



Six weeks industrial training report

(5th semester)

On

Student management system -PROJECT

Submitted for the partial fulfillment of the degree

Of

Bachelor of Technology

(Computer Science of Technology)

Submitted by:

Rohan Kashyap

University roll no:- 2101818

Submitted to:

Mrs.Priyanka kumari

DECLARATION

I Rohan Kashyap, student of B.Tech (computer science engineering) 5th semester, studying at Chandigarh Group of Colleges, Landran, hereby declare that the six-week of Industrial training The report, undergone at “Mrs.Priyanka kumari”, submitted to Punjab Technical University, Kapurthala, in partial fulfillment of the award of the degree of Bachelor of Technology in Computer Science, Engineering is the original work conducted by me.

The information and the data in the report are authentic to the best of my knowledge.

Rohan Kashyap

ACKNOWLEDGEMENT

I student of Chandigarh College of Engineering, Landran, have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them. I am highly indebted to the “Solitaire Infosys” for their guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project. I would like to express my gratitude towards my teachers for their kind cooperation and encouragement which helps me in the completion of this project.

Rohan Kashyap

TABLE OF CONTENTS

Introduction

Technology used

- Front end
- Back end
 - Database work
- Code Editor

Python

- Intro
- Why to learn Python
- Characteristics of Python
- Application of Python
- Tkinter framework
- Vs code

Source code

- Login.py
- main.py

Conclusion

Introduction

The Student Management System is an essential tool for educational institutions to efficiently manage student records, streamline administrative tasks, and enhance overall organization. This Python program showcases a comprehensive Student Management System implemented through a graphical user interface (GUI) using the Tkinter library.

Features:

- **User-Friendly Interface:** The GUI provides an intuitive interface for administrators to interact with the system. It offers buttons for adding, searching, updating, showing, and deleting student records.
- **Real-Time Clock and Slider:** The GUI includes a real-time clock display and a sliding text element, enhancing the visual appeal and user experience.
- **Student Records Display:** A table view powered by the Treeview widget presents student records, including their IDs, names, contact information, and other essential details.
- **Data Entry and Update:** Administrators can add new student records or update existing ones through a dedicated data entry form. The form includes fields for inputting the student's name, contact details, address, gender, and date of birth.
- **View Details:** The system allows administrators to view detailed information about a student by clicking on a corresponding button in the student records table.
- **Data Validation:** The program performs data validation to ensure that all required fields are filled out correctly before storing or updating student records.
- **MySQL Integration:** The application integrates with a MySQL database to store and retrieve student records securely.

Technology used

PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public

License (GPL). This tutorial gives enough understanding on Python programming language.

Why to Learn Python?

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented – Python supports an Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language – Python is a great language for beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

The following are important characteristics of Python Programming –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Applications of Python

As mentioned before, Python is one of the most widely used language over the web.

I'm going to list a few of them here:

- Easy-to-learn – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read – Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain – Python's source code is fairly easy-to-maintain.
- A broad standard library – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases – Python provides interfaces to all major commercial databases.
- GUI Programming – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable – Python provides a better structure and support for large programs than shell scripting.

Front end

In the Front end I used Tkinter. Tkinter is a graphical user interface (GUI) toolkit and library for creating front-end interfaces in Python. It provides a set of modules and classes that allow developers to create windows, dialogs, buttons, labels, and other GUI elements for desktop applications. Tkinter is not a full-fledged framework but rather a library that simplifies the process of creating graphical user interfaces.

The front end is the user interface of the application that interacts with users. In your program, the front end is primarily created using the Tkinter library, and it's responsible for presenting the user with visual elements, receiving user input, displaying data, and providing a way for users to interact with the system.

Key front-end components in this program:

- Buttons for adding, searching, updating, showing, and deleting student records.
- Treeview widget that displays student records and their details.
- Labels, entry fields, and combo boxes for entering and displaying student information.
- Images, clocks, and sliders are used for visual representation.

Back End:

The back end is responsible for handling data processing, storage, and business logic. In your program, the back end consists of the following components:

- Database Connection: The program connects to a MySQL database to store and retrieve student records. The interactions with the database, including inserting, updating, deleting, and querying data, are part of the back end.
- Data Processing Functions: Functions that manipulate the data before storing or after retrieving it from the database are part of the back end. These functions perform tasks like validation, data transformation, and formatting.
- Function Definitions: The functions you've defined for adding, searching, updating, and deleting student records are part of the back end. They handle the core logic of the application.

- Event Handling: The event handling, where actions like button clicks trigger specific functions, is also considered part of the back end. The back end responds to user interactions and updates the front end accordingly.
- Database Queries: SQL queries and database-related operations fall under the back-end category. These queries retrieve, update, or manipulate data within the database.
- In this program, the back end interacts with the database, processes user input, performs data validation, executes database queries, and manages the overall functionality of the application.

VS CODE

Visual Studio Code (famously known as VS Code) is a free open source text editor by Microsoft. VS Code is available for Windows, Linux, and macOS. Although the editor is relatively lightweight, it includes some powerful features that have made VS Code one of the most popular development environment tools in recent times.

Features

VS Code supports a wide array of programming languages from Java, C++, and Python to CSS, Go, and Dockerfile. Moreover, VS Code allows you to add on and even creating new extensions including code linters, debuggers, and cloud and web development support. The VS Code user interface allows for a lot of interaction compared to other text editors. To simplify user experience, VS Code is divided into five main regions:

- The activity bar
- The side bar
- Editor groups
- The panel
- The status bar

Source code

Login.py

```
import tkinter as tk
from tkinter import messagebox
from tkinter import PhotoImage
import mysql.connector
from PIL import ImageTk
global_username = ""
# Connect to MySQL database
db = mysql.connector.connect(
    host='localhost',
    username='root',
    password='*****',
    database='studentmanagementsystem'
)
cursor = db.cursor()

root = tk.Tk()
root.title("Login and Create Account")
root.geometry('1280x700+0+0')

root.resizable(False,False)

def create_table():
    global_username = create_username_entry.get()
    table_name = create_username_entry.get()
    query=f'create table {table_name}(id int, name varchar(30),mobile varchar(10),email
varchar(30),' \
        'address varchar(100),gender varchar(20),dob varchar(20),date
varchar(50), time varchar(50))'
    # query = f"CREATE TABLE {table_name} (id varchar(10) PRIMARY KEY, name VARCHAR(25),
mobile varchar(10), email varchar(30), address varchar(100), gender varchar(10),dob
varchar(20), date varchar(50), time varchar(50))"
    cursor = db.cursor()
    try:
        cursor.execute(query)
        db.commit()
        messagebox.showinfo("Success", f"account '{table_name}' created successfully now
login.")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Error creating table: {err}")
    cursor.close()
    return

def show_login_frame():
    login_frame.place(x=480,y=400)
    create_account_frame.place_forget()

def show_create_account_frame():
    create_account_frame.place(x=480,y=400)
    login_frame.place_forget()
```

```

login_frame.place_forget()
create_account_frame.place(x=480,y=400)

def login():
    global global_username
    username = login_username_entry.get()
    password = login_password_entry.get()

    cursor.execute("SELECT * FROM accounts WHERE username = %s AND password = %s",
(username, password))
    account = cursor.fetchone()

    if account:
        global_username = login_username_entry.get()
        result_label.config(text="Logging in as: " + username)
        root.destroy()
        #import test
    else:
        result_label.config(text="Invalid username or password.")

def create_account():
    new_username = create_username_entry.get()
    new_password = create_password_entry.get()
    confirm_new_password = confirm_password_entry.get()

    if new_username==' ' or new_password==' ':
        messagebox.showerror('Error','Fields cannot be empty')
    else:
        # Check if the username already exists
        cursor.execute("SELECT * FROM accounts WHERE username = %s", (new_username,))
        existing_account = cursor.fetchone()
        if existing_account:
            messagebox.showerror('Error',"Username already exists.")
            result_label.config(text="Username already exists.")
            return

        if new_password != confirm_new_password:
            messagebox.showerror('Error',"Passwords do not match.")
            result_label.config(text="Passwords do not match.")
            return

        cursor.execute("INSERT INTO accounts (username, password) VALUES (%s, %s)",
(new_username, new_password))
        db.commit()
        #var = create_username_entry.get()
        create_table()
        #root.destroy()
        #result_label.config(text="Account created for: " + new_username)

# Set up background image
backgroundImage=ImageTk.PhotoImage(file="bg.jpg")
bgLabel=tk.Label(image=backgroundImage)
bgLabel.place(x=0,y=0 )

```

```

# Set up logo image
logo_image = PhotoImage(file="logo.png")
logo_label = tk.Label(root, image=logo_image , bg='white')
logo_label.place(x=500,y=90)

# Create login frame
login_frame = tk.Frame(root , bg= 'white')
login_label = tk.Label(login_frame, text="Login")
login_label.place(x=500,y=300)
usernameImage=PhotoImage(file="user.png")
login_username_label = tk.Label(login_frame,image=usernameImage, text="Username:",
compound=tk.LEFT,font=('times new roman',20,'bold'),bg='white')
login_username_label.pack()
login_username_entry = tk.Entry(login_frame, font=('times new
roman',20,'bold'),bd=5,fg='royalblue')
login_username_entry.pack()

passwordImage=PhotoImage(file="password.png")
login_password_label = tk.Label(login_frame, image=passwordImage,
text="Password:",font=('times new roman',20,'bold'),bg='white', compound=tk.LEFT)
login_password_label.pack()
login_password_entry = tk.Entry(login_frame, show="*", font=('times new
roman',20,'bold'),bd=5,fg='royalblue')
login_password_entry.pack()

login_button = tk.Button(login_frame, text="Login", command=login,font=('times new
roman',14,'bold'),width=15
                        ,fg='white',bg='cornflowerblue',activebackground='cornflowerblue',
                        activeforeground='white',cursor='hand2')
login_button.pack()

# Create create account frame
create_account_frame = tk.Frame(root, bg= 'white')

create_username_label = tk.Label(create_account_frame, text="Username:",
image=usernameImage,font=('times new roman',20,'bold'),bg='white', compound=tk.LEFT)
create_username_label.pack()
create_username_entry = tk.Entry(create_account_frame, font=('times new
roman',20,'bold'),bd=5,fg='royalblue')
create_username_entry.pack()

create_password_label = tk.Label(create_account_frame,image=passwordImage, text="Set
Password:",font=('times new roman',20,'bold'),bg='white', compound=tk.LEFT)
create_password_label.pack()
create_password_entry = tk.Entry(create_account_frame, show="*", font=('times new
roman',20,'bold'),bd=5,fg='royalblue')
create_password_entry.pack()

confirm_password_label = tk.Label(create_account_frame, text="Confirm
Password:",font=('times new roman',20,'bold'),bg='white')

```

```
confirm_password_label.pack()
confirm_password_entry = tk.Entry(create_account_frame, show="*", font=('times new
roman',20,'bold'),bd=5,fg='royalblue')
confirm_password_entry.pack()

create_button = tk.Button(create_account_frame, text="Create Account",font=('times new
roman',14,'bold'),width=15
                        ,fg='white',bg='cornflowerblue',activebackground='cornflowerblue',
                        activeforeground='white',cursor='hand2', command=create_account)
create_button.pack()

result_label = tk.Label(root, text="")
result_label.pack()

login_button = tk.Button(root, text="Login",font=('times new roman',14,'bold'),width=15
                        ,fg='white',bg='cornflowerblue',activebackground='cornflowerblue',
                        activeforeground='white',cursor='hand2', command=show_login_frame)
login_button.place(x=540,y=290)

create_account_button = tk.Button(root, text="Create Account",font=('times new
roman',14,'bold'),width=15
                        ,fg='white',bg='cornflowerblue',activebackground='cornflowerblue',
                        activeforeground='white',cursor='hand2',
command=show_create_account_frame)
create_account_button.place(x=540,y=350)

root.mainloop()

# Close the database connection
db.close()
```

Main.py

```
from tkinter import *
import time
import ttkthemes
from tkinter import ttk,messagebox,filedialog
from tkcalendar import DateEntry
from PIL import ImageTk
#import pymysql
import pandas
import mysql.connector
from tkinter import PhotoImage
from PIL import Image, ImageTk
from login import global_username

# Now you can use the global_username variable in this file
print("Username from the first file:", global_username)

conn = mysql.connector.connect (
    host ='localhost',
    username = 'root',
    password = '*****',
    database = 'studentmanagementsystem'
)
mycursor=conn.cursor()

#functionality Part
def view_details(event):
    item = studentTable.identify("item", event.x, event.y)
    if item:
        student_id = studentTable.item(item, "values")[0]
        view_student_details(student_id)

def view_student_details(student_id):
    student_info_screen = Toplevel(root)
    student_info_screen.title("Student Details")
    student_info_screen.geometry("400x350")

    query = f"SELECT * FROM {global_username} WHERE id = %s"
    values = (student_id,)
    mycursor.execute(query, values)
    student_data = mycursor.fetchone()

    if student_data:
        image_path = f"studentimg/{student_data[1]}.jpeg" # Replace with actual image
path
        try:
            image = Image.open(image_path)
            image = image.resize((150, 150), Image.LANCZOS)
            photo = ImageTk.PhotoImage(image)
```

```

        image_label = Label(student_info_screen, image=photo)
        image_label.image = photo # Keep a reference to the PhotoImage to prevent it
from being garbage collected
        image_label.pack()
    except Exception as e:
        print("Error loading image:", e)
    Label(student_info_screen, text=f"Student ID: {student_data[0]}").pack()
    Label(student_info_screen, text=f"Name: {student_data[1]}").pack()
    Label(student_info_screen, text=f"Mobile: {student_data[2]}").pack()
    Label(student_info_screen, text=f"Email: {student_data[3]}").pack()
    Label(student_info_screen, text=f"Address: {student_data[4]}").pack()
    Label(student_info_screen, text=f"Gender: {student_data[5]}").pack()
    Label(student_info_screen, text=f"Date of Birth: {student_data[6]}").pack()

    # Load the student's image and display it

else:
    Label(student_info_screen, text="Student data not found.").pack()

def iexit():
    result=messagebox.askyesno('Confirm','Do you want to exit?')
    if result:
        root.destroy()
    else:
        pass

def toplevel_data(title,button_text,command):
    global idEntry,phoneEntry,nameEntry,emailEntry,addressEntry,genderEntry,dob_var,screen
    screen = Toplevel()
    screen.title(title)
    screen.grab_set()
    screen.resizable(False, False)
    idLabel = Label(screen, text='Id', font=('times new roman', 20, 'bold'))
    idLabel.grid(row=0, column=0, padx=30, pady=15, sticky=W)
    idEntry = Entry(screen, font=('roman', 15, 'bold'), width=24)
    idEntry.grid(row=0, column=1, pady=15, padx=10)

    nameLabel = Label(screen, text='Name', font=('times new roman', 20, 'bold'))
    nameLabel.grid(row=1, column=0, padx=30, pady=15, sticky=W)
    nameEntry = Entry(screen, font=('roman', 15, 'bold'), width=24)
    nameEntry.grid(row=1, column=1, pady=15, padx=10)

    phoneLabel = Label(screen, text='Phone', font=('times new roman', 20, 'bold'))
    phoneLabel.grid(row=2, column=0, padx=30, pady=15, sticky=W)
    phoneEntry = Entry(screen, font=('roman', 15, 'bold'), width=24)
    phoneEntry.grid(row=2, column=1, pady=15, padx=10)

    emailLabel = Label(screen, text='Email', font=('times new roman', 20, 'bold'))
    emailLabel.grid(row=3, column=0, padx=30, pady=15, sticky=W)
    emailEntry = Entry(screen, font=('roman', 15, 'bold'), width=24)
    emailEntry.grid(row=3, column=1, pady=15, padx=10)

```



```

addressLabel = Label(screen, text='Address', font=('times new roman', 20, 'bold'))
addressLabel.grid(row=4, column=0, padx=30, pady=15, sticky=W)
addressEntry = Entry(screen, font=('roman', 15, 'bold'), width=24)
addressEntry.grid(row=4, column=1, pady=15, padx=10)

genderLabel = Label(screen, text='Gender', font=('times new roman', 20, 'bold'))
genderLabel.grid(row=5, column=0, padx=30, pady=15, sticky=W)
gender_var = StringVar()
gender_choices = ['Male', 'Female', 'Other']
genderEntry = ttk.Combobox(screen, textvariable=gender_var, values=gender_choices)
genderEntry.grid(row=5, column=1, pady=15, padx=10)

dobLabel = Label(screen, text='D.O.B', font=('times new roman', 20, 'bold'))
dobLabel.grid(row=6, column=0, padx=30, pady=15, sticky=W)
dob_var = StringVar()
dob_cal = DateEntry(screen, textvariable=dob_var, date_pattern="yyyy-mm-dd",
locale="en_US")
dob_cal.grid(row=6, column=1, pady=15, padx=30)

#dob_var = Entry(screen, font=('roman', 15, 'bold'), width=24)
#dob_var.grid(row=6, column=1, pady=15, padx=10)

student_button = ttk.Button(screen, text=button_text, command=command)
student_button.grid(row=7, columnspan=2, pady=15)
if title=='Update Student':
    indexing = studentTable.focus()

    content = studentTable.item(indexing)
    listdata = content['values']
    idEntry.insert(0, listdata[0])
    nameEntry.insert(0, listdata[1])
    phoneEntry.insert(0, listdata[2])
    emailEntry.insert(0, listdata[3])
    addressEntry.insert(0, listdata[4])
    genderEntry.insert(0, listdata[5])
    dob_var.insert(0, listdata[6])

def update_data():
    query = f'update {global_username} set name=%s, mobile=%s, email=%s, address=%s,
gender=%s, dob=%s, date=%s, time=%s where id=%s'
    values = (nameEntry.get(), phoneEntry.get(), emailEntry.get(), addressEntry.get(),
genderEntry.get(), dob_var.get(), date, currenttime, idEntry.get())
    mycursor.execute(query, values)
    conn.commit()
    messagebox.showinfo('Success', f'Id {idEntry.get()} is modified successfully',
parent=screen)
    screen.destroy()
    show_student()

def show_student():

```

```

query = f"select * from {global_username}"
mycursor.execute(query)
fetched_data = mycursor.fetchall()
studentTable.delete(*studentTable.get_children())
for data in fetched_data:
    studentTable.insert('', END, values=data)

def delete_student():
    indexing=studentTable.focus()
    print(indexing)
    content=studentTable.item(indexing)
    content_id=content['values'][0]
    query=f'delete from {global_username} where id=%s'
    values = (content_id,)
    mycursor.execute(query,values)
    conn.commit()
    messagebox.showinfo('Deleted',f'Id {content_id} is deleted succesfully')
    query=f'select * from {global_username}'
    mycursor.execute(query)
    fetched_data=mycursor.fetchall()
    studentTable.delete(*studentTable.get_children())
    for data in fetched_data:
        studentTable.insert('',END,values=data)

def search_data():
    query=f'select * from {global_username} where id=%s or name=%s or email=%s or
mobile=%s or address=%s or gender=%s or dob=%s'
    values =
(idEntry.get(),nameEntry.get(),emailEntry.get(),phoneEntry.get(),addressEntry.get(),gender
Entry.get(),dob_var.get())
    mycursor.execute(query, values)
    studentTable.delete(*studentTable.get_children())
    fetched_data=mycursor.fetchall()
    for data in fetched_data:
        studentTable.insert('',END,values=data)

def add_data():
    if idEntry.get()==' ' or nameEntry.get()==' ' or phoneEntry.get()==' ' or
emailEntry.get()==' ' or addressEntry.get()==' ' or genderEntry.get()==' ' or
dob_var.get()==' ':
        messagebox.showerror('Error','All Feilds are required',parent=screen)

    else:
        #try:
            query= f'insert into {global_username} values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)'

```

```

        values
=(idEntry.get(),nameEntry.get(),phoneEntry.get(),emailEntry.get(),addressEntry.get(),
                                genderEntry.get(),dob_var.get(),date,currenttime)
        mycursor.execute(query,values)
        conn.commit()
        result=messagebox.askyesno('Confirm','Data added successfully. Do you want to
clean the form?',parent=screen)
        if result:
            idEntry.delete(0,END)
            nameEntry.delete(0,END)
            phoneEntry.delete(0,END)
            emailEntry.delete(0,END)
            addressEntry.delete(0,END)
            genderEntry.delete(0,END)
            #dob_var.delete(0,END)
        else:
            pass
        #except:
            # messagebox.showerror('Error','Id cannot be repeated',parent=screen)
            # return

        query=f'select *from {global_username}'
        mycursor.execute(query)
        fetched_data=mycursor.fetchall()
        studentTable.delete(*studentTable.get_children())
        for data in fetched_data:
            studentTable.insert('',END,values=data)

count=0
text=' '
def slider():
    global text,count
    if count==len(s):
        count=0
        text=' '
    text=text+s[count]
    sliderLabel.config(text=text)
    count+=1
    sliderLabel.after(300,slider)

def clock():
    global date,currenttime
    date=time.strftime('%d/%m/%Y')
    currenttime=time.strftime('%H:%M:%S')
    datetimeLabel.config(text=f'    Date: {date}\nTime: {currenttime}')
    datetimeLabel.after(1000,clock)

```

```

#GUI Part
root=ttkthemes.ThemedTk()

root.get_themes()

root.set_theme('radiance')

root.geometry('1174x680+0+0')
root.resizable(0,0)
root.title('Student Management System')

datetimeLabel=Label(root,font=('times new roman',18,'bold'))
datetimeLabel.place(x=5,y=5)
clock()
s='Student Management System' #s[count]=t when count is 1
sliderLabel=Label(root,font=('arial',28,'italic bold'),width=30)
sliderLabel.place(x=200,y=0)
slider()


leftFrame=Frame(root)
leftFrame.place(x=920,y=-10,width=300,height=800)

logo_image=PhotoImage(file='logo.png')
logo_Label=Label(leftFrame,image=logo_image)
logo_Label.grid(row=0,column=0)

addstudentButton=ttk.Button(leftFrame,text='Add Student',width=25,command=lambda
:toplevel_data('Add Student','Add',add_data))
addstudentButton.grid(row=1,column=0,pady=20)

searchstudentButton=ttk.Button(leftFrame,text='Search Student',width=25,command=lambda
:toplevel_data('Search Student','Search',search_data))
searchstudentButton.grid(row=2,column=0,pady=20)

deletestudentButton=ttk.Button(leftFrame,text='Delete
Student',width=25,command=delete_student)
deletestudentButton.grid(row=3,column=0,pady=20)

updatestudentButton=ttk.Button(leftFrame,text='Update Student',width=25,command=lambda
:toplevel_data('Update Student','Update',update_data))
updatestudentButton.grid(row=4,column=0,pady=20)

showstudentButton=ttk.Button(leftFrame,text='Show Student',width=25,command=show_student)
showstudentButton.grid(row=5,column=0,pady=20)

exitButton=ttk.Button(leftFrame,text='Exit',width=25,command=iexit)
exitButton.grid(row=6,column=0,pady=20)

rightFrame=Frame(root)
rightFrame.place(x=10,y=100,width=920,height=550)

scrollBarX=Scrollbar(rightFrame,orient=HORIZONTAL)

```

```

scrollBarY=Scrollbar(rightFrame,orient=VERTICAL)

studentTable=ttk.Treeview(rightFrame,columns=('Id','Name','Mobile','Email','Address','Gender',
                                             'D.O.B','Added Date','Added Time','View Details'),
                           xscrollcommand=scrollBarX.set,yscrollcommand=scrollBarY.set)

scrollBarX.config(command=studentTable.xview)
scrollBarY.config(command=studentTable.yview)

scrollBarX.pack(side=BOTTOM,fill=X)
scrollBarY.pack(side=RIGHT,fill=Y)

studentTable.pack(expand=1,fill=BOTH)

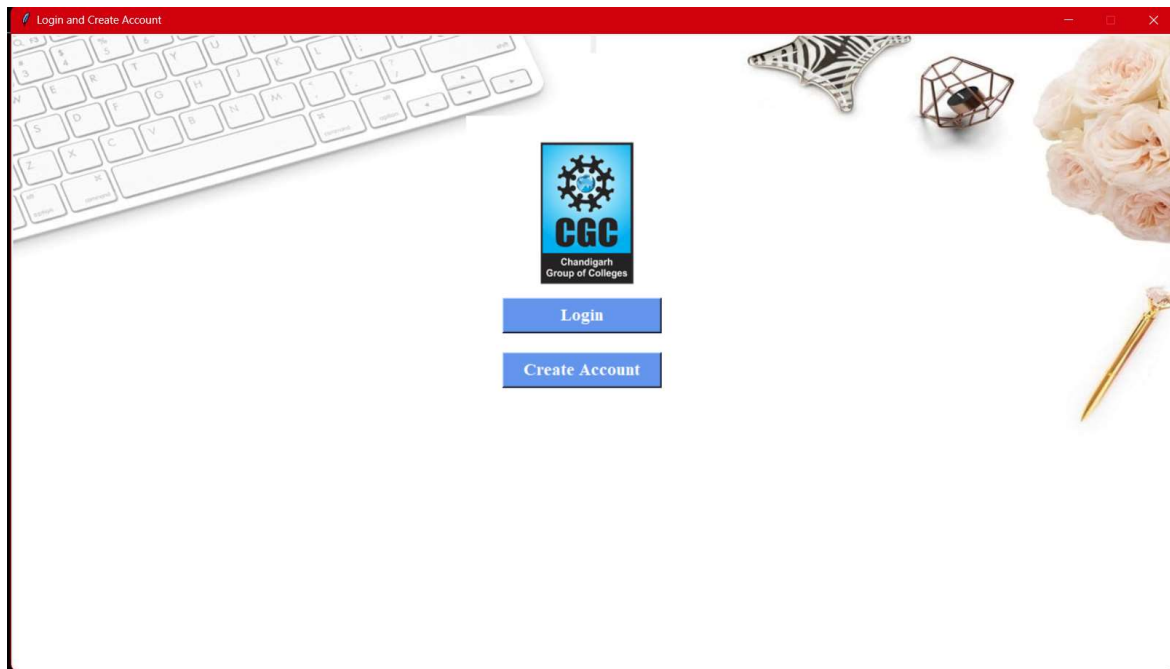
studentTable.heading('Id',text='Id')
studentTable.heading('Name',text='Name')
studentTable.heading('Mobile',text='Mobile No')
studentTable.heading('Email',text='Email Address')
studentTable.heading('Address',text='Address')
studentTable.heading('Gender',text='Gender')
studentTable.heading('D.O.B',text='D.O.B')
studentTable.heading('Added Date',text='Added Date')
studentTable.heading('Added Time',text='Added Time')
studentTable.heading('View Details',text='View Details')

studentTable.column('Id',width=50,anchor=CENTER)
studentTable.column('Name',width=200,anchor=CENTER)
studentTable.column('Email',width=300,anchor=CENTER)
studentTable.column('Mobile',width=200,anchor=CENTER)
studentTable.column('Address',width=300,anchor=CENTER)
studentTable.column('Gender',width=100,anchor=CENTER)
studentTable.column('D.O.B',width=200,anchor=CENTER)
studentTable.column('Added Date',width=200,anchor=CENTER)
studentTable.column('Added Time',width=200,anchor=CENTER)
studentTable.column('View Details',width=200,anchor=CENTER)
studentTable.bind("<Button-1>", view_details)
style=ttk.Style()

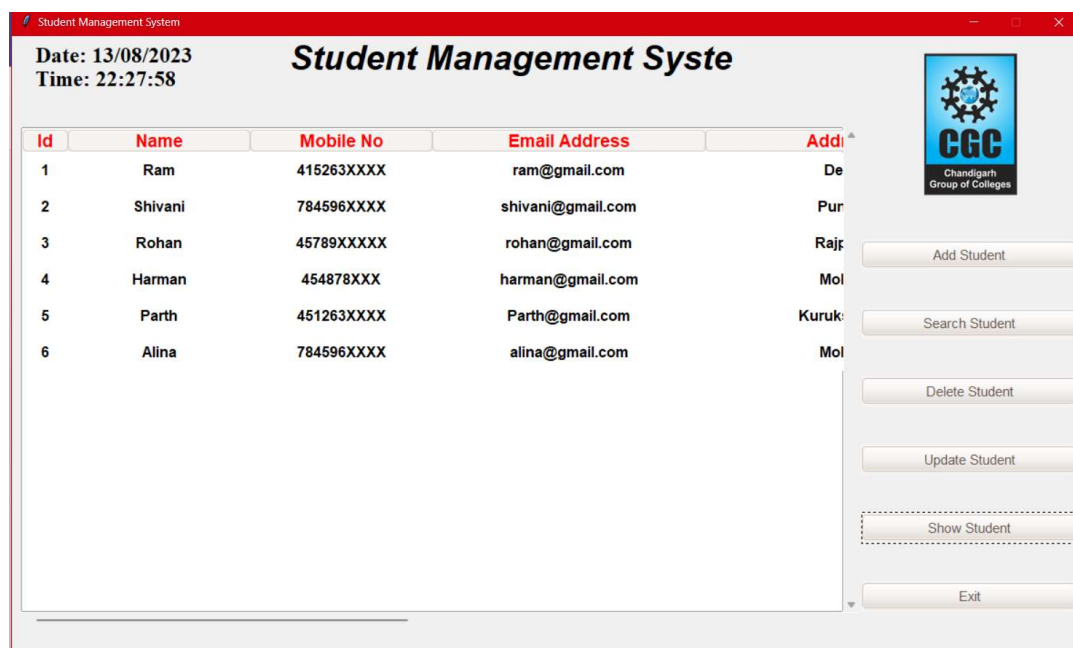
#style.configure('Treeview', rowheight=40,font=('arial', 12, 'bold'),
#fieldbackground='white', background='white',)
#style.configure('Treeview.Heading',font=('arial', 14, 'bold'),foreground='red')
# Update the style to change the text color of the button and Treeview
style.configure('Treeview.Heading', font=('arial', 14, 'bold'), foreground='red')
style.configure('Treeview', rowheight=40, font=('arial', 12, 'bold'),
#fieldbackground='white', background='white', foreground='black')
studentTable.config(show='headings')
root.mainloop()

```

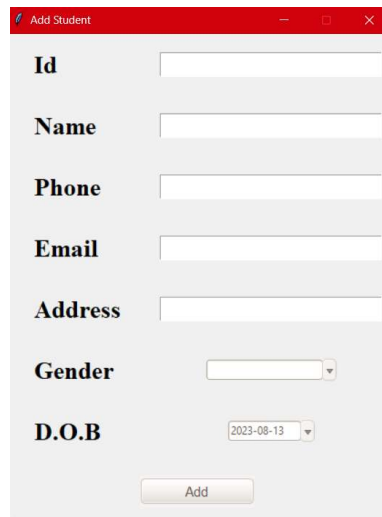
Login page



Main page

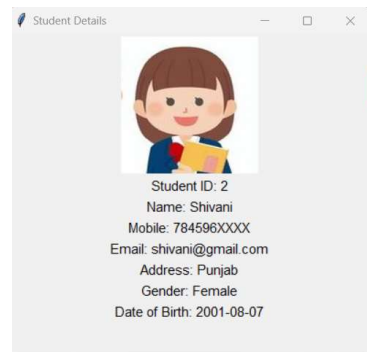
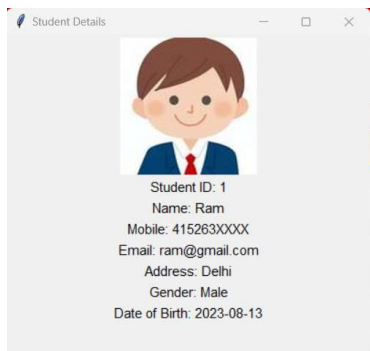


Page for adding and editing already uploaded data.

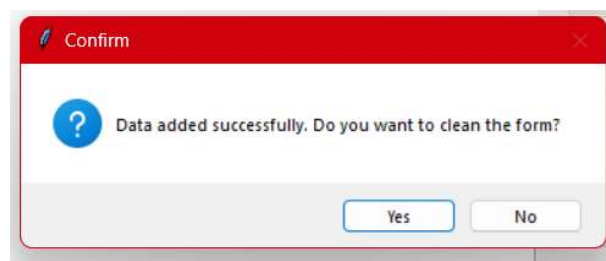
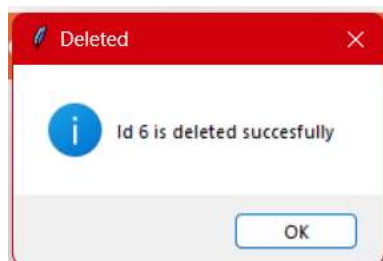


A form titled "Add Student" with a red header bar. It contains the following fields: "Id" (text input), "Name" (text input), "Phone" (text input), "Email" (text input), "Address" (text input), "Gender" (dropdown menu), and "D.O.B" (date picker showing "2023-08-13"). An "Add" button is at the bottom.

Specific student data and image



Some responsive message boxes



Conclusion:

In conclusion, the Student Management System GUI developed using the Tkinter library serves as a powerful tool for educational institutions to manage student records effectively and seamlessly. By combining intuitive user interface design with robust back-end functionality, this application offers a range of features designed to streamline administrative tasks and enhance the overall organization of student information.

The Student Management System GUI exemplifies the potential of combining front-end design with back-end functionality to create a valuable tool for educational institutions. By simplifying administrative tasks, enhancing user experiences, and ensuring data accuracy, this application contributes to efficient management and organization of student information.