**ITM Student management system USING mongodb**

**A Project Report submitted in the fulfill of the requirement**

**for the degree of**

**INFORMATIONAL TECHNOLOGY AND MANAGEMENT**

**UNDER**

**UTKAL UNIVERSITY,**

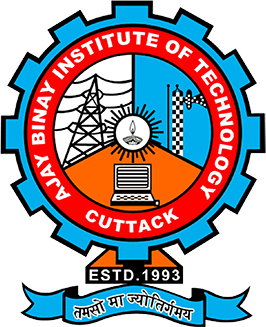
**BHUBANESWAR, ODISHA**

**Submitted by: Guided by:**

**Sanjib Sahu**

**& Dr. Amaresh Sahu**

**Deepak Kumar Das**



**AJAY BINAYA DEGREE COLLEGE**

**CUTTACK, ODISHA**

**ABSTRACT**

1. **Introduction:**

The ITM Student Management System is a desktop application developed using Python and MongoDB. It is designed to manage the student details in a university or educational institution. The application provides a user-friendly interface to add, delete, and update student information. The system is built with the Tkinter library, which provides a simple way to create graphical user interfaces.

This project is a simple implemented in Python using the Tkinter GUI toolkit and the MongoDB NoSQL database. The main purpose of the application is to manage student records, such as their name, email, batch, and mobile number. Additionally, the application provides the ability to add, delete, and update student records, as well as to upload and display student images.

1. **Prerequisites:**

Before running the application, you need to make sure that the following requirements are installed on your system:

* Install Python 3.10 on your machine if not already installed.
* Install the necessary Python libraries and packages. The required packages are pymongo, gridfs, pillow, and tkinter.
* Install MongoDB and start the MongoDB server.
* PyMongo driver for MongoDB
* The GUI of the application will open up, and you can use the features of the application.

1. **Installation:**

To run this project on your computer, you need to follow the steps below:

1. **Install Python:**

Go to <https://www.python.org/downloads/> and download the latest version of Python that suits your operating system.

1. **Install MongoDB:**

Go to <https://www.mongodb.com/try/download/community> and download the latest version of MongoDB that suits your operating system.

1. **Install the required Python libraries:**

Open the terminal and run the following commands to install the required

Python libraries:

**pip install pymongo**

**pip install gridfs**

**pip install pillow**

1. **Download the project:**

You can download the project from GitHub using the following link: (<https://github.com/vivekyadav44/Student-Management-System>.

1. **Run the project:**

To run the project, open the terminal and navigate to the project directory. Then, run the following command: **app.py**

1. **Technology Stack & Architecture:**

The ITM Student Management System is built using the following architecture:

1. **Python Programming Language:**

The core programming language used for the development of this project is Python. It provides the necessary libraries and modules for interacting with the database and developing the user interface.

1. **Tkinter (tk) Module:**

Tkinter is a standard GUI (Graphical User Interface) library for Python. It provides the necessary tools for creating user interfaces, such as buttons, labels, text boxes, etc.

1. **MongoDB Database:**

MongoDB is a document-oriented NoSQL database used for storing and retrieving data in JSON-like documents. It is a scalable and flexible database that provides high performance and availability.

1. **Database:**

The database used in this project is MongoDB. It contains a collection called **"students"** that stores the information of all the students. The collection has the following fields:

1. **REGNO:** This field stores the student's registration number.
2. **NAME:** This field stores the student's name.
3. **EMAIL**: This field stores the student's email.
4. **BATCH:** This field stores the student's batch.
5. **MOBILE:** This field stores the student's mobile number.
6. **IMAGE:** This field stores the student's image.
7. **Technical Details:**

The application is implemented in Python using the Tkinter GUI toolkit and the PyMongo driver for MongoDB. The application consists of a single Python script **app.py**, which contains the main application logic and user interface.

The script connects to a MongoDB database using the PyMongo driver and creates a new database called **Assignment08** with a single collection called **students**. The student’s collection stores student records, which are represented as Python dictionaries with the following fields:

* **REGNO:** registration number (string)
* **NAME:** student name (string)
* **EMAIL:** student email (string)
* **BATCH:** student batch (string)
* **MOBILE:** student mobile number (string)
* **IMAGE:** student image (binary)

The application provides a simple user interface implemented using the Tkinter GUI toolkit. The user interface consists of several windows, each of which corresponds to a different functionality of the application, such as adding, deleting, and updating student records.

The user interface allows the user to enter student details using text fields and select a student image using a file browser. The selected image is converted to binary format and stored in the IMAGE field of the student record before being inserted into the database.

When a user selects the "View Students" option, the application retrieves all student records from the student’s collection and displays them in a new window using a scrollable list.

1. **User Interface:**

The user interface of this project is developed using the **Tkinter** library. It consists of the following components:

1. **Main Window:**

This is the main window of the application. It contains buttons to perform various operations such as adding, deleting, updating, and viewing student information.

1. **Add Student Window:**

This window is used to add a new student to the database. It contains fields for the student's registration number, name, email, batch, and mobile number. It also contains a browse button to select an image file for the student. Once the fields are filled, the user can click the submit button to add the student to the database.

1. **Delete Student Window:**

This window is used to delete an existing student from the database. It contains a field for the student's registration number. Once the registration number is entered, the user can click the delete button to delete the student from the database.

1. **Update Student Window:**

This window is used to update an existing student's information in the database. It contains fields for the student's registration number, name, email, batch, and mobile number. Once the fields are filled, the user can click the submit button to update the student's information in the database.

1. **View Student Window:**

This window is used to view the list of students in the database. It contains a table that displays the students' registration numbers, names, emails, batches, mobile numbers, and images.

1. **Functionality:**

To run the application, simply execute the main script **app.py** using the Python interpreter. The script will open a new window displaying the main application screen, which provides the following options:

1. **Add Student:**

The user can add a new student to the database by providing the student's details such as registration number, name, email, batch, mobile number, and an optional image of the student. The system generates a random registration number for the student and stores the student's information in the database.

1. **View Students:**

The user can view a list of all the students in the database. The system displays the student's registration number, name, email, batch, and mobile number in a tabular format. The user can select a student from the list and view their details.

1. **Update Student:**

The user can update the details of a student in the database by providing their registration number. The system retrieves the student's information from the database and displays it in the interface. The user can then update the details and save the changes to the database.

1. **Delete Student:**

The user can delete a student from the database by providing their registration number. The system checks if the student exists in the database and deletes their record if it is found.

1. **Exit**:

Closes the application.

The ITM Student Management System also provides basic validation for user inputs. For example, when adding a new student, the application ensures that all required fields are filled and that the registration number is unique. When updating a student's details, the application checks that the registration number exists in the database before allowing changes to be made.

Overall, the ITM Student Management System is a user-friendly application that allows you to manage and maintain student records easily. Its intuitive interface and basic validation features make it an ideal solution for small schools, colleges, or other educational institutions.

1. **Project Structure:**

The project consists of only one files **app.py:**

The code for the **app.py** file is as follows:

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import unicode\_literals

from pymongo import MongoClient

import pymongo

from bson import Binary

from random import randint

from tkinter import \*

from tkinter import messagebox

import gridfs

from tkinter import Button

import sys

from PIL import Image, ImageTk

from tkinter import filedialog

import tkinter as tk

from io import BytesIO

import base64

import io

from tkinter.filedialog import askopenfilename

try:

    client = MongoClient(port=27017)

    db=client.Assignment08

    print("Connected to MongoDB")

except :

    print("Database connection Error ")

    print("No connection could be made because the target machine actively refused it ")

    messagebox.showerror("Error", "Connection Error")

    sys.exit(1)

root=tk.Tk()

root.geometry('400x350')

root.title("Student Management System")

def add\_STUDENTS(root,db):

    def add\_query():

        global root

        regno = E1.get()

        name = E2.get()

        email = E3.get()

        batch = E4.get()

        mobile = E5.get()

        REGNO = [regno]

        NAME = [name]

        EMAIL = [email]

        BATCH = [batch]

        MOBILE = [mobile]

        # check if an image file was selected

        if img\_file\_path != '':

            # read the contents of the image file

            with open(img\_file\_path, 'rb') as f:

                img\_data = f.read()

        Assignment08 = {

        'REGNO' : REGNO[randint(0, (len(REGNO)-1))] ,

        'NAME' : NAME[randint(0, (len(NAME)-1))],

        'EMAIL' : EMAIL[randint(0, (len(EMAIL)-1))],

        'BATCH' : BATCH[randint(0, (len(BATCH)-1))],

        'MOBILE' : MOBILE[randint(0, (len(MOBILE)-1))],

        'IMAGE' : Binary(img\_data)}

        if len(regno) == 0 or len(name) == 0 or len(email) == 0 or len(batch) == 0:

            messagebox.showwarning("WARNING", "All fields are compulsory (except mobile number)")

            return

        if db.students.count\_documents({'REGNO': regno}, limit=1) > 0:

            messagebox.showwarning("ERROR", "Student already exists")

            return

        result = db.students.insert\_one(Assignment08)

        newwin.destroy()

        messagebox.showinfo("Add Student", "Student added successfully")

    def browse\_img\_file():

        global img\_file\_path

        img\_file\_path = askopenfilename()

        img\_file\_label.config(text=img\_file\_path)

    newwin = Toplevel(root)

    newwin.geometry('400x500')

    newwin.title("Add Students")

    L1 = Label(newwin, text="REGNO")

    L1.place(x=10, y=50)

    E1 = Entry(newwin, bd=7)

    E1.place(x=100, y=50)

    L2 = Label(newwin, text="NAME")

    L2.place(x=10, y=100)

    E2 = Entry(newwin, bd=7)

    E2.place(x=100, y=100)

    L3 = Label(newwin, text="EMAIL")

    L3.place(x=10, y=150)

    E3 = Entry(newwin, bd=7)

    E3.place(x=100, y=150)

    L4 = Label(newwin, text="BATCH")

    L4.place(x=10, y=200)

    E4 = Entry(newwin, bd=7)

    E4.place(x=100, y=200)

    L5 = Label(newwin, text="MOBILE")

    L5.place(x=10, y=250)

    E5 = Entry(newwin, bd=7)

    E5.place(x=100, y=250)

    # add a label and button for browsing image files

    img\_file\_label = Label(newwin, text="No image file selected")

    img\_file\_label.place(x=10, y=300)

    browse\_button = Button(newwin, text="Browse", command=browse\_img\_file)

    browse\_button.place(x=10, y=330)

    sub=Button(newwin,text="Submit",command=add\_query)

    sub.place(x=120,y=350)

def del\_data(root,db):

    def delete():

        global root

        regno = E1.get()

        if(len(regno)==0):

            messagebox.showwarning("WARNING", "Enter a Valid REG.NO")

            return

        if db.students.count\_documents({ 'REGNO': regno }, limit = 1)==0:

            messagebox.showwarning("ERROR", "STUDENT Does Not Exist")

            return

        else:

            db.students.delete\_one({'REGNO':regno})

        newwin.destroy()

        messagebox.showinfo("Delete Student", "Student Deleted")

    newwin=Toplevel(root)

    newwin.geometry('400x350')

    newwin.title("Delete STUDENT")

    L1 = Label(newwin, text="REGNO")

    L1.place(x=10, y=50)

    E1 = Entry(newwin,bd=5)

    E1.place(x=100, y=50)

    sub = Button(newwin, text="Delete Entry", command=delete)

    sub.place(x=120, y=200)

def update\_data(root,db):

    def UPDD():

        global root

        regno = E6.get()

        name = E7.get()

        email = E8.get()

        batch = E9.get()

        mobile = E10.get()

        if(len(regno)==0):

            messagebox.showwarning("WARNING", "Enter a Valid REG.NO")

            return

        if db.students.count\_documents({ 'REGNO': regno }, limit = 1)==0:

            messagebox.showwarning("ERROR", "STUDENT Does Not Exist")

            return

        if(len(name)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'NAME' : name}})

        if(len(email)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'EMAIL' : email}})

        if(len(batch)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'BATCH' : batch}})

        if(len(mobile)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'MOBILE' : mobile}})

        # check if an image file was selected

        if img\_file\_path != '':

            # read the contents of the image file

            with open(img\_file\_path, 'rb') as f:

                img\_data = f.read()

            # update the image field for the student with the new image data

            db.students.update\_one({"REGNO":regno},{"$set": {'IMAGE' : Binary(img\_data)}})

        newwin.destroy()

        messagebox.showinfo("Update Student", "Student Updated")

    def browse\_img\_file():

        global img\_file\_path

        img\_file\_path = askopenfilename()

        img\_file\_label.config(text=img\_file\_path)

    newwin = Toplevel(root)

    newwin.geometry('400x500')

    newwin.title("Update STUDENTS")

    L6 = Label(newwin, text="REGNO")

    L6.place(x=10,y=50)

    E6 = Entry(newwin, bd=7)

    E6.place(x=100,y=50)

    L7 = Label(newwin, text="NAME")

    L7.place(x=10,y=100)

    E7 = Entry(newwin, bd=7)

    E7.place(x=100,y=100)

    L8 = Label(newwin, text="EMAIL")

    L8.place(x=10,y=150)

    E8 = Entry(newwin, bd=7)

    E8.place(x=100,y=150)

    L9 = Label(newwin, text="BATCH")

    L9.place(x=10,y=200)

    E9 = Entry(newwin, bd=7)

    E9.place(x=100,y=200)

    L10 = Label(newwin, text="MOBILE")

    L10.place(x=10,y=250)

    E10 = Entry(newwin, bd=7)

    E10.place(x=100,y=250)

    # add a label and button for browsing image files

    img\_file\_label = Label(newwin, text="No image file selected")

    img\_file\_label.place(x=10, y=300)

    browse\_button = Button(newwin, text="Browse", command=browse\_img\_file)

    browse\_button.place(x=10, y=330)

    sub=Button(newwin,text="Submit",command=UPDD)

    sub.place(x=120,y=350)

def display(root,db):

    newwin=Toplevel(root)

    newwin.geometry('600x600')

    newwin.title("STUDENT Details")

    L1=Label(newwin,text="REGNO")

    L1.grid(row=0,column=0)

    L2 = Label(newwin, text="NAME")

    L2.grid(row=0, column=2)

    L3=Label(newwin,text="EMAIL")

    L3.grid(row=0,column=4)

    L4=Label(newwin,text="BATCH")

    L4.grid(row=0,column=6)

    L5=Label(newwin,text="PHOTO")

    L5.grid(row=0,column=8)

    L6=Label(newwin,text="MOBILE")

    L6.grid(row=0,column=10)

    i=1

    students=db.students.find().sort("regno", pymongo.ASCENDING)

    for x in students:

        photo\_path = x['IMAGE']

        photo=Image.open(io.BytesIO(photo\_path))

        photo = photo.resize((50, 50), resample=Image.LANCZOS)

        photo = ImageTk.PhotoImage(photo)

        L1 = Label(newwin, text=x['REGNO'])

        L1.grid(row=i, column=0)

        L2 = Label(newwin, text=x['NAME'])

        L2.grid(row=i, column=2)

        L3 = Label(newwin, text=x['EMAIL'])

        L3.grid(row=i, column=4)

        L4 = Label(newwin, text=x['BATCH'])

        L4.grid(row=i, column=6)

        L5 = Label(newwin, image=photo)

        L5.image = photo

        L5.grid(row=i, column=8)

        if 'MOBILE' in x:

            L6 = Label(newwin, text=x['MOBILE'])

            L6.grid(row=i, column=10)

        i+=1

add= Button(root,text='Add New STUDENTS',command=lambda:add\_STUDENTS(root,db))

delete= Button(root,text='Delete STUDENTS Entry',command=lambda:del\_data(root,db))

update= Button(root,text='Update STUDENTS Info',command=lambda:update\_data(root,db))

show= Button(root,text='Show STUDENTS Details',command=lambda:display(root,db))

add.place(x=100,y=100)

delete.place(x=100,y=150)

update.place(x=100,y=200)

show.place(x=100,y=250)

root.mainloop()

The above code is a Python script for a basic GUI application that allows the user to add, delete and update student data in a MongoDB database. The application uses the Tkinter library for creating the GUI.

The script first imports the necessary libraries including pymongo, tkinter, PIL, and io. It then establishes a connection to a MongoDB server running on the local machine on port 27017, and selects the "Assignment08" database. The user interface for the application is created using the Tkinter library.

The script defines three main functions for adding, deleting and updating student data in the database. Each of these functions creates a new window that allows the user to input the required data.

The add\_STUDENTS() function allows the user to enter data for a new student, including REGNO, NAME, EMAIL, BATCH, MOBILE, and an optional IMAGE field. If the user selects an image file, the function reads the contents of the file and adds it to the database as a Binary field. The function first checks if all the required fields are entered, and then checks if the student already exists in the database before adding the new data. If the data is successfully added to the database, a message is displayed to the user indicating that the student has been added.

The del\_data() function allows the user to delete a student from the database. The function prompts the user to enter the REGNO of the student to be deleted. If the student exists in the database, it is deleted, and a message is displayed to the user indicating that the student has been deleted.

The update\_data() function allows the user to update the data for an existing student. The function prompts the user to enter the REGNO of the student whose data is to be updated, and then allows the user to enter the new values for the student's fields. If the student exists in the database, its data is updated with the new values entered by the user.

The script also defines helper functions such as browse\_img\_file() that allows the user to browse for an image file to be added to the database, and resize\_image() that resizes the image to fit the dimensions of the GUI window.

Overall, the script provides a basic user interface for managing student data in a MongoDB database using Python and Tkinter.

1. **Code explanation:**

The code begins with importing necessary libraries such as pymongo, Binary, randint, tkinter, gridfs, and others that will be used to interact with MongoDB database, load and display images, handle GUI events, and read files.

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import unicode\_literals

from pymongo import MongoClient

import pymongo

from bson import Binary

from random import randint

from tkinter import \*

from tkinter import messagebox

import gridfs

from tkinter import Button

import sys

from PIL import Image, ImageTk

from tkinter import filedialog

import tkinter as tk

from io import BytesIO

import base64

import io

from tkinter.filedialog import askopenfilename

The code sets up a connection with the MongoDB database using the MongoClient class. If the connection is successful, it prints a message indicating that the connection is successful. If there is an error connecting to the database, it prints an error message and exits the program with a return value of 1.

try:

    client = MongoClient(port=27017)

    db=client.Assignment08

    print("Connected to MongoDB")

except :

    print("Database connection Error ")

    print("No connection could be made because the target machine actively refused it ")

    messagebox.showerror("Error", "Connection Error")

    sys.exit(1)

The code creates a new tkinter object named root which will be the main window of the GUI application. The geometry function sets the width and height of the window, and the title function sets the title of the window.

root=tk.Tk()

root.geometry('400x350')

root.title("Student Management System")

The code defines a function named add\_STUDENTS which takes two arguments, root and db. This function is responsible for creating a new window to add new students to the database.

def add\_STUDENTS(root,db):

    def add\_query():

        global root

        regno = E1.get()

        name = E2.get()

        email = E3.get()

        batch = E4.get()

        mobile = E5.get()

        REGNO = [regno]

        NAME = [name]

        EMAIL = [email]

        BATCH = [batch]

        MOBILE = [mobile]

        # check if an image file was selected

        if img\_file\_path != '':

            # read the contents of the image file

            with open(img\_file\_path, 'rb') as f:

                img\_data = f.read()

        Assignment08 = {

        'REGNO' : REGNO[randint(0, (len(REGNO)-1))] ,

        'NAME' : NAME[randint(0, (len(NAME)-1))],

        'EMAIL' : EMAIL[randint(0, (len(EMAIL)-1))],

        'BATCH' : BATCH[randint(0, (len(BATCH)-1))],

        'MOBILE' : MOBILE[randint(0, (len(MOBILE)-1))],

        'IMAGE' : Binary(img\_data)}

        if len(regno) == 0 or len(name) == 0 or len(email) == 0 or len(batch) == 0:

            messagebox.showwarning("WARNING", "All fields are compulsory (except mobile number)")

            return

        if db.students.count\_documents({'REGNO': regno}, limit=1) > 0:

            messagebox.showwarning("ERROR", "Student already exists")

            return

        result = db.students.insert\_one(Assignment08)

        newwin.destroy()

        messagebox.showinfo("Add Student", "Student added successfully")

    def browse\_img\_file():

        global img\_file\_path

        img\_file\_path = askopenfilename()

        img\_file\_label.config(text=img\_file\_path)

    newwin = Toplevel(root)

    newwin.geometry('400x500')

    newwin.title("Add Students")

    L1 = Label(newwin, text="REGNO")

    L1.place(x=10, y=50)

    E1 = Entry(newwin, bd=7)

    E1.place(x=100, y=50)

    L2 = Label(newwin, text="NAME")

    L2.place(x=10, y=100)

    E2 = Entry(newwin, bd=7)

    E2.place(x=100, y=100)

    L3 = Label(newwin, text="EMAIL")

    L3.place(x=10, y=150)

    E3 = Entry(newwin, bd=7)

    E3.place(x=100, y=150)

    L4 = Label(newwin, text="BATCH")

    L4.place(x=10, y=200)

    E4 = Entry(newwin, bd=7)

    E4.place(x=100, y=200)

    L5 = Label(newwin, text="MOBILE")

    L5.place(x=10, y=250)

    E5 = Entry(newwin, bd=7)

    E5.place(x=100, y=250)

    # add a label and button for browsing image files

    img\_file\_label = Label(newwin, text="No image file selected")

    img\_file\_label.place(x=10, y=300)

    browse\_button = Button(newwin, text="Browse", command=browse\_img\_file)

    browse\_button.place(x=10, y=330)

    sub=Button(newwin,text="Submit",command=add\_query)

    sub.place(x=120,y=350)

The code defines a function named del\_data which takes two arguments, root and db. This function is responsible for creating a new window to delete a student from the database.

def del\_data(root,db):

    def delete():

        global root

        regno = E1.get()

        if(len(regno)==0):

            messagebox.showwarning("WARNING", "Enter a Valid REG.NO")

            return

        if db.students.count\_documents({ 'REGNO': regno }, limit = 1)==0:

            messagebox.showwarning("ERROR", "STUDENT Does Not Exist")

            return

        else:

            db.students.delete\_one({'REGNO':regno})

        newwin.destroy()

        messagebox.showinfo("Delete Student", "Student Deleted")

    newwin=Toplevel(root)

    newwin.geometry('400x350')

    newwin.title("Delete STUDENT")

    L1 = Label(newwin, text="REGNO")

    L1.place(x=10, y=50)

    E1 = Entry(newwin,bd=5)

    E1.place(x=100, y=50)

    sub = Button(newwin, text="Delete Entry", command=delete)

    sub.place(x=120, y=200)

The code defines a function named update\_data which takes two arguments, root and db. This function is responsible for creating a new window to update the details of a student in the database.

def update\_data(root,db):

    def UPDD():

        global root

        regno = E6.get()

        name = E7.get()

        email = E8.get()

        batch = E9.get()

        mobile = E10.get()

        if(len(regno)==0):

            messagebox.showwarning("WARNING", "Enter a Valid REG.NO")

            return

        if db.students.count\_documents({ 'REGNO': regno }, limit = 1)==0:

            messagebox.showwarning("ERROR", "STUDENT Does Not Exist")

            return

        if(len(name)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'NAME' : name}})

        if(len(email)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'EMAIL' : email}})

        if(len(batch)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'BATCH' : batch}})

        if(len(mobile)!=0):

            db.students.update\_one({"REGNO":regno},{"$set": {'MOBILE' : mobile}})

        # check if an image file was selected

        if img\_file\_path != '':

            # read the contents of the image file

            with open(img\_file\_path, 'rb') as f:

                img\_data = f.read()

            # update the image field for the student with the new image data

            db.students.update\_one({"REGNO":regno},{"$set": {'IMAGE' : Binary(img\_data)}})

        newwin.destroy()

        messagebox.showinfo("Update Student", "Student Updated")

    def browse\_img\_file():

        global img\_file\_path

        img\_file\_path = askopenfilename()

        img\_file\_label.config(text=img\_file\_path)

    newwin = Toplevel(root)

    newwin.geometry('400x500')

    newwin.title("Update STUDENTS")

    L6 = Label(newwin, text="REGNO")

    L6.place(x=10,y=50)

    E6 = Entry(newwin, bd=7)

    E6.place(x=100,y=50)

    L7 = Label(newwin, text="NAME")

    L7.place(x=10,y=100)

    E7 = Entry(newwin, bd=7)

    E7.place(x=100,y=100)

    L8 = Label(newwin, text="EMAIL")

    L8.place(x=10,y=150)

    E8 = Entry(newwin, bd=7)

    E8.place(x=100,y=150)

    L9 = Label(newwin, text="BATCH")

    L9.place(x=10,y=200)

    E9 = Entry(newwin, bd=7)

    E9.place(x=100,y=200)

    L10 = Label(newwin, text="MOBILE")

    L10.place(x=10,y=250)

    E10 = Entry(newwin, bd=7)

    E10.place(x=100,y=250)

    # add a label and button for browsing image files

    img\_file\_label = Label(newwin, text="No image file selected")

    img\_file\_label.place(x=10, y=300)

    browse\_button = Button(newwin, text="Browse", command=browse\_img\_file)

    browse\_button.place(x=10, y=330)

    sub=Button(newwin,text="Submit",command=UPDD)

    sub.place(x=120,y=350)

The code defines a function named display which takes two arguments, root and db. This function is responsible for creating a new window to view all the students in the database.

def display(root,db):

    newwin=Toplevel(root)

    newwin.geometry('600x600')

    newwin.title("STUDENT Details")

    L1=Label(newwin,text="REGNO")

    L1.grid(row=0,column=0)

    L2 = Label(newwin, text="NAME")

    L2.grid(row=0, column=2)

    L3=Label(newwin,text="EMAIL")

    L3.grid(row=0,column=4)

    L4=Label(newwin,text="BATCH")

    L4.grid(row=0,column=6)

    L5=Label(newwin,text="PHOTO")

    L5.grid(row=0,column=8)

    L6=Label(newwin,text="MOBILE")

    L6.grid(row=0,column=10)

    i=1

    students=db.students.find().sort("regno", pymongo.ASCENDING)

    for x in students:

        photo\_path = x['IMAGE']

        photo=Image.open(io.BytesIO(photo\_path))

        photo = photo.resize((50, 50), resample=Image.LANCZOS)

        photo = ImageTk.PhotoImage(photo)

        L1 = Label(newwin, text=x['REGNO'])

        L1.grid(row=i, column=0)

        L2 = Label(newwin, text=x['NAME'])

        L2.grid(row=i, column=2)

        L3 = Label(newwin, text=x['EMAIL'])

        L3.grid(row=i, column=4)

        L4 = Label(newwin, text=x['BATCH'])

        L4.grid(row=i, column=6)

        L5 = Label(newwin, image=photo)

        L5.image = photo

        L5.grid(row=i, column=8)

        if 'MOBILE' in x:

            L6 = Label(newwin, text=x['MOBILE'])

            L6.grid(row=i, column=10)

        i+=1

The code defines a function named browse\_img\_file which is called when the "Browse" button is clicked in the "Add Students" window. This function is responsible for opening a file dialog boxto allow the user to select an image file.

def browse\_img\_file():

        global img\_file\_path

        img\_file\_path = askopenfilename()

        img\_file\_label.config(text=img\_file\_path)

The file dialog box is created using the filedialog module of the tkinter package, which provides a set of dialogs for selecting files or directories. In this case, the filedialog.askopenfilename method is used to display a dialog that allows the user to select a single image file.

Once the user selects an image file, the full path of the selected file is returned by the askopenfilename method, and this path is then set as the value of the img\_path variable using the set method of the StringVar class.

The img\_preview Label widget is then updated with the selected image by using the PIL (Python Imaging Library) to open the image file and resize it to fit within the boundaries of the Label widget. The ImageTk module is then used to convert the resized image to a format that can be displayed in the Label widget.

Finally, the img\_path variable is returned from the function so that it can be used by other parts of the program to access the selected image file.

That's a great summary! Just to add a bit more detail on the PIL and ImageTk modules:

The Python Imaging Library (PIL) is a library for opening, manipulating, and saving many different images file formats. In this code, PIL is used to open the selected image file and resize it to fit within the boundaries of the Label widget.

The ImageTk module is a module in PIL that provides a way to convert PIL images to a format that can be displayed in a tkinter Label widget. The PhotoImage class of the ImageTk module is used to create a PhotoImage object from the resized PIL image, which is then set as the value of the img\_preview Label widget using the configure method.

Overall, the browse\_img\_file function is a useful function that provides a way for the user to select an image file and preview it in the program.

1. **Future Scope:**

This project is a simple Student Management System developed using Python and MongoDB. It helps educational institutions to manage their students' information easily. This project can be useful for teachers or administrators who want to manage the student's information efficiently. The project can be further enhanced by adding more features such as attendance management, fee management, and result management.

Finally, the application can be enhanced by adding an option to export the list of students in the database to a CSV file. This can be useful for keeping a backup of the database or for analysing the data using other tools. To export the data, click on the "Export Data" button in the main window and select a location to save the file.

1. **Conclusion:**

The ITM Student Management System is a simple yet effective desktop application designed to manage student information in a university or educational institution. The application provides a user-friendly interface that allows users to add, delete, update, view, and search student information. The application is built using Python and MongoDB, and the Tkinter library is used to create the graphical user interface.