ABSTRACT OF THE WORK :

Student Information Management System can be used by education institutes to maintain the records of students easily. Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. All these problems are solved using this project.

The main purpose of this project is to build a face recognition-based attendance monitoring system for educational institution to enhance and upgrade the current attendance system into more efficient and effective as compared to before. The current old system has a lot of ambiguity that caused inaccurate and inefficient of attendance taking. Many problems arise when the authority is unable to enforce the regulation that exist in the old system. The technology working behind will be the face recognition system. The human face is one of the natural traits that can uniquely identify an individual. Therefore, it is used to trace identity as the possibilities for a face to deviate or being duplicated is low. In this project, face databases will be created to pump data into the recognizer algorithm. Then, during the attendance taking session, faces will be compared against the database to seek for identity. When an individual is identified, its attendance will be taken down automatically saving necessary information into a excel sheet. At the end of the day, the excel sheet containing attendance information regarding all individuals are mailed to the respective faculty.

OBJECTIVES:

It is mainly useful for educational establishments to manage student data which also facilitates all individual associated Information for easier navigation on daily basis. It provides capabilities for entering student details, tracking student fee data needs in a college. Our easy-to-use, integrated college administration application would be used to reduce time spent on administrative tasks, as to concentrate on other skillful practical activities other than book worming. It can accept, process and generate reports at any given point of time accurately.

TABLE OF CONTENTS

TITLE

ABSTRACT

LIST OF FIGURE

CHAPTER 1 - 1. INTRODUCTION

CHAPTER 2 - 2. LITERATURE SURVEY

CHAPTER 3 - 3.AIM & SCOPE OF THE PRESENT INVESTIGATION

CHAPTER 4 - 4. METHODOLOGY

CHAPTER 5 - 5. RESULT AND DISCUSSION

CHAPTER 6 - 6.CONCLUSION AND FUTURE WORK

CHAPTER 7 - 7.REFERENCES

CHAPTER 8 - 8.APPENDICIES

CHAPTER 1

1.INTRODUCTION

The mission of the Student Information Management system is to create an integrated information technology environment for students, HOD, faculty, staff and administration. Our goal is to focus on services and integration for end users. It is a simple GUI based self service environment for an administrative transaction processing environment for yearly admissions; It is mainly useful for educational establishments to manage student data which also facilitates all individual associated information for easier navigation on daily basis. It provides capabilities for entering student details, tracking student fee data needs in a college. Our easy-to-use, integrated college administration application would be used to reduce time spent on administrative tasks, as to concentrate on other skill ful practical activities other than book worming. It can accept, process and generate reports at any given point of time accurately. Talking about the system, it consists of basic features which encompass Add students, view college students, search college students and get rid of the student. Also it maintains the fee payment details of the students. The following presentation provides the specification for the system.

CHAPTER 2

2. LITRERATURE OF SURVEY

2.1 Automatic Attendance Monitoring System

According to (P. Padma Rekha, V.Narendhiran, D. Amudhan, S Ramya and N. Pavithra 2016) attendance is taken in every organization. The traditional approach for attendance is the professor calls student name and records attendance. For each lecture this is wastage of time. The study maintained that to avoid losses, an automatic process based on image processing is to be utilized. This project approach is to use face detection and face recognition system.

2.2 Biometric Student Identification

According to (Raymond J. D. Anne Marie Dunphy March 2015), One of the many challenges facing schools today is accurately identifying students. The bottom line is that schools receive federal and state money based on accurate and auditable records. In addition, schools need to provide a safe and secure environment for everyone on campus. Now more than ever, accurate student identification is the key to the efficient operation of a school. Over the past few decades, schools have been implementing all kinds of new technologies to both enhance and improve operations. Smart boards, laptop and real-time internet resources are just a part of a student's everyday experience. Schools worldwide have been implementing biometric finer scanning to streamline operations, increase teaching time and enhances security.

2.3 Attendance Monitoring

According to Casas, Ibanez, and Pilongco (2013), this is a study that tries to formulate an easy way of monitoring the attendance of students and faculty. This study includes the use of a radio frequency identification card reader and tags. With the help of this study, the reader will be able to see the flow of attendance of every Faculty and students in their respective classes.

2.4 School Management Software

According to (Ren Web 2019) is school information System software. It allows the user to take attendance, track grades, and generate report cards, as well as a HOST of otherfeatures. Very features-rich. In years past, Renweb sot of ost its way with not really understanding who its coe customer was k-12 schools.

2.4 Veyon

According to (Morpus 2017) Veyon is an open sourced digital classroom management software built off of ITALC that gives you complete control of your student's computers, allowing you to display lessons, aid in corrections, view workstation progress, and lock out particular users. The first impression I had about Veyon was how clean the user interface was for a free and open-source software. Veyon is visually pleasing and straightforward with 3D icons, simple prompts, and easy navigation. Adding to its user-friendly nature, Veyon is also easy to install and run on any major operating System (Windows, Linux OSX).

CHAPTER 3

3.1 AIM

To develop software for solving maintaining the details of a students in administration in order to nurture the needs of an student adminstrators by providing various ways to perform tasks. The project is totally built at administrative end and thus only the administrator is guaranteed the access.

3.2 SCOPE AND OBJECTIVE

- This system is aimed at total user-friendly as well as efficient management of varied tasks. These tasks may range from registering new students, managing fees payment, examination management to all the essential features necessary for making the administrative division of school effective.

- It is mainly useful for educational establishments to manage student data which also facilitates all individual associated information for easier navigation on daily basis. It provides capabilities for entering student details, tracking student fee data needs in a college.

1. The main objective of the Student Management System is to manage the details of Profiles, Course, Logins, Exams, Fees. It manages all the information about Profiles, Student, Fees, Profiles.
2. Our easy-to-use, integrated college administration application would be used to reduce time spent on administrative tasks, as to concentrate on other skill full practical activities other than book worming.
3. It can accept, process and generate reports at any given point of time accurately.

SYSTEM REQUIREMENTS

3.3.1 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. The minimal hardware requirements are as follows,

1.Processor : Pentium IV Above

2. RAM : 4 GB and Above

3. Processor : 2.4 GHz

4. Main Memory : 80 GB and Above

5. Hard Disk Drive : 1 TB

6. Keyboard : 104 Keys

3.3.2 Software Requirements

Software requirements deals with defining resource requirements and prerequisites that needs to be installed on a computer to provide functioning of an application. The minimal software requirements are as follows,

1. Front end : python
2. Dataset : sqlite3
3. IDE : pycharm (platform)
4. Operating System : Windows 10

3.4 SOFTWARE USED:

3.4.1 Python Language

Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances. Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python is widely considered one of the easiest programming languages for a beginner to learn, but it is also difficult to master.

3.4.2 Python Characteristics

- Python is an extensible programming language. Python can be written in C/C++ language and compiled in its compiler too.
- It is a platform independent scripted language with full access to operating system
  API's
- Compared to other programming languages, it allows more run-time flexibility
- Python supports functional and structured programming as well as OOP
- Python is an interpreted language means its programming code can be executed line by line at a time. This makes debugging easier and handier.
- It is a dynamic and high-level open-source programming language that supports both object-oriented and procedural-oriented language.
- Graphical User interfaces can be made using a module such as wx Python, or Tk in python.

CHAPTER 4.

4. METHODOLOGY

The project is divided into 2 Main modules such as:

4.1. Student record

4.2. Fees detail

We have sub modules for this:

4.3 Add record modules

4.4 Delete record modules

4.5 Reset record modules

4.6 User modules

4.1 Student record

Student record management system is **designed to help manage the daily school activities of recording and maintaining by automating it**. It is also known as the student information system (SIS) or school records system (SRS).

4.2 Fees detail

This module represents how the transactions are doing everyday. It manages all the fees transactions of students. So it's easier the maintain accounting books of students. In this screen, it also shows all the fee payment details of the students. While entering the fee details .It indicated the balance payment to make the student.

4.3 Add record

Add Record Module is used to add details about students like Name, Date of birth, Gender, Department, Mobile Number and Total Fees

After Adding a Record you can see the indication Record Was Successfully Added

4.4 Delete record

Delete Record Module Is used to Delete the saved details easily

Select the name which you have delete it

Click the delete record button

Now can see the indication Record was deleted Successfully

4.5 Reset record

Reset Field Module is used to clear the details to add another details

Now you can add another details easily

4.6 User modules

Purpose of project is to maintain details of the students such as storing information about:

- Student Name
- Department
- Mobile Number
- Gender
- Total Fees

CHAPTER 5

5. RESULTS AND DISCUSSIONS

5.1. Performance Requirements:

Our software will perform and fulfill all the tasks that any Student Adminstrator would desire. It is developed as a software program for managing the Student Adminstration related to keep each every track about their details efficiently. Hereby, our main objective is the customer's satisfaction considering today's faster world.

5.2. Security Requirements:

Student who will be viewing either all or some specific information form the database. Depending upon the category of user the access rights are decided. It means if the user is an administrator then he can be able to modify the data, append etc.

5.3 DISCUSSION

5.3.1 Purpose:

Talking about the system, it consists of basic features which encompass Add students,view college students, search college students and get rid of the student. Also it maintains the fee payment details of the students. The following presentation provides the specification for the system.

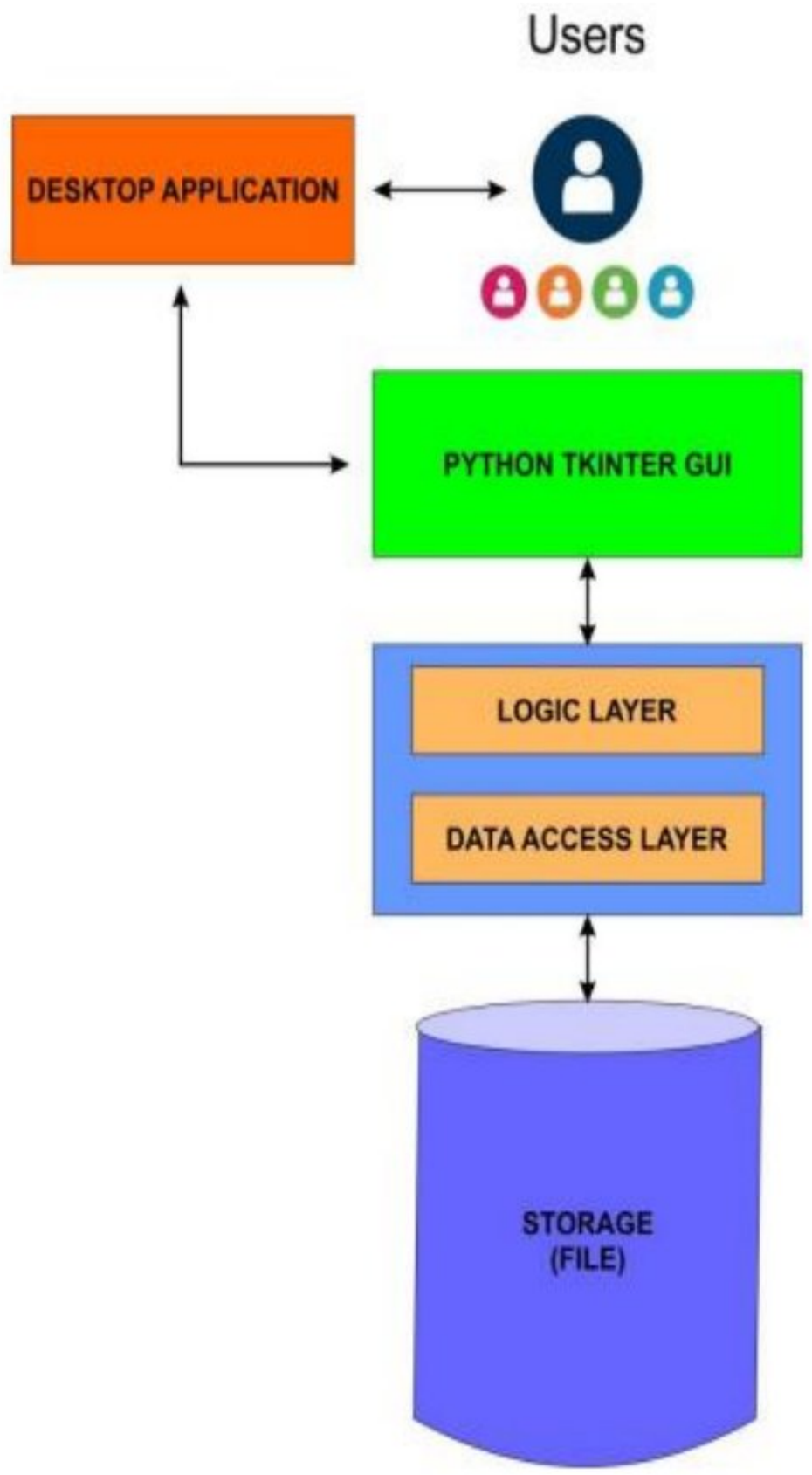
5.3.2 Overall System Design Objectives:

The availability of the software's for almost every process or every system has taken the world in its top-gear and fastens the day-to-day life. So, we have tried our best to develop the software program for the Student Management System where all the tasks to manage the admin operations are performed easily and efficiently. Thus, above features of this software will save transaction time and therefore increase the efficiency of the system.

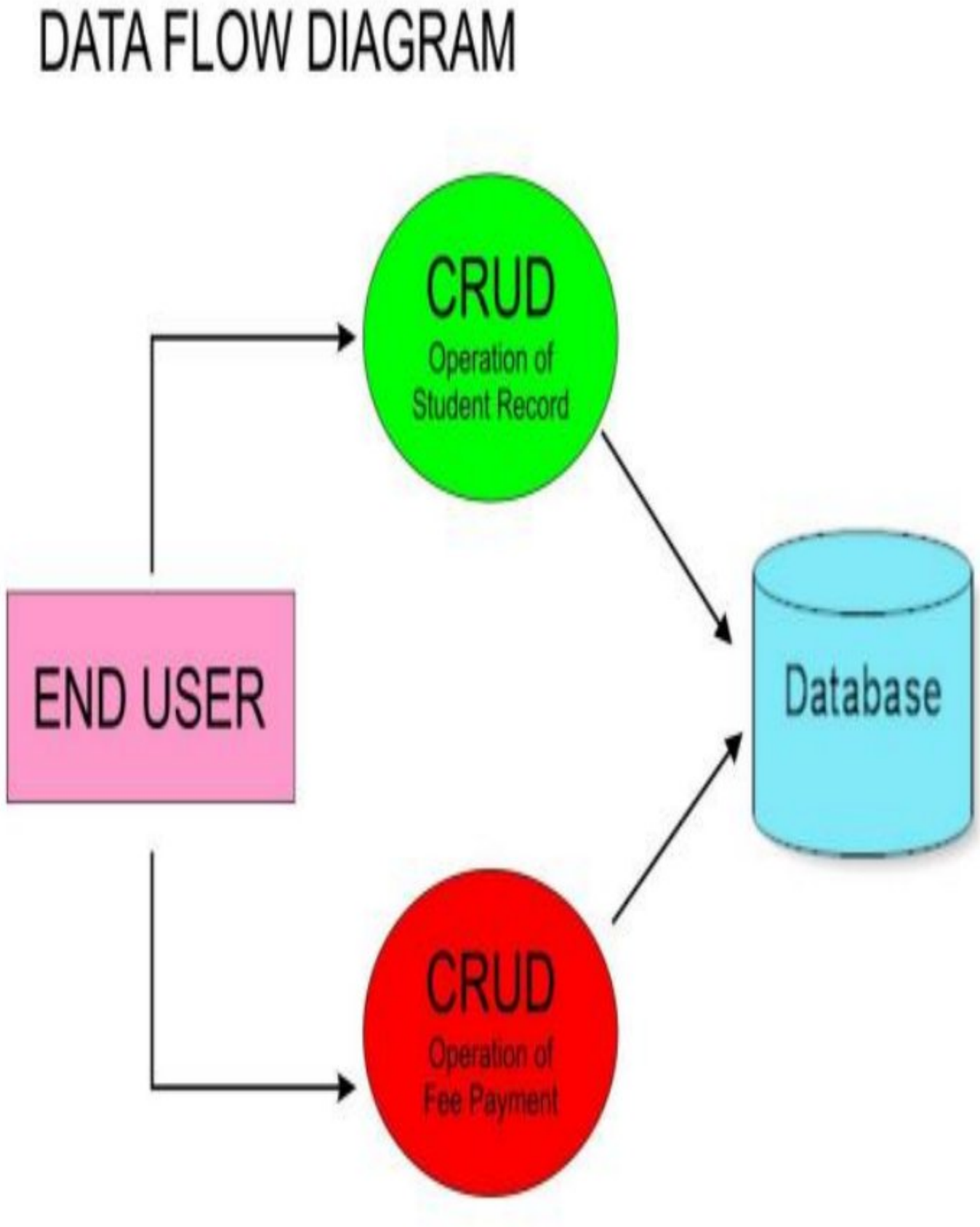
CHAPTER 6

6.CONCLUSION AND FUTURE WORK:

6.1 APPLICATION ARCHITECTURE DIAGRAM:



6.2 - DATA FLOW DESIGN:

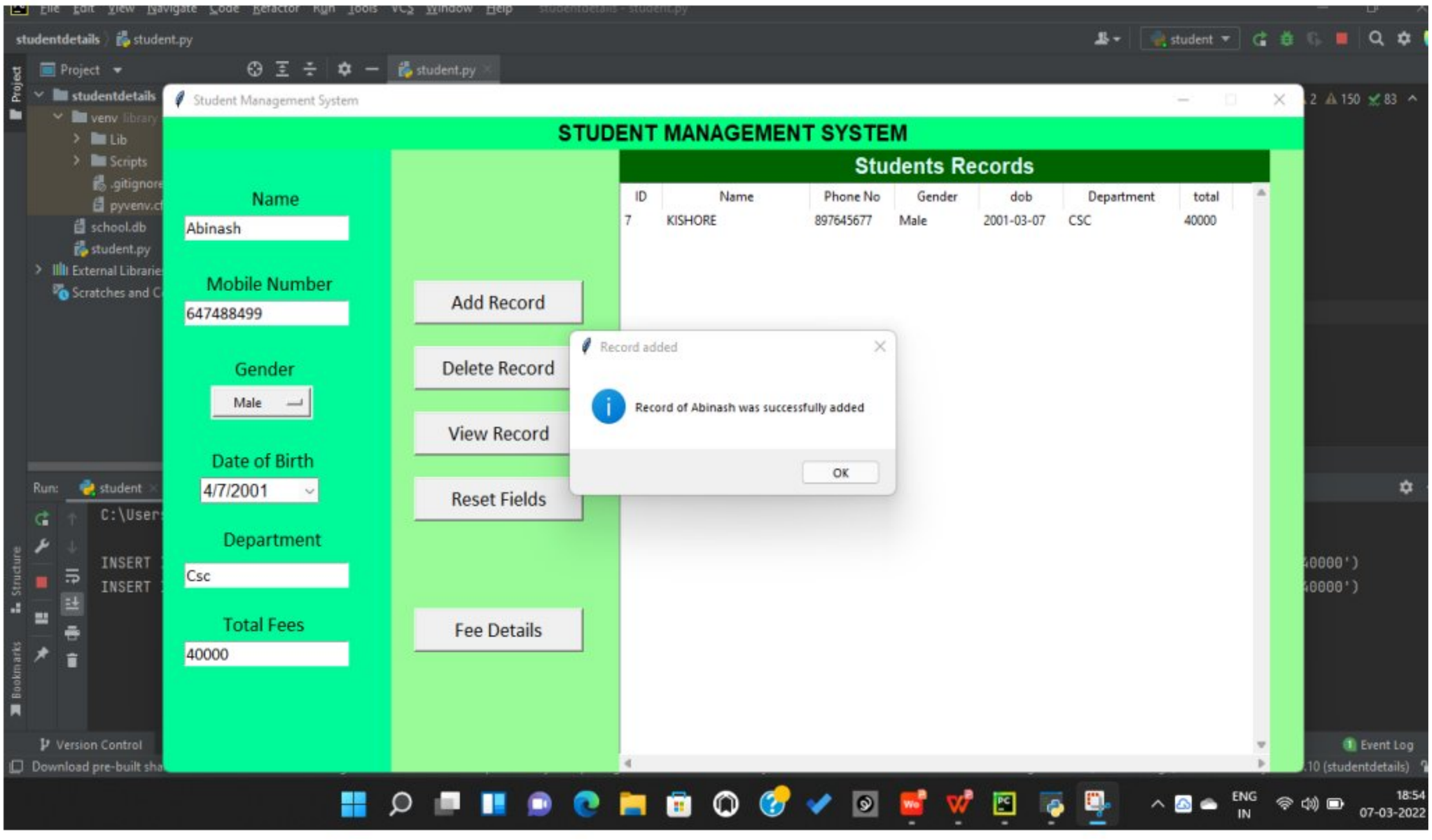


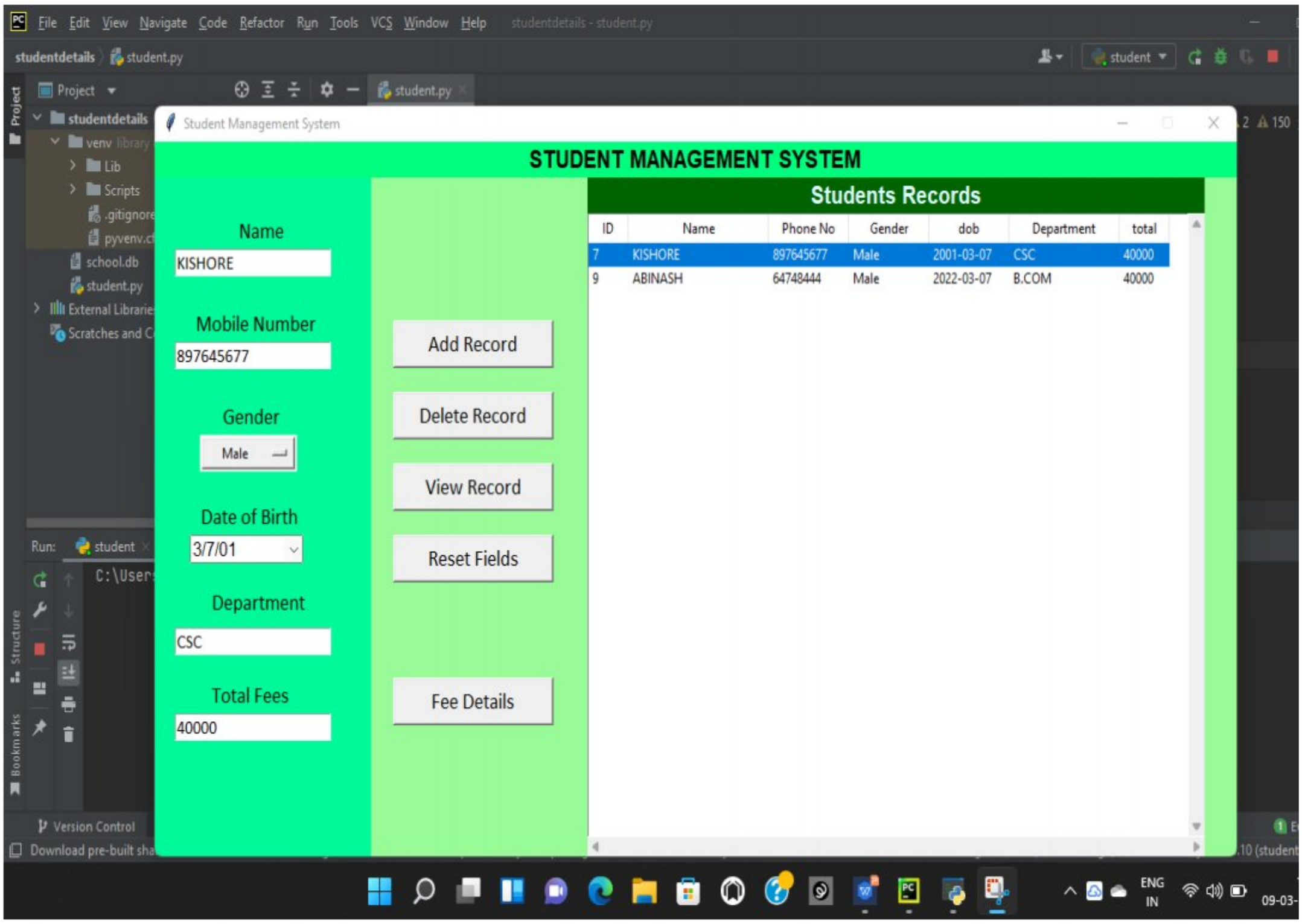
6.3 - FINAL CONCLUSION:

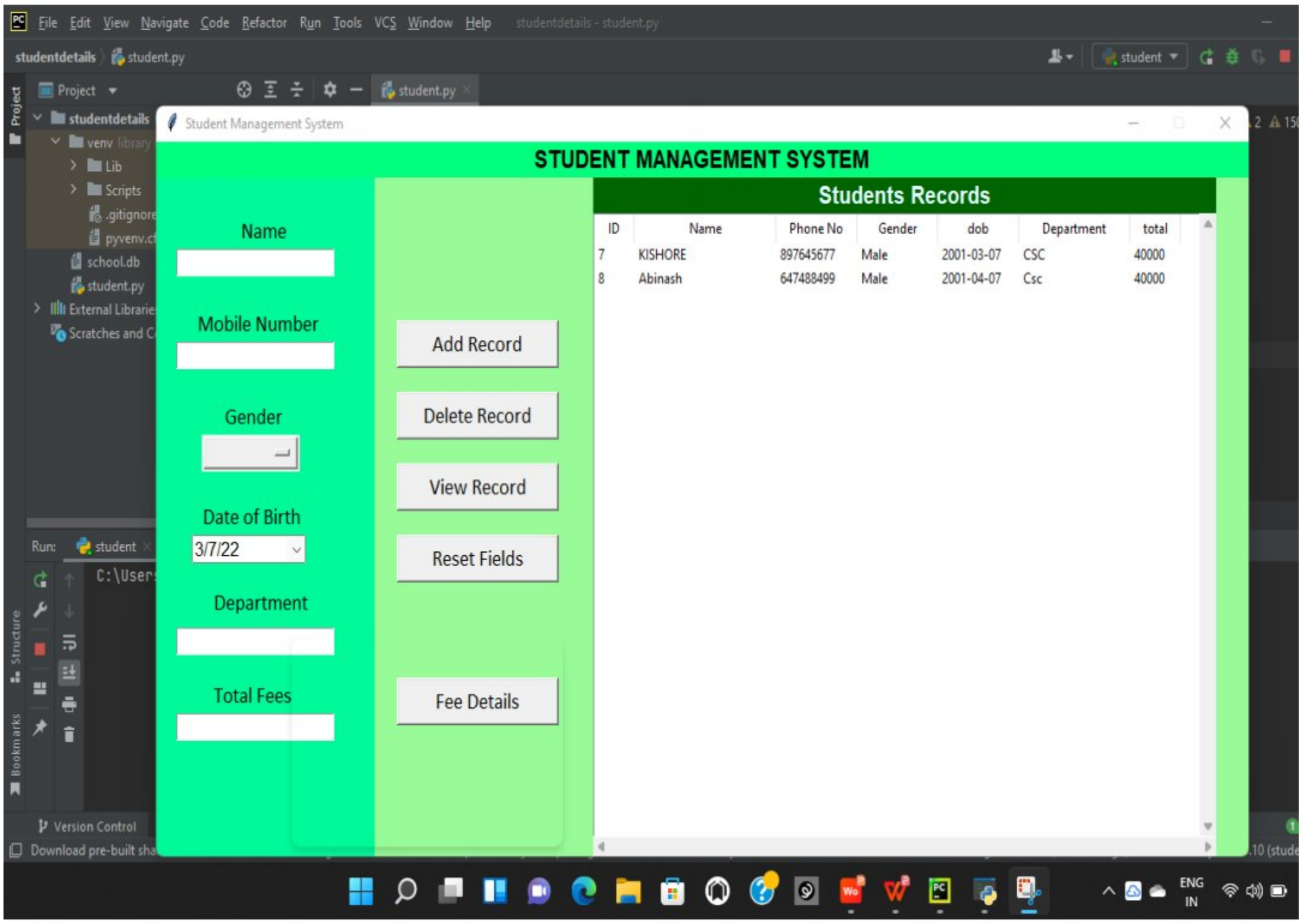
It is always prudent to opt for a student information system that is designed using modern system architecture to cope with changing requirements. This system should encompass very solid information coding and distinctly outlined business applications. The overview of system elaborates the ease of information delivery at the tip of your fingers with precise data and increases the retention rate of student and teaches them how to manage their time efficiently

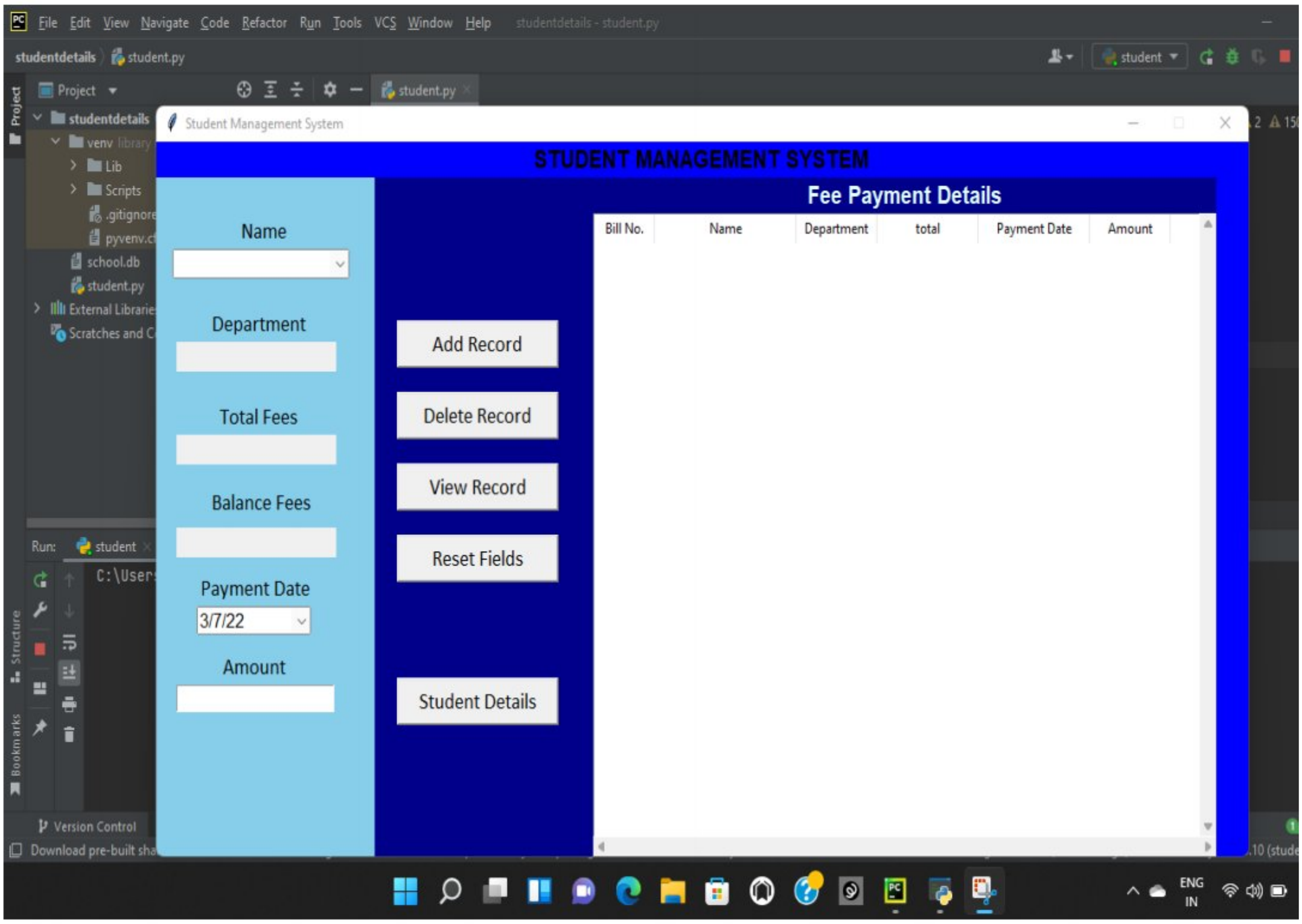
7.REFERENCES:

- Python For Desktop Applications: How to develop, pack and deliver Python applications with TkInter by Tran Duc Loi
- Python in A Nutshell: A Desktop Quick Reference, Third Edition by Alex Martelli ,
  Anna Ravenscroft , Steve Holden
- The Python Language Reference Manual (version 3.2) by Guido van Rossum, and Fred L. Drake, Jr. (Editor)









8. APPENIDICES:

SOURCE CODE:

8.1 - SAMPLE CODE:

```
import datetime

from tkinter import *

import tkinter.messagebox as mb

from tkcalendar import DateEntry

import sqlite3

import ctypes

from tkinter import ttk



user32 = ctypes.windll.user32

xpos = int((user32.GetSystemMetrics(0) - 1080) / 2)

ypos = int((user32.GetSystemMetrics(1) - 600) / 2)



headlabelfont = ("Arial", 15, 'bold')

labelfont = ('Calibri', 14)

entryfont = ('Calibri', 12)

rows=[]

connector = sqlite3.connect('school.db')

cursor = connector.cursor()

currentwindow=''
```

```
connector.execute("CREATE TABLE IF NOT EXISTS MANAGEMENT "

                  "(STUDENT_ID INTEGER PRIMARY KEY AUTOINCREMENT
NOT NULL, NAME TEXT, "

                  "PHONE_NO TEXT, GENDER TEXT, DOB TEXT,
DEPARTMENT TEXT, TOTALFEES TEXT)"

                  )

connector.execute("CREATE TABLE IF NOT EXISTS FEES "

                    "(BILL_NO INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, STUDENT_ID INTEGER, "

                    "BILL_DATE TEXT, AMOUNT TEXT)"

                  )



def reset_fields():

    global name_strvar, contact_strvar, gender_strvar, department_strvar,
total,dob

    for i in ['name_strvar', 'contact_strvar', 'gender_strvar',
'department_strvar','total']:

        exec(f"{i}.set('')")

    dob.set_date(datetime.datetime.now().date())



def reset_fee_fields():

    global namebox, dept_strvar, tfees_strvar,amount,db

    for i in ['namebox',  'dept_strvar', 'tfees_strvar','amount']:

        exec(f"{i}.set('')")
```

```
    db.set_date(datetime.datetime.now().date())


def display_records():
    tree.delete(*tree.get_children())
    curr = connector.execute('SELECT * FROM management')
    data = curr.fetchall()

    for records in data:
        tree.insert('', END, values=records)


def add_record():
    global name_strvar, contact_strvar, gender_strvar, dob,
department_strvar, total

    name = name_strvar.get()
    contact = contact_strvar.get()
    gender = gender_strvar.get()
    dobv = dob.get_date()
    department = department_strvar.get()
    totalfees = total.get()
```

```
    if not name  or not contact or not gender or not dobv or not department
or not totalfees:
        mb.showerror('Error!', "Please fill all the missing fields!!")
    else:
        try:
            print("INSERT INTO management (NAME, PHONE_NO,
GENDER, DOB, DEPARTMENT, TOTALFEES) VALUES
('{}','{}','{}','{}','{}','{}')".format(name, contact, gender, dob, department,
totalfees))
            connector.execute('INSERT INTO MANAGEMENT (NAME,
PHONE_NO, GENDER, DOB, DEPARTMENT, TOTALFEES) VALUES
(?,?,?,?,?,?)', (name,  contact, gender, dobv, department, totalfees))
            connector.commit()
            mb.showinfo('Record added', f"Record of {name} was successfully
added")
            reset_fields()
            display_records()
        except Exception as e:
            print(e)
            mb.showerror('Wrong type', 'The type of the values entered is not
accurate. Pls note that the contact field can only contain numbers')

def remove_record():
    global tree
```

```
if not tree.selection():
    mb.showerror('Error!', 'Please select an item from the database')
else:
    current_item = tree.focus()
    values = tree.item(current_item)
        selection = values["values"]

        tree.delete(current_item)

        connector.execute('DELETE FROM management WHERE
STUDENT_ID=%d' % selection[0])
        connector.commit()
        mb.showinfo('Done', 'The record you wanted deleted was
successfully deleted.')

        display_records()

def load_student():
    global namebox,rows
curr = connector.execute('SELECT * FROM management')
data = curr.fetchall()
names=[]
```

```
rows=[]
for x in data:
    rows.append(x)
    names.append(x[1])
namebox['values']=names
```

```
def view_record():
    global name_strvar, contact_strvar, gender_strvar, dob,
department_strvar, total

    current_item = tree.focus()
    values = tree.item(current_item)
    selection = values["values"]

    if len(selection)==0:
        mb.showinfo('Done', 'Select any Student to view their details')
        return
    date = datetime.date(int(selection[4][:4]), int(selection[4][5:7]),
int(selection[4][8:]))

    name_strvar.set(selection[1])
    contact_strvar.set(selection[2])
    gender_strvar.set(selection[3])
```

```
    dob.set_date(date)

    department_strvar.set(selection[5])

    total.set(selection[6])


def mainwindow():
    global main,tree,fee,currentwindow

    global name_strvar, contact_strvar, gender_strvar, dob,
department_strvar, total

    print(currentwindow)

    if currentwindow=='fee':

        fee.destroy()

    currentwindow="main"

    main = Tk()

    main.title('Student Management System')

    main.geometry('1080x600+{}+{}'.format(xpos, ypos))

    main.resizable(0, 0)

    main.configure(background="PaleGreen")

    lf_bg = 'MediumSpringGreen' # bg color for the left_frame

    cf_bg = 'PaleGreen' # bg color for the center_frame


    name_strvar = StringVar()

    contact_strvar = StringVar()

    gender_strvar = StringVar()
```

```
department_strvar = StringVar()

total= StringVar()


Label(main, text="STUDENT MANAGEMENT SYSTEM",
font=headlabelfont, bg='SpringGreen').pack(side=TOP, fill=X)

left_frame = Frame(main, bg=lf_bg)

left_frame.place(x=0, y=30, relheight=1, relwidth=0.2)


center_frame = Frame(main, bg=cf_bg)

center_frame.place(relx=0.2, y=30, relheight=1, relwidth=0.2)


right_frame = Frame(main, bg="Gray35")

right_frame.place(relx=0.4, y=30, relheight=1, relwidth=0.57)


Label(left_frame, text="Name", font=labelfont,
bg=lf_bg).place(relx=0.375, rely=0.05)

Label(left_frame, text="Mobile Number", font=labelfont,
bg=lf_bg).place(relx=0.175, rely=0.18)

Label(left_frame, text="Gender", font=labelfont,
bg=lf_bg).place(relx=0.3, rely=0.31)

Label(left_frame, text="Date of Birth ", font=labelfont,
bg=lf_bg).place(relx=0.2, rely=0.45)

Label(left_frame, text="Department", font=labelfont,
bg=lf_bg).place(relx=0.25, rely=0.57)
```

```
    Label(left_frame, text="Total Fees", font=labelfont,
bg=lf_bg).place(relx=0.25, rely=0.70)

    Entry(left_frame, width=19, textvariable=name_strvar,
font=entryfont).place(x=20, rely=0.1)

    Entry(left_frame, width=19, textvariable=contact_strvar,
font=entryfont).place(x=20, rely=0.23)

    OptionMenu(left_frame, gender_strvar, 'Male', "Female").place(x=45,
rely=0.36, relwidth=0.45)

    Entry(left_frame, width=19, textvariable=department_strvar,
font=entryfont).place(x=20, rely=0.63)

    Entry(left_frame, width=19, textvariable=total,
font=entryfont).place(x=20, rely=0.75)

    dob = DateEntry(left_frame, font=("Arial", 12), width=10)

    dob.place(x=35, rely=0.5)

    Button(center_frame, text='Add Record', font=labelfont,
command=add_record, width=15).place(relx=0.1, rely=0.20)

    Button(center_frame, text='Delete Record', font=labelfont,
command=remove_record, width=15).place(relx=0.1, rely=0.30)

    Button(center_frame, text='View Record', font=labelfont,
command=view_record, width=15).place(relx=0.1, rely=0.40)

    Button(center_frame, text='Reset Fields', font=labelfont,
command=reset_fields, width=15).place(relx=0.1, rely=0.50)
```

```
    Button(center_frame, text='Fee Details', font=labelfont,
command=feewindow, width=15).place(relx=0.1, rely=0.70)

    Label(right_frame, text='Students Records', font=headlabelfont,
bg='DarkGreen', fg='LightCyan').pack(side=TOP, fill=X)

    tree = ttk.Treeview(right_frame, height=100, selectmode=BROWSE,
                        columns=('Student ID', "Name", "Contact Number",
"Gender", "Date of Birth", "Department","Fees"))

    X_scroller = Scrollbar(tree, orient=HORIZONTAL, command=tree.xview)
    Y_scroller = Scrollbar(tree, orient=VERTICAL, command=tree.yview)

    X_scroller.pack(side=BOTTOM, fill=X)
    Y_scroller.pack(side=RIGHT, fill=Y)

    tree.config(yscrollcommand=Y_scroller.set,
xscrollcommand=X_scroller.set)

    tree.heading('Student ID', text='ID', anchor=CENTER)
    tree.heading('Name', text='Name', anchor=CENTER)
    tree.heading('Contact Number', text='Phone No', anchor=CENTER)
    tree.heading('Gender', text='Gender', anchor=CENTER)
    tree.heading('Date of Birth', text='dob', anchor=CENTER)
```

```
tree.heading('Department', text='Department', anchor=CENTER)

tree.heading('Fees', text='total', anchor=CENTER)


tree.column('#0', width=0, stretch=NO)

tree.column('#1', width=40, stretch=NO)

tree.column('#2', width=140, stretch=NO)

tree.column('#3', width=80, stretch=NO)

tree.column('#4', width=80, stretch=NO)

tree.column('#5', width=80, stretch=NO)

tree.column('#6', width=110, stretch=NO)

tree.column('#7', width=50, stretch=NO)


tree.place(y=30, relwidth=1, relheight=0.9, relx=0)


display_records()


main.update()

main.mainloop()


def add_fee_record():

    global namebox, amount, db,rows,tfees_strvar

    print(tfees_strvar.get())

    pay_date = db.get_date()
```

```
amt = amount.get()

std_id= rows[namebox.current()][0]

if not pay_date  or not amt :

    mb.showerror('Error!', "Please fill all the missing fields!!")

else:

        try:


            connector.execute('INSERT INTO FEES (STUDENT_ID,
BILL_DATE, AMOUNT) VALUES (?,?,?)', (std_id,pay_date, amt ))

            connector.commit()

            mb.showinfo('Record added', f"Fee Payment of {namebox.get()}
was successfully added")

            reset_fee_fields()

            display_fee_records()

        except Exception as e:

            print(e)

            mb.showerror('Wrong type', 'The type of the values entered is not
accurate. Pls note that the contact field can only contain numbers')
```

```
def display_fee_records():

    global ftree

    ftree.delete(*ftree.get_children())
```

```
    curr = connector.execute('Select bill_no,name,department, totalfees,
bill_date,amount from management a, fees b'+
                             ' where a.student_id=b.student_id')
    data = curr.fetchall()
    for records in data:
        ftree.insert('', END, values=records)


def remove_fee_record():
    global ftree
    if not ftree.selection():
        mb.showerror('Error!', 'Please select an item from the database')
    else:
        current_item = ftree.focus()
        values = ftree.item(current_item)
        selection = values["values"]

        ftree.delete(current_item)

        connector.execute('DELETE FROM Fees WHERE BILL_NO=%d' %
selection[0])
        connector.commit()

        mb.showinfo('Done', 'The record you wanted deleted was
successfully deleted.')
```

```
        display_fee_records()


def view_fee_record():

    global  db, dept_strvar, tfees_strvar,amount, namebox

    current_item = ftree.focus()

    values = ftree.item(current_item)

    selection = values["values"]


    if len(selection)==0:

        mb.showinfo('Done', 'Select any Student to view their details')

        return

    print(selection)

    date = datetime.date(int(selection[4][:4]), int(selection[4][5:7]),
int(selection[4][8:]))


    namebox.set(selection[1])

    db.set_date(date)

    dept_strvar.set(selection[2])

    tfees_strvar.set(selection[3])

    amount.set(selection[5])
```

```
    print('Select sum(amount) from fees  where student_id=(select
student_id from fees where bill_no=' + str(selection[0]) +")")

    curr = connector.execute('Select sum(amount) from fees  where
student_id=(select student_id from fees where bill_no=' + str(selection[0])
+")")

    data = curr.fetchone()
    balance_strvar.set(selection[3]-data[0])


def get_student(evt):
    global namebox, rows
    global dept_strvar,tfees_strvar,balance_strvar
    print(namebox.current())
    row=rows[namebox.current()]
    dept_strvar.set(row[5])
    tfees_strvar.set(row[6])
    print(row)
    curr = connector.execute('Select sum(amount) from fees  where
student_id=' + str(row[0]))
    data = curr.fetchone()
    if data[0]==None:
        balance_strvar.set(row[6])
    else:
        balance_strvar.set(int(row[6])-data[0])
def feewindow():
```

```
    global main, namebox, dept_strvar,tfees_strvar,amount, db,
ftree,balance_strvar,fee, currentwindow

    main.destroy()

    currentwindow='fee'

    fee = Tk()

    fee.title('Student Management System')

    fee.geometry('1080x600+{}+{}'.format(xpos, ypos))

    fee.resizable(0, 0)

    fee.configure(background="Blue")

    lf_bg = 'skyblue' # bg color for the left_frame

    cf_bg = 'darkblue' # bg color for the center_frame

    dept_strvar = StringVar()

    tfees_strvar= StringVar()

    amount=StringVar()

    tfees_strvar=StringVar()

    balance_strvar=StringVar()

    Label(fee, text="STUDENT MANAGEMENT SYSTEM",
font=headlabelfont, bg='Blue').pack(side=TOP, fill=X)

    left_frame = Frame(fee, bg=lf_bg)

    left_frame.place(x=0, y=30, relheight=1, relwidth=0.2)
```

```
center_frame = Frame(fee, bg=cf_bg)

center_frame.place(relx=0.2, y=30, relheight=1, relwidth=0.2)


right_frame = Frame(fee, bg="Gray35")

right_frame.place(relx=0.4, y=30, relheight=1, relwidth=0.57)

   Label(left_frame, text="Name", font=labelfont,
bg=lf_bg).place(relx=0.375, rely=0.05)

   Label(left_frame, text="Department", font=labelfont,
bg=lf_bg).place(relx=0.24, rely=0.18)

   Label(left_frame, text="Total Fees", font=labelfont,
bg=lf_bg).place(relx=0.28, rely=0.31)

   Label(left_frame, text="Balance Fees", font=labelfont,
bg=lf_bg).place(relx=0.24, rely=0.43)

   Label(left_frame, text="Payment Date", font=labelfont,
bg=lf_bg).place(relx=0.19, rely=0.55)

   Label(left_frame, text="Amount", font=labelfont,
bg=lf_bg).place(relx=0.29, rely=0.66)


   namebox=ttk.Combobox(left_frame, width=19, font=entryfont)

   namebox.place(x=16, rely=0.1)

   namebox.bind("<<ComboboxSelected>>", get_student)

   load_student()

   Label(left_frame, width=19, textvariable=dept_strvar,
font=entryfont).place(x=20, rely=0.23)
```

```
    Label(left_frame,          width=19,          textvariable=tfees_strvar,
font=entryfont).place(x=20, rely=0.36)

    Label(left_frame,          width=19,          textvariable=balance_strvar,
font=entryfont).place(x=20, rely=0.49)

    Entry(left_frame,          width=19,          textvariable=amount,
font=entryfont).place(x=20, rely=0.71)
```

```
    db = DateEntry(left_frame, font=("Arial", 12), width=10)

    db.place(x=40, rely=0.60)
```

```
    Button(center_frame,       text='Add       Record',       font=labelfont,
command=add_fee_record, width=15).place(relx=0.1, rely=0.20)

    Button(center_frame,       text='Delete      Record',      font=labelfont,
command=remove_fee_record, width=15).place(relx=0.1, rely=0.30)

    Button(center_frame,       text='View       Record',       font=labelfont,
command=view_fee_record, width=15).place(relx=0.1, rely=0.40)

    Button(center_frame,       text='Reset       Fields',       font=labelfont,
command=reset_fee_fields, width=15).place(relx=0.1, rely=0.50)

    Button(center_frame,       text='Student       Details',       font=labelfont,
command=mainwindow, width=15).place(relx=0.1, rely=0.70)

    Label(right_frame, text='Fee Payment Details', font=headlabelfont,
bg='DarkBlue', fg='LightCyan').pack(side=TOP, fill=X)
```

```
ftree = ttk.Treeview(right_frame, height=100, selectmode=BROWSE,

                columns=('Bill                No.',                "Name",
"Department","Fees","Payment Date","Amount"))
```

```
    X_scroller = Scrollbar(ftree, orient=HORIZONTAL,
command=ftree.xview)

    Y_scroller = Scrollbar(ftree, orient=VERTICAL, command=ftree.yview)

    X_scroller.pack(side=BOTTOM, fill=X)

    Y_scroller.pack(side=RIGHT, fill=Y)


    ftree.config(yscrollcommand=Y_scroller.set,
xscrollcommand=X_scroller.set)


    ftree.heading('Bill No.', text='Bill No.', anchor=CENTER)

    ftree.heading('Name', text='Name', anchor=CENTER)

    ftree.heading('Department', text='Department', anchor=CENTER)

    ftree.heading('Fees', text='total', anchor=CENTER)

    ftree.heading('Payment Date', text='Payment Date', anchor=CENTER)

    ftree.heading('Amount', text='Amount', anchor=CENTER)


    ftree.column('#0', width=0, stretch=NO)

    ftree.column('#1', width=60, stretch=NO)

    ftree.column('#2', width=140, stretch=NO)

    ftree.column('#3', width=80, stretch=NO)

    ftree.column('#4', width=100, stretch=NO)
```

```
ftree.column('#5', width=110, stretch=NO)

ftree.column('#6', width=80, stretch=NO)



ftree.place(y=30, relwidth=1, relheight=0.9, relx=0)

    display_fee_records()



    fee.update()

    fee.mainloop()



mainwindow()
```