

**Project Report**

**On**

**Student**

**Management System**

# **INDEX**

<b>No.</b>	<b>Title</b>	<b>Remark</b>
<b>1</b>	<b>INTRODUCTION OF PROJECT</b> 1.1 Objective 1.2 Introduction 1.3 Scope	
<b>2</b>	<b>HARDWARE &amp; SOFTWARE REQUIREMENT</b> 2.1 Hardware Requirement 2.2 Software requirement	
<b>3</b>	<b>TECHNICAL DESCRIPTION</b> 3.1 Front End Description	
<b>4</b>	<b>SOFTWARE ANALYSIS AND DESIGN</b> 4.1 Software Analysis 4.2 Software development Life Cycle 4.3 Description of Used model 4.4 Software Design	
<b>5</b>	<b>DEVELOPMENT OF PROJECT</b> 5.1 Source Code 5.2 Snapshots of Output	
<b>6</b>	<b>DIAGRAMS</b> 6.1 Data Flow Diagram 6.2 ER Diagram 6.3 Use Case Diagram 6.4 Flow Chart	
<b>7</b>	<b>TESTING</b>	
<b>8</b>	<b>BENEFITS</b>	
<b>9</b>	<b>LIMITATIONS</b>	
<b>10</b>	<b>FUTURE ENHANCEMENT</b>	
<b>11</b>	<b>CONCLUSION</b>	
<b>12</b>	<b>BIBLIOGRAPHY &amp; REFERENCES</b>	

## **ABSTRACT**

Our project student Management System includes registration of students , storing their details into the system, I,e, computerized the whole process. Our software has the facility to give a unique id for every student and stores the details of every student. It includes a search facility . it also search by name , contact and roll number. The data can be retrieved easily. The Interface is very UserFriendly. The data are well protected for the personal use and makes the data processing very first.

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1: OBJECTIVE**

Student Management System is a software which is useful for college as well as the school authorities . In the current system all the activities are done manually .It is very time consuming and costly.Our Student Management System deals with various activities related to managing student records. Our objective is computerizing the process of student records management.

### **1.2 INTRODUCTION**

This project is to automate the Student Management System .This project is developed mainly to administrate the student records.The purpose of project entitled as STUDENT MANAGEMENT SYSTEM is to computerize the Front Office Management of Student records in colleges, schools and coaching center's to develop software which is user friendly , simple, fast and cost- effective. Traditionally it done manually.

The main function of the system is to register and store student details, retrieve these details as and when required and also to manipulate these details meaningfully.

### **1.3 . SCOPE**

The purposed software product is the Student Management System .The system will be used in any school, College and coaching institute to get the information from the student and then storing the data for future usage.

The current system in use is a paper-based system. It is too slow and cannot provide updated lists of students within a reasonable timeframe. The intentions of the system are to reduce over-time pay and increase the productivity.Requirements statements in this document are both functional and non-functional.

## **CHAPTER 2:**

### **HARDWARE & SOFTWARE REQUIREMENT**

#### **2.1 HARDWARE REQUIREMENT**

- PENTIUM 4 processor or higher
- 128 MB RAM (or above)
- 40 GB or more HARDDISK
- Mouse/Keyboard

#### **2.2 SOFTWARE REQUIREMENT**

- OS-Windows 8/9/10/11
- Python Interpreter
- VS Code
- XAMPP( For Mysql ) or Mysql Workbench

## **CHAPTER 3:**

### **TECHNICAL DESCRIPTION**

#### **3.1: FRONT END DESCRIPTION**

##### **LANGUAGE: Python**

Python is widely general-purpose , high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural , object-oriented, and functional programming. Python is often described as a “batteries included” language due to its comprehensive standard library.

##### **Tkinter**

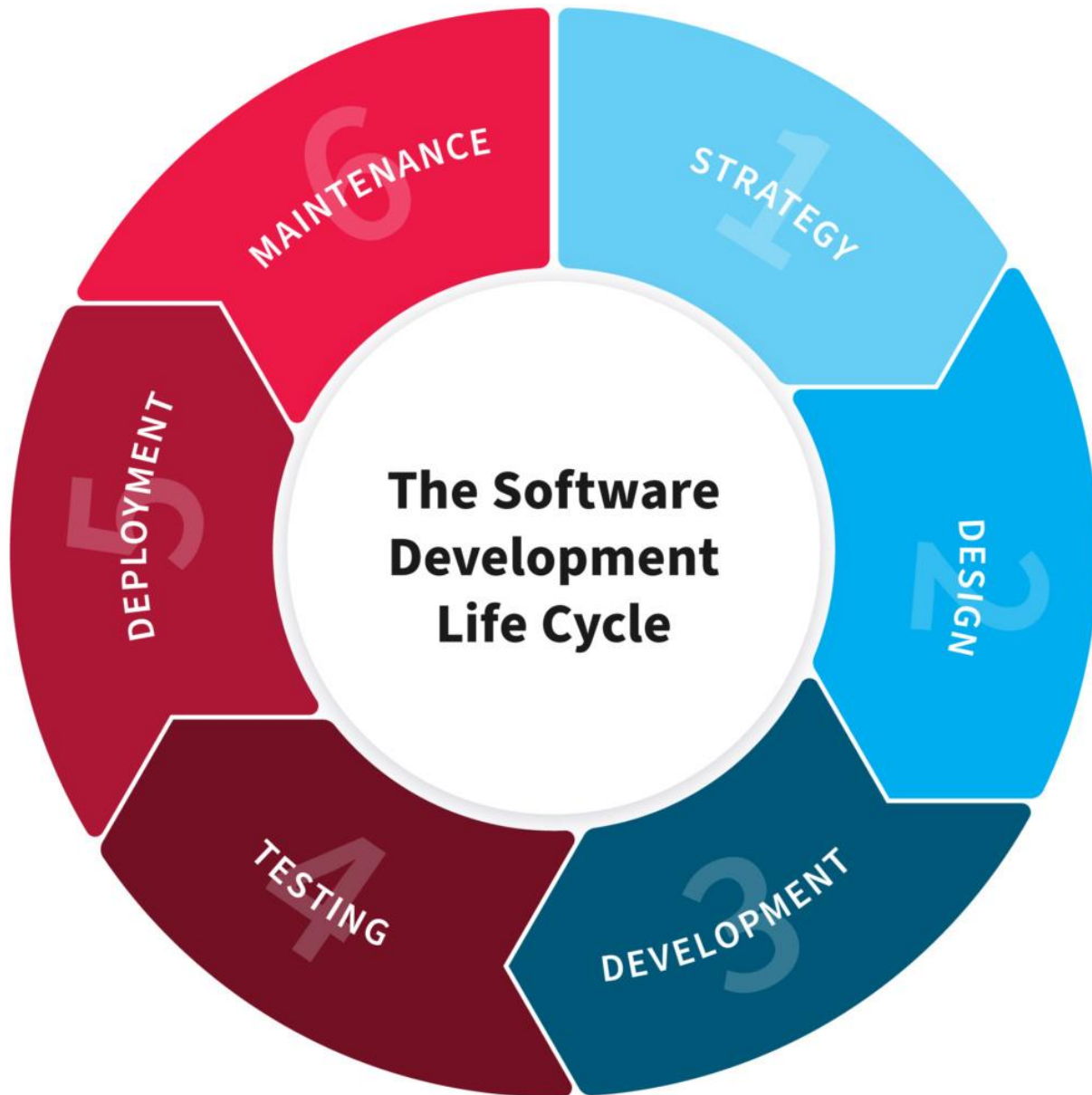
Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python’s de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python.

The name Tkinter comes from Tk interface. Tkinter was written by Fredrik Lundh.

## CHAPTER 4

### SOFTWARE ANALYSIS AND DESIGN

#### 4.1.1. Software Development Life Cycle



#### 4.2.2 Description of Used Model

The Waterfall model is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design (validation), Construction, Testing and Maintenance.

To follow the Waterfall model , one proceeds from one phase to the next in a sequential manner. For example, one first completes requirements specification, which after sign-off are considered “set in stone.” When the requirements are fully completed, one proceeds to design .The software in question is designed and a blueprint is drawn for implementors(coders) to follow—this design should be a plan for implementing the requirements given. When the design is fully completed, an implementation of that design is made by coders. Towards the later stages of this implementation phase, separate software components produced are combined to introduce new functionality and reduced risk through the removal of errors.

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is completed and perfected. However, there are various modified waterfall models (including Royce’s final model) that may include slight or major variations upon this process. Time spent early in the software production cycle can lead to greater economy at later stages. It has been shown that a bug found in the early stages (such as requirements specification or design ) is cheaper in terms of money, effort and time , to fix than the same bug found later on in the process. To take an extreme example, if a program design turns out to be impossible to implement , it is easier to fix the design at the design stage than to realize months later, when program components are being integrated , that all the work done so far has to be scrapped because of a broken design.

This is the central idea behind the waterfall model—time spent early on making sure that requirements and design are absolutely correct will save you much time and effort later . Thus , the thinking of those who follow the waterfall process goes , one should make sure that each phase is 100% complete and absolutely correct before proceeding to the next phase of program creation . Program requirements should be set in stone before design is started (otherwise work put into a design based on incorrect requirements is wasted); the program’s design should be perfect before begin work on implementing the design (otherwise they are implementing the wrong design and their work is wasted ), etc.

A further argument for the waterfall model is that it places emphasis on documentation (such as requirements documents and design documents ) as well as source code . In less designed and documented methodologies , should team members leave , much knowledge is lost and may be difficult for project to recover from. Should a fully working design document be present (as is the intent of Big Design Up Front and the waterfall model) new team members or even entirely new teams should be able to familiarize themselves by reading the documents.

Basic principles of the waterfall model are :

Project is divided into sequential phases , with some overlap and splash back acceptable between phases .

Emphasis is on planning , time schedules , target dates , budgets and implementation of an entire system at one time.

Tight control is maintained over the life of the project through the use of extensive written documentation , as well as through formal reviews and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase.



## CHAPTER 5

### DEVELOPMENT

#### 5.1 Source code

credentials.py

```
# User Credentials
host = 'localhost'
user = 'root'
password = ''
database = 'student_management'
```

custom.py

```
color_1 = "deep sky blue"
color_2 = "gray95"
color_3 = "black"
color_4 = "white"
font_1 = "times new roman"
font_2 = "helvetica"
```

main.py

```
"""student management system project in python using tkinter and mysql"""
from functools import partial
from tkinter import *
from tkinter import messagebox
import pymysql
import custom as cs
import credentials as cr

class Management:
    def __init__(self, root):
        self.window = root
        self.window.title("Student Management System")
        self.window.geometry("780x480")
        self.window.config(bg = "white")

        # Customization
        self.color_1 = cs.color_1
        self.color_2 = cs.color_2
        self.color_3 = cs.color_3
```

```

self.color_4 = cs.color_4
self.font_1 = cs.font_1
self.font_2 = cs.font_2

# User Credentials
self.host = cr.host
self.user = cr.user
self.password = cr.password
self.database = cr.database

# Left Frame
self.frame_1 = Frame(self.window, bg=self.color_1)
self.frame_1.place(x=0, y=0, width=540, relheight = 1)

# Right Frame
self.frame_2 = Frame(self.window, bg = self.color_2)
self.frame_2.place(x=540,y=0,relwidth=1, relheight=1)

# Buttons
self.add_bt = Button(self.frame_2, text='Add New', font=(self.font_1, 12), bd=2,
command=self.AddStudent, cursor="hand2", bg=self.color_2,fg=self.color_3).place(x=68,y=40,width=100)
self.view_bt = Button(self.frame_2, text='View Details', font=(self.font_1, 12), bd=2,
command=self.GetContact_View, cursor="hand2",
bg=self.color_2,fg=self.color_3).place(x=68,y=100,width=100)
self.update_bt = Button(self.frame_2, text='Update', font=(self.font_1, 12), bd=2,
command=self.GetContact_Update, cursor="hand2",
bg=self.color_2,fg=self.color_3).place(x=68,y=160,width=100)
self.delete_bt = Button(self.frame_2, text='Delete', font=(self.font_1, 12), bd=2,
command=self.GetContact_Delete,cursor="hand2",
bg=self.color_2,fg=self.color_3).place(x=68,y=220,width=100)
self.clear_bt = Button(self.frame_2, text='Clear', font=(self.font_1, 12), bd=2, command=self.ClearScreen,
cursor="hand2", bg=self.color_2,fg=self.color_3).place(x=68,y=280,width=100)
self.exit_bt = Button(self.frame_2, text='Exit', font=(self.font_1, 12), bd=2, command=self.Exit,
cursor="hand2", bg=self.color_2,fg=self.color_3).place(x=68,y=340,width=100)

"""Widgets for adding student data"""
def AddStudent(self):
    self.ClearScreen()

    self.name = Label(self.frame_1, text="First Name", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=30)
    self.name_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
    self.name_entry.place(x=40,y=60, width=200)

```

```

        self.surname = Label(self.frame_1, text="Last Name", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=30)
        self.surname_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.surname_entry.place(x=300,y=60, width=200)

        self.course = Label(self.frame_1, text="Course", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=100)
        self.course_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.course_entry.place(x=40,y=130, width=200)

        self.subject = Label(self.frame_1, text="Subject", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=100)
        self.subject_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.subject_entry.place(x=300,y=130, width=200)

        self.year = Label(self.frame_1, text="Year", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=170)
        self.year_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.year_entry.place(x=40,y=200, width=200)

        self.age = Label(self.frame_1, text="Age", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=170)
        self.age_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.age_entry.place(x=300,y=200, width=200)

        self.gender = Label(self.frame_1, text="Gender", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=240)
        self.gender_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.gender_entry.place(x=40,y=270, width=200)

        self.birth = Label(self.frame_1, text="Birthday", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=240)
        self.birth_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.birth_entry.place(x=300,y=270, width=200)

        self.contact = Label(self.frame_1, text="Contact", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=310)
        self.contact_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.contact_entry.place(x=40,y=340, width=200)

        self.email = Label(self.frame_1, text="Email", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=310)
        self.email_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.email_entry.place(x=300,y=340, width=200)

```

```

        self.submit_bt_1 = Button(self.frame_1, text='Submit', font=(self.font_1, 12), bd=2, command=self.Submit,
cursor="hand2", bg=self.color_2, fg=self.color_3).place(x=200, y=389, width=100)

    """Get the contact number to show a student details"""
    def GetContact_View(self):
        self.ClearScreen()

        self.getInfo = Label(self.frame_1, text="Enter Phone Number", font=(self.font_2, 18, "bold"),
bg=self.color_1).place(x=140, y=70)
        self.getInfo_entry = Entry(self.frame_1, font=(self.font_1, 12), bg=self.color_4, fg=self.color_3)
        self.getInfo_entry.place(x=163, y=110, width=200, height=30)
        self.submit_bt_2 = Button(self.frame_1, text='Submit', font=(self.font_1, 10), bd=2,
command=self.CheckContact_View, cursor="hand2",
bg=self.color_2, fg=self.color_3).place(x=220, y=150, width=80)

    """To update a student details, get the contact number"""
    def GetContact_Update(self):
        self.ClearScreen()

        self.getInfo = Label(self.frame_1, text="Enter Phone Number", font=(self.font_2, 18, "bold"),
bg=self.color_1).place(x=140, y=70)
        self.getInfo_entry = Entry(self.frame_1, font=(self.font_1, 12), bg=self.color_4, fg=self.color_3)
        self.getInfo_entry.place(x=163, y=110, width=200, height=30)
        self.submit_bt_2 = Button(self.frame_1, text='Submit', font=(self.font_1, 10), bd=2,
command=self.CheckContact_Update, cursor="hand2",
bg=self.color_2, fg=self.color_3).place(x=220, y=150, width=80)

    """Get the contact number to delete a student record"""
    def GetContact_Delete(self):
        self.ClearScreen()

        self.getInfo = Label(self.frame_1, text="Enter Phone Number", font=(self.font_2, 18, "bold"),
bg=self.color_1).place(x=140, y=70)
        self.getInfo_entry = Entry(self.frame_1, font=(self.font_1, 12), bg=self.color_4, fg=self.color_3)
        self.getInfo_entry.place(x=163, y=110, width=200, height=30)
        self.submit_bt_2 = Button(self.frame_1, text='Submit', font=(self.font_1, 10), bd=2,
command=self.DeleteData, cursor="hand2", bg=self.color_2, fg=self.color_3).place(x=220, y=150, width=80)

    """Remove all widgets from the frame 1"""
    def ClearScreen(self):
        for widget in self.frame_1.winfo_children():
            widget.destroy()

    """Exit window"""

```

```

def Exit(self):
    self.window.destroy()

'''
Checks whether the contact number is available or not. If available,
the function calls the 'ShowDetails' function to display the result.
'''

def CheckContact_View(self):
    if self.getInfo_entry.get() == "":
        messagebox.showerror("Error!", "Please enter your contact number",parent=self.window)
    else:
        try:
            connection = pymysql.connect(host=self.host, user=self.user, password=self.password,
database=self.database)
            curs = connection.cursor()
            curs.execute("select * from student_register where contact=%s", self.getInfo_entry.get())
            row=curs.fetchone()

            if row == None:
                messagebox.showerror("Error!", "Contact number doesn't exists",parent=self.window)
            else:
                self.ShowDetails(row)
                connection.close()
        except Exception as e:
            messagebox.showerror("Error!",f"Error due to {str(e)}",parent=self.window)

'''
Checks whether the contact number is available or not. If available,
the function calls the 'GetUpdateDetails' function to get the new data to perform
update operation.
'''

def CheckContact_Update(self):
    if self.getInfo_entry.get() == "":
        messagebox.showerror("Error!", "Please enter your contact number",parent=self.window)
    else:
        try:
            connection = pymysql.connect(host=self.host, user=self.user, password=self.password,
database=self.database)
            curs = connection.cursor()
            curs.execute("select * from student_register where contact=%s", self.getInfo_entry.get())
            row=curs.fetchone()

            if row == None:
                messagebox.showerror("Error!", "Contact number doesn't exists",parent=self.window)

```

```

        else:
            self.GetUpdateDetails(row)
            connection.close()
    except Exception as e:
        messagebox.showerror("Error!",f"Error due to {str(e)}",parent=self.window)

    """Clears a student record"""
    def DeleteData(self):
        if self.getInfo_entry.get() == "":
            messagebox.showerror("Error!", "Please enter your contact number",parent=self.window)
        else:
            try:
                connection = pymysql.connect(host=self.host, user=self.user, password=self.password,
                database=self.database)
                curs = connection.cursor()
                curs.execute("select * from student_register where contact=%s", self.getInfo_entry.get())
                row=curs.fetchone()

                if row == None:
                    messagebox.showerror("Error!", "Contact number doesn't exists",parent=self.window)
                else:
                    curs.execute("delete from student_register where contact=%s", self.getInfo_entry.get())
                    connection.commit()
                    messagebox.showinfo('Done!', "The data has been deleted")
                    connection.close()
                    self.ClearScreen()
            except Exception as e:
                messagebox.showerror("Error!",f"Error due to {str(e)}",parent=self.window)

    """Gets the data that the user wants to update to perform the update operation"""
    def GetUpdateDetails(self, row):
        self.ClearScreen()

        self.name = Label(self.frame_1, text="First Name", font=(self.font_2, 15, "bold"),
        bg=self.color_1).place(x=40,y=30)
        self.name_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.name_entry.insert(0, row[0])
        self.name_entry.place(x=40,y=60, width=200)

        self.surname = Label(self.frame_1, text="Last Name", font=(self.font_2, 15, "bold"),
        bg=self.color_1).place(x=300,y=30)
        self.surname_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.surname_entry.insert(0, row[1])

```

```

self.surname_entry.place(x=300,y=60, width=200)

        self.course = Label(self.frame_1, text="Course", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=100)
        self.course_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.course_entry.insert(0, row[2])
        self.course_entry.place(x=40,y=130, width=200)

        self.subject = Label(self.frame_1, text="Subject", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=100)
        self.subject_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.subject_entry.insert(0, row[3])
        self.subject_entry.place(x=300,y=130, width=200)

        self.year = Label(self.frame_1, text="Year", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=170)
        self.year_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.year_entry.insert(0, row[4])
        self.year_entry.place(x=40,y=200, width=200)

        self.age = Label(self.frame_1, text="Age", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=170)
        self.age_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.age_entry.insert(0, row[5])
        self.age_entry.place(x=300,y=200, width=200)

        self.gender = Label(self.frame_1, text="Gender", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=240)
        self.gender_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.gender_entry.insert(0, row[6])
        self.gender_entry.place(x=40,y=270, width=200)

        self.birth = Label(self.frame_1, text="Birthday", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=240)
        self.birth_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)
        self.birth_entry.insert(0, row[7])
        self.birth_entry.place(x=300,y=270, width=200)

        contact = Label(self.frame_1, text="Contact", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=310)
        contact_data = Label(self.frame_1, text=row[8], font=(self.font_1, 10)).place(x=40, y=340)

        self.email = Label(self.frame_1, text="Email", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=310)
        self.email_entry = Entry(self.frame_1, bg=self.color_4, fg=self.color_3)

```

```

self.email_entry.insert(0, row[9])
self.email_entry.place(x=300,y=340, width=200)

self.submit_bt_1 = Button(self.frame_1, text='Submit', font=(self.font_1, 12), bd=2,
command=partial(self.UpdateDetails, row), cursor="hand2",
bg=self.color_2,fg=self.color_3).place(x=160,y=389,width=100)
self.cancel_bt = Button(self.frame_1, text='Cancel', font=(self.font_1, 12), bd=2,
command=self.ClearScreen, cursor="hand2", bg=self.color_2,fg=self.color_3).place(x=280,y=389,width=100)

"""Within frame 1, it displays information about a student"""
def ShowDetails(self, row):
    self.ClearScreen()
    name = Label(self.frame_1, text="First Name", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=30)
    name_data = Label(self.frame_1, text=row[0], font=(self.font_1, 10)).place(x=40, y=60)

    surname = Label(self.frame_1, text="Last Name", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=30)
    surname_data = Label(self.frame_1, text=row[1], font=(self.font_1, 10)).place(x=300, y=60)

    course = Label(self.frame_1, text="Course", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=100)
    course_data = Label(self.frame_1, text=row[2], font=(self.font_1, 10)).place(x=40, y=130)

    subject = Label(self.frame_1, text="Subject", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=100)
    subject_data = Label(self.frame_1, text=row[3], font=(self.font_1, 10)).place(x=300, y=130)

    year = Label(self.frame_1, text="Year", font=(self.font_2, 15, "bold"), bg=self.color_1).place(x=40,y=170)
    year_data = Label(self.frame_1, text=row[4], font=(self.font_1, 10)).place(x=40, y=200)

    age = Label(self.frame_1, text="Age", font=(self.font_2, 15, "bold"), bg=self.color_1).place(x=300,y=170)
    age_data = Label(self.frame_1, text=row[5], font=(self.font_1, 10)).place(x=300, y=200)

    gender = Label(self.frame_1, text="Gender", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=240)
    gender_data = Label(self.frame_1, text=row[6], font=(self.font_1, 10)).place(x=40, y=270)

    birth = Label(self.frame_1, text="Birthday", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=240)
    birth_data = Label(self.frame_1, text=row[7], font=(self.font_1, 10)).place(x=300, y=270)

    contact = Label(self.frame_1, text="Contact", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=40,y=310)

```



```

contact_data = Label(self.frame_1, text=row[8], font=(self.font_1, 10)).place(x=40, y=340)

        email    = Label(self.frame_1, text="Email", font=(self.font_2, 15, "bold"),
bg=self.color_1).place(x=300,y=310)
        email_data = Label(self.frame_1, text=row[9], font=(self.font_1, 10)).place(x=300, y=340)


    """Updates student data"""
    def UpdateDetails(self, row):
        if self.name_entry.get() == "" or self.surname_entry.get() == "" or self.course_entry.get() == "" or
self.subject_entry.get() == "" or self.year_entry.get() == "" or self.age_entry.get() == "" or
self.gender_entry.get() == "" or self.birth_entry.get() == "" or self.email_entry.get() == "":
            messagebox.showerror("Error!", "Sorry!, All fields are required", parent=self.window)
        else:
            try:
                connection = pymysql.connect(host=self.host, user=self.user, password=self.password,
database=self.database)
                curs = connection.cursor()
                curs.execute("select * from student_register where contact=%s", row[8])
                row=curs.fetchone()

                if row==None:
                    messagebox.showerror("Error!", "The contact number doesn't exists", parent=self.window)
                else:
                    curs.execute("update student_register set f_name=%s,l_name=%s, course=%s, subject=%s,
year=%s, age=%s, gender=%s, birth=%s, email=%s where contact=%s",
                                (
                                    self.name_entry.get(),
                                    self.surname_entry.get(),
                                    self.course_entry.get(),
                                    self.subject_entry.get(),
                                    self.year_entry.get(),
                                    self.age_entry.get(),
                                    self.gender_entry.get(),
                                    self.birth_entry.get(),
                                    self.email_entry.get(),
                                    row[8]
                                ))
                    connection.commit()
                    connection.close()
                    messagebox.showinfo('Done!', "The data has been updated")
                    self.ClearScreen()
            except Exception as e:
                messagebox.showerror("Error!", f"Error due to {str(e)}", parent=self.window)

```

```

"""It adds the information of new students"""
def Submit(self):
    if self.name_entry.get() == "" or self.surname_entry.get() == "" or self.course_entry.get() == "" or
self.subject_entry.get() == "" or self.year_entry.get() == "" or self.age_entry.get() == "" or
self.gender_entry.get() == "" or self.birth_entry.get() == "" or self.contact_entry.get() == "" or
self.email_entry.get() == "":
        messagebox.showerror("Error!", "Sorry!, All fields are required", parent=self.window)
    else:
        try:
            connection = pymysql.connect(host=self.host, user=self.user, password=self.password,
database=self.database)
            curs = connection.cursor()
            curs.execute("select * from student_register where contact=%s", self.contact_entry.get())
            row=curs.fetchone()

            if row!=None:
                messagebox.showerror("Error!", "The contact number is already exists, please try again with another
number", parent=self.window)
            else:
                curs.execute("insert into student_register
(f_name,l_name,course,subject,year,age,gender,birth,contact,email)
values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",
                (
                    self.name_entry.get(),
                    self.surname_entry.get(),
                    self.course_entry.get(),
                    self.subject_entry.get(),
                    self.year_entry.get(),
                    self.age_entry.get(),
                    self.gender_entry.get(),
                    self.birth_entry.get(),
                    self.contact_entry.get(),
                    self.email_entry.get()
                ))
            connection.commit()
            connection.close()
            messagebox.showinfo("Done!", "The data has been submitted")
            self.reset_fields()
        except Exception as e:
            messagebox.showerror("Error!", f"Error due to {str(e)}", parent=self.window)

"""Reset all the entry fields"""
def reset_fields(self):
    self.name_entry.delete(0, END)

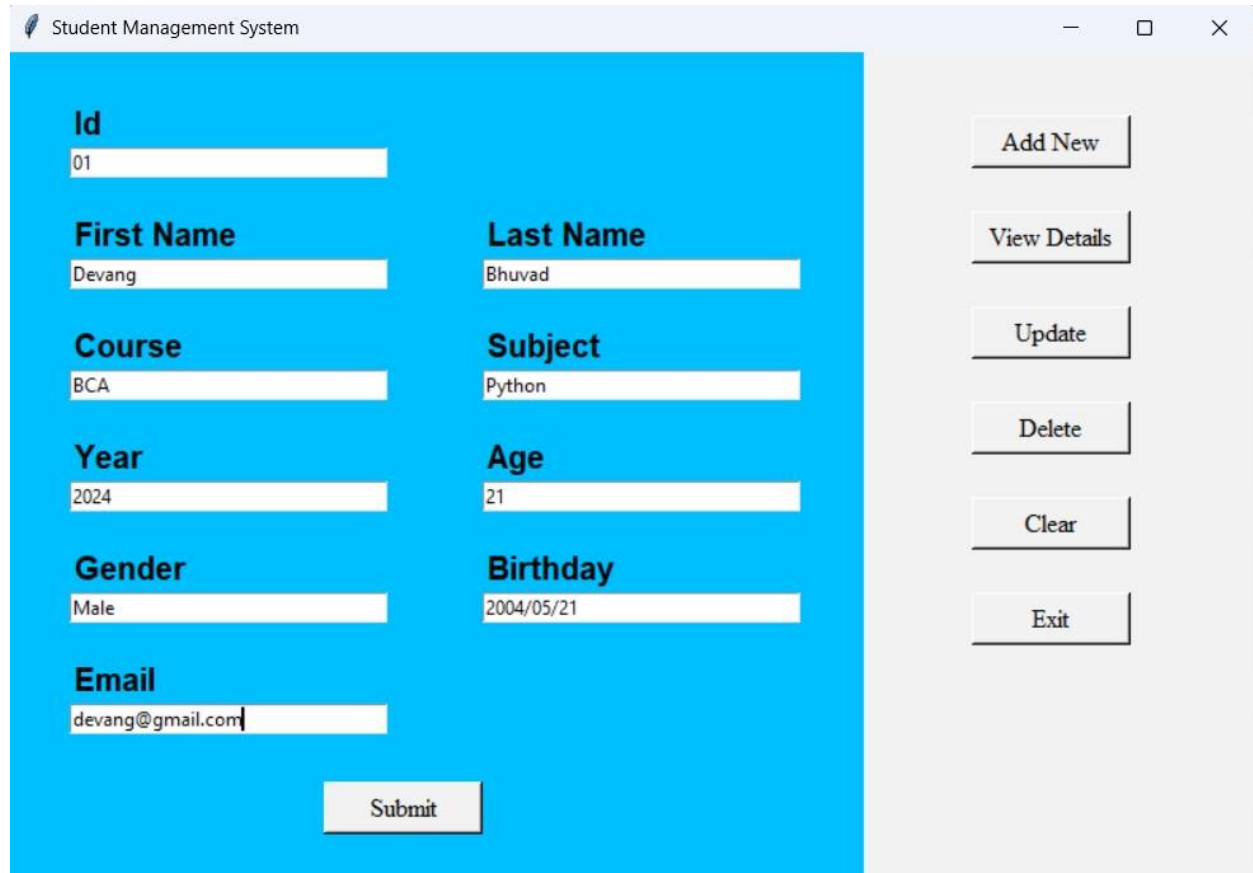
```

```
self.surname_entry.delete(0, END)
self.course_entry.delete(0, END)
self.subject_entry.delete(0, END)
self.year_entry.delete(0, END)
self.age_entry.delete(0, END)
self.gender_entry.delete(0, END)
self.birth_entry.delete(0, END)
self.contact_entry.delete(0, END)
self.email_entry.delete(0, END)

# The main function
if __name__ == "__main__":
    root = Tk()
    obj = Management(root)
    root.mainloop()
```

## 5.2 Snapshots

- **Add New Details:**

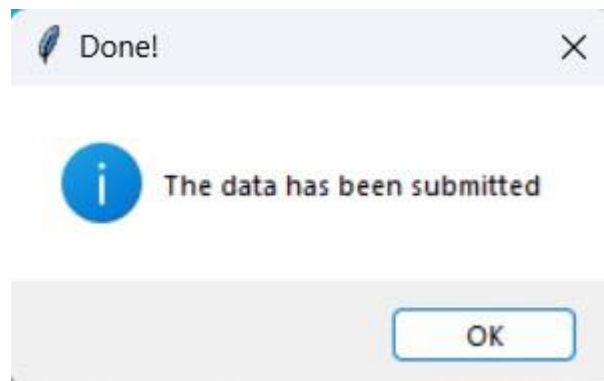


The screenshot shows a web application window titled "Student Management System". The main content area has a blue background and contains a form with the following fields:

- Id**: Input field with "01".
- First Name**: Input field with "Devang".
- Last Name**: Input field with "Bhuvad".
- Course**: Input field with "BCA".
- Subject**: Input field with "Python".
- Year**: Input field with "2024".
- Age**: Input field with "21".
- Gender**: Input field with "Male".
- Birthday**: Input field with "2004/05/21".
- Email**: Input field with "devang@gmail.com".

A "Submit" button is located at the bottom center of the form. To the right of the form, there is a vertical sidebar with the following buttons: "Add New", "View Details", "Update", "Delete", "Clear", and "Exit".

- **After Submitting Details :**



- **View Details:**

Student Management System

**Enter ID**

01

Submit

Add New

View Details

Update

Delete

Clear

Exit

Student Management System

<b>First Name</b>	<b>Last Name</b>
Devang	Bhuvad
<b>Course</b>	<b>Subject</b>
BCA	Python
<b>Year</b>	<b>Age</b>
2024	21
<b>Gender</b>	<b>Birthday</b>
Male	2004-05-21
<b>ID</b>	<b>Email</b>
01	devang@gmail.com

Add New

View Details

Update

Delete

Clear

Exit

- **Update Details**

Student Management System

Enter ID

01

Submit

Add New

View Details

Update

Delete

Clear

Exit

Student Management System

First Name

Devang

Last Name

Bhuvad

Course

BCA

Subject

Python

Year

2024

Age

21

Gender

Male

Birthday

2004-05-21

ID

01

Email

devang@gmail.com

Submit

Cancel

Add New

View Details

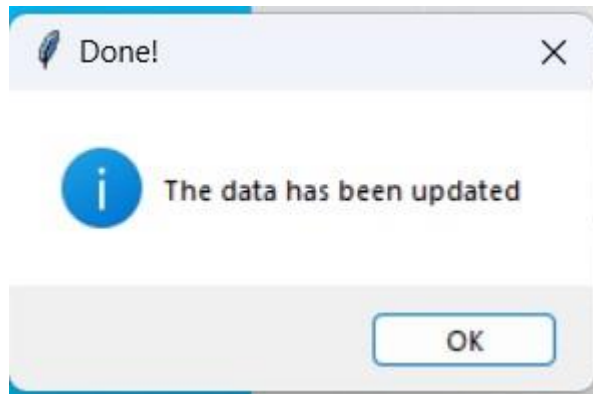
Update

Delete

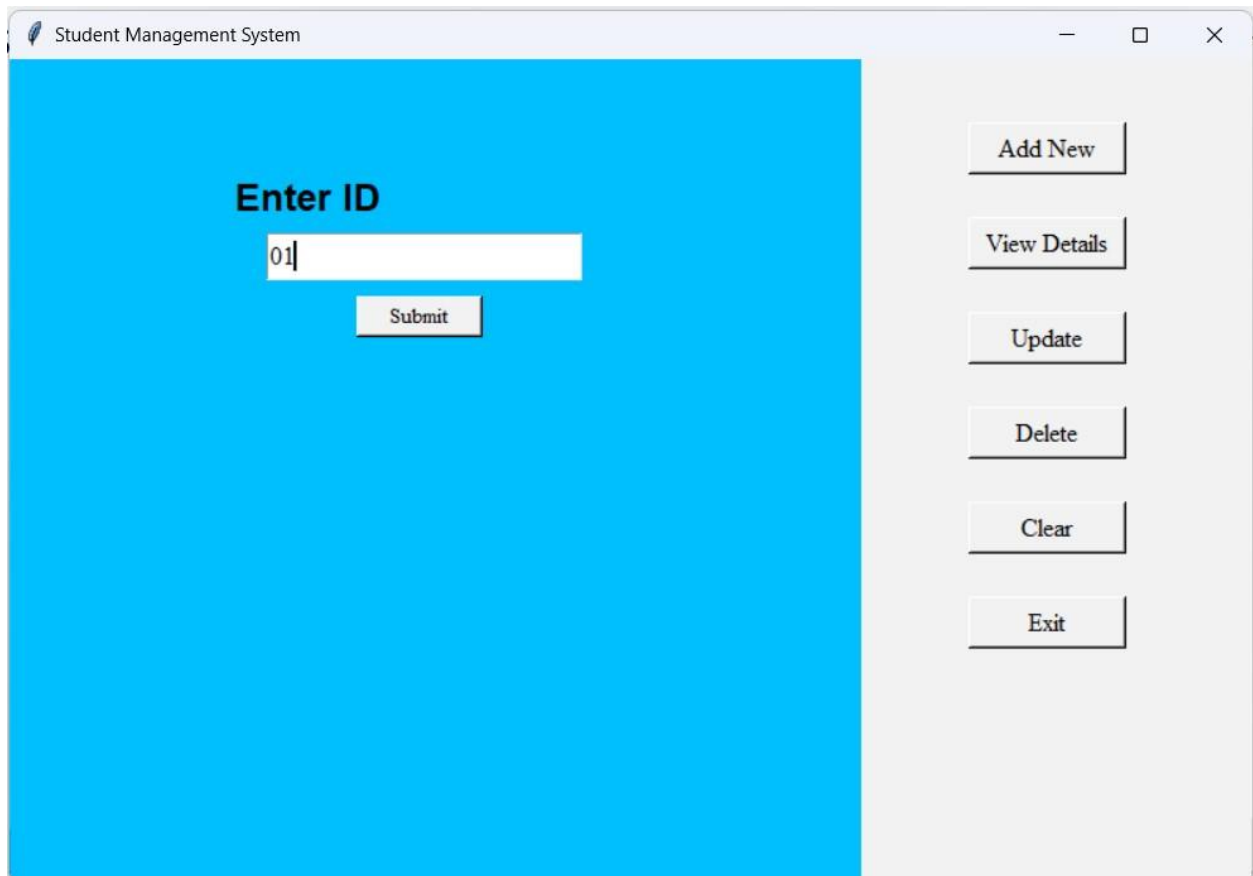
Clear

Exit

### After Update Details:

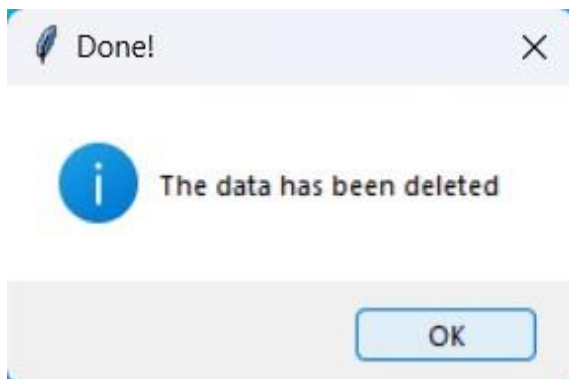


- **Delete Details:**



The screenshot shows a window titled "Student Management System". The main area has a blue background with the text "Enter ID" in bold. Below this is a text input field containing "01" and a "Submit" button. To the right of the main area is a vertical sidebar with six buttons: "Add New", "View Details", "Update", "Delete", "Clear", and "Exit". The "Delete" button is highlighted, indicating it is the active function.

- **After Delete Details:**

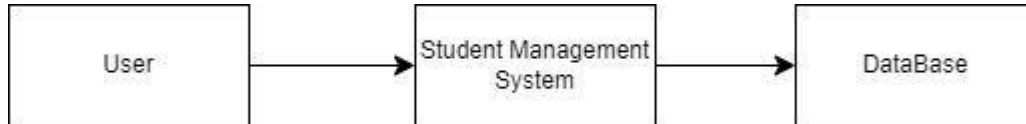




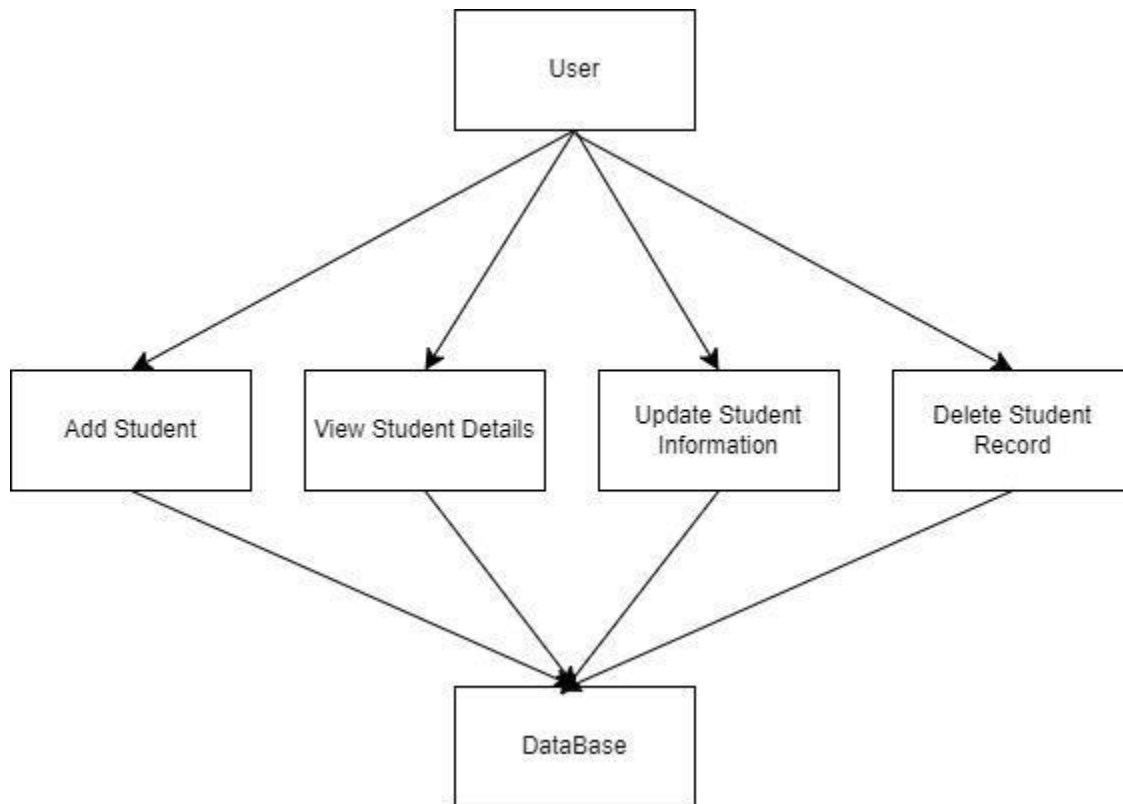
## CHAPTER 6 DIAGRAMS

### 6.1 Data Flow Diagram:-

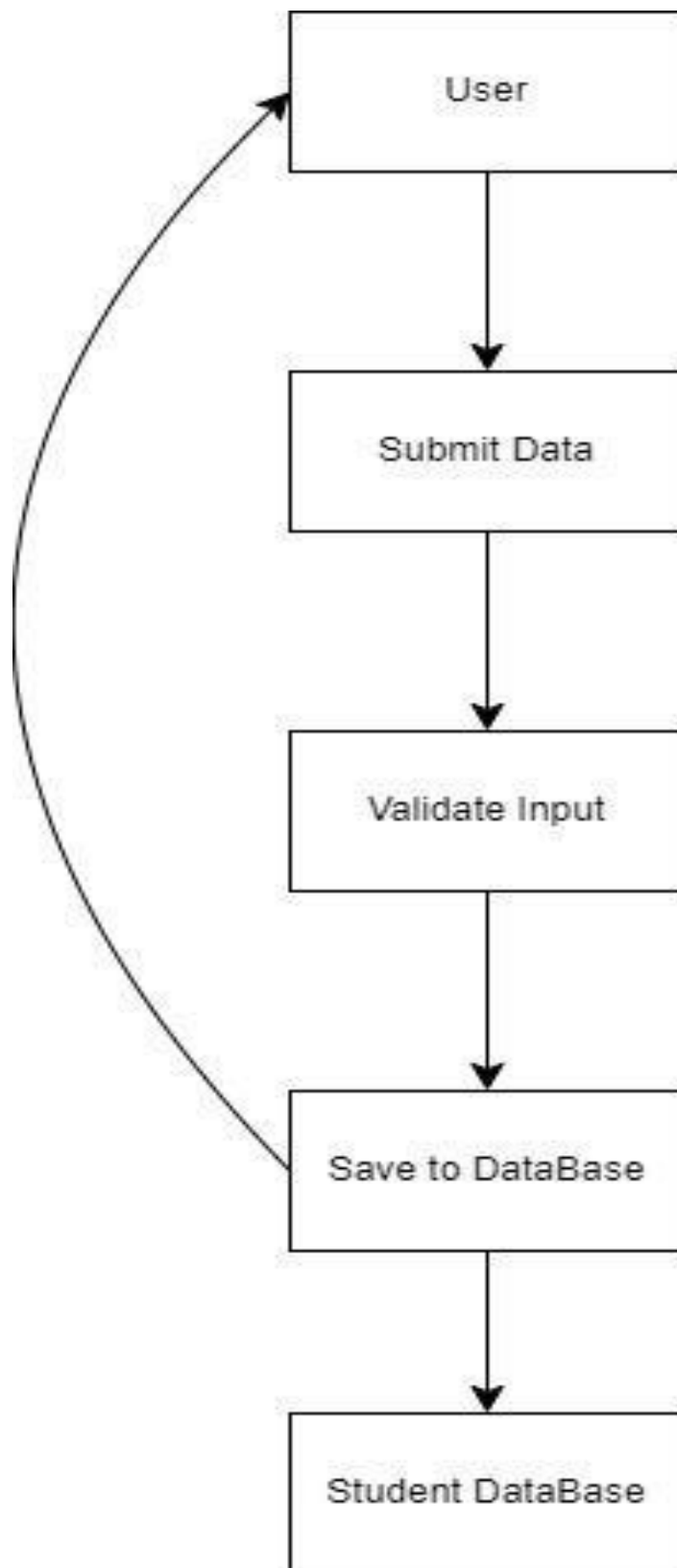
6.1.1 DFD 0



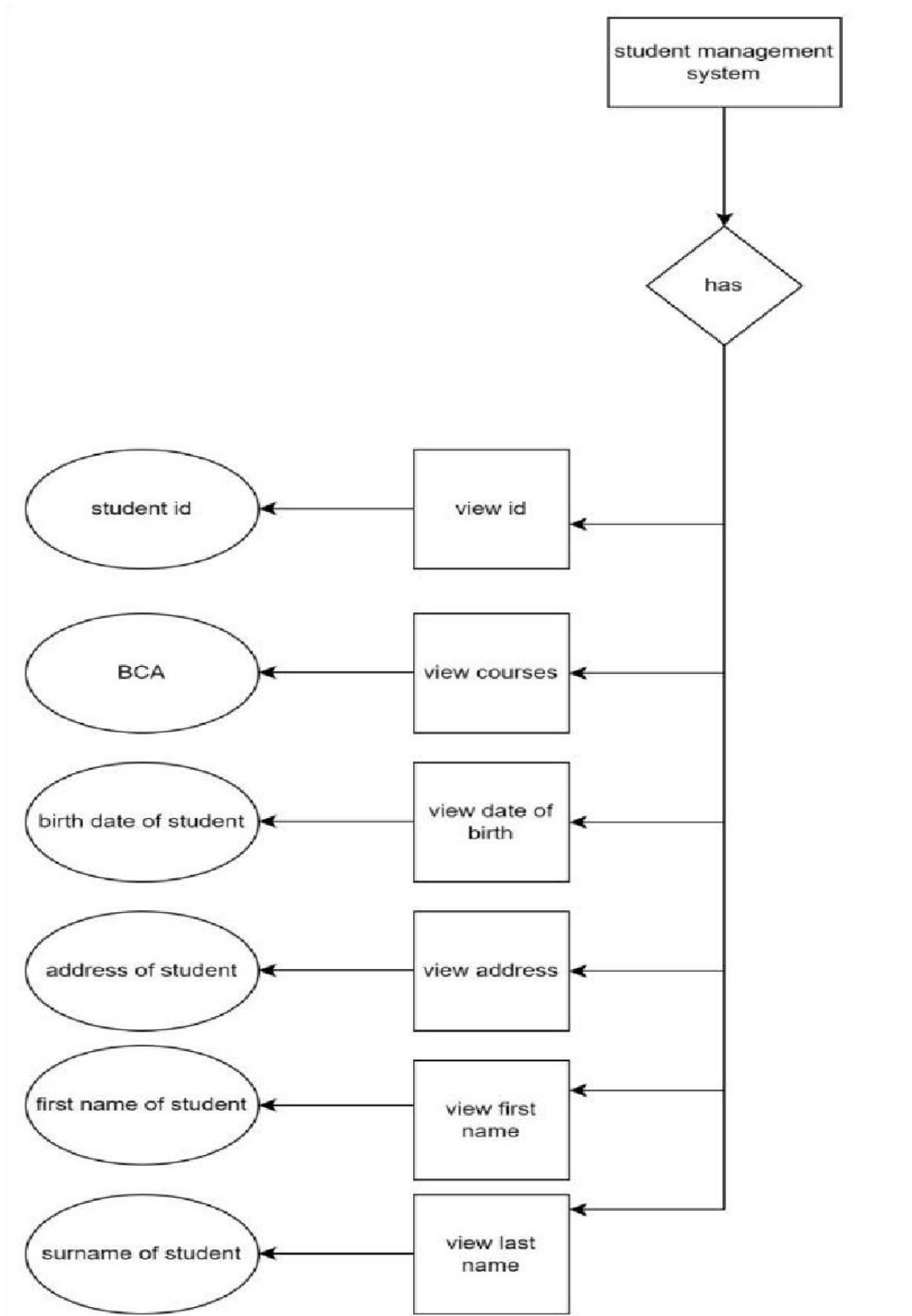
6.1.2 DFD 1



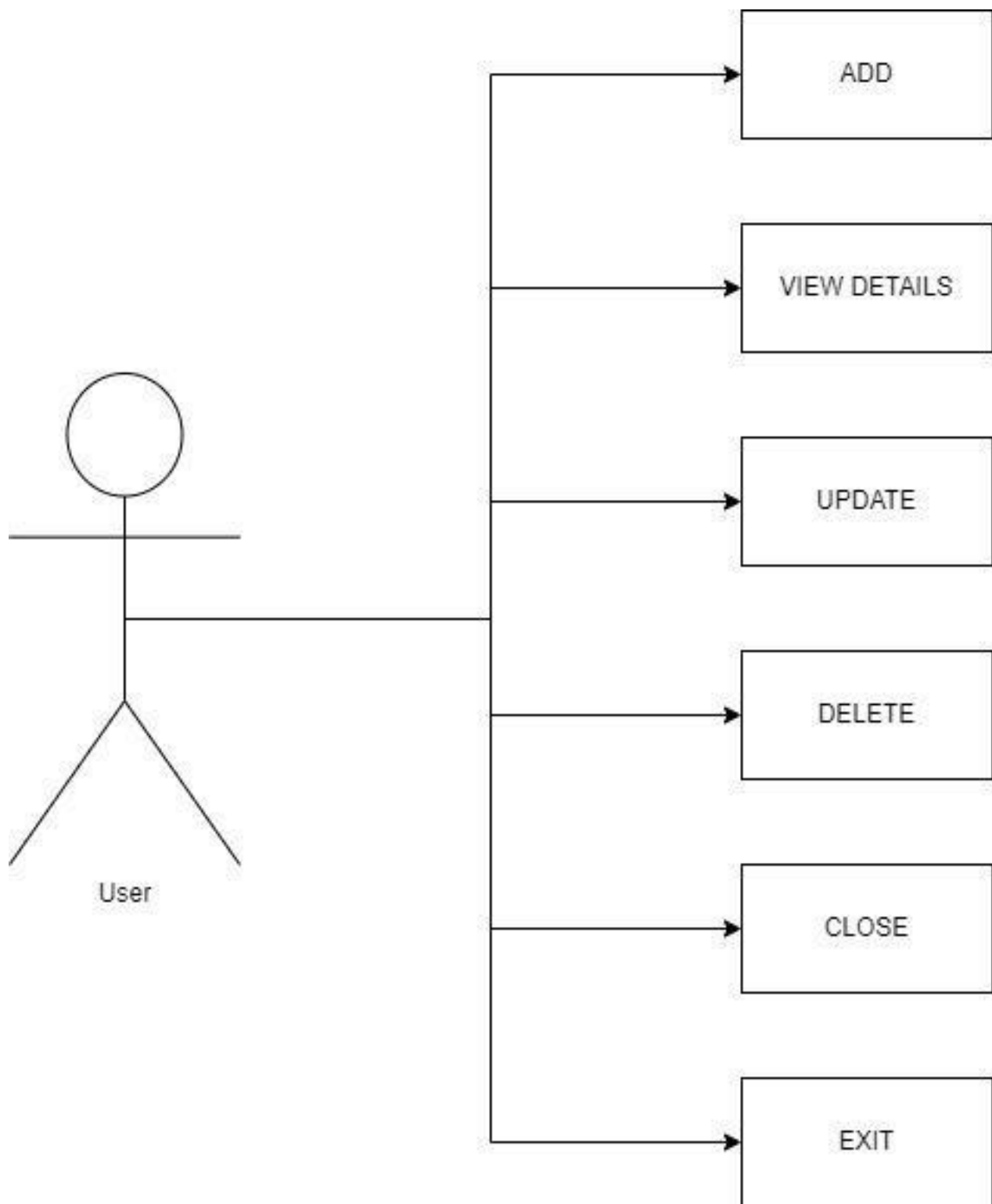
### 6.1.3 DFD 2



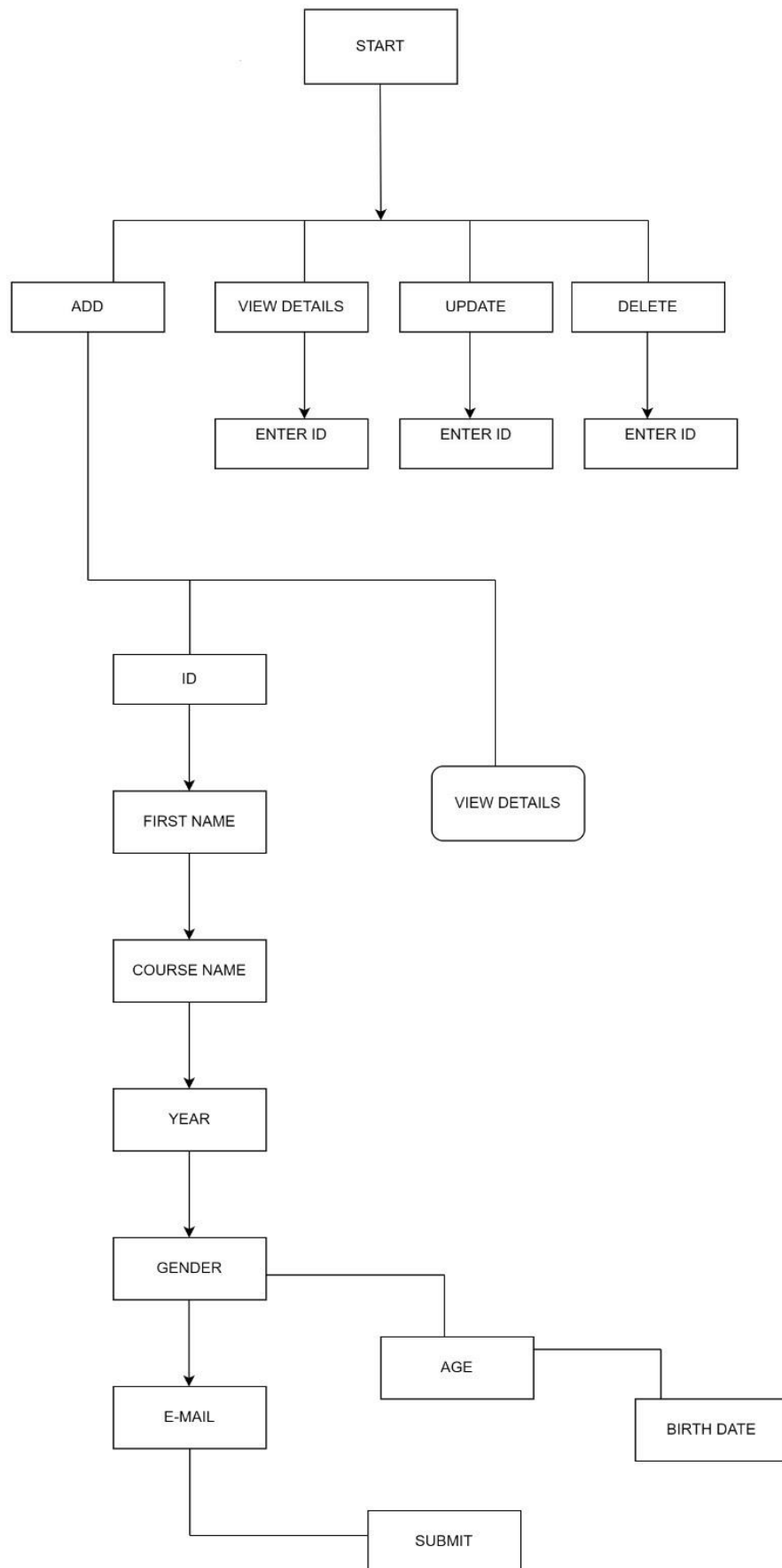
## 6.2 ER Diagram:-



### 6.3 Use Diagram:-



### 6.3 Flow Chart:-



## CHAPTER 7

### TESTING

Testing is more than just debugging. The purpose of testing can be quality assurance, verification and validation or reliability estimation. Correctness testing and reliability testing are two major areas of testing. Software testing is a trade -off between budget, time and quality.

#### SOFTWARE TESTING

Software testing is the process of excuting a program or system with the intent of finding errors.Or it involves any activity aimed at evaluating an attribute or capable or capability of a program or system and determining that it meets its required results.Software is not unlike other physical processes where inputs are received and outputs are produced.Where software differs is in the manner in which it fails .Unlike most physical systems, most of the defects in software are design errors , not manipulating defects.

#### To improve quality

As computer and software are used in critical applications, the outcome of a bug can be severe. Bugs can cause huge losses.

#### For Verification and Validation(V & V)

Another important purpose of testing is verification and validation(V & V).It is heavily used as a tool in the V &V process. Testers can make claims based on interpretations of the testing results, which either the product works under certain situations, or it does not work

#### Software Testing Types:

##### **1. Black-Box testing**

The Black-box approach is a testing method in which test data are derived from the specified functional requirements without regard to the final program structure.It is also termed data-driven, input/output driven or requirements-based testing. A testing method emphasized on executing the functions and examination of their input and output data.

##### **2. White-box testing**

Contrary to black-box testing , software is viewed as a white-box , or glas-box in white-box testing, as the structure and flow of the software under test are visible to the tester. This testing is based on knowledge of the internal logic of an application's code. Testing plans are made according to the details of the software implementation, such as programming language, logic and styles. Test cases are derived from the program structure . White-box testing is also called glass-box testing, logic-driven testing or design -based testing.

##### **3. Unit testing**

This involves testing of individual software components or modules.Typically done by the programmer and not by the testers, as it requires the detailed knowledge of the internal program design and code.

#### **4. System testing**

Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.

#### **5. End-to-end testing**

Similar to system testing , involves testing of a complete environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications , or systems if appropriate.

#### **6. Usability testing**

User-friendliness check application flow is tested, Can new user understand the application easily ,Proper help documented whenever user stuck at any point. Basically system navigation is checked in this testing.

#### **7. Install/uninstall testing**

Testing for full, partial, or upgrade install/uninstall processes on different operating systems under different hardware, software environment.

#### **8. Recovery testing**

Testing how well a system recovers from crashes , hardware failures or other catastrophic problems.

#### **9. Security testing**

Can system be penetrated by any hacking way. Testing how well the system protects against unauthorized internal or external access Checked if system, database is safe from external attack.

#### **10. Compatibility testing**

Testing how well software performs in a particular hardware/software/operating system/network environment and different combination of above.

#### **11. Comparison Testing**

Comparison of the product strengths and weaknesses with previous versions or other similar products.

#### **12. Alpha testing**

In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

#### **13. Beta testing**

Testing typically done by end-users or others. Final testing before releasing application for commercial purpose.

## **CHAPTER 8**

### **BENEFITS**

- ✓ Software provides easy management of student records.
- ✓ Software has very user friendly interface which is very easy to handle and understand.
- ✓ Software provides security to private data by hiding them.
- ✓ Software uses very less memory and takes less time to startup.



## **CHAPTER 9**

### **LIMITATIONS**

- ✓ Software is limited to desktop only.
- ✓ System requires python interpreter installed on the system.
- ✓ All options of student management are not included in current version.
- ✓ Security options provide only low level security against beginner attackers.
- ✓ GUI is English only.

## **CHAPTER 10**

### **FUTURE ENHANCEMENT**

- ✓ This software can be made for all OS.
- ✓ Higher security features can be included in the software.
- ✓ Program scheduling can be included in the software.
- ✓ This software can be developed to use as tutorial to teach basic concepts of OS to new users.
- ✓ This system can be implemented with OS to reduce overhead of installing and running interface of each and every tool at different place.
- ✓ Automatic shutdown through SMS service can be implemented in this.

## **CHAPTER 11**

### **CONCLUSION**

The project entitled “Student Management System” is developed using Python Tkinter as front end and MYSQL database in back end to computerize the process of management of student records. This project covers only the basic features required.

## CHAPTER 12

### BIBLIOGRAPHY & REFERENCES

#### **Bibliography:-**

- “Python Programming” by Dr. Ms. Manisha Bharambe
- “Python” by Dr. Ms. Manisha Bharambe
- “Python GUI Programming with Tkinter” by Alan D. Moore

#### **References:-**

- Wikipedia
- <https://www.geeksforgeeks.org/python-gui-tkinter/>
- <https://www.javatpoint.com/python-tkinter>
- <https://www.python.org/>