SRMS

An Innovative Student Result Management System

A MINI PROJECT REPORT

Submitted by

PRASEETHA S 220701202

SAKTHI SHALINI R 220701241

In partial Fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI – 602105

2023-2024

BONAFIDE CERTIFICATE

Certified that this project report

“ SRMS : An Innovative Student Result Management System “

is the bonafide work of

“**PRASEETHA S (220701202) , SAKTHI SHALINI R (220701241)** “

Who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA
Professor and II Year Academic Head
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105

SIGNATURE

Mrs.K.MAHESHMEENA
Assistant Professor,
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105

ABSTRACT

A **Student Result Management System (SRMS)** is a digital solution designed to streamline the process of recording, analyzing, and disseminating student academic performance data. It automates grade entry, calculates GPA, and generates comprehensive reports, enhancing accuracy and efficiency. By providing secure, centralized access to results, the SRMS facilitates communication between educators, students, and parents. It supports data-driven decision-making, identifies academic trends, and helps in early detection of students needing intervention. This system modernizes traditional result management, ensuring timely and transparent dissemination of academic records, ultimately contributing to improved educational outcomes.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

A **Student Result Management System (SRMS)** is an innovative digital platform designed to enhance the efficiency and accuracy of managing student academic records. In educational institutions, handling vast amounts of data related to student performance can be a complex and time-consuming task. Traditional methods, often reliant on manual entry and paper-based systems, are prone to errors, delays, and inefficiencies. The SRMS addresses these challenges by automating the entire process of result management, from grade entry to report generation.

This system enables educators to input grades, calculate GPA, and produce comprehensive academic reports quickly and accurately. By centralizing data, it provides a secure and accessible repository for student performance records, facilitating seamless communication between teachers, students, and parents. The SRMS also supports advanced features such as statistical analysis, trend identification, and the early detection of students who may require additional support.

1.2 OBJECTIVES

The primary objectives of the **Student Result Management System (SRMS)** project are as follows:

- **Accuracy and Efficiency:**

    i) Automated Data Entry: Reduce manual errors by automating grade input, ensuring accuracy in student records.
    ii) Swift Processing: Speed up the process of calculating GPAs and generating reports, saving time for educators.

- **Centralized Data Management:**

    i) Unified Repository: Maintain all student performance data in a single, secure location for easy access and management.
    ii) Enhanced Accessibility: Allow authorized users, such as teachers, students, and parents, to access relevant information anytime, anywhere.

- **Improved Communication:**

    i) Timely Updates: Provide instant updates on student performance to parents and students, fostering transparency.
    ii) Collaborative Platform: Enable better communication between educators and parents, facilitating discussions on student progress and areas for improvement.

- **Data-Driven Insights:**

    i) Performance Analytics: Utilize analytics tools to identify academic trends and student performance patterns.
    ii) Informed Decision-Making: Support data-driven decisions by providing detailed insights into student achievements and areas needing attention.

- **Early Intervention:**

    i) Risk Identification: Detect students at risk of academic underperformance early through continuous monitoring.
    ii) Targeted Support: Enable educators to design targeted interventions and support strategies to assist struggling students.

- **Enhanced Reporting:**

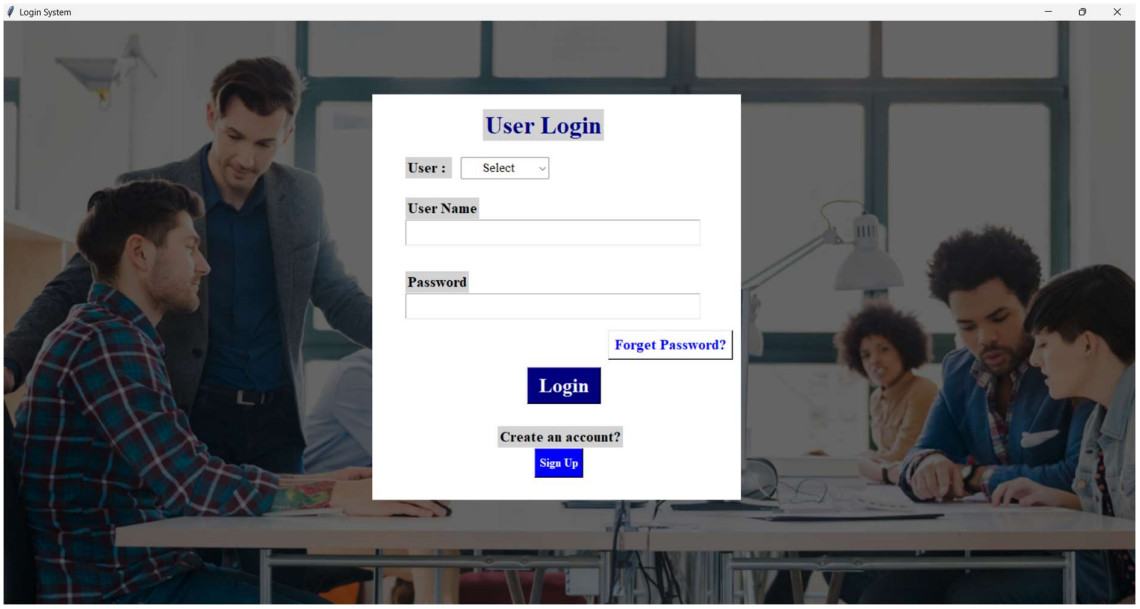
    i) Comprehensive Reports: Generate detailed academic reports that encompass a wide range of performance metrics.
    ii) Customizable Output: Allow customization of report formats to meet the specific needs of different stakeholders, such as parents, school administrators, and regulatory bodies.

1.3 MODULES

The Student Result Management System project is divided into several modules, each responsible for different aspects of the system. The key modules are as follows:

- **User Authentication Module:**

  i) Registration: Allows new users to create an account by providing their details.
  ii) Login: Authenticates users based on their credentials and provides access to the system.
  iii) Logout: Allows users to securely log out of the application.



- **Menu Module:**

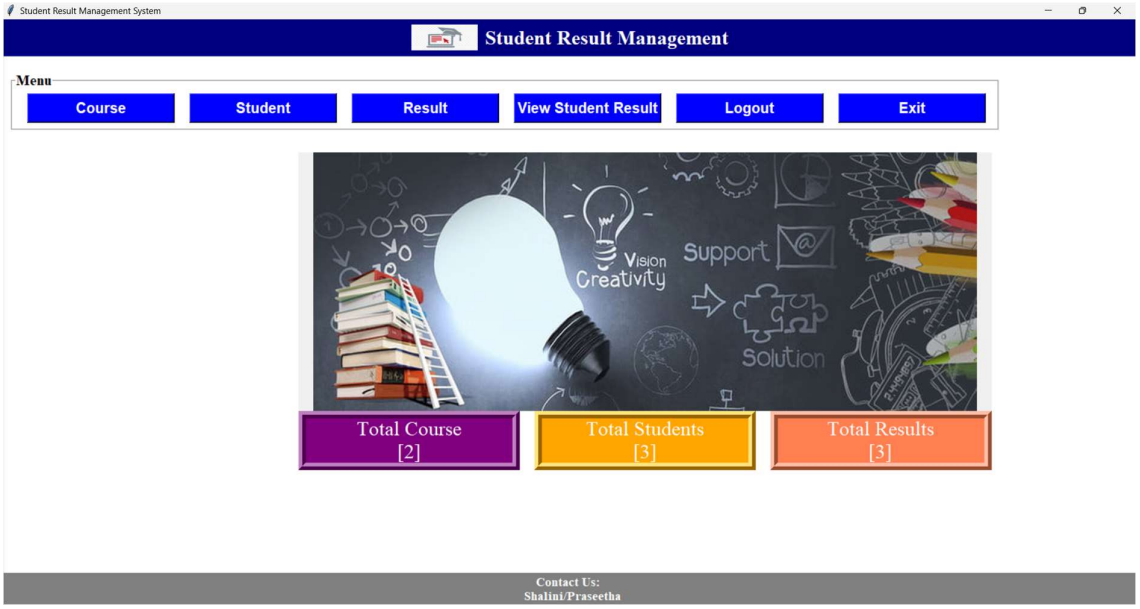
i) Course:
This option allows you to manage the courses offered. You can add, edit, or delete courses from the system.

ii) Student:
This section is for managing student data. You can add new students, update existing student information, or remove students.

iii) Result:
This is where you can input and manage student results. You can add, update, or delete results for each student.

iv) View Student Result:
This option allows you to view the results of individual students. You can search for a student and view their academic performance.

v) Logout:
Click here to safely log out of the system.



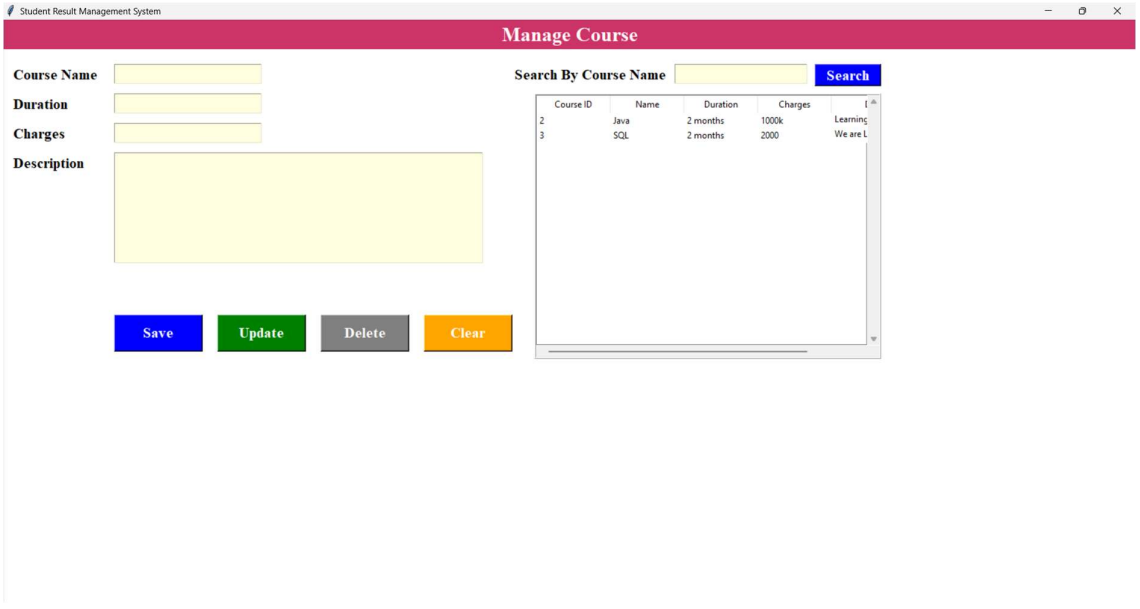
- **Course Module:**

i) Input Fields: On the left side, there are input fields labeled “Course Name”, “Duration”, “Charges”, and “Description”. These fields are likely used to enter

ii) Buttons: Below the input fields, there are four buttons: “Save”, “Update”, “Delete”, and “Clear”. These buttons are probably used to perform actions on the data entered in the input fields.

iii) Search Functionality: On the right side, there’s a search functionality section with an input field labeled “Search By Course Name” followed by a “Description”. These fields are likely used to enter and manage course information.

iv) Data Table: Below the search section, there’s a table with columns titled ‘Course ID’, ‘Name’, ‘Duration’, ‘Charges’, and ‘%’. This table seems to display the course data, possibly the search results or all available courses.



- **Result Module:**

i) Select Student: This is a dropdown menu where you can select a student. There's also a 'Search' button next to it, which is likely used to find students in the system.

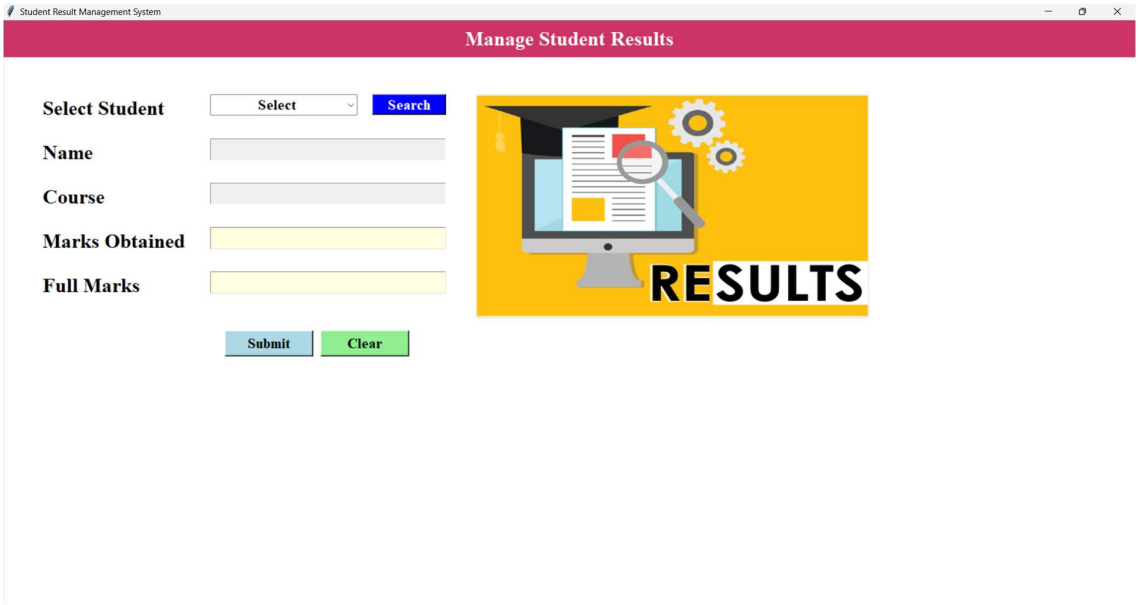
ii) Name: This field is probably used to display the selected student's name.

iii) Course: This field is likely used to display the course that the selected student is enrolled in.

iv) Marks Obtained: This field is likely used to enter or display the marks obtained by the student in the selected course

v) Full Marks: This field is likely used to enter or display the full marks of the selected course.

vi) Submit and Clear Buttons: The 'Submit' button is probably used to save the entered data to the system. The 'Clear' button is likely used to clear all input fields.



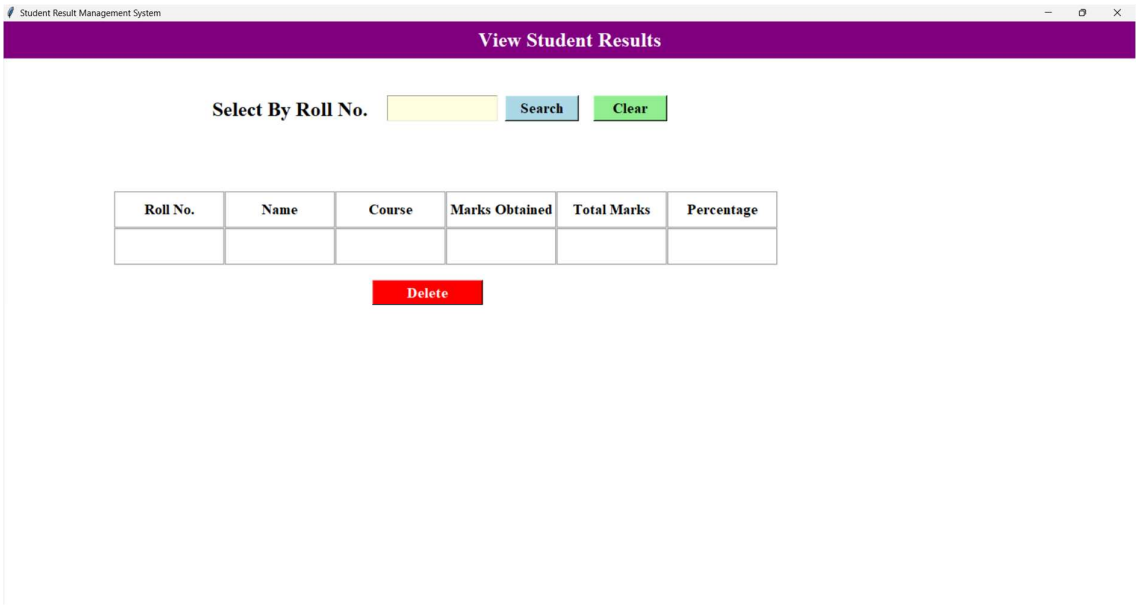
- **View Student Result Module:**

i) Select By Roll No.: This is a dropdown menu where you can select a student by their roll number. There's also a 'Search' button next to it, which is likely used to find students in the system.

ii) Table: Below the search section, there's a table with columns for 'Roll No.', 'Name', 'Course', 'Marks Obtained', 'Total Marks', and 'Percentage'. This table seems to display the student results, possibly the search results or all available students.

iii) Delete Button: At the bottom of the table, there is a button labeled "Delete". This button is probably used to delete the selected student's result from the system.

iv) Clear Button: Next to the 'Search' button, there's a 'Clear' button. This button is likely used to clear the selected option in the "Select By Roll No." dropdown menu.



2.SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

The Student Result Management project utilizes various software technologies to ensure a robust and scalable system. The core technologies include Python for server-side scripting, SQLite for database management.

2.2 LANGUAGES

The Student Result Management project leveraged several programming languages and technologies to build the system. This language was chosen for its specific strengths and contributions to different aspects of the project.

2.2.1 PYTHON

PYTHON is a widely-used open-source scripting language suited for web development. It was employed for server-side scripting to handle data processing, database interactions, and dynamic content generation. Key features of PYTHON utilized in this project include:

- Server-side scripting
- Form handling
- Database connectivity using SQLite
- Session management

LIBRARIES: -

1) TKINTER:
   Tkinter is a standard GUI (Graphical User Interface) library in Python. It provides tools to create windows, dialogs, buttons, and other graphical elements, enabling interactive applications development easily.

2) PIL :
   PIL (Python Imaging Library) is a Python library for opening, manipulating, and saving image files. It supports numerous file formats and provides powerful image processing capabilities, such as resizing, filtering, and enhancing.

3) SQLite:
   SQLite is a lightweight, self-contained SQL database engine. It's serverless, zero-configuration, and transactional, making it ideal for embedded database applications, prototyping, and small to medium-sized data storage needs.

4) OS :
   The OS module in Python provides functions for interacting with the operating system. It offers operations like file management, directory handling, process management, and system-specific functionalities, facilitating cross-platform development.

2.2.2 TOOLS USED: -

1) SQLite3: -

SQLite3 is a lightweight, self-contained SQL database engine included in Python's standard library. It allows for creating, querying, and managing relational databases using SQL commands. SQLite3 databases are stored as a single file, making them easy to manage, deploy, and integrate into applications without requiring a separate server.

2) VISUAL STUDIO CODE: -

Visual Studio Code (VS Code) is a lightweight, open-source code editor developed by Microsoft, renowned for its versatility and extensive feature set. It offers a customizable and intuitive interface with support for various programming languages, extensions, and debugging tools. With features like IntelliSense, Git integration, and built-in terminal, VS Code enhances developer productivity, making it a favourite among developers for building web, mobile, and cloud applications.

3) GITHUB: -

GitHub is a web-based platform for version control using Git, enabling collaboration and code sharing among developers worldwide. It provides a centralized repository for managing and tracking changes to code, facilitating seamless collaboration through features like pull requests, code reviews, and issue tracking. With its vast community, robust documentation, and integration with popular development tools, GitHub has become an essential platform for software development, fostering innovation and open-source contributions across various domains.

3.REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

- User Requirements:

  The system should be accessible via web browsers and mobile devices. Users should be able to create, edit, and delete inmate details, add inmates, and predict sentence.

- System Requirements:

  The system should ensure data integrity and provide a secure environment for user information.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

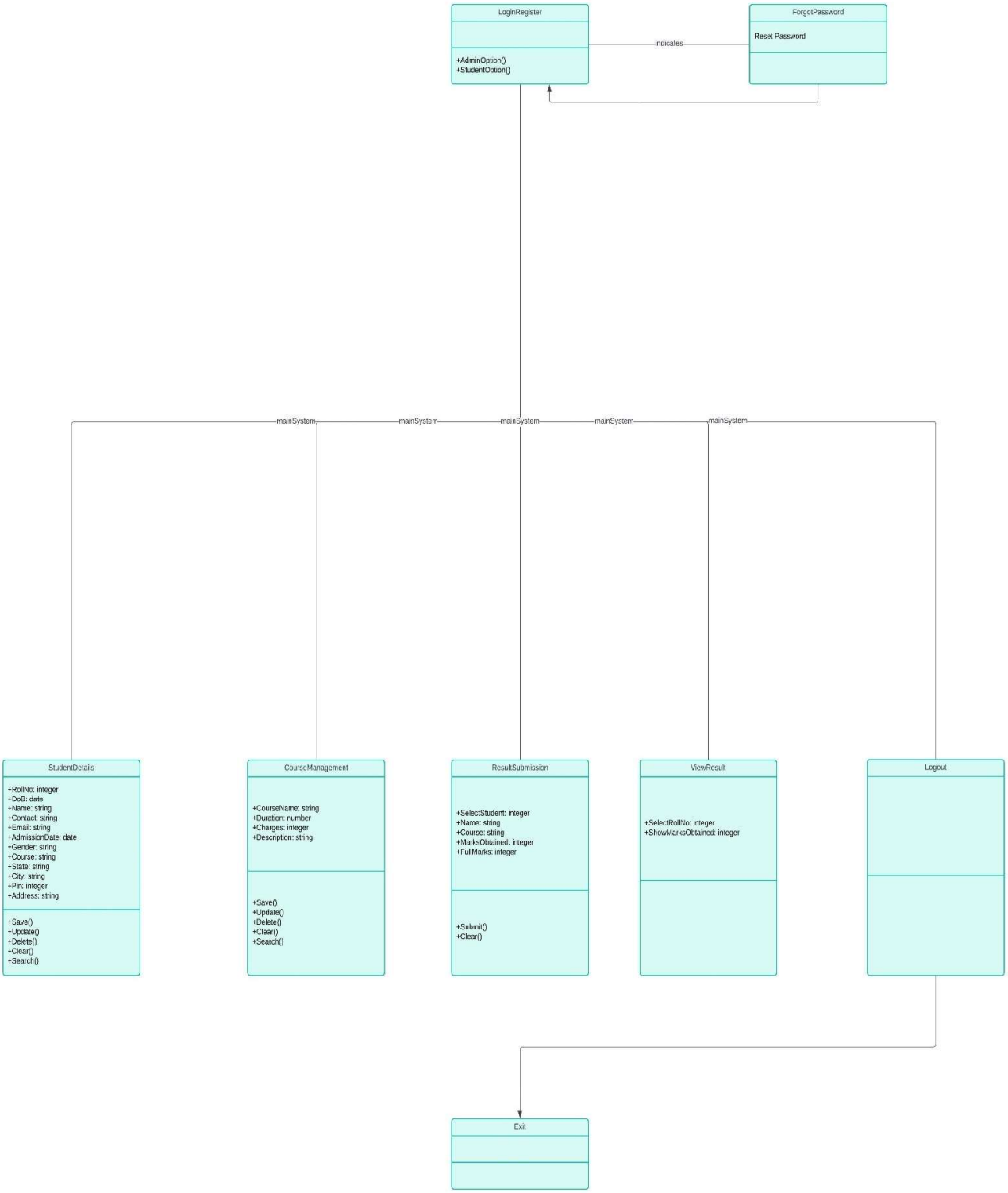
- Hardware:

  i) Server with minimum 4GB RAM and 100GB storage.
  ii) Client devices with internet access.

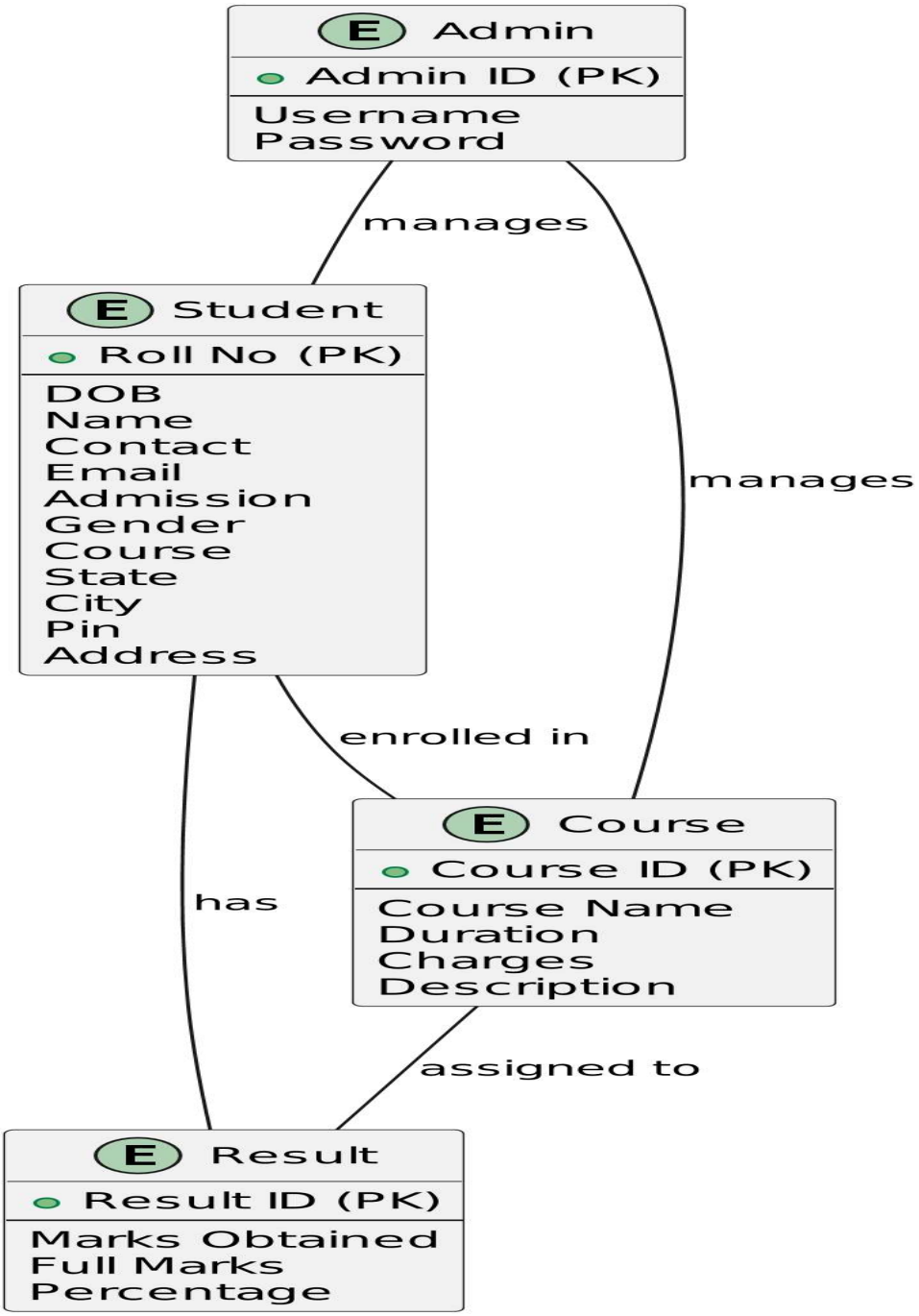
- Software:

  i) Operating System: Linux/Windows
  ii) Web Server: Apache
  iii) Database: SQLite3
  iv) Languages: Python
  v) Libraries:TKInter,PIL,OS and SQLite

3.3 ARCHITECTURE DIAGRAM



3.4 ER DIAGRAM



4.PROGRAM CODE

4.1 Module Codes:-

Login:

```
#Login System
from tkinter import *
from PIL import Image, ImageTk
from tkinter import messagebox,ttk
import sqlite3
import os
class Login:
    def __init__(self, home):
        self.home=home
        self.home.title("Login System")
        self.home.geometry("1400x700+0+0")
        #====BG IMage=====
        self.bg=ImageTk.PhotoImage(file="images/6.jpg")
        self.bg_image=Label(self.home,image=self.bg).place(x=0,y=0,relwidth=1,relheight=1)
        Frame_login=Frame(self.home,bg="white")
        Frame_login.place(x=500,y=100,height=550,width=500)
        # Login title
        title=Label(Frame_login,text="User Login",font=("Times New Roman",25,"bold"),fg="navy
blue",bg="light gray").place(x=150,y=20)
        #Username and password lable
        lbl_user=Label(Frame_login,text="User Name",font=("Times New
Roman",15,"bold"),fg="black",bg="light gray").place(x=45,y=140)
        self.txt_user=Entry(Frame_login,font=("times new roman",15),bg="white")
        self.txt_user.place(x=45,y=170,width=400,height=35)
        lbl_pass=Label(Frame_login,text="Password",font=("Times New Roman",15,"bold"),fg="black",bg="light
gray").place(x=45,y=240)
        self.txt_pass=Entry(Frame_login,font=("times new roman",15),bg="white")
        self.txt_pass.place(x=45,y=270,width=400,height=35)
        forget=Button(Frame_login,text="Forget
Password?",cursor="hand2",command=self.forget_window,bg="white",fg="blue",font=("times new
roman",15,"bold")).place(x=320,y=320)
        #Buttons
        signup=Button(Frame_login,text="Sign
Up",cursor="hand2",command=self.register_window,bg="blue",fg="white",font=("times new
roman",12,"bold")).place(x=220,y=480,height=40)
        login=Button(Frame_login,command=self.login_function,text="Login",cursor="hand2",bg="navy
blue",fg="white",font=("times new roman",20,"bold")).place(x=210,y=370,width=100,height=50)
        lbl_create=Label(Frame_login,text="Create an account?",font=("Times New
Roman",15,"bold"),fg="black",bg="light gray").place(x=170,y=450)
        #Checkbuttons
        UserType=Label(Frame_login,text="User : ", font=("times new roman", 15, "bold"), bg="light
gray",fg="black").place(x=45,y=85)
```

```
        self.txt_UserType=ttk.Combobox (Frame_login,font=("times new roman",13),
state="readonly",justify=CENTER)
        self.txt_UserType['values']=("Select","Student","Admin" )
        self.txt_UserType.place(x=120, y=85,width=120,height=30)
        self.txt_UserType.current (0)
    def reset(self):
        self.cmb_quest.current(0)
        self.txt_new_passwd.delete(0,END)
        self.txt_answer.delete(0,END)
        self.txt_pass.delete(0,END)
        self.txt_user.delete(0,END)
# Function for forgot password
    def forget_passwd(self):
        if self.cmb_quest.get()=="Select"or self.txt_answer.get()=="" or self.txt_new_passwd.get()=="":
            messagebox.showerror("Error","All fields are required",parent=self.home2)
        else:
            try:
                conn=sqlite3.connect(database="ResultManagementSystem.db")
                cur=conn.cursor()
                cur.execute("Select * from AllUsers where email=? and question=? and
answer=?",(self.txt_user.get(),self.cmb_quest.get(),self.txt_answer.get()))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error","Please Select the Correct security question and
answer",parent=self.home2)
                else: #Coding for Password Pattern
                    l,u,p,d=0,0,0,0
                    if len(self.txt_new_passwd.get())>=8:
                        for i in self.txt_new_passwd.get():
                            if(i.islower()):
                                l+=1
                            elif(i.isupper()):
                                u+=1
                            elif(i.isdigit()):
                                d+=1
                            elif(i=='@'or i=='&'or i=='#' or i=='!' or i=='_'):
                                p+=1
                        if(l>=1 and u>=1 and d>=1 and p>=1):
                            cur.execute("update AllUsers set password=? where email=?
",(self.txt_new_passwd.get(),self.txt_user.get()))
                            conn.commit()
                            conn.close()
                            messagebox.showinfo("Success","Your Password has been Reset, Please Login with new
password",parent=self.home2)
                            self.reset()
                            self.home2.destroy()
```

```
                else:
                    messagebox.showerror("Error","Password should have 8 characters ,atleast one upper case
letter,lower case letter,numeric value and special character '@','&','#','!','_'",parent=self.home2)
            else:
                messagebox.showerror("Error","Password should have 8 characters ,atleast one upper case
letter,lower case letter,numeric value and special character '@','&','#','!','_'",parent=self.home2)
        except Exception as es:
            messagebox.showerror("Error",f"Error Due to: {str(es)}",parent=self.home)
#Forget Password Window
    def forget_window(self):
        if self.txt_user.get()=="":
            messagebox.showerror("Error","Please enter email to reset your password",parent=self.home)
        else:
            try:
                conn=sqlite3.connect(database="ResultManagementSystem.db")
                cur=conn.cursor()
                cur.execute("Select * from AllUsers where email=?",(self.txt_user.get(),))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error","Please Enter Valid Email address to reset your
password",parent=self.home)
                else:
                    conn.close()
                    self.home2=Toplevel()
                    self.home2.title("Forget Password")
                    self.home2.geometry("400x400+500+150")
                    self.home2.focus_force()
                    self.home2.grab_set()
                #Title for Forget Interface
                    title=Label(self.home2,text="Forget Password",font=("Times New
Roman",25),fg="red",bg="white").place(x=0,relwidth=1)
                #Contents in Forget Password Page
                    question=Label(self.home2, text="Security Question", font=("times new roman", 15, "bold"),
bg="white",fg="gray").place(x=50,y=100)
                    self.cmb_quest=ttk.Combobox (self.home2, font=("times new roman",13),
state="readonly",justify=CENTER)
                    self.cmb_quest['values']=("Select", "Your First Pet Name", "Your Birth Place", "Your Best Friend
Name")
                    self.cmb_quest.place(x=50, y=130,width=250)
                    self.cmb_quest.current (0)
                    answer=Label(self.home2, text="Answer", font=("times new roman",15,"bold"),bg="white",
fg="gray").place(x=50,y=180)
                    self.txt_answer=Entry(self.home2, font=("times new roman", 15),bg="lightgray")
                    self.txt_answer.place(x=50, y=210,width=250)
                    new_passwd=Label(self.home2, text="New Password", font=("times new
roman",15,"bold"),bg="white", fg="gray").place(x=50,y=260)
                    self.txt_new_passwd=Entry(self.home2, font=("times new roman", 15),bg="lightgray")
                    self.txt_new_passwd.place(x=50, y=290,width=250)
```

```
                btn_change_passwd=Button(self.home2,text="Reset
Password",command=self.forget_passwd,bg="Green",fg="White",font=("times new
roman",15,"bold")).place(x=90,y=340)
            except Exception as es:
                messagebox.showerror("Error",f"Error Due to: {str(es)}",parent=self.home)
#By clicking on Sign-Up button going directly on sign-up page by destroying login page
    def register_window(self):
        self.home.destroy()
        import Register
#Function for Login Part
    def login_function(self):
        if self.txt_pass.get()=="" or self.txt_user.get()=="" or self.txt_UserType.get()=="select":
            messagebox.showerror("Error","All fields are requires",parent=self.home)
        else:
            try:
                conn=sqlite3.connect(database="ResultManagementSystem.db")
                cur=conn.cursor()
                cur.execute("Select * from AllUsers where email=? and password=? and u_name=?
",(self.txt_user.get(),self.txt_pass.get(),self.txt_UserType.get()))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error","Invalid Username or Password or UserType",parent=self.home)
                else:
                    if (self.txt_UserType.get()=="Student"):
                        messagebox.showinfo("Success",f"Welcome :- {self.txt_user.get()}",parent=self.home)
                        self.home.destroy()
                        os.system("python dashboardStudent.py")
                    else:
                        messagebox.showinfo("Success",f"Welcome :- {self.txt_user.get()}",parent=self.home)
                        self.home.destroy()
                        os.system("python dashboard.py")
                    conn.close()
            except Exception as es:
                messagebox.showerror("Error",f"Error Due to: {str(es)}",parent=self.home)
home=Tk()
obj=Login(home)
home.mainloop()
```

Register;-

```
#Register Page
from tkinter import*
from tkinter import ttk,messagebox
from turtle import width
from PIL import Image, ImageTk #pip install pillow
import sqlite3
import os
class Register:
    def __init__(self, home):
```

```
        self.home=home
        self.home.title("Registeration Window")
        self.home.geometry("1350x700+0+0")
        self.home.config(bg="white")
        #===Bg Image===
        self.bg=ImageTk.PhotoImage(file="Images/6.1.jpg")
        bg =Label(self.home, image=self.bg).place(x=250, y=0, relwidth=1,relheight=1)
        #===LEFT Images===
        self.left=ImageTk.PhotoImage(file="Images/left.jpg")
        left=Label(self.home, image=self.left).place(x=80, y=100, width=400,height=500)
        #===Register frame ====
        frame1=Frame(self.home, bg="white")
        frame1.place(x=480, y=100,width=700,height=550)
        title=Label(frame1, text="REGISTER HERE", font=("times new roman", 20, "bold"),
bg="white",fg="green").place(x=50,y=30)
        #----------------Row1
        self.var_fname=StringVar()
        f_name=Label(frame1, text="First Name", font=("times new roman",15,
"bold"),bg="white",fg="gray").place(x=50,y=100)
        self.txt_fname=Entry(frame1,font=("times new roman",15),bg="lightgray")
        self.txt_fname.place(x=50,y=130,width=250)
        l_name=Label(frame1, text="Last Name", font=("times new roman", 15, "bold"),
bg="white",fg="gray").place(x=370,y=100)
        self.txt_lname=Entry(frame1, font=("times new roman",15), bg="lightgray")
        self.txt_lname.place(x=370, y=130,width=250)
        #-------------Row2
        contact=Label(frame1, text="Contact No.", font=("times new roman", 15, "bold"),
bg="white",fg="gray").place(x=50,y=170)
        self.txt_contact=Entry(frame1, font=("times new roman", 15), bg="lightgray")
        self.txt_contact.place(x=50, y=200,width=250)
        #-----------------Row3
        email=Label(frame1, text="Email", font=("times new roman",15,
"bold"),bg="white",fg="gray").place(x=370,y=170)
        self.txt_email=Entry(frame1, font=("times new roman",15), bg="lightgray")
        self.txt_email.place(x=370, y=200,width=250)
        #----------------Row4
        question=Label(frame1, text="Security Question", font=("times new roman", 15, "bold"),
bg="white",fg="gray").place(x=50,y=240)
        self.cmb_quest=ttk.Combobox (frame1, font=("times new roman",13), state="readonly",justify=CENTER)
        self.cmb_quest['values']=("Select", "Your First Pet Name", "Your Birth Place", "Your Best Friend Name")
        self.cmb_quest.place(x=50, y=270,width=250)
        self.cmb_quest.current (0)
        answer=Label(frame1, text="Answer", font=("times new roman",15,"bold"),bg="white",
fg="gray").place(x=370,y=240)
        self.txt_answer=Entry(frame1, font=("times new roman", 15),bg="lightgray")
        self.txt_answer.place(x=370, y=270,width=250)
```

```
        #-------Row5
        password=Label(frame1, text="Password", font=("times new
roman",15,"bold"),bg="white",fg="gray").place(x=50,y=310)
        self.txt_password=Entry(frame1, font=("times new roman",15),bg="lightgray")
        self.txt_password.place(x=50, y=340,width=250)
        cpassword=Label(frame1, text="Confirm Password", font=("times new roman", 15, "bold"),
bg="white",fg="gray").place(x=370,y=310)
        self.txt_cpassword=Entry(frame1, font=("times new roman",15), bg="lightgray")
        self.txt_cpassword.place(x=370, y=340,width=250)
        UserType=Label(frame1, text="User", font=("times new roman", 15, "bold"),
bg="white",fg="gray").place(x=220,y=380)
        self.txt_UserType=ttk.Combobox (frame1, font=("times new roman",13),
state="readonly",justify=CENTER)
        self.txt_UserType['values']=("Select", "Student", "Admin")
        self.txt_UserType.place(x=270, y=380,width=100)
        self.txt_UserType.current (0)
        #-------Terms-----
        self.var_chk=IntVar()
        chk=Checkbutton (frame1, text="I Agree The Terms & Conditions",variable=self.var_chk,onvalue=1,
offvalue=0, font=("times new roman",12),bg="white").place(x=50,y=430)
        self.btn_img=ImageTk.PhotoImage(file="Images/register.png")
        btn_register=Button(frame1,image=self.btn_img,bd=0,
cursor="hand2",command=self.register_data).place(x=50,y=470)
        btn_login=Button(self.home,text="Sign In",command=self.login_window,font=("times new
roman",20,"bold"),fg="white",bg="blue").place(x=280,y=580)
    def login_window(self):
        self.home.destroy()
        os.system("python login.py")
    def clear(self):
        self.txt_fname.delete(0,END)
        self.txt_lname.delete(0,END)
        self.txt_contact.delete(0,END)
        self.txt_email.delete(0,END)
        self.txt_answer.delete(0,END)
        self.txt_password.delete(0,END)
        self.txt_cpassword.delete(0,END)
        self.cmb_quest.current(0)
        self.txt_UserType.current (0)
#Register Function
    def register_data(self):
        if self.txt_fname.get()=="" or self.txt_contact.get()=="" or self.txt_email.get()=="" or
self.cmb_quest.get()=="Select"or self.txt_answer.get()=="" or self.txt_password.get()==""or
self.txt_cpassword.get()=="" or self.txt_UserType.get()=="Select" :
            messagebox.showerror("Error", "All Fields Are Required",parent=self.home)
        elif self.txt_password.get()!=self.txt_cpassword.get():
            messagebox.showerror("Error", "Password & Confirm Password should be same",parent=self.home)
        elif self.var_chk.get()==0:
            messagebox.showerror("Error", "Please Agree our terms & condition",parent=self.home)
```

```
        else:
            try:
                fn = 0
                ln = 0
                #Validation for first and last name
                for i in self.txt_fname.get():
                    if i.isupper() or i.islower():
                        fn+=1
                for i in self.txt_lname.get():
                    if i.isupper() or i.islower():
                        ln+=1
                if (fn==len(self.txt_fname.get())) and (ln==len(self.txt_lname.get())):
                    #Validation for contact number
                    s = 0
                    for i in self.txt_contact.get():
                        if i.isdigit():
                            s+=1
                    if s==10:
                        #Validation for Email
                        k=self.txt_email.get()[-10:-1]
                        j=k+"m"
                        if(j=="@gmail.com") and len(self.txt_email.get())>=12:
                            #Validation for Password
                            l,u,p,d=0,0,0,0
                            if len(self.txt_password.get())>=8:
                                for i in self.txt_password.get():
                                    if(i.islower()):
                                        l+=1
                                    elif(i.isupper()):
                                        u+=1
                                    elif(i.isdigit()):
                                        d+=1
                                    elif(i=='@'or i=='&'or i=='#' or i=='!' or i=='_'):
                                        p+=1
                                if(l>=1 and u>=1 and d>=1 and p>=1):
                                    #Connection with database
                                    conn=sqlite3.connect(database="ResultManagementSystem.db")
                                    cur=conn.cursor()
                                    cur.execute("select *from AllUsers where email=?",(self.txt_email.get(),))
                                    row=cur.fetchone()
                                    if row!=None:
                                        messagebox.showerror("Error", "User already exist please try with another
email",parent=self.home)
                                    else:
                                        cur.execute("insert into AllUsers (f_name,l_namen, contact, email, question, answer,
password,u_name) values(?,?,?,?,?,?,?,?)",
                                                    (self.txt_fname.get(),
                                                    self.txt_lname.get(),
```

```
                                        self.txt_contact.get(),
                                        self.txt_email.get(),
                                        self.cmb_quest.get(),
                                        self.txt_answer.get(),
                                        self.txt_password.get(),
                                        self.txt_UserType.get()
                                        ))
                                    conn.commit()
                                    conn.close()
                                    messagebox.showinfo("Success", "Register Successful",parent=self.home)
                                    self.clear()
                                    self.login_window()
                                else:
                                    messagebox.showerror("Error","Password should have 8 characters ,upper case
letter,lower case letter,numeric value and special character '@','&','#','!','_'",parent=self.home)
                            else:
                                messagebox.showerror("Error","Password should have 8 characters ,atleast one upper case
letter,lower tcase letter,numeric value and special character '@','&','#','!','_'",parent=self.home)
                        else:
                            messagebox.showerror("Error","Please Enter Valid Email address")
                    else:
                        messagebox.showerror("Error","Please Enter Valid Contact Number.")
                else:
                    messagebox.showerror("Error","Please Enter Valid First name or Last name.")
            except Exception as es:
                    messagebox.showerror("Error", f"Error due to:{str(es)}",parent=self.home)

home=Tk()
obj=Register(home)
home.mainloop()
```

Password:-

```
#Password

pass1 =input("Enter Password: ")
l,u,p,d=0,0,0,0
if len(pass1)>=8:
    for i in pass1:
        if(i.islower()):
            l+=1
        elif(i.isupper()):
            u+=1
        elif(i.isdigit()):
            d+=1
        elif(i=='@'or i=='&'or i=='#' or i=='!' or i=='_'):
            p+=1
    if(l>=1 and u>=1 and d>=1 and p>=1):
        print("Valid Password")
    else:
```

```
        print("Invalid Password")
else:
    print("Invalid Password")
#First Name and Last Name
FirstName = 'Shubham'
LastName= "kumbhar"
fn = 0
ln = 0
for i in FirstName:
    if i.isupper() or i.islower():
        fn+=1
for i in LastName:
    if i.isupper() or i.islower():
        ln+=1
if fn == len(FirstName) and ln == len(LastName):
    print("OK")
else:
    print("Error")
contact="1234567288"
s = 0
for i in contact:
    if i.isdigit():
        s+=1
if s==10:
    print("Continue")
else:
    print("error")
#Right Email address Verification
pass1 ="shubham@gmail.com"
k=pass1[-10:-1]
j=k+"m"
print(j)

if(j!="@gmail.com"):
    print("error")
else:
    print("Ok")
```

4.2 WebApplication:-

Course:-

```
#Course Module
from tkinter import*
from PIL import Image,ImageTk
from tkinter import ttk,messagebox
import sqlite3
class CourseClass:
    def __init__(self,home):
        self.home=home
        self.home.title("Student Result Management System")
        self.home.geometry("1200x500+80+170")
        self.home.config(bg="white")
        self.home.focus_force()
    #Title of Course
        title=Label(self.home,text="Manage Course",font=("times new
roman",20,"bold"),bg="#CC3366",fg="white").place(x=0,y=0,relwidth=1,height=40)
    #Variables
        self.var_course=StringVar()
        self.var_duration=StringVar()
        self.var_charges=StringVar()
    #Categories of Courses
        courseName = Label(self.home,text="Course Name",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=60)
        duration = Label(self.home,text="Duration",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=100)
        charges = Label(self.home,text="Charges",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=140)
        description = Label(self.home,text="Description",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=180)
    #Entry Fields
        self.courseName1 = Entry(self.home,textvariable=self.var_course,font=("times new
roman",15,"bold"),bg="lightyellow")
        self.courseName1.place(x=150,y=60,width=200)
        duration1 = Entry(self.home,textvariable=self.var_duration,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=150,y=100,width=200)
        charges1 = Entry(self.home,textvariable=self.var_charges,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=150,y=140,width=200)
        self.description1 = Text(self.home,font=("times new roman",15,"bold"),bg="lightyellow")
        self.description1.place(x=150,y=180,width=500,height=150)
    # Buttons
        self.add_btn=Button(self.home,text="Save",font=("times new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.add)
        self.add_btn.place(x=150,y=400,width=120,height=50)
        self.update_btn=Button(self.home,text="Update",font=("times new
roman",15,"bold"),bg="green",fg="white",cursor="hand2",command=self.update)
        self.update_btn.place(x=290,y=400,width=120,height=50)
```

```
        self.delete_btn=Button(self.home,text="Delete",font=("times new
roman",15,"bold"),bg="grey",fg="white",cursor="hand2",command=self.delete)
        self.delete_btn.place(x=430,y=400,width=120,height=50)
        self.clear_btn=Button(self.home,text="Clear",font=("times new
roman",15,"bold"),bg="orange",fg="white",cursor="hand2",command=self.clear)
        self.clear_btn.place(x=570,y=400,width=120,height=50)
    #Search Panel
        self.var_search=StringVar()
        search_courseName = Label(self.home,text="Search By Course Name",font=("times new
roman",15,"bold"),bg="white").place(x=690,y=60)
        search_courseName1 = Entry(self.home,textvariable=self.var_search,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=910,y=60,width=180)
        btn_search=Button(self.home,text="Search",font=("times new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.search).place(x=1100,y=60,width=90,
height=30)
    #Content
        self.C_Frame=Frame(self.home,bd=2,relief=RIDGE)
        self.C_Frame.place(x=720,y=100,width=470,height=360)
    #Table
        #Scroll bar for table to view all headings
        scroly=Scrollbar(self.C_Frame,orient=VERTICAL)
        scrolx=Scrollbar(self.C_Frame,orient=HORIZONTAL)
        # Columns and headings and adding commands for the functioning of scroll bar
        self.CourseTable=ttk.Treeview(self.C_Frame,columns=("cid","name","duration","charges","description"),x
scrollcommand=scrolx.set,yscrollcommand=scroly.set)
        scrolx.pack(side=BOTTOM,fill=X)
        scroly.pack(side=RIGHT,fill=Y)
        scrolx.config(command=self.CourseTable.xview)
        scroly.config(command=self.CourseTable.yview)
    #Headings and Coloumns for the tables
        self.CourseTable.heading("cid",text="Course ID")
        self.CourseTable.heading("name",text="Name")
        self.CourseTable.heading("duration",text="Duration")
        self.CourseTable.heading("charges",text="Charges")
        self.CourseTable.heading("description",text="Description")
        self.CourseTable["show"]="headings"
        self.CourseTable.column("cid",width=100)
        self.CourseTable.column("name",width=100)
        self.CourseTable.column("duration",width=100)
        self.CourseTable.column("charges",width=100)
        self.CourseTable.column("description",width=150)
        self.CourseTable.pack(fill=BOTH,expand=1)
        self.CourseTable.bind("<ButtonRelease-1>",self.get_data) #When you click on any cid row it will show
details on their sections get_data function is defined below
        self.show()  #It is help to show details in table the function is defined at the bottom
    def clear(self):
        self.show()
        self.var_course.set("")
```

```
        self.var_duration.set("")
        self.var_charges.set("")
        self.var_search.set("")
        self.description1.delete('1.0',END)
        self.courseName1.config(state=NORMAL)
    def delete(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            if self.var_course.get()=="":
                messagebox.showerror("Error","Course name should be required",parent=self.home)
            else:
                cur.execute("Select * from course where name=?",(self.var_course.get(),))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error, Select The Course From the List first",parent=self.home)
                else:
                    p=messagebox.askyesno("Confirm","Do you really want to delete",parent=self.home)
                    if p==True:
                        cur.execute("Delete from course where name=? ",(self.var_course.get(),))
                        conn.commit()
                        messagebox.showinfo("Delete","Course deleted Successfully",parent=self.home)
                        self.clear() #We are calling clear because we declare show in to that
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def get_data(self,event):
        self.courseName1.config(state="readonly")
        self.courseName1
        r=self.CourseTable.focus()
        content=self.CourseTable.item(r)
        row=content["values"]
        self.var_course.set(row[1])
        self.var_duration.set(row[2])
        self.var_charges.set(row[3])
        self.description1.delete('1.0',END)
        self.description1.insert(END,row[4])
# Adding Details and Saving
    def add(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            if self.var_course.get()=="":
                messagebox.showerror("Error","Course name should be required",parent=self.home)
            else:
                cur.execute("Select * from course where name=?",(self.var_course.get(),)) #Due to tupple we added ,
                row=cur.fetchone()
                if row!=None:
                    messagebox.showerror("Error, Course name already Present",parent=self.home)
```

```
            else:
                cur.execute("Insert into course (name,duration,charges,description) values(?,?,?,?)",(
                    self.var_course.get(),
                    self.var_duration.get(),
                    self.var_charges.get(),
                    self.description1.get("1.0",END)
                ) )
                conn.commit()
                messagebox.showinfo("Great","Course Added Successfully",parent=self.home)
                self.show()
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    #Updating Details
    def update(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            if self.var_course.get()=="":
                messagebox.showerror("Error","Course name should be required",parent=self.home)
            else:
                cur.execute("Select * from course where name=?",(self.var_course.get(),))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error","Select Course From List",parent=self.home)
                else:
                    cur.execute("Update course set duration=?,charges=?,description=? where name=? ",(
                        self.var_duration.get(),
                        self.var_charges.get(),
                        self.description1.get("1.0",END),
                        self.var_course.get()
                    ) )
                    conn.commit()
                    messagebox.showinfo("Great","Course Update Successfully",parent=self.home)
                    self.show()
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def show(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            cur.execute("Select * from course")
            rows=cur.fetchall()
            self.CourseTable.delete(*self.CourseTable.get_children())
            for row in rows:
                self.CourseTable.insert('',END,values=row)
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
```

```
    def search(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            cur.execute(f"Select * from course where name LIKE '%{self.var_search.get()}%'")
            rows=cur.fetchall()
            self.CourseTable.delete(*self.CourseTable.get_children())
            for row in rows:
                self.CourseTable.insert('',END,values=row)
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
if __name__=="__main__":
    home=Tk()
    obj=CourseClass(home)
    home.mainloop()
```

Dashboard:-

```
#Admin Dashboard
from tkinter import*
from tkinter import messagebox
from PIL import Image,ImageTk
from course import CourseClass
from student import StudentClass
from result import ResultClass
from ViewResult import ViewClass
import sqlite3
import os
class ResultManagementSystem:
    def __init__(self,home):
        self.home=home
        self.home.title("Student Result Management System")
        self.home.geometry("1450x700+0+0")
        self.home.config(bg="white")
        #Importing image logo (icons)
        self.logo=Image.open("images/logo.jpg")
        self.logo=self.logo.resize((90,35),Image.Resampling.LANCZOS)
        self.logo=ImageTk.PhotoImage(self.logo)
        #Title of project
        title=Label(self.home,text="Student Result
Management",padx=10,compound=LEFT,image=self.logo,font=("times new roman",20,"bold"),bg="Navy
Blue",fg="white").place(x=0,y=0,relwidth=1,height=50)
        # Menu
        Frame = LabelFrame(self.home,text="Menu",font=("times new roman",15,"bold"),bg="white")
        Frame.place(x=10,y=70,width=1340,height=80)
        #SubMenu
        button_Course=Button(Frame,text="Course",font=("ties new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.add_course).place(x=20,y=5,width=20
0,height=40)
```

```
        button_Student=Button(Frame,text="Student",font=("ties new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.add_student).place(x=240,y=5,width=
200,height=40)
        button_Result=Button(Frame,text="Result",font=("ties new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.add_result).place(x=460,y=5,width=20
0,height=40)
        button_View=Button(Frame,text="View Student Result",font=("ties new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.add_view).place(x=680,y=5,width=20
0,height=40)
        button_Logout=Button(Frame,text="Logout",font=("ties new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.logout).place(x=900,y=5,width=200,h
eight=40)
        button_Exit=Button(Frame,text="Exit",font=("ties new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.exit).place(x=1120,y=5,width=200,hei
ght=40)
        # Content Window
        self.bgImage=Image.open("Images/7.jpg")
        self.bgImage=self.bgImage.resize((900,350),Image.Resampling.LANCZOS)
        self.bgImage=ImageTk.PhotoImage(self.bgImage)
        self.lbl_bg=Label(self.home,image=self.bgImage).place(x=400,y=180,width=940,height=350)
        # Update Details
        self.totalCourse=Label(self.home,text="Total Courses \n 0 ",font=("times new
roman",20),bd=10,relief=RIDGE,bg="purple",fg="white")
        self.totalCourse.place(x=400,y=530,width=300,height=80)
        self.totalstudent=Label(self.home,text="Total Student \n 0 ",font=("times new
roman",20),bd=10,relief=RIDGE,bg="orange",fg="white")
        self.totalstudent.place(x=720,y=530,width=300,height=80)
        self.totalresults=Label(self.home,text="Total Results \n 0 ",font=("times new
roman",20),bd=10,relief=RIDGE,bg="coral",fg="white")
        self.totalresults.place(x=1040,y=530,width=300,height=80)
        #Footer
        footer=Label(self.home,text="Contact Us: \n  Shalini/Praseetha",font=("times new
roman",13,"bold"),bg="grey",fg="white").pack(side=BOTTOM,fill=X)
        self.update_details()
    #Adding function for bottom buttons(Total corses,students,results)
    def update_details(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            cur.execute("Select * from course")
            cr=cur.fetchall()
            self.totalCourse.config(text=f"Total Course\n[{str(len(cr))}]")
            self.totalCourse.after(200,self.update_details)
            cur.execute("Select * from student")
            cr=cur.fetchall()
            self.totalstudent.config(text=f"Total Students\n[{str(len(cr))}]")
            cur.execute("Select * from result")
            cr=cur.fetchall()
```

```
            self.totalresults.config(text=f"Total Results\n[{str(len(cr))}]")
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    #Adding Sub-Menus for Functioning
    def add_course(self):
        self.window1=Toplevel(self.home)
        self.obj1=CourseClass(self.window1)
    def add_student(self):
        self.window1=Toplevel(self.home)
        self.obj1=StudentClass(self.window1)
    def add_result(self):
        self.window1=Toplevel(self.home)
        self.obj1=ResultClass(self.window1)
    def add_view(self):
        self.window1=Toplevel(self.home)
        self.obj1=ViewClass(self.window1)
#Functioning of Exit and Logout Button
    def logout(self):
        op=messagebox.askyesno("Confirm Again","Do You really Want to Logout ?",parent=self.home)
        if op==True:
            self.home.destroy()
            os.system("Python Login.py")
    def exit(self):
        op=messagebox.askyesno("Confirm Again","Do You really Want to Exit ?",parent=self.home)
        if op==True:
            self.home.destroy()
if __name__=="__main__":
    home=Tk()
    obj=ResultManagementSystem(home)
    home.mainloop()
```

Dashboard For Student:-

```
#Student Dashboard
from tkinter import*
from tkinter import messagebox,ttk
from PIL import Image,ImageTk
from course import CourseClass
from student import StudentClass
from result import ResultClass
from ViewResult import ViewClass
import sqlite3
import os
class studentSystem:
    def __init__(self,home):
        self.home=home
        self.home.title("Student Page")
        self.home.geometry("1450x700+0+0")
        self.home.config(bg="white")
```

```
        #Title of Student Page
        title=Label(self.home,text="Student Result",font=("times new
roman",20,"bold"),bg="purple",fg="white").place(x=0,y=0,relwidth=1,height=50)
        #Searching Button
        self.var_search=StringVar()
        lbl_rollno = Label(self.home,text="Enter Roll No. ",font=("times new
roman",30,"bold"),bg="white").place(x=450,y=60)
        txt_rollno1 = Entry(self.home,textvariable=self.var_search,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=700,y=70,width=180,height=35)
        btn_search=Button(self.home,text="Search",font=("times new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.search).place(x=900,y=70,width=100,
height=35)
        btn_clear=Button(self.home,text="Clear",font=("times new
roman",15,"bold"),bg="orange",fg="white",cursor="hand2",command=self.clear).place(x=1020,y=70,width=10
0,height=35)
        button_Logout=Button(text="Logout",font=("ties new
roman",15,"bold"),bg="red",fg="white",cursor="hand2",command=self.logout).place(x=1170,y=70,width=100,
height=35)
        #Result Of Student and content to show
        lbl_roll = Label(self.home,text="Roll No.",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=100,y=200,width=190,height=90)
        lbl_name = Label(self.home,text="Name",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=290,y=200,width=190,height=90)
        lbl_course = Label(self.home,text="Course",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=480,y=200,width=190,height=90)
        lbl_marks = Label(self.home,text="Marks Obtained",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=670,y=200,width=190,height=90)
        lbl_full = Label(self.home,text="Total Marks",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=860,y=200,width=190,height=90)
        lbl_percentage = Label(self.home,text="Percentage",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=1050,y=200,width=190,height=90)
        self.roll = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.roll.place(x=100,y=290,width=190,height=90)
        self.name = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.name.place(x=290,y=290,width=190,height=90)
        self.course = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.course.place(x=480,y=290,width=190,height=90)
        self.marks = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.marks.place(x=670,y=290,width=190,height=90)
        self.full = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.full.place(x=860,y=290,width=190,height=90)
        self.percentage = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.percentage.place(x=1050,y=290,width=190,height=90)
#Functions
    def search(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
```

```
            if self.var_search.get()=="":
                messagebox.showerror("Error","Roll No. should be required",parent=self.home)
            else:
                cur.execute("Select * from result where roll=?",(self.var_search.get(),))
                row=cur.fetchone()
                if row !=None:
                    self.var_id=row[0]
                    self.roll.config(text=row[1])
                    self.name.config(text=row[2])
                    self.course.config(text=row[3])
                    self.marks.config(text=row[4])
                    self.full.config(text=row[5])
                    self.percentage.config(text=row[6])
                else:
                    messagebox.showerror("Error","No record Found",parent=self.home)
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def clear(self):
        self.var_id=""
        self.roll.config(text="")
        self.name.config(text="")
        self.course.config(text="")
        self.marks.config(text="")
        self.full.config(text="")
        self.percentage.config(text="")
        self.var_search.set("")
    def logout(self):
        op=messagebox.askyesno("Confirm Again","Do You really Want to Logout ?",parent=self.home)
        if op==True:
            self.home.destroy()
            os.system("Python Login.py")
if __name__=="__main__":
    home=Tk()
    obj=studentSystem(home)
    home.mainloop()
```

Student:-

```
#Student Module
from os import name
from tkinter import*
from PIL import Image,ImageTk
from tkinter import ttk,messagebox
import sqlite3
class StudentClass:
    def __init__(self,home):
        self.home=home
        self.home.title("Student Result Management System")
        self.home.geometry("1200x500+80+170")
```

```
        self.home.config(bg="white")
        self.home.focus_force()
    #Title of Course
        title=Label(self.home,text="Manage Student Details",font=("times new
roman",20,"bold"),bg="#CC3366",fg="white").place(x=0,y=0,relwidth=1,height=40)
    #Variables
        self.var_roll=StringVar()
        self.var_name=StringVar()
        self.var_email=StringVar()
        self.var_gender=StringVar()
        self.var_dob=StringVar()
        self.var_contact=StringVar()
        self.var_course=StringVar()
        self.var_adm_date=StringVar()
        self.var_state=StringVar()
        self.var_city=StringVar()
        self.var_pin=StringVar()
    #Categories of student details 1 side
        rollno = Label(self.home,text="Roll No.",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=60)
        name = Label(self.home,text="Name",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=100)
        email = Label(self.home,text="Email",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=140)
        gender = Label(self.home,text="Gender",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=180)
        state = Label(self.home,text="State",font=("times new roman",15,"bold"),bg="white").place(x=10,y=220)
        self.state1 = Entry(self.home,textvariable=self.var_state,font=("times new
roman",15,"bold"),bg="lightyellow")
        self.state1.place(x=150,y=220,width=150)
        city = Label(self.home,text="City",font=("times new roman",15,"bold"),bg="white").place(x=330,y=220)
        self.city1 = Entry(self.home,textvariable=self.var_city,font=("times new
roman",15,"bold"),bg="lightyellow")
        self.city1.place(x=380,y=220,width=110)
        pin = Label(self.home,text="Pin",font=("times new roman",15,"bold"),bg="white").place(x=510,y=220)
        self.pin1 = Entry(self.home,textvariable=self.var_pin,font=("times new
roman",15,"bold"),bg="lightyellow")
        self.pin1.place(x=560,y=220,width=120)
        address = Label(self.home,text="Address",font=("times new
roman",15,"bold"),bg="white").place(x=10,y=260)
    #Entry Fields 1
        self.rollno1 = Entry(self.home,textvariable=self.var_roll,font=("times new
roman",15,"bold"),bg="lightyellow")
        self.rollno1.place(x=150,y=60,width=200)
        name1 = Entry(self.home,textvariable=self.var_name,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=150,y=100,width=200)
        email1 = Entry(self.home,textvariable=self.var_email,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=150,y=140,width=200)
```

```
        self.gender1 =
ttk.Combobox(self.home,textvariable=self.var_gender,values=("Select","Male","Female","Other"),font=("times
new roman",15,"bold"),state="readonly",justify=CENTER)
        self.gender1.place(x=150,y=180,width=200)
        self.gender1.current(0)
    #Address
        self.address = Text(self.home,font=("times new roman",15,"bold"),bg="lightyellow")
        self.address.place(x=150,y=260,width=540,height=100)
    #Categories of student details 2 side
        dob = Label(self.home,text="D.O.B",font=("times new roman",15,"bold"),bg="white").place(x=360,y=60)
        contact = Label(self.home,text="Contact",font=("times new
roman",15,"bold"),bg="white").place(x=360,y=100)
        admission = Label(self.home,text="Admission",font=("times new
roman",15,"bold"),bg="white").place(x=360,y=140)
        course = Label(self.home,text="Course",font=("times new
roman",15,"bold"),bg="white").place(x=360,y=180)
    #Entry Fields 2
        self.course_list=[]
        #Function call to update list
        self.fetch_course()
        self.dob1 = Entry(self.home,textvariable=self.var_dob,font=("times new
roman",15,"bold"),bg="lightyellow")
        self.dob1.place(x=480,y=60,width=200)
        contact1 = Entry(self.home,textvariable=self.var_contact,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=480,y=100,width=200)
        admission1 = Entry(self.home,textvariable=self.var_adm_date,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=480,y=140,width=200)
        self.course1 = ttk.Combobox(self.home,textvariable=self.var_course,values=self.course_list,font=("times
new roman",15,"bold"),state="readonly",justify=CENTER)
        self.course1.place(x=480,y=180,width=200)
        self.course1.set("Select")
    # Buttons
        self.add_btn=Button(self.home,text="Save",font=("times new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.add)
        self.add_btn.place(x=150,y=400,width=120,height=50)
        self.update_btn=Button(self.home,text="Update",font=("times new
roman",15,"bold"),bg="green",fg="white",cursor="hand2",command=self.update)
        self.update_btn.place(x=290,y=400,width=120,height=50)
        self.delete_btn=Button(self.home,text="Delete",font=("times new
roman",15,"bold"),bg="grey",fg="white",cursor="hand2",command=self.delete)
        self.delete_btn.place(x=430,y=400,width=120,height=50)
        self.clear_btn=Button(self.home,text="Clear",font=("times new
roman",15,"bold"),bg="orange",fg="white",cursor="hand2",command=self.clear)
        self.clear_btn.place(x=570,y=400,width=120,height=50)
    #Search Panel
        self.var_search=StringVar()
        search_rollno = Label(self.home,text="Search By Roll No. ",font=("times new
roman",15,"bold"),bg="white").place(x=720,y=60)
```

```
        search_rollno1 = Entry(self.home,textvariable=self.var_search,font=("times new
roman",15,"bold"),bg="lightyellow").place(x=890,y=60,width=160,height=30)
        btn_search=Button(self.home,text="Search",font=("times new
roman",15,"bold"),bg="blue",fg="white",cursor="hand2",command=self.search).place(x=1070,y=60,width=100
,height=30)
    #Content
        self.C_Frame=Frame(self.home,bd=2,relief=RIDGE)
        self.C_Frame.place(x=720,y=100,width=470,height=360)
    #Table
        #Scroll bar for table to view all headings
        scroly=Scrollbar(self.C_Frame,orient=VERTICAL)
        scrolx=Scrollbar(self.C_Frame,orient=HORIZONTAL)
        # Columns and headings and adding commands for the functioning of scroll bar
        self.CourseTable=ttk.Treeview(self.C_Frame,columns=("roll","name","email","gender","dob","contact","a
dmission","course","state","city","pin","address"),xscrollcommand=scrolx.set,yscrollcommand=scroly.set)
        scrolx.pack(side=BOTTOM,fill=X)
        scroly.pack(side=RIGHT,fill=Y)
        scrolx.config(command=self.CourseTable.xview)
        scroly.config(command=self.CourseTable.yview)
        self.CourseTable.heading("roll",text="Roll No")
        self.CourseTable.heading("name",text="Name")
        self.CourseTable.heading("email",text="Email")
        self.CourseTable.heading("gender",text="Gender")
        self.CourseTable.heading("dob",text="D.O.B")
        self.CourseTable.heading("contact",text="Contact")
        self.CourseTable.heading("admission",text="Admission")
        self.CourseTable.heading("course",text="Course")
        self.CourseTable.heading("state",text="State")
        self.CourseTable.heading("city",text="City")
        self.CourseTable.heading("pin",text="PIN")
        self.CourseTable.heading("address",text="Address")
        self.CourseTable["show"]="headings"
        self.CourseTable.column("roll",width=100)
        self.CourseTable.column("name",width=100)
        self.CourseTable.column("email",width=100)
        self.CourseTable.column("gender",width=100)
        self.CourseTable.column("dob",width=100)
        self.CourseTable.column("contact",width=100)
        self.CourseTable.column("admission",width=100)
        self.CourseTable.column("course",width=100)
        self.CourseTable.column("state",width=100)
        self.CourseTable.column("city",width=100)
        self.CourseTable.column("pin",width=100)
        self.CourseTable.column("address",width=100)
        self.CourseTable.pack(fill=BOTH,expand=1)
        self.CourseTable.bind("<ButtonRelease-1>",self.get_data) #When you click on any cid row it will show
details on their sections get_data function is defined below
```

```
        self.show()  #It is help to show details in table the function is defined at the bottom
#---------------database---------------------------------
#Adding name,duration, discription and showing pop messages on pc accordint to that
    def clear(self):
        self.var_roll.set("")
        self.var_name.set("")
        self.var_email.set("")
        self.var_gender.set("Select")
        self.var_dob.set("")
        self.var_contact.set("")
        self.var_adm_date.set("")
        self.var_course.set("Select")
        self.var_state.set("")
        self.var_city.set("")
        self.var_pin.set("")
        self.address.delete("1.0",END)
        self.rollno1.config(state=NORMAL)
        self.var_search.set("")
    def delete(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            if self.var_roll.get()=="" :
                messagebox.showerror("Error","Roll No should be required ",parent=self.home)
            else:
                cur.execute("Select * from student where roll=?",(self.var_roll.get(),))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error, Select The Student From the List first",parent=self.home)
                else:
                    p=messagebox.askyesno("Confirm","Do you really want to delete",parent=self.home)
                    if p==True:
                        cur.execute("Delete from student where roll=? ",(self.var_roll.get(),))
                        conn.commit()
                        messagebox.showinfo("Delete","Student deleted Successfully",parent=self.home)
                        self.clear() #We are calling clear because we declare show in to that
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def get_data(self,event):
        self.rollno1.config(state="readonly")
        self.rollno1
        r=self.CourseTable.focus()
        content=self.CourseTable.item(r)
        row=content["values"]
        self.var_roll.set(row[0])
        self.var_name.set(row[1])
        self.var_email.set(row[2])
        self.var_gender.set(row[3])
```

```
        self.var_dob.set(row[4])
        self.var_contact.set(row[5])
        self.var_adm_date.set(row[6])
        self.var_course.set(row[7])
        self.var_state.set(row[8])
        self.var_city.set(row[9])
        self.var_pin.set(row[10])
        self.address.delete("1.0",END)
        self.address.insert(END,row[11])
# Adding Details and Saving
    def add(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            if self.var_roll.get()==""or self.var_name.get()=="" or self.var_email.get()=="" or
self.var_course=="Select":
                messagebox.showerror("Error","Roll No., Student name, Email and Course must
required",parent=self.home)
            else:
                cur.execute("Select * from student where roll=?",(self.var_roll.get(),)) #Due to tupple we added , at
last
                row=cur.fetchone()
                if row!=None:
                    messagebox.showerror("Error, Roll No. is already Present",parent=self.home)
                else:
                    cur.execute("Insert into student
(roll,name,email,gender,dob,contact,admission,course,state,city,pin,address) values(?,?,?,?,?,?,?,?,?,?,?,?)",(
                        self.var_roll.get(),
                        self.var_name.get(),
                        self.var_email.get(),
                        self.var_gender.get(),
                        self.var_dob.get(),
                        self.var_contact.get(),
                        self.var_adm_date.get(),
                        self.var_course.get(),
                        self.var_state.get(),
                        self.var_city.get(),
                        self.var_pin.get(),
                        self.address.get("1.0",END)
                    ) )
                    conn.commit()
                    messagebox.showinfo("Great","Student Added Successfully",parent=self.home)
                    self.show()
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    #Updating Details
    def update(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
```

```
        cur=conn.cursor()
        try:
            if self.var_roll.get()=="":
                messagebox.showerror("Error","Roll No should be required",parent=self.home)
            else:
                cur.execute("Select * from student where roll=?",(self.var_roll.get(),))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error","Select Student From List",parent=self.home)
                else:
                    cur.execute("Update student set
name=?,email=?,gender=?,dob=?,contact=?,admission=?,course=?,state=?,city=?,pin=?,address=? where  roll=?
",(
                        self.var_name.get(),
                        self.var_email.get(),
                        self.var_gender.get(),
                        self.var_dob.get(),
                        self.var_contact.get(),
                        self.var_adm_date.get(),
                        self.var_course.get(),
                        self.var_state.get(),
                        self.var_city.get(),
                        self.var_pin.get(),
                        self.address.get("1.0",END),
                        self.var_roll.get()
                    ) )
                    conn.commit()
                    messagebox.showinfo("Great","Student Update Successfully",parent=self.home)
                    self.show()
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def show(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            cur.execute("Select * from student")
            rows=cur.fetchall()
            self.CourseTable.delete(*self.CourseTable.get_children())
            for row in rows:
                self.CourseTable.insert(',END,values=row)
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def fetch_course(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            cur.execute("Select name from course")
            rows=cur.fetchall()
```

```
            if len(rows)>0:
                for row in rows:
                    self.course_list.append(row[0])
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def search(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            cur.execute("Select * from student where roll=?",(self.var_search.get(),))
            row=cur.fetchone()
            if row !=None:
                self.CourseTable.delete(*self.CourseTable.get_children())
                self.CourseTable.insert('',END,values=row)
            else:
                messagebox.showerror("Error","No record Found",parent=self.home)
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
if __name__=="__main__":
    home=Tk()
    obj=StudentClass(home)
    home.mainloop()
```

View Result:-

```
#View Result
from tkinter import*
from PIL import Image,ImageTk
from tkinter import ttk,messagebox
import sqlite3
class ViewClass:
    def __init__(self,home):
        self.home=home
        self.home.title("Student Result Management System")
        self.home.geometry("1200x500+80+170")
        self.home.config(bg="white")
        self.home.focus_force()
    #Title of result
        title=Label(self.home,text="View Student Results",font=("times new
roman",20,"bold"),bg="purple",fg="white").place(x=0,y=0,relwidth=1,height=50)
    #Search
        self.var_search=StringVar()
        self.var_id=""
        lbl_select = Label(self.home,text="Select By Roll No.",font=("times new
roman",20,"bold"),bg="white").place(x=280,y=100)
        txt_select = Entry(self.home,textvariable=self.var_search,font=("times new
roman",20),bg="lightyellow").place(x=520,y=100,width=150)
```

```
        btn_search=Button(self.home,text="Search",font=("times new
roman",15,"bold"),bg="lightblue",fg="black",cursor="hand2",command=self.search).place(x=680,y=100,width
=100,height=35)
        btn_clear=Button(self.home,text="Clear",font=("times new
roman",15,"bold"),bg="lightgreen",fg="black",cursor="hand2",command=self.clear).place(x=800,y=100,width
=100,height=35)
        lbl_roll = Label(self.home,text="Roll No.",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=150,y=230,width=150,height=50)
        lbl_name = Label(self.home,text="Name",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=300,y=230,width=150,height=50)
        lbl_course = Label(self.home,text="Course",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=450,y=230,width=150,height=50)
        lbl_marks = Label(self.home,text="Marks Obtained",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=600,y=230,width=150,height=50)
        lbl_full = Label(self.home,text="Total Marks",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=750,y=230,width=150,height=50)
        lbl_percentage = Label(self.home,text="Percentage",font=("times new
roman",15,"bold"),bg="white",bd=2,relief=GROOVE).place(x=900,y=230,width=150,height=50)
        self.roll = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.roll.place(x=150,y=280,width=150,height=50)
        self.name = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.name.place(x=300,y=280,width=150,height=50)
        self.course = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.course.place(x=450,y=280,width=150,height=50)
        self.marks = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.marks.place(x=600,y=280,width=150,height=50)
        self.full = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.full.place(x=750,y=280,width=150,height=50)
        self.percentage = Label(self.home,font=("times new roman",15,"bold"),bg="white",bd=2,relief=GROOVE)
        self.percentage.place(x=900,y=280,width=150,height=50)
        #Delete button
        btn_delete=Button(self.home,text="Delete",font=("times new
roman",15,"bold"),bg="red",fg="white",cursor="hand2",command=self.delete).place(x=500,y=350,width=150,
height=35)
    #------------------------------
    def search(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            if self.var_search.get()=="":
                messagebox.showerror("Error","Roll No. should be required",parent=self.home)
            else:
                cur.execute("Select * from result where roll=?",(self.var_search.get(),))
                row=cur.fetchone()
                if row !=None:
                    self.var_id=row[0]
                    self.roll.config(text=row[1])
                    self.name.config(text=row[2])
```

```
                self.course.config(text=row[3])
                self.marks.config(text=row[4])
                self.full.config(text=row[5])
                self.percentage.config(text=row[6])
            else:
                messagebox.showerror("Error","No record Found",parent=self.home)
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
    def clear(self):
        self.var_id=""
        self.roll.config(text="")
        self.name.config(text="")
        self.course.config(text="")
        self.marks.config(text="")
        self.full.config(text="")
        self.percentage.config(text="")
        self.var_search.set("")
    def delete(self):
        conn=sqlite3.connect(database="ResultManagementSystem.db")
        cur=conn.cursor()
        try:
            if self.var_id=="":
                messagebox.showerror("Error","search Student Result First",parent=self.home)
            else:
                cur.execute("Select * from result where rid=?",(self.var_id,))
                row=cur.fetchone()
                if row==None:
                    messagebox.showerror("Error","Invalid Student Result",parent=self.home)
                else:
                    p=messagebox.askyesno("Confirm","Do you really want to delete",parent=self.home)
                    if p==True:
                        cur.execute("Delete from result where rid=? ",(self.var_id,))
                        conn.commit()
                        messagebox.showinfo("Delete","Result deleted Successfully",parent=self.home)
                        self.clear() #We are calling clear because we declare show in to that
        except Exception as ex:
            messagebox.showerror("Error",f"Error due to {str(ex)}")
if __name__=="__main__":
    home=Tk()
    obj=ViewClass(home)
    home.mainloop()
```

4.3 Database Connectivity

```
#Database File
import sqlite3
def create_db():
    conn=sqlite3.connect(database="ResultManagementSystem.db")
    cur=conn.cursor()
#Table Creation for Course Page
    cur.execute("Create table if not exists course(cid INTEGER primary key AutoIncrement,name text,duration text, charges text,description text)")
    conn.commit()
#Table Creation for Student Page
    cur.execute("Create table if not exists student(roll INTEGER primary key AutoIncrement,name text,email test,gender text,dob text,contact text,admission text,course text,state text,city text,pin text,address text)")
    conn.commit()
#Table Creation for Result Page
    cur.execute("Create table if not exists result(rid INTEGER primary key AutoIncrement,roll text,name text, course text,marks_obtain text,full_marks text,percentage text)")
    conn.commit()
#Table Creation for Sign_Up Page
    cur.execute("Create table if not exists AllUsers(eid INTEGER primary key AutoIncrement,f_name text,l_namen text, contact text, email text, question text, answer text, password text,u_name text)")
    conn.commit()

    conn.close()

create_db()
```

5.EXISTING SOFTWARES

There are several Student Result Management software solutions currently in use, designed to help correctional facilities manage information, Marks,Results, security, and administrative tasks. Here are some examples:

- **Blackboard:-**

A comprehensive Learning Management System (LMS) facilitating course management, assessment, and grading, offering educators tools for tracking student performance.

  - **Use Case:-**Facilitates course management, assessment, and grading for educators to track student performance efficiently.

- **Canvas:-**

An open-source LMS with features for grading, course organization, and communication, providing educators and students a user-friendly platform for managing academic progress.

  - Use Case:-Enables educators and students to manage academic progress through grading, course organization, and communication features.

- **Moodle:-**

A widely-used LMS supporting online learning, offering features for creating quizzes, assignments, and tracking student progress through gradebooks and analytics.

  - **Use Case:-**Supports online learning by offering tools for creating quizzes, assignments, and tracking student progress via gradebooks and analytics.

- **PowerSchool:-**

A student information system (SIS) with modules for grading, attendance, and scheduling, providing educators comprehensive tools for managing student data and academic progress.

  - **Use Case:-** Provides comprehensive tools for educators to manage student data, including grading, attendance, and scheduling functionalities.

- **Google Classroom:-**

A free LMS integrated with G Suite for Education, offering educators tools for creating assignments, grading, and communication, providing a streamlined platform for digital learning.

  - **Use Case:-**Offers educators a streamlined platform within G Suite for Education for creating assignments, grading, and communication with students, enhancing digital learning experiences.

6.NEW ADDITION

SRMS (Student Result Management Systems) provides modern educational tools empower educators with deep result analysis, seamless communication, and robust data security. These systems promote personalized learning experiences, foster collaboration, and ensure compliance, driving continuous improvement and student success in today's digital educational landscape.

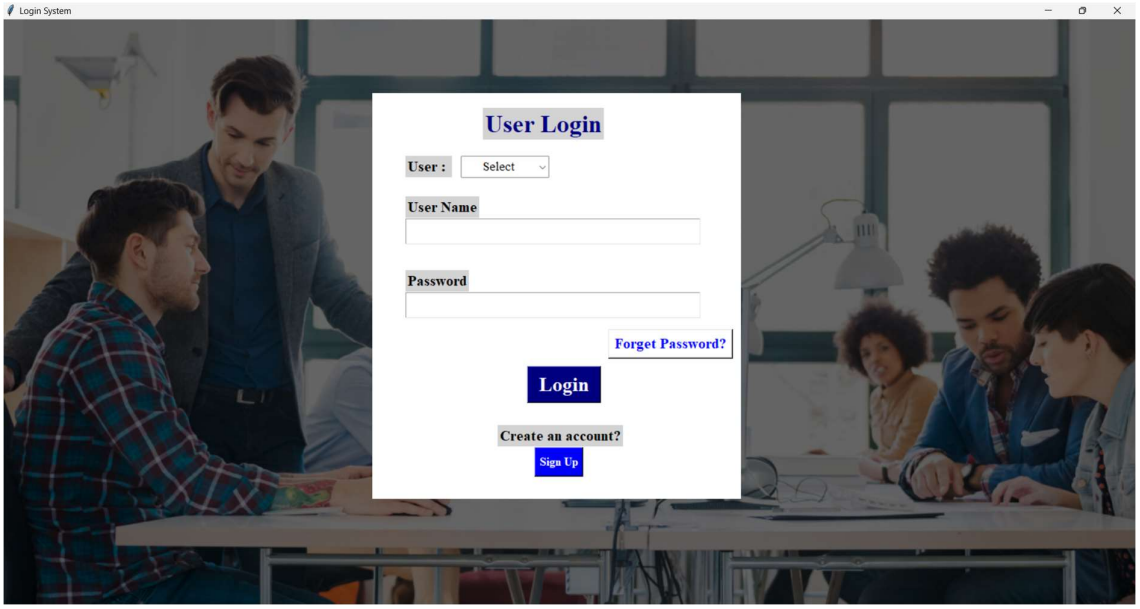
- **Enhanced Result Analysis Tools:** Implementing advanced analytics features to provide insights into student performance trends, helping educators identify areas for improvement and personalize learning experiences effectively.

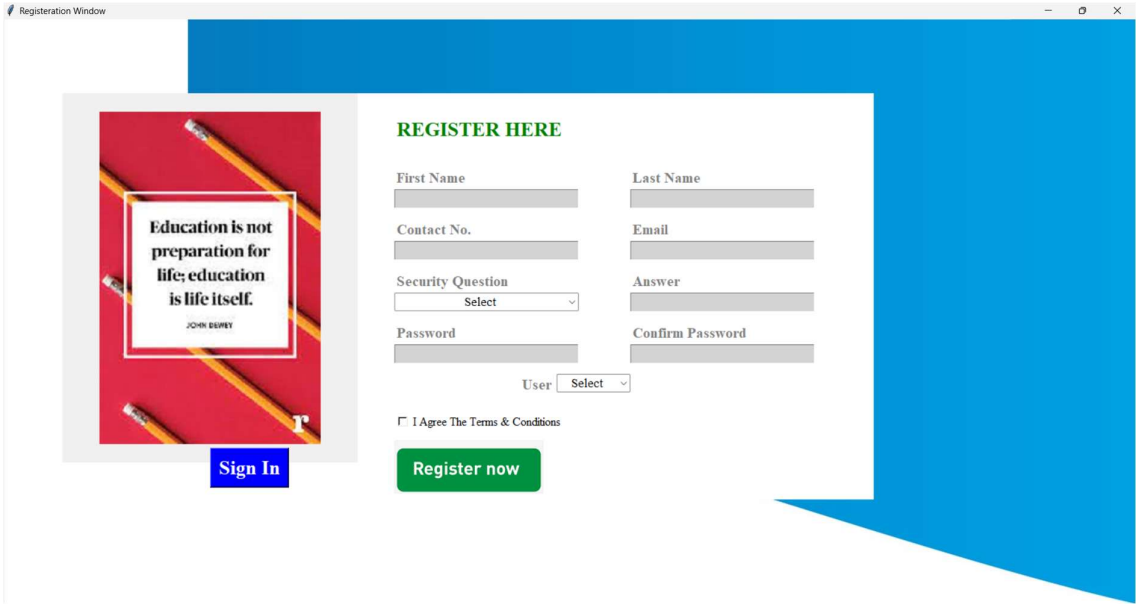
- **Integrated Communication Platform:** Developing a seamless communication system within the student result management software, facilitating collaboration between educators, students, and parents to foster a supportive learning environment.

- **Data Security and Privacy Measures:** Implementing robust data security protocols and privacy measures to ensure the confidentiality and integrity of student information, adhering to regulatory compliance standards such as GDPR and FERPA..

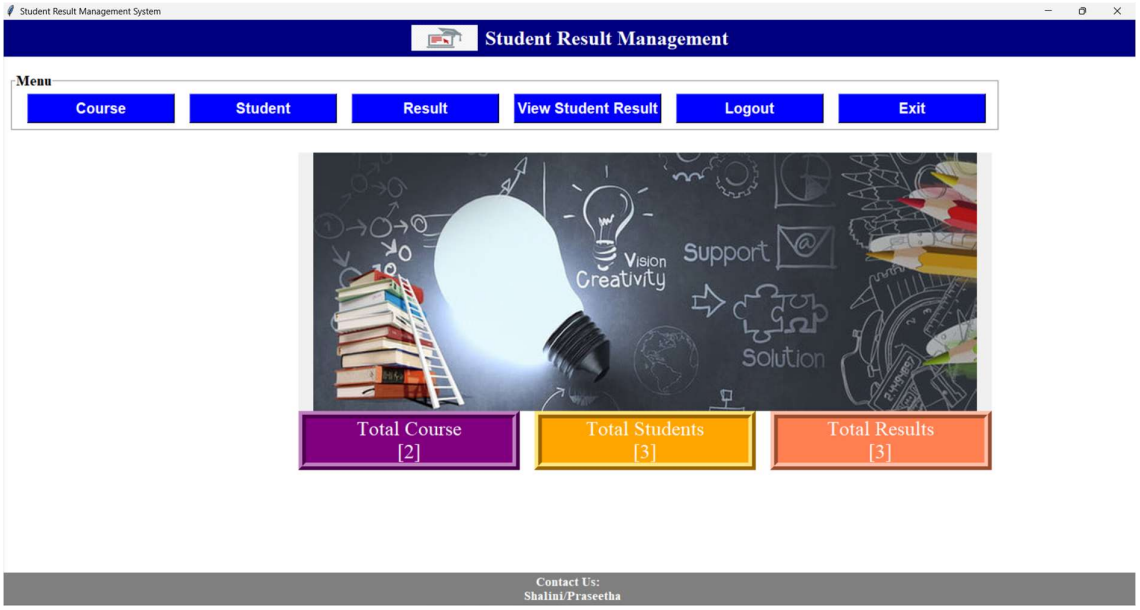
7. RESULTS AND DISCUSSION

LOGIN/REGISTER INTERFACE

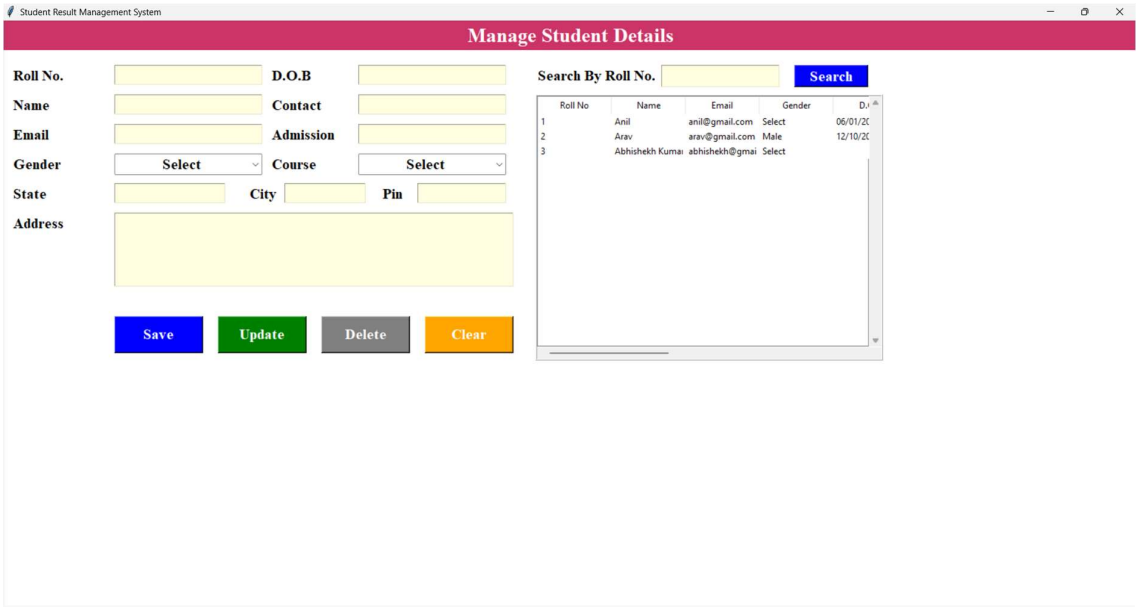




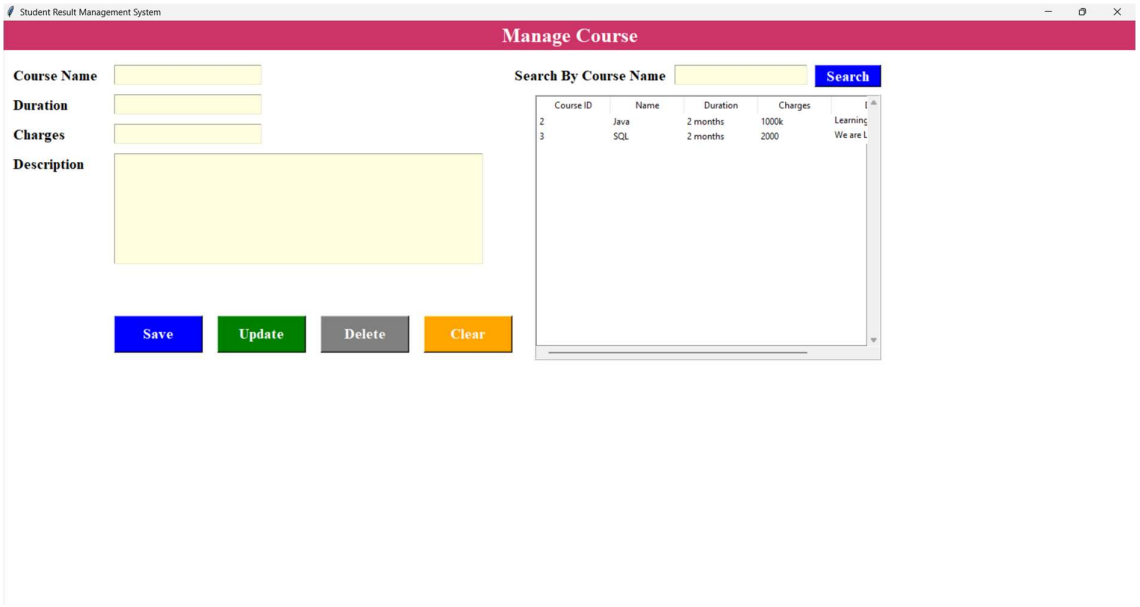
DASHBOARD INTERFACE FOR ADMIN



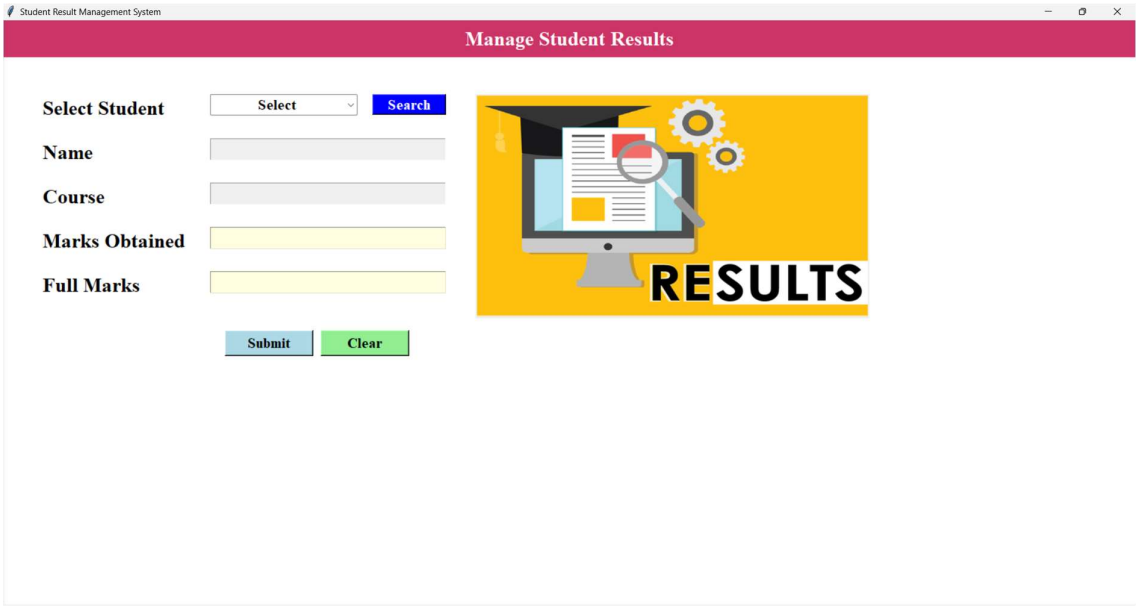
INTERFACE FOR MANAGING STUDENT DETAILS



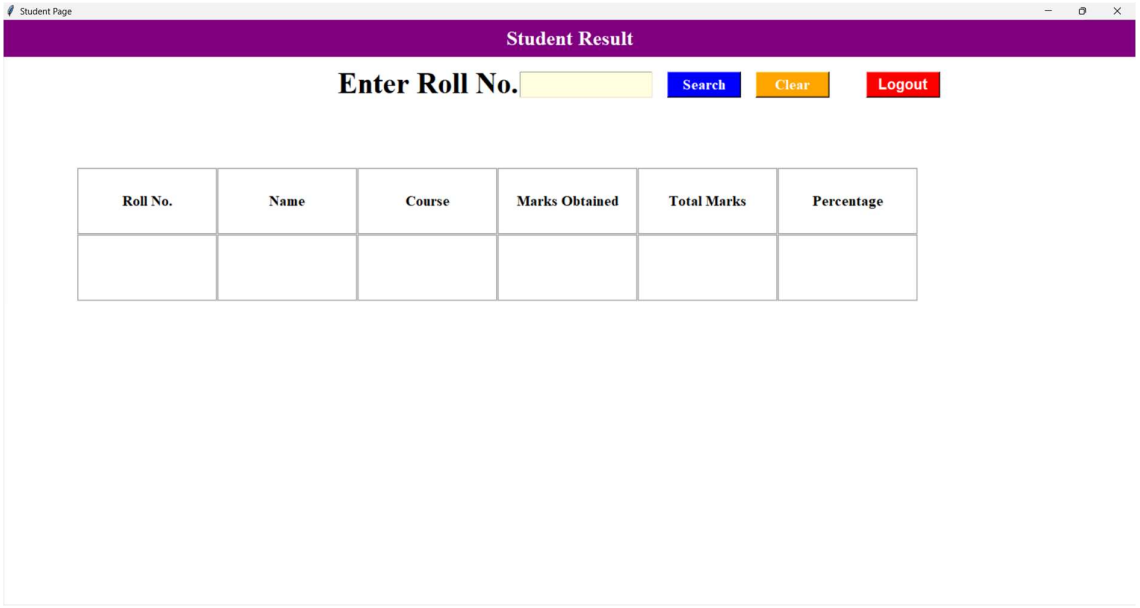
INTERFACE FOR MANAGING COURSE DETAILS



INTERFACE FOR UPDATING STUDENT RESULTS



INTERFACE FOR STUDENTS TO VIEW RESULT



8.CONCLUSION

In conclusion, Student Result Management Systems (SRMS) are instrumental in modern education, facilitating efficient data management, communication, and academic success. Through sophisticated result analysis tools and seamless communication platforms, SRMS empower educators to personalize learning experiences, identify trends, and foster collaboration among stakeholders. Moreover, stringent data security measures ensure the protection of sensitive student information, maintaining compliance with regulatory standards. As educational institutions continue to embrace digital innovation, SRMS serve as catalysts for positive change, driving continuous improvement and preparing students for success in a rapidly evolving global landscape. In essence, SRMS represent a transformative force in education, shaping the future of teaching and learning.

9.REFERENCE

Below is a list of references and resources used during the development of the Student Result Management System Project. These include documentation, libraries, frameworks, and tools that were essential for building and testing the application.

- **Python Documentation:**

  Official Python Documentation

  https://docs.python.org/3/

- **TKInter:**

  Python Software Foundation. (n.d.). TKInter documentation.

  https://docs.python.org/3/library/tkinter.html

- **PIL (Python Imaging Library):**

  PythonWare. (n.d.). Pillow (PIL Fork) documentation.

  https://pillow.readthedocs.io/en/stable/index.html

- **OS Module:**

  Python Software Foundation. (n.d.). OS documentation.

  https://docs.python.org/3/library/os.html

- **SQLite:**

  SQLite. (n.d.). SQLite documentation.

  https://www.sqlite.org/docs.html

These references provide comprehensive documentation and resources for learning and using each of these libraries and modules in Python programming.