STUDENT MANAGEMENT SYSTEM
USING PYTHON, MYSQL

---

1. OBJECTIVE, TOOLS AND TECHNOLOGY, KEY FEATURES

The objective of this project is to develop a GUI-based Student Database Management System that allows educational institutions to manage student data efficiently. It provides functionality to perform basic SQL CRUD operations, real-time clock and date display, and data export capabilities.

TOOLS AND TECHNOLOGIES USED:

Category	**Tools / Technologies**
Programming Language	Python
GUI Library	Tkinter, TTKThemes, PIL
Database	MySQL
IDE/Environment	VS Code

SYSTEM FEATURES:

- Login authentication for secured access
- Add, update, delete, and search student data
- View all student records in a table
- Export student data as CSV
- Animated system title
- Real-time date and time display

2. DATABASE DESIGN

Table: Student

```
CREATE TABLE student (
  id INT PRIMARY KEY,
  name VARCHAR(30),
  mobile VARCHAR(10),
  email VARCHAR(30),
  address VARCHAR(100),
  gender VARCHAR(20),
  dob VARCHAR(20),
  date VARCHAR(50),
  time VARCHAR(50)
);
```

3. MODULE-WISE CODE EXPLANATION

MAIN.PY:

4.1 Login Window

This file handles the login screen.

```
from tkinter import *
from tkinter import messagebox
from PIL import ImageTk
```

•Imports necessary modules for GUI and image handling.

```
def login():
    if usernameEntry.get()=='' or passwordEntry.get()=='':
        messagebox.showerror('Error','Fields cannot be empty')
    elif usernameEntry.get()=='Nand' and passwordEntry.get()=='2003':
        messagebox.showinfo('Success','Welcome')
        window.destroy()
        import sms
    else:
        messagebox.showerror('Error','Please enter correct credentials')
```

•Validates user credentials. Imports sms.py if login is successful.

```
window=Tk()
window.geometry('1280x854')
window.resizable(False,False)
```

•Sets up the main window.

```
backgroundImage=ImageTk.PhotoImage(file='background1.jpg')
bgLabel=Label(window,image=backgroundImage)
bgLabel.place(x=0,y=0)

loginFrame=Frame(window,bg='white')
loginFrame.place(x=400,y=150)
logoImage=PhotoImage(file='student1.png')
logoLabel=Label(loginFrame,image=logoImage)
logoLabel.grid(row=0,column=0,columnspan=2,pady=10)
usernameImage=PhotoImage(file='user.png')
usernameLabel=Label(loginFrame,image=usernameImage,text='Username',compound=LE
FT
                    ,font=('times new roman',20,'bold'),bg='white')
usernameLabel.grid(row=1,column=0,pady=10,padx=20)

usernameEntry=Entry(loginFrame,font=('times new
roman',20,'bold'),bd=5,fg='royalblue')
```

```
usernameEntry.grid(row=1,column=1,pady=10,padx=20)

passwordImage=PhotoImage(file='password.png')
passwordLabel=Label(loginFrame,image=passwordImage,text='Password',compound=LE
FT
                    ,font=('times new roman',20,'bold'),bg='white')
passwordLabel.grid(row=2,column=0,pady=10,padx=20)

passwordEntry=Entry(loginFrame,font=('times new
roman',20,'bold'),bd=5,fg='royalblue',show='*')
passwordEntry.grid(row=2,column=1,pady=10,padx=20)

loginButton=Button(loginFrame,text='Login',font=('times new
roman',14,'bold'),width=15
                   ,fg='white',bg='cornflowerblue',activebackground='cornflowe
rblue',
                   activeforeground='white',cursor='hand2',command=login)
loginButton.grid(row=3,column=1,pady=10)
```
•Displays background and login UI components.

SMS.PY
4.2 Main Student Management System
This file handles all student database operations and GUI components.

Imports:
```
from tkinter import *
import time
import ttkthemes
from tkinter import ttk,messagebox,filedialog
import pymysql
import pandas
```
• Includes modules for GUI, database, and file handling.

iexit Function:
```
def iexit():
    result=messagebox.askyesno('Confirm','Do you want to exit?')
    if result:
        root.destroy()
    else:
        pass
```
• Shows a confirmation box to exit the application.
• If user selects “Yes,” it closes the GUI.

export_data Function:

```
def export_data():
    url=filedialog.asksaveasfilename(defaultextension='.csv')
    indexing=studentTable.get_children()
    newlist=[]
    for index in indexing:
        content=studentTable.item(index)
        datalist=content['values']
        newlist.append(datalist)

    table=pandas.DataFrame(newlist,columns=['Id','Name','Mobile','Email','Addr
ess','Gender','DOB','Added Date','Added Time'])
    table.to_csv(url,index=False)
    messagebox.showinfo('Success','Data is saved succesfully')
```

• Exports student data from the Tree view to a CSV file using pandas.
• Opens file dialog, gathers table content, and writes it to CSV.

toplevel_data Function:

```
def toplevel_data(title,button_text,command):
    global
idEntry,phoneEntry,nameEntry,emailEntry,addressEntry,genderEntry,dobEntry,scre
en
    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)
    genderEntry = Entry(screen, font=('roman', 15, 'bold'), width=24)
    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)
    dobEntry = Entry(screen, font=('roman', 15, 'bold'), width=24)
    dobEntry.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])
        dobEntry.insert(0, listdata[6])
```

- Creates a new popup form for **Add**, **Update**, or **Search** operations.
- Fields: Id, Name, Phone, Email, Address, Gender, DOB.
- Dynamically adds values if the title is 'Update Student'.

update_data Function:

```
def update_data():
    query='update student set
name=%s,mobile=%s,email=%s,address=%s,gender=%s,dob=%s,date=%s,time=%s where
id=%s'
    mycursor.execute(query,(nameEntry.get(),phoneEntry.get(),emailEntry.get(),
addressEntry.get(),
                            genderEntry.get(),dobEntry.get(),date,currenttime,
idEntry.get()))
```

```
    con.commit()
    messagebox.showinfo('Success',f'Id {idEntry.get()} is modified 
successfully',parent=screen)
    screen.destroy()
    show_student()
```

•Updates the selected student's info in the MySQL database.
•Uses the fields from toplevel_data.
•Refreshes the table after updating

show_student Function:

```
def show_student():
    query = 'select * from student'
    mycursor.execute(query)
    fetched_data = mycursor.fetchall()
    studentTable.delete(*studentTable.get_children())
    for data in fetched_data:
        studentTable.insert('', END, values=data)
```

•Fetches all student records from the database.
•Clears the current table and displays fresh data.

delete_student Function:

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

•Deletes the selected student from the database.
•Refreshes the table after deletion.

search_data Function:

```
def search_data():
    query='select * from student where id=%s or name=%s or email=%s or 
mobile=%s or address=%s or gender=%s or dob=%s'
    mycursor.execute(query,(idEntry.get(),nameEntry.get(),emailEntry.get(),pho
neEntry.get(),addressEntry.get(),genderEntry.get(),dobEntry.get()))
```

```
    studentTable.delete(*studentTable.get_children())
    fetched_data=mycursor.fetchall()
    for data in fetched_data:
        studentTable.insert('',END,values=data)
```

•Searches the student table by any matching field (Id, Name, etc.).
•Displays only matched results in the table.

add_data Function:

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

    else:
        try:
            query='insert into student values(%s,%s,%s,%s,%s,%s,%s,%s,%s)'
            mycursor.execute(query,(idEntry.get(),nameEntry.get(),phoneEntry.g
et(),emailEntry.get(),addressEntry.get(),
                                    genderEntry.get(),dobEntry.get(),date,curr
enttime))
            con.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)
                dobEntry.delete(0,END)
            else:
                pass
        except:
            messagebox.showerror('Error','Id cannot be 
repeated',parent=screen)
            return

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

•Adds a new student record.
•Checks if fields are empty before adding.

•If ID already exists, throws an error.
•After successful insert, asks user whether to clear form.

connect_database Function:

```
def connect_database():
    def connect():
        global mycursor,con
        try:
            con=pymysql.connect(host=hostEntry.get(),user=usernameEntry.get(),
password=passwordEntry.get())
            mycursor=con.cursor()
        except:
            messagebox.showerror('Error','Invalid
Details',parent=connectWindow)
            return

        try:
            query='create database studentmanagementsystem'
            mycursor.execute(query)
            query='use studentmanagementsystem'
            mycursor.execute(query)
            query='create table student(id int not null primary key, name
varchar(30),mobile varchar(10),email varchar(30),' \
                  'address varchar(100),gender varchar(20),dob
varchar(20),date varchar(50), time varchar(50))'
            mycursor.execute(query)
        except:
            query='use studentmanagementsystem'
            mycursor.execute(query)
        messagebox.showinfo('Success', 'Database Connection is successful',
parent=connectWindow)
        connectWindow.destroy()
        addstudentButton.config(state=NORMAL)
        searchstudentButton.config(state=NORMAL)
        updatestudentButton.config(state=NORMAL)
        showstudentButton.config(state=NORMAL)
        exportstudentButton.config(state=NORMAL)
        deletestudentButton.config(state=NORMAL)


    connectWindow=Toplevel()
    connectWindow.grab_set()
    connectWindow.geometry('470x250+730+230')
    connectWindow.title('Database Connection')
    connectWindow.resizable(0,0)
    hostnameLabel=Label(connectWindow,text='Host
Name',font=('arial',20,'bold'))
    hostnameLabel.grid(row=0,column=0,padx=20)

    hostEntry=Entry(connectWindow,font=('roman',15,'bold'),bd=2)
    hostEntry.grid(row=0,column=1,padx=40,pady=20)

    usernameLabel = Label(connectWindow, text='User Name', font=('arial', 20,
'bold'))
    usernameLabel.grid(row=1, column=0, padx=20)

    usernameEntry = Entry(connectWindow, font=('roman', 15, 'bold'), bd=2)
    usernameEntry.grid(row=1, column=1, padx=40, pady=20)

    passwordLabel = Label(connectWindow, text='Password', font=('arial', 20,
'bold'))
    passwordLabel.grid(row=2, column=0, padx=20)

    passwordEntry = Entry(connectWindow, font=('roman', 15, 'bold'),
bd=2,show='*')
    passwordEntry.grid(row=2, column=1, padx=40, pady=20)

    connectButton=ttk.Button(connectWindow,text='CONNECT',command=connect)
    connectButton.grid(row=3,columnspan=2)

count=0
text=''
```
•Opens a new window to take MySQL **host**, **username**, and **password**.
•Connects to MySQL and creates studentmanagementsystem database and student table if not exists.
•Enables all buttons after successful connection.

slider Function:

```
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)
```
•Creates a text animation in the title "Student Management System".
•Updates every 300ms for a sliding effect.

clock Function:

```
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)
```

•Displays live date and time on the top-left corner.
•Updates every second using after.

GUI Initialization:

```
root=ttkthemes.ThemedTk()

root.get_themes()

root.set_theme('radiance')

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

•Initializes themed root window using ttkthemes.
•Sets size, title, and disables resizing.

GUI Components:

Datetime + Slider Title

```
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()

connectButton=ttk.Button(root,text='Connect
database',command=connect_database)
connectButton.place(x=980,y=0)
```

•Top bar showing current date and time (updated by clock).
•Scrolling text below it (handled by slider).

Left Frame (Sidebar):

```
leftFrame=Frame(root)
leftFrame.place(x=50,y=80,width=300,height=600)
```

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

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

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

exportstudentButton=ttk.Button(leftFrame,text='Export
data',width=25,state=DISABLED,command=export_data)
exportstudentButton.grid(row=6,column=0,pady=20)

exitButton=ttk.Button(leftFrame,text='Exit',width=25,command=iexit)
exitButton.grid(row=7,column=0,pady=20)
```
•Contains: Student image, Buttons for: Add, Search, Delete, Update, Show, Export, Exit.
•Buttons disabled initially. Enabled after DB connect.

Right Frame (Treeview Table):

```
rightFrame=Frame(root)
rightFrame.place(x=350,y=80,width=820,height=600)

scrollBarX=Scrollbar(rightFrame,orient=HORIZONTAL)
scrollBarY=Scrollbar(rightFrame,orient=VERTICAL)

studentTable=ttk.Treeview(rightFrame,columns=('Id','Name','Mobile','Email','Ad
dress','Gender',
                                        'D.O.B','Added Date','Added Time'),
                                xscrollcommand=scrollBarX.set,yscrollcommand=scrollB
arY.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.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)
```
•Shows student records in a table format.
•Scrollbars for navigation.
•Custom styles for better appearance.

Treeview Configuration:
```
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')

studentTable.config(show='headings')
```
•Sets headings and column properties for Treeview.
•Applies font and color styles for header and rows.

Main Loop:

```
root.mainloop()
```

•Keeps the GUI window running and responsive.

4. SAMPLE SQL QUERIES

- View all students
  ```
  SELECT * FROM student;
  ```

- Search student by ID
  ```
  SELECT * FROM student WHERE id = 101;
  ```

- Delete student by ID
  ```
  DELETE FROM student WHERE id = 101;
  ```

- Update student email
  ```
  UPDATE student SET email = 'newemail@example.com'
  WHERE id = 101;
  ```

5. CHALLENGES AND LEARNINGS

Challenges:

1. Managing the GUI state before/after DB connection
   - ➢ All operation buttons remain disabled until a successful database connection is established, ensuring the system is not used prematurely.
2. Preventing duplicate entries and handling errors gracefully
   - ➢ User inputs are validated and wrapped in try-except blocks to prevent duplicate IDs and handle insertion or connection errors without crashing.
3. Linking the GUI components with database logic efficiently
   - ➢ Each GUI action is tightly integrated with backend SQL queries through well-structured functions, allowing smooth data flow between interface and database.

Learnings:

- ➢ Mastered CRUD operations with MySQL in Python.
- ➢ Built a responsive and user-friendly GUI with Tkinter.
- ➢ Learned modular programming by separating login and functionality logic.
- ➢ Used pandas for real-world data export tasks.