete_reason, width=30, height=3, font=("Helvetica", 10), wrap="word",
            relief="solid")
        delete_text.place(x=50, y=70)
        delete_msg = delete_text.get("1.0", "end-1c")
        confirm_button = Button(delete_reason, text="Confirm", width=10, height=1, font=("Helvetica",
            10), bg="white", fg="green", command=delete_student).place(x=50, y=150)
        cancel_button = Button(delete_reason, text="Cancel", width=10, height=1, font=("Helvetica", 10),
            bg="white", fg="red", command=delete_reason.destroy).place(x=160, y=150)
        delete_reason.mainloop()

    if not tree.selection():
        tkMessageBox.showinfo("", "Please Select a Student Record to Delete")
    else:
        delete_reason()

def search():
    """
    This function creates a search window with a label, text box, search button, and close button, and
    performs a search query on a database based on the keywords entered by the user.
    """
    search_window = Tk()
    global SEARCH

```

```

SEARCH = StringVar()
# create a new window
search_window.title("Search Data")
search_window.resizable(width=False, height=False)
search_window.configure(background="#1338BE")
# create a label and text box for the user to enter search keywords
search_label = Label(search_window, text="Enter keywords to search:", bg="#1338BE", fg="white")
search_label.pack(side=LEFT, padx=5, pady=5)
search_entry = Entry(search_window, textvariable=SEARCH)
search_entry.pack(side=LEFT, padx=5, pady=5)

def advanced_search():
    """
    The function "advanced_search" is used to retrieve data such as
    Marks Range or semester or HSC or SSC results also used to retrieve results of semesters
    eg:
    [] > [] or [] < []
    [] >= [] or [] <= []
    """
    tkinterMessageBox.showinfo("Alert", "This feature is under progress!")

# create a function to search for the keywords entered by the user
def search_keyword():
    try:
        database()
        #checking search text is empty or not
        if SEARCH.get() != "":
            #clearing current display data
            tree.delete(*tree.get_children())

            #select query with where clause
            cur.execute("""SELECT sr_no, stu_name, roll_no, sap_no, branch, present_year FROM student_data
            WHERE
            sr_no = %s OR
            stu_name LIKE %s OR
            roll_no LIKE %s OR
            sap_no LIKE %s OR
            branch LIKE %s OR
            present_year LIKE %s""",
            (SEARCH.get(), '%' + SEARCH.get() + '%', '%' + SEARCH.get() + '%', '%' + SEARCH.get() + '%',
            '%' + SEARCH.get() + '%', '%' + SEARCH.get() + '%'))

            #fetch all matching records
            fetch = cur.fetchall()

            #loop for displaying all records into GUI
            if len(fetch) == 0 or SEARCH.get() == "" or SEARCH.get() is None:
                tkinterMessageBox.showinfo("Alert", "Record Not Found")
            else:
                for data in fetch:

```

```

tree.insert("", 'end', values=data)
# set foreground color for inserted items
if SEARCH.get().casefold() in data[1].casefold() or data[2].casefold() or data[3].casefold() or
data[4].casefold() or data[5].casefold():
tree.tag_configure('found', foreground='red')
tree.item(tree.get_children()[-1], tags=('found',))
except Exception as e:
print("Database error => ", e)

finally:
if con:
con.close()
print("Database closed successfully")
print()

# create a button to perform the search
search_button = Button(search_window, text="Search", command=search_keyword, bg="white")
search_button.pack(side=LEFT, padx=5, pady=5)

# create a button to close the search window
close_button = Button(search_window, text="Close", command=search_window.destroy,
bg="white")
close_button.pack(side=LEFT, padx=5, pady=5)

adv_button = Button(search_window, text="Advanced Search", command=advanced_search,
bg="white")
adv_button.pack(side=LEFT, padx=5, pady=5)

search_window.mainloop()

def sort():
'''
In this function i have created a login window for admin
which have input fields such as username and password as well as submit button to submit the form
'''
tkMessageBox.showinfo("Alert", "This module is not been completed yet")

def updateDisplay(event):
database()
update = Tk()
window_width = 700
window_height = 700
screen_width = update.winfo_screenwidth()
screen_height = update.winfo_screenheight()
x = (screen_width/2) - (window_width/2)
y = (screen_height/2) - (window_height/2)
update.geometry('%dx%d+%d+%d' % (window_width, window_height, x, y-50))
update.resizable(width=False, height=False)
update.configure(bg='white')
item = tree.selection()[0]

```

```

sap_no = tree.item(item, "values")[3]
cur.execute("SELECT * FROM student_data WHERE sap_no=%s", (sap_no,))
record = cur.fetchone()

def revert_info():
    database()
    cur.execute("SELECT * FROM student_data WHERE sap_no=%s", (sap_no,))
    record = cur.fetchone()

    if record:
        # Clear the entry fields
        contact1_entry.delete(0, END)
        contact2_entry.delete(0, END)
        email1_entry.delete(0, END)
        email2_entry.delete(0, END)
        address_entry.delete("1.0", END)
        y1.delete(0,END)
        y2.delete(0,END)
        y3.delete(0,END)
        y4.delete(0,END)
        y5.delete(0,END)
        y6.delete(0,END)
        p1.delete(0,END)
        p2.delete(0,END)
        p3.delete(0,END)
        p4.delete(0,END)
        p5.delete(0,END)
        p6.delete(0,END)

        # Restore the original values
        contact1_entry.insert(END, record[7])
        contact2_entry.insert(END, record[8])
        email1_entry.insert(END, record[9])
        email2_entry.insert(END, record[10])
        address_entry.insert(END, record[12])
        y1.insert(END, record[18])
        y2.insert(END, record[19])
        y3.insert(END, record[20])
        y4.insert(END, record[21])
        y5.insert(END, record[22])
        y6.insert(END, record[23])
        p1.insert(END, record[24])
        p2.insert(END, record[25])
        p3.insert(END, record[26])
        p4.insert(END, record[27])
        p5.insert(END, record[28])
        p6.insert(END, record[29])
        tkMessageBox.showinfo("Success", "Reverted to original values")

    con.close()

```

```

def update_info():
    database()
    cur.execute("SELECT * FROM student_data WHERE sap_no=%s", (sap_no,))
    record = cur.fetchone()
    if record:
        new_address = address_entry.get("1.0", "end-1c")
        new_contact1 = contact1_entry.get()
        new_contact2 = contact2_entry.get()
        new_email1 = email1_entry.get()
        new_email2 = email2_entry.get()
        new_y1 = y1.get()
        new_y2 = y2.get()
        new_y3 = y3.get()
        new_y4 = y4.get()
        new_y5 = y5.get()
        new_y6 = y6.get()
        new_p1 = p1.get()
        new_p2 = p2.get()
        new_p3 = p3.get()
        new_p4 = p4.get()
        new_p5 = p5.get()
        new_p6 = p6.get()
        cur.execute("UPDATE student_data SET address=%s, contact1=%s, contact2=%s, email1=%s,
        email2=%s, sem1=%s, sem2=%s, sem3=%s, sem4=%s, sem5=%s, sem6=%s, sem1_percent=%s,
        sem2_percent=%s, sem3_percent=%s, sem4_percent=%s, sem5_percent=%s, sem6_percent=%s
        WHERE sap_no=%s",
        (new_address, new_contact1, new_contact2, new_email1, new_email2, new_y1, new_y2, new_y3,
        new_y4, new_y5, new_y6, new_p1, new_p2, new_p3, new_p4, new_p5, new_p6, sap_no))
        con.commit()
        tkMessageBox.showinfo("Success", "Record Updated Successfully")

    if record:
        update.title(f"{record[2]} Record Details")

# Add labels to display the record details

roll_label = Label(update, text="Roll No", font=("Helvetica", 10, "bold"), bg="white")
roll_label.place(x=370, y=50)
Label(update, text=record[2], bg="white", font=("Helvetica", 10)).place(x=430, y=50)

sap_label = Label(update, text="Sap No", font=("Helvetica", 10, "bold"), bg="white")
sap_label.place(x=150, y=50)
Label(update, text=record[1], font=("Helvetica", 10), bg="white").place(x=210, y=50)

year_of_admission_label = Label(update, text="Year of Admission: ", bg="white",
font=("Helvetica", 10, "bold"))
year_of_admission_label.place(x=20, y=120)
Label(update, text=record[3], font=("Helvetica", 10), bg="white").place(x=150, y=120)

```

```

type_of_admission_label = Label(update, text="Type of Admission: ", bg="white",
font=("Helvetica", 10, "bold"))
type_of_admission_label.place(x=20,y=155)
Label(update, text=record[4], font=("Helvetica", 10), bg="white").place(x=150,y=155)

branch_label = Label(update, text="Branch: ", bg="white", font=("Helvetica", 10, "bold"))
branch_label.place(x=20,y=210)
Label(update, text=record[5], font=("Helvetica", 10), bg="white").place(x=150,y=210)

name_label = Label(update, text="Student Name: ", bg="white", font=("Helvetica", 10, "bold"))
name_label.place(x=20, y=245)
Label(update, text=record[6], font=("Helvetica", 10), bg="white").place(x=150, y=245)

contact1_entry = Entry(update, width=20, font=("Helvetica", 10), relief="groove", bd=2)
contact1_entry.place(x=150, y=290)
contact1_entry.insert(END, record[7])

contact2_entry = Entry(update, width=20, font=("Helvetica", 10), relief="groove", bd=2)
contact2_entry.place(x=150, y=315)
contact2_entry.insert(END, record[8])

email1_entry = Entry(update, width=20, font=("Helvetica", 10), relief="groove", bd=2)
email1_entry.place(x=150, y=360)
email1_entry.insert(END, record[9])

email2_entry = Entry(update, width=20, font=("Helvetica", 10), relief="groove", bd=2)
email2_entry.place(x=150, y=390)
email2_entry.insert(END, record[10])

date_of_birth_label = Label(update, text="Date of Birth:", bg="white", font=("Helvetica", 10,
"bold"))
date_of_birth_label.place(x=20,y=445)
Label(update, text=record[11], font=("Helvetica", 10), bg="white").place(x=150, y=445)

address_label = Label(update, text="Address:", bg="white", font=("Helvetica", 10, "bold"))
address_label.place(x=20,y=505)
address_entry = Text(update, width=20, font=("Helvetica", 10), height=2, wrap="word",
relief="groove", bd=2)
address_entry.place(x=150, y=505)
address_entry.insert(END, record[12])

achedemic_label = Label(update, text="Achedemic Record: ", bg="white", font=("Helvetica", 10,
"bold"))
achedemic_label.place(x=20, y=560)

l1 = Label(update, text="Exam ", bg="white", font=("Helvetica", 10)).place(x=20, y=580)
l2 = Label(update, text="Year of Passing", bg="white", font=("Helvetica", 10)).place(x=80, y=580)
l3 = Label(update, text="Percentage(%)", bg="white", font=("Helvetica", 10)).place(x=200,y=580)
l4 = Label(update, text="SSC", bg="white", font=("Helvetica", 10)).place(x=20, y=620)
l5 = Label(update, text="HSC", bg="white", font=("Helvetica", 10)).place(x=20, y=660)

```



```

Label(update, text=record[13], font=("Helvetica", 10), bg="white").place(x=80,y=620)
Label(update, text=record[14], font=("Helvetica", 10), bg="white").place(x=200,y=620)
Label(update, text=record[15], font=("Helvetica", 10), bg="white").place(x=80,y=660)
Label(update, text=record[16], font=("Helvetica", 10), bg="white").place(x=200,y=660)

sem_label = Label(update, text="Semesters", bg="white", font=("Helvetica", 10,
"bold")),place(x=400, y=400)
year_label = Label(update, text="Year, Month", bg="white", font=("Helvetica", 10,
"bold")),place(x=500, y=400)
percent_label = Label(update, text="%", bg="white", font=("Helvetica", 10, "bold")).place(x=580,
y=400)

sem1_label = Label(update, text="Semester 1", bg="white", font=("Helvetica", 10)).place(x=400,
y=440)
sem2_label = Label(update, text="Semester 2", bg="white", font=("Helvetica", 10)).place(x=400,
y=460)
sem3_label = Label(update, text="Semester 3", bg="white", font=("Helvetica", 10)).place(x=400,
y=480)
sem4_label = Label(update, text="Semester 4", bg="white", font=("Helvetica", 10)).place(x=400,
y=500)
sem5_label = Label(update, text="Semester 5", bg="white", font=("Helvetica", 10)).place(x=400,
y=520)
sem6_label = Label(update, text="Semester 6", bg="white", font=("Helvetica", 10)).place(x=400,
y=540)

y1 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
y1.place(x=500, y=442)
y1.insert(END, record[18])
y2 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
y2.place(x=500, y=462)
y2.insert(END, record[19])
y3 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
y3.place(x=500, y=482)
y3.insert(END, record[20])
y4 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
y4.place(x=500, y=502)
y4.insert(END, record[21])
y5 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
y5.place(x=500, y=522)
y5.insert(END, record[22])
y6 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
y6.place(x=500, y=542)
y6.insert(END, record[23])
p1 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
p1.place(x=585, y=442)
p1.insert(END, record[24])
p2 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
p2.place(x=585, y=462)
p2.insert(END, record[25])

```

```

p3 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
p3.place(x=585, y=482)
p3.insert(END, record[26])
p4 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
p4.place(x=585, y=502)
p4.insert(END, record[27])
p5 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
p5.place(x=585, y=522)
p5.insert(END, record[28])
p6 = Entry(update, font=("Helvetica", 10), bg="white", width=8, relief="groove", bd=2)
p6.place(x=585, y=542)
p6.insert(END, record[29])

update_record = Button(update, text="Update", width=10, bg="blue", fg="white", font=("Helvetica",
10, "bold"), command=update_info)
update_record.place(x=400,y=600)

revert_record = Button(update, text="Revert", width=10, bg="blue", fg="white", font=("Helvetica",
10, "bold"), command=revert_info)
revert_record.place(x=500,y=600)

image_data = record[30]

# Create an image from the retrieved data
image = Image.open(io.BytesIO(image_data))

# Calculate the aspect ratio of the image
image_width, image_height = image.size
aspect_ratio = image_width / image_height

# Determine the dimensions for cropping
canvas_width = 150
canvas_height = 150
if aspect_ratio > 1:
    # Landscape image, crop the width
    crop_width = image_height * aspect_ratio
    crop_height = image_height
else:
    # Portrait image or square image, crop the height
    crop_width = image_width
    crop_height = image_width / aspect_ratio
    crop_x = (image_width - crop_width) / 2
    crop_y = (image_height - crop_height) / 2

# Crop the image to desired dimensions
cropped_image = image.crop((crop_x, crop_y, crop_x + crop_width, crop_y + crop_height))

# Resize the cropped image to fit the canvas
resized_image = cropped_image.resize((canvas_width, canvas_height), Image.ANTIALIAS)

```

```

# Display the resized image in a canvas
image_upload = Canvas(update, width=canvas_width, height=canvas_height)
image_upload.place(x=500, y=120)
image_object = ImageTk.PhotoImage(resized_image)
image_upload.image = image_object # Keep a reference to the image object
image_upload.create_image(int(canvas_width / 2), int(canvas_height / 2), image=image_object)
image_upload.update()

update.mainloop()

def update():
    if not tree.selection():
        tkMessageBox.showinfo("", "Please Select a Student Record to Update")
    else:
        selected_item = tree.selection()
        sap_no = tree.item(selected_item)['values'][3]

# Perform the detail display of the particular SAP number
updateDisplay(sap_no)

def help_option():
    help_window = Tk()
    help_window.title('Help')
    help_window.geometry("500x500")

# Define the font style
title_font = Font(family="Helvetica", size=11, weight="bold")
text_font = Font(family="Helvetica", size=9)

# Insert label and description
insert_label = Label(help_window, text="Insert", font=title_font, anchor="w", justify=LEFT)
insert_label.pack()
insert_description = Label(help_window, text="The insert feature allows users to add new student
data into the record management system. It prompts the user to input information such as the student's
name, age, grade, and other relevant details. The system then stores this data, enabling easy retrieval
and management of student records", font=text_font, anchor="w", justify=LEFT, wraplength=480)
insert_description.pack()

# Delete label and description
delete_label = Label(help_window, text="Delete", font=title_font, anchor="w", justify=LEFT)
delete_label.pack()
delete_description = Label(help_window, text="The delete feature enables users to remove specific
student data from the record management system. Users can select a student's record to delete based
on unique identifiers such as student ID or name. Deleting a record ensures that outdated or erroneous
information is removed from the system, maintaining data accuracy.", font=text_font, anchor="w",
justify=LEFT, wraplength=480)
delete_description.pack()

# Search label and description

```

```

search_label = Label(help_window, text="Search", font=title_font, anchor="w", justify=LEFT)
search_label.pack()
search_description = Label(help_window, text="The search feature allows users to find specific
student data within the record management system. Users can enter search criteria such as student
name, ID, or any other relevant information. The system then retrieves and displays the corresponding
student records, making it efficient to locate specific information quickly.", font=text_font,
anchor="w", justify=LEFT, wraplength=480)
search_description.pack()

# Sort label and description
sort_label = Label(help_window, text="Sort", font=title_font, anchor="w", justify=LEFT)
sort_label.pack()
sort_description = Label(help_window, text="The sort feature organizes the student records within
the system based on a specific criterion. Users can choose to sort the records alphabetically,
numerically, or based on other relevant attributes such as grades or enrollment dates. Sorting helps in
presenting the student records in a structured and easily navigable manner.", font=text_font,
anchor="w", justify=LEFT, wraplength=480)
sort_description.pack()

# Display label and description
display_label = Label(help_window, text="Display", font=title_font, anchor="w", justify=LEFT)
display_label.pack()
display_description = Label(help_window, text="The display feature presents all the student records
stored in the system. It provides an overview of all the information available, such as student names,
ages, grades, and any other relevant data. Displaying the records allows users to review the complete
dataset, facilitating analysis, decision-making, or generating reports as needed.", font=text_font,
anchor="w", justify=LEFT, wraplength=480)
display_description.pack()

help_window.mainloop()

def aboutUs():
    about_window = Tk()
    about_window.title('About Us')
    about_window.geometry("500x300")

    name_label = Label(about_window, text="Jainam Barbhaya", font=("Varenda", 15, "bold"))
    name_label.place(x=20, y=20)

    message = """
    Email : jainambarbhaya1509@gmail.com \n
    Contact : +91 9702288992\n
    Department : Information Technology\n
    SAP No : 57498210010\n
    Roll No : T010\n
    Semester: 4\n
    """

    info = Message(about_window, text=message, width=400, font=("Varenda", 8, "bold"))
    info.place(x=0, y=50)

```

```

# Canvas for image
img_canvas = Canvas(about_window, height=150, width=150)
img_canvas.place(x=300, y=20)

# Load and crop the image
image = Image.open("C:\\Users\\Jainam Barbhaya\\Desktop\\Python-GUI\\Python Mini
Project\\Images\\image.jpeg")
image = image.resize((150, 150), Image.ANTIALIAS) # Resize the image to fit the canvas
cropped_image = ImageTk.PhotoImage(image)

# Display the cropped image on the canvas
img_canvas.create_image(0, 0, anchor="nw", image=cropped_image)

about_window.mainloop()

def window():
'''
In this function i have created main window after admin logins
this module contains clickable labels such as insert, update, delete, sort, search & display
on clicking the label, it will be redirected to a new screen where the admin can perform the function

This function uses TreeView to display the data in tabular format
'''

# creating a window
global window

window = Tk()
window.title("Student Bio-Data RMS")

# setting window on the center of the screen
window_width = 1000
window_height = 500
screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()
x = (screen_width/2) - (window_width/2)
y = (screen_height/2) - (window_height/2)
window.geometry('%dx%d+%d+%d' % (window_width, window_height, x, y))
window.resizable(width=False, height=False)
window.configure(bg='white')

# setting canvas taskbar
canvas = Canvas(window, width=1000, height=30, bg="blue")
canvas.pack()

#setting scrollbar
global tree
scrollbary = Scrollbar(window, orient=VERTICAL)

```

```

tree = ttk.Treeview(window, columns=("Sr No.", "Full Name", "Roll No.", "Sap Id",
"Department", "Year"),selectmode="extended", height=21, yscrollcommand=scrollbary.set)
scrollbary.config(command=tree.yview)
scrollbary.pack(side=RIGHT, fill=Y)

tree.heading('Sr No.', text="Sr No.", anchor=W)
tree.heading('Full Name', text="Full Name", anchor=W)
tree.heading('Roll No.', text="Roll No.", anchor=W)
tree.heading('Sap Id', text="Sap Id", anchor=W)
tree.heading('Department', text="Department", anchor=W)
tree.heading('Year', text="Year", anchor=W)

#setting width of the columns
tree.column('#0', stretch=NO, minwidth=0, width=0)
tree.column('#1', stretch=NO, minwidth=0, width=140)
tree.column('#2', stretch=NO, minwidth=0, width=140)
tree.column('#3', stretch=NO, minwidth=0, width=140)
tree.column('#4', stretch=NO, minwidth=0, width=140)
tree.place(x=10,y=40)

# adding CRUD labes and their function in list and iterating them

# display()

add_label = Label(window,text="Add", fg="white", bg="blue", cursor="hand2", font=("Helvetica", 8,
"bold"))
add_label.place(x=20, y=7)
add_label.bind("<Button-1>", lambda event: insert())

delete_label = Label(window,text="Delete", fg="white", bg="blue", cursor="hand2",
font=("Helvetica", 8, "bold"))
delete_label.place(x=70, y=7)
delete_label.bind("<Button-1>", lambda event: delete())

search_label = Label(window,text="Search", fg="white", bg="blue", cursor="hand2",
font=("Helvetica", 8, "bold"))
search_label.place(x=130, y=7)
search_label.bind("<Button-1>", lambda event: search())

display_label = Label(window,text="Display", fg="white", bg="blue", cursor="hand2",
font=("Helvetica", 8, "bold"))
display_label.place(x=190, y=7)
display_label.bind("<Button-1>", lambda event: display())

update_label = Label(window,text="Update", fg="white", bg="blue", cursor="hand2",
font=("Helvetica", 8, "bold"))
update_label.place(x=250, y=7)
update_label.bind("<Button-1>", lambda event: update())

```

```

sort_label = Label(window,text="Sort", fg="white", bg="blue", cursor="hand2", font=("Helvetica", 8,
"bold"))
sort_label.place(x=310, y=7)
sort_label.bind("<Button-1>", lambda event: sort())

help_label = Label(window,text="Help", fg="white", bg="blue", cursor="hand2", font=("Helvetica",
8, "bold"))
help_label.place(x=850, y=7)
help_label.bind("<Button-1>", lambda event: help_option())

about_label = Label(window,text="About Us", fg="white", bg="blue", cursor="hand2",
font=("Helvetica", 8, "bold"))
about_label.place(x=900, y=7)
about_label.bind("<Button-1>", lambda event: aboutUs())

window.mainloop()

'''
# Future Scope:
> Complete Advanced Search Function
> Improve UI
> Reduce Complexity
'''

```

10. System Testing

| Bug Id | Bug Description | Steps to Reproduce | Expected Result | Actual Result | Status |
|--------|-----------------------------------|--|-------------------------------|---|--------|
| PY_01 | Database Error | 1. Run program 2. Enter Id & password | Database Connected | Database connection failed | Fail |
| PY_02 | Error in inserting record | 1. Fill enter form 2. Click Submit | Record Inserted | Not all attributes were passed. | Fail |
| PY_03 | Image not retrieved from database | 1. Click Display 2. Double click on record | Entire Form Is Been Displayed | TCLError | Fail |
| PY_04 | Incorrect Datatype | 1. Insert Record 2. Fill Details 3. Click Submit | Record Inserted | Incorrect datatype in program or database | Fail |

Current Bug Status: Resolved Above Bugs


11. Results

SBMP Student Profile RMS: Admin Panel

Admin Login

Username:

Password:

 **Shri Bhagubhai Mafatlal Polytechnic**

Student Bio-Data RMS

Add Delete Search Display Update Sort Help About Us

| Sr No. | Full Name | Roll No. | Sap Id | Department | Year |
|--------|-----------------|----------|-------------|------------------------|------|
| 1 | Jainam Barbhaya | T010 | 57498210010 | Information Technology | 2 |
| 2 | Chintan Dodia | T002 | 57498210002 | Information Technology | 2 |
| 3 | Khushi Sanghavi | T003 | 57498210003 | Information Technology | 2 |
| 4 | Het Shah | T031 | 57498210031 | Information Technology | 2 |
| 5 | Rashmi Kamdar | T035 | 57498210035 | Information Technology | 2 |
| 6 | Ayush Vora | T021 | 57498210021 | Information Technology | 2 |
| 7 | Iqra Khatib | T048 | 57498210048 | Information Technology | 2 |
| 8 | Vanshita Shah | T007 | 57498210007 | Information Technology | 2 |

Total Records Found: 8

OK

T010 Record Details

Sap No57498210010

Roll NoT010

Year of Admission:2021


Type of Admission:First Year

Branch:Information Technology

Student Name:Jainam Barbhaya

Contact No.:9702288992
9702288993

Email:jainambarbhaya1509@gmail.com



Date of Birth:15-09-2005

Current Address:1403, Jakerial Lane Malad
West Mumbai 400064

| Semesters | Year | Percentages(%) |
|------------|------|----------------|
| Semester 1 | 2021 | 83.64 |
| Semester 2 | 2022 | 82 |
| Semester 3 | 2022 | 79.81 |
| Semester 4 | | |
| Semester 5 | | |
| Semester 6 | | |

Achedemic Record:

| Exam | Year of Passing | Percentage(%) |
|------|-----------------|---------------|
| SSC | 2021 | 90.2 |
| HSC | | |

Add Student

Sap No

Roll No

Year of Admission:

Upload Photo:

Choose Photo

Type of Admission:
☒ First Year
☐ Direct Second Year

Branch:

Student Name:

Contact No.:

Email:

Date of Birth:

Current Address:

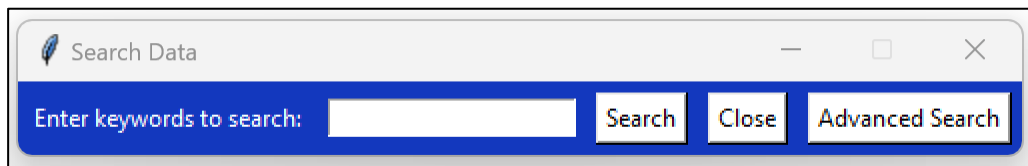
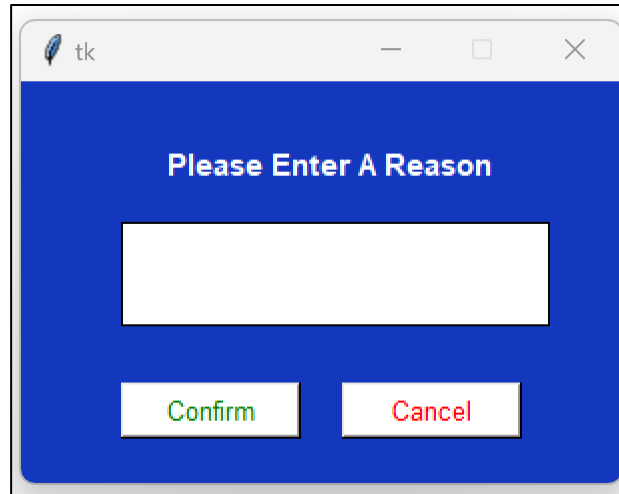
| Semesters | Year, Month | % |
|------------|----------------------|----------------------|
| Semester 1 | <input type="text"/> | <input type="text"/> |
| Semester 2 | <input type="text"/> | <input type="text"/> |
| Semester 3 | <input type="text"/> | <input type="text"/> |
| Semester 4 | <input type="text"/> | <input type="text"/> |
| Semester 5 | <input type="text"/> | <input type="text"/> |
| Semester 6 | <input type="text"/> | <input type="text"/> |

Achedemic Record:

| Exam | Year of Passing | Percentage(%) |
|------|----------------------|----------------------|
| SSC | <input type="text"/> | <input type="text"/> |
| HSC | <input type="text"/> | <input type="text"/> |

Submit

Clear



Roll No T010

Type of Admission: First Year

Student Name: Jainam Barbhaya



9702288993

Address: 1403, Jakerial Lane
Malad West Mumbai

| Exam | Year of Passing | Percentage(%) |
|------|-----------------|---------------|
|------|-----------------|---------------|

HSC

| | | |
|------------|------|-------|
| Semester 1 | 2021 | 83.64 |
| Semester 2 | 2022 | 82 |
| Semester 3 | 2022 | 79.81 |
| Semester 4 | | |
| Semester 5 | | |
| Semester 6 | | |

Revert

12. Future Scope

While the student bio data record management system implemented in this project is effective, there are still areas for further development. Future updates will focus on enhancing the user interface by incorporating advanced functionalities such as "advancedSearch()". Additionally, efforts will be made to reduce the complexity of the project, ensuring a more streamlined and efficient user experience. These improvements will contribute to the overall effectiveness and usability of the system.

13. Conclusion

To summarize, the implementation of a student bio data record management system using tkinter and MySQL Connector in Python showcases the effectiveness of these tools in creating a streamlined solution for storing and retrieving student information. The project's user-friendly interface, coupled with the robustness of the MySQL database, enables efficient data management and enhances overall productivity. This mini project highlights the versatility and practicality of Python for developing data management systems.

14. References

- <https://www.javatpoint.com/python-tkinter-canvas>
- <https://www.javatpoint.com/tree-view-widgets-and-tree-view-scrollbar-in-tkinter-python>
- <https://www.plus2net.com/python/tkinter-blob-display.php>