PSD Project

Studentify App

Akhil Binoy
Joel Varughese Thomas

Contents

1. Preface 2
2. Introduction 2
3. User requirements definition 2
   3.1. User requirements 2
   3.2. Non-functional requirements 2
4. System Architecture 3
5. System requirements specification 4
   5.1. System Requirements 4
   5.2. Functional Requirements 5
6. System models 5
7. System Evolution 8
8. Appendices 8

1. Preface

This is the first version of Studentify app developed for students of Hochschule Rhein Waal, Kleve to socialize among their peers. This is the initial iteration of software and more modifications and improvements can be made based on the feedbacks of students which can be given to student representative.

2. Introduction

The application is designed to assist prospective, newly admitted, and already admitted students who want to socialize, make friends, and join interesting activities within the university area.

This software will be installed on a single computer where students can come to browse, add, or join events. Every month, all student organizations will update their information accordingly, ensuring that students have access to the latest happenings on campus.

3. User requirements definition

3.1. User requirements

1. The software shall have the login feature for the user to provide security and privacy to the personal data.
2. The software shall include distinct sections or separations within the user interface to clearly differentiate between various options like add events, view events, forum post, inquiries etc.
3. The software shall allow the user to add their own events.
4. The software shall give alerts or notifications when user responds to an event or a reply to an inquiry is received.
5. The software shall facilitate a platform for students to post queries or concerns, with provision for other students to comment and engage in discussion.
6. The software shall feature a dedicated functionality enabling students to submit inquiries requiring personal attention to the student representative, regarding educational concerns.
7. The software shall allow users to respond to events and to view the list of people responded.

3.2. Non-functional requirements

1. The software shall be developed using Python programming concepts.
2. The software shall utilize Python's GUI capabilities, specifically leveraging libraries such as Tkinter, to provide a user-friendly front-end interface.
3. The software shall run in Windows OS.

4. Only registered user shall login into the app.
5. All the tabs are clearly visible to the user such that user don't need to find the hidden features in the software.
6. The personal details of the users shall only be visible to oneself

4. System Architecture

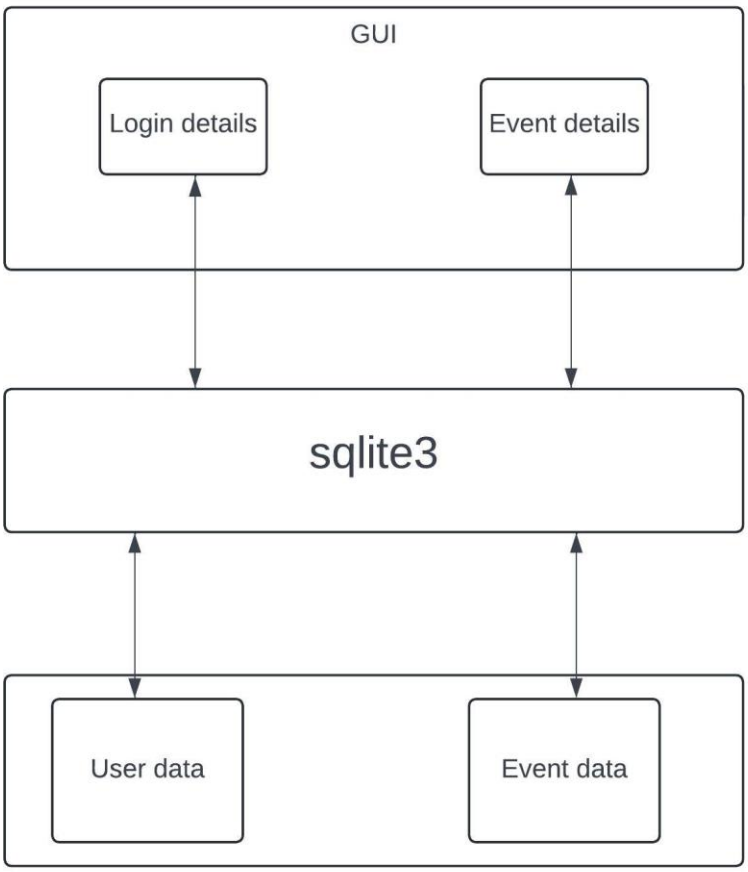

Figure 1: System Architecture

The figure illustrates the system architecture of the studentify app. The data base is set up using sqlite3 used in Python. The login data of the users and the event data for the events added are stored and retrieved from a local database in our computer. When people respond to events, it is retrieved from this database and even notification is setup which we will see better further in the document.

5. System requirements specification

5.1. System Requirements

1.1. The software shall provision create account feature for the new users(including student organisation) using their full name, enrollment id, email and password.

1.2. The software shall provide login feature for known user accounts from successfully created details.

1.3. The software shall have two account types: student and student representative, which is selected.

1.3. The software shall provide a incorrect warning incase of incorrect student ID or incorrect password.

1.4. The software shall have the logout feature.

2.1. The software shall show 5 sections: View Profile, notification, Add/view Event, post/view open queries and inquiries.

2.2. Each section shall be in differenct buttons.

3.1. The events shall have a event name, location, date, time and details.

4.1. The software shall give in-app notification to host when someone responds to event with yes, no or may be.

4.2. The software shall give notification when a student receives reply from their student representative.

4.3. The notification should be cleared once it is read.

5.1. The forum shall have fields to add title and to type the post.

5.2. The software shall give option to comment and shall view all comments.

5.3. If the comments contain any improper words, the software shall find it and give a warning about it and terminte the process.

6.1. The submitted inquiries shall be catogorized based on thir nature, for instance examination, re-registration, studies, transcripts and documents, workshops, co-curricular activities, Sports etc.

7.1. The software shall give an option to respond to the events like yes/no/maybe

7.2. The software shall show response as drop down.

5.2. Functional Requirements

1. The user shall be able to input the login details such as login id and password for the access to the software.
2. The sofware shall allow the user to post on the forum or react to any information posted.
3. Different tabs/sections of the GUI shall interact based on the selection of user.
4. The address/location of the events or program shall be visible to everyone in the group.

6. System models

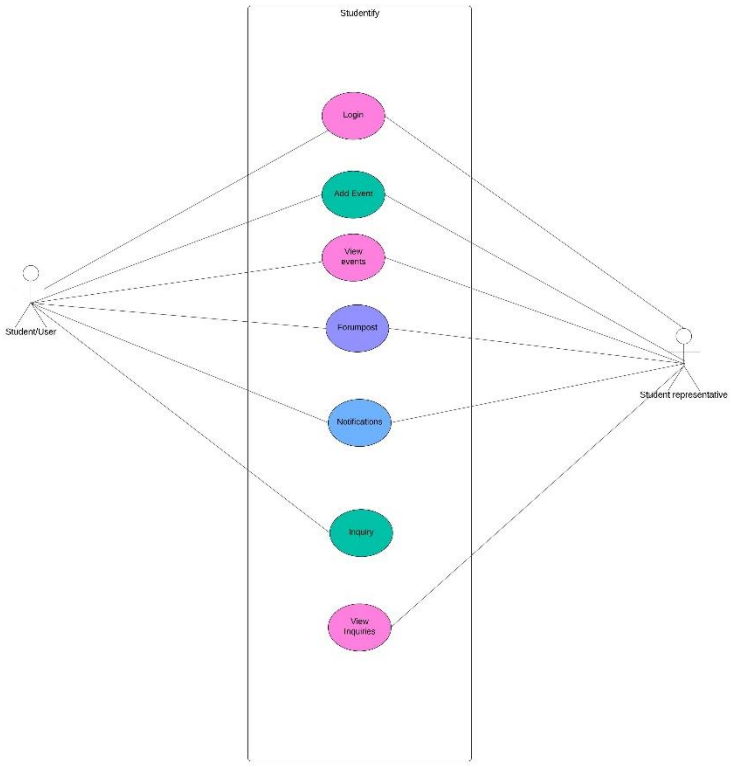

Figure 2: Use Case diagram

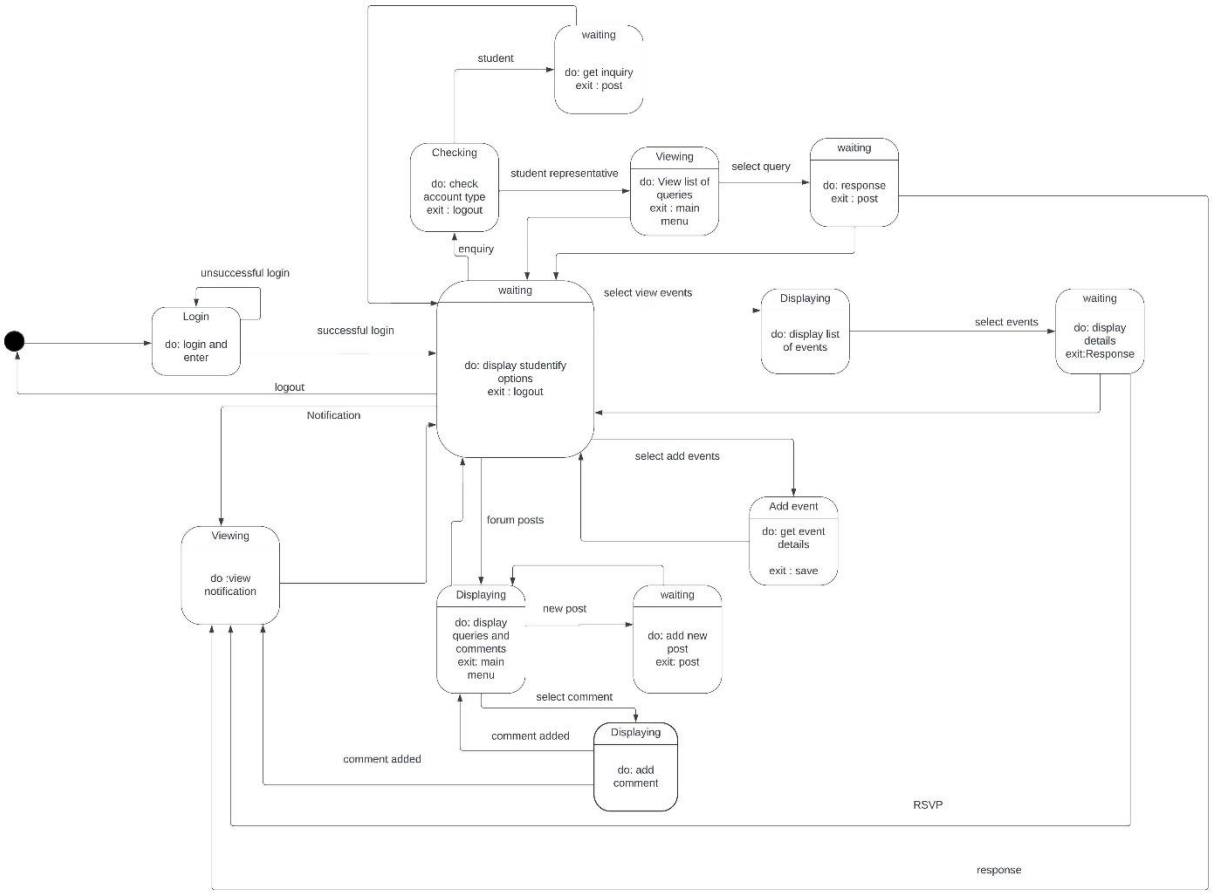

Figure 3: State diagram

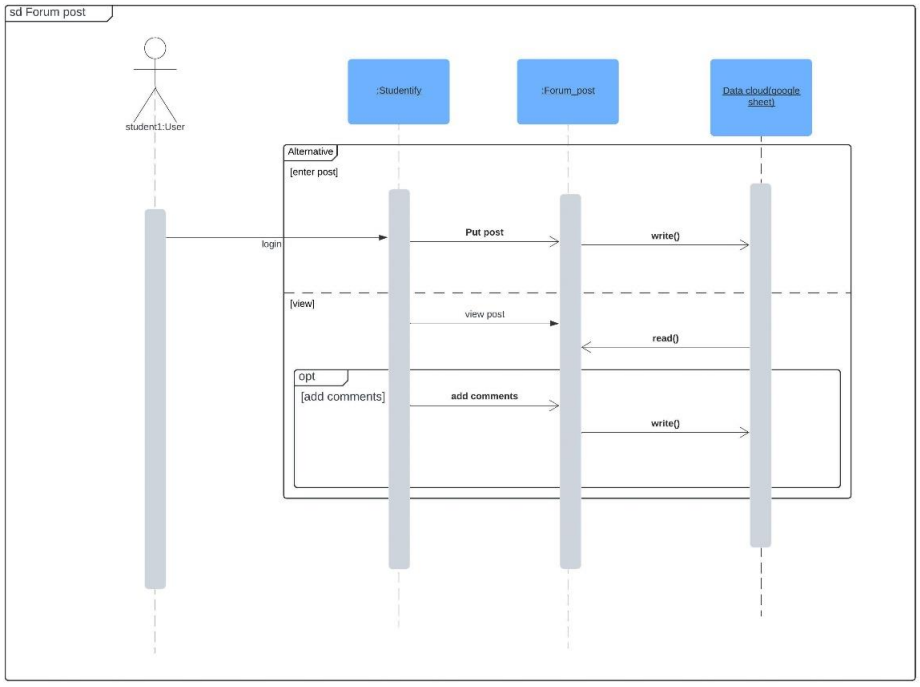

Figure 4: Sequence diagram for forum post

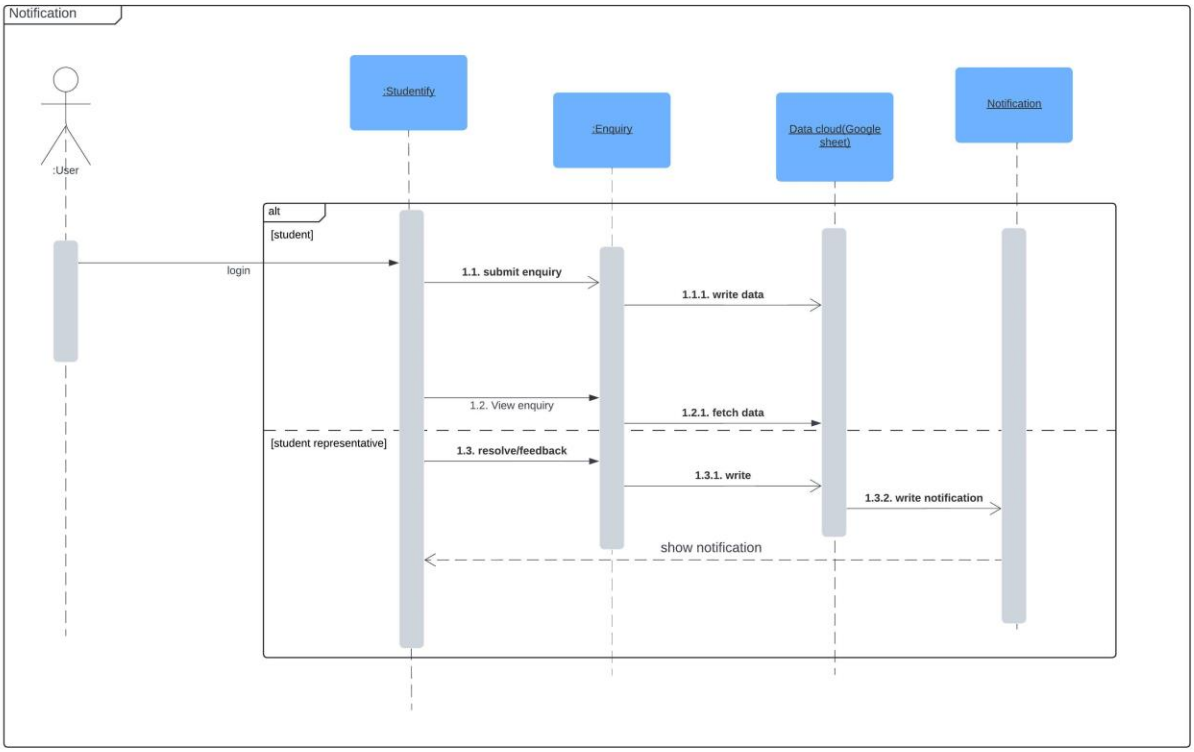

Figure 5: sequence diagram for notification

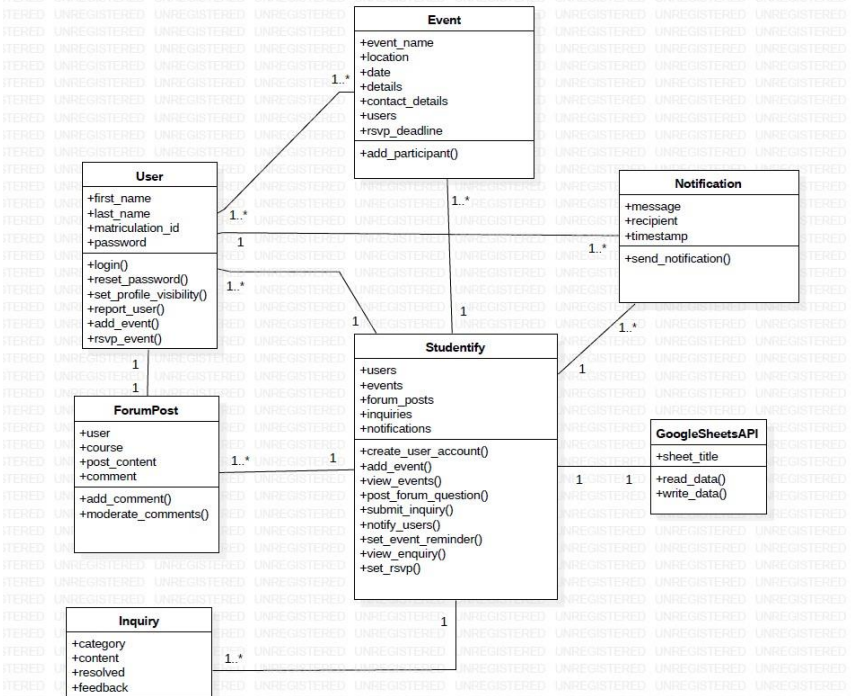

Figure 6: Class diagram

7. System Evolution

- For better load handling, flexible scaling and for remote access, Cloud database services could be used.
- Feature expansion: Developing a mobile version of Studentify to make it accessible on smartphones and tablets, increasing convenience and accessibility for users.
- Event Management enhancements: Introducing features like event reminders and integration with calendar applications.
- Multilingual support to cater to a diverse user base starting with implementing German.
- AI integration: Perhaps integration of a chatbot.

8. Appendices

#main.py

```
from singelton_Homescreen import Singelton_Homescreen

application = Singelton_Homescreen()
application.home_screen.mainloop()
```

singelton_Homescreen

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

from user import User
from data_base_connect import LoginDatabase

from singelton_register_screen import Singelton_Register_Screen
from singelton_studentify_screen import Singelton_Studentify_Screen

from GUI import IGui
```

```
class Singelton_Homescreen(IGui):
    __instance = None

    def __new__(cls):
        if (cls.__instance==None):
            cls.__instance = super(Singelton_Homescreen,cls).__new__(cls)
            cls.__instance.create_widget()
        return cls.__instance

    def create_widget(self):
        ## create tkinter object and set properties
        self.home_screen = Tk()
        self.home_screen.title("STUDENTIFY")
        self.home_screen.geometry("1200x750")
        self.home_screen.resizable(False, False)

        ## set backround ##
        self.bg_image =
ImageTk.PhotoImage(Image.open('DEVELOPMENT/background.jpg'))
        self.bg_label = Label(self.home_screen, image=self.bg_image)
        self.bg_label.image = self.bg_image
        self.bg_label.pack()

        # Username label and entry
        self.enrollment_label = Label(self.home_screen, font=("times new roman",
20), text='Enrollment ID')
        self.enrollment_label.place(x=550, y=250, width=250, height=40)
        self.enrollment_id: str = Entry(self.home_screen, font=("times new
roman", 20), bg="lightgray")
        self.enrollment_id.place(x=800, y=250, width=250, height=40)

        # Password label and entry
        self.password_label = Label(self.home_screen, font=("times new roman",
20), text='Password')
        self.password_label.place(x=550, y=350, width=250, height=40)
        self.password:str = Entry(self.home_screen, font=("times new roman", 20),
bg="lightgray")
        self.password.config(show='*')
        self.password.place(x=800, y=350, width=250, height=40)

        # Login button
        self.login_btn = Button(self.home_screen, text="Login",
command=self.login, font=("times new roman", 15),
                                bg="lightblue", bd=2, width=12, height=2)
        self.login_btn.place(x=850, y=450)
```

```
        # Register button
        self.register_btn = Button(self.home_screen,command=self.register_here,
text="Register Here",
                                   font=("times new roman", 15), bg="lightblue", bd=2,
width=12, height=2)
        self.register_btn.place(x=850, y=550)


    def get_enrollment_id(self):
        return self.enrollment_id.get()

    def get_password(self):
        return self.password.get()

    def create_user(self):
        user = User(self.get_enrollment_id(), self.get_password())
        return user

    def check_empty_feilds(self):
        user=self.create_user()
        if not user.get_enrollment_id() or not user.get_userpassw():
            messagebox.showerror("Error","All feilds are required.")
            return True
        else:
            return False

    def check_valid_user(self):
        row = LoginDatabase.is_existing_user(self.create_user())

        if row is None:
            messagebox.showerror("Error", "Invalid enrollment id and password")
            return False
        else:
            return True

    def login(self):
        try:
            if not self.check_empty_feilds() and self.check_valid_user():

                row = LoginDatabase.is_existing_user(self.create_user())
                user_account_type = row[2]
                name = row[0]
                enrollment_id= row[1]
                Singelton_Homescreen().home_screen.destroy()
```

```
                Singelton_Studentify_Screen().set_user_label(name,user_account_ty
pe,enrollment_id)


        except Exception as e:
            messagebox.showerror("Error", str(e))

    def register_here(self):
        Singelton_Homescreen().home_screen.destroy()
        Singelton_Register_Screen()
```

#user.py

```
from tkinter import *
from tkinter import messagebox
from data_base_connect import InquiryDatabase, NotificationDatabase,
LoginDatabase

class Singleton_ViewInquiriesScreen:
    __instance = None

    def __new__(cls, root):
        if cls.__instance is None:
            cls.__instance = super(Singleton_ViewInquiriesScreen,
cls).__new__(cls)
            cls.__instance.create_widget(root)
        return cls.__instance

    def create_widget(self, root):
        self.view_inquiries_window = Toplevel(root)
        self.view_inquiries_window.title("View Inquiries")
        self.view_inquiries_window.geometry("1200x750")
        self.view_inquiries_window.resizable(True, True)

        # Inquiries Listbox
        self.inquiries_listbox = Listbox(self.view_inquiries_window, font=("times
new roman", 12), width=100, height=15)
        self.inquiries_listbox.place(x=50, y=50, width=400, height=600)
```

```
        self.inquiries_listbox.bind("<Double-Button-1>",
self.show_inquiry_details)

        # Inquiry Details
        self.details_label = Label(self.view_inquiries_window, text="",
font=("times new roman", 14), wraplength=750, anchor="w")
        self.details_label.place(x=500, y=50, width=650, height=400)

        # Reply Section
        self.reply_label = Label(self.view_inquiries_window, text="Reply:",
font=("times new roman", 15))
        self.reply_label.place(x=500, y=460)

        self.reply_text = Text(self.view_inquiries_window, font=("times new
roman", 15), height=8, width=60)
        self.reply_text.place(x=500, y=490)

        self.submit_reply_btn = Button(self.view_inquiries_window, text="Submit
Reply", font=("times new roman", 15), command=self.submit_reply)
        self.submit_reply_btn.place(x=650, y=650)

        # Load Inquiries
        self.load_inquiries()

        # Handle window close
        self.view_inquiries_window.protocol("WM_DELETE_WINDOW", self.on_close)

    def load_inquiries(self):
        inquiries = InquiryDatabase.get_inquiries()
        self.inquiries_listbox.delete(0, END)  # Clear any existing items
        for inquiry in inquiries:
            # Adjust the unpacking logic to match the number of columns in the
database
            inquiry_id, name, enrollment_id, email, department, inquiry_type,
subject, details, status = inquiry
            self.inquiries_listbox.insert(END, f"{subject} - {name}")

    def show_inquiry_details(self, event):
        selected_index = self.inquiries_listbox.curselection()
        if selected_index:
            subject_name = self.inquiries_listbox.get(selected_index)
            inquiries = InquiryDatabase.get_inquiries()
            for inquiry in inquiries:
                # Adjust the unpacking logic to match the number of columns in
the database
```

```
                inquiry_id, name, enrollment_id, email, department, inquiry_type,
subject, details, status = inquiry
                if f"{subject} - {name}" == subject_name:
                    self.details_label.config(text=f"From: {name}
({email})\nDepartment: {department}\nType: {inquiry_type}\nDetails: {details}")
                    self.current_inquiry_id = inquiry_id
                    self.current_inquirer_id = enrollment_id  # Track who asked
the inquiry
                    break

    def submit_reply(self):
        reply_content = self.reply_text.get("1.0", END).strip()
        if reply_content:

            current_user =
LoginDatabase.get_user_details(self.current_inquirer_id)
            print(f"current_user is {self.current_inquirer_id}")
            if current_user:

                InquiryDatabase.add_reply(self.current_inquiry_id,
current_user['enrollment_id'], reply_content)

                NotificationDatabase.add_notification(current_user['enrollment_id
'], f"Your inquiry has been replied: {reply_content}")

                messagebox.showinfo("Success", "Reply submitted and notification
sent successfully.")
                self.reply_text.delete("1.0", END)
                self.load_inquiries()

    def on_close(self):
        Singleton_ViewInquiriesScreen.__instance = None
        self.view_inquiries_window.destroy()
```

GUI.py

```
from abc import ABC, abstractmethod

class IGui(ABC):

    @abstractmethod
    def create_widget():
```

```
        #create tkinter objetc
        #addign required attributes
        pass
```

#singelton_register_screen

```
from tkinter import *
from tkinter import messagebox, ttk
from PIL import Image, ImageTk

from GUI import IGui
from user import NewUser
from data_base_connect import LoginDatabase
from singelton_studentify_screen import Singelton_Studentify_Screen




class Singelton_Register_Screen(IGui):
    __instance = None

    def __new__(cls):
        if (cls.__instance==None):
            cls.__instance = super(Singelton_Register_Screen,cls).__new__(cls)
            cls.__instance.create_widget()
        return cls.__instance


    def create_widget(self):
        ## create tkinter object and set properties
        self.register_screen = Tk()
        self.register_screen.title("Registration Window")
        self.register_screen.geometry("1200x750")
        self.register_screen.resizable(False, False)

        # Background image
        self.bgimage =
ImageTk.PhotoImage(Image.open('DEVELOPMENT/background.jpg'))
        self.bgimage_label = Label(self.register_screen,image=self.bgimage)
        self.bgimage_label.image = self.bgimage
        self.bgimage_label.pack()
```

```
        # Enrollment
        self.enrollment_id_label = Label(self.register_screen,font=("times new
roman", 20), text='Enrollment ID')
        self.enrollment_id_label.place(x=550, y=150, width=250, height=40)
        self.enrollment_id = Entry(self.register_screen,font=("times new roman",
20), bg="lightgray")
        self.enrollment_id.place(x=800, y=150, width=250, height=40)

        #  Name
        self.name_label = Label(self.register_screen,font=("times new roman",
20), text='Full Name')
        self.name_label.place(x=550, y=200, width=250, height=40)
        self.name = Entry(self.register_screen,font=("times new roman", 20),
bg="lightgray")
        self.name.place(x=800, y=200, width=250, height=40)

        # Account type in drop down
        self.account_type_label = Label(self.register_screen, font=("times new
roman", 20), text='Account Type')
        self.account_type_label.place(x=550, y=250, width=250, height=40)
        # Create a Combobox widget instead of Entry
        self.account_type = ttk.Combobox(self.register_screen, font=("times new
roman", 20), values=["Student", "Student Representative"], state="readonly")
        self.account_type.place(x=800, y=250, width=250, height=40)
        #set default as student
        self.account_type.set("Student")

        # Email
        self.email_label = Label(self.register_screen,font=("times new roman",
20), text='Email')
        self.email_label.place(x=550, y=300, width=250, height=40)
        self.email = Entry(self.register_screen,font=("times new roman", 20),
bg="lightgray")
        self.email.place(x=800, y=300, width=250, height=40)

        # Password
        self.password_label = Label(self.register_screen,font=("times new roman",
20), text='Password')
        self.password_label.place(x=550, y=350, width=250, height=40)
        self.password = Entry(self.register_screen,font=("times new roman", 20),
bg="lightgray")
        self.password.place(x=800, y=350, width=250, height=40)

        # Confirm Password
```

```
        self.conpassword_label = Label(self.register_screen,font=("times new
roman", 20), text='Confirm Password')
        self.conpassword_label.place(x=550, y=400, width=250, height=40)
        self.conpassword = Entry(self.register_screen,font=("times new roman",
20), bg="lightgray")
        self.conpassword.place(x=800, y=400, width=250, height=40)

        # Register button
        self.register_btn =
Button(self.register_screen,text="Register",command=self.register,
                                    font=("times new roman", 15), bg="lightblue",
bd=2, width=12, height=2)
        self.register_btn.place(x=850, y=550)

    ## define getters
    def get_enrollment_id(self):
        return self.enrollment_id.get()
    def get_password(self):
        return self.password.get()
    def get_name(self):
        return self.name.get()
    def get_account_type(self):
        return self.account_type.get()
    def get_user_email(self):
        return self.email.get()
    def get_conpassword(self):
        return self.conpassword.get()

    ## create a new user from entry
    def create_new_user(self):

        new_user = NewUser(
            self.get_enrollment_id(),
            self.get_name(),
            self.get_account_type(),
            self.get_user_email(),
            self.get_password(),
            self.get_conpassword()
        )
        return new_user

    ## check empty fields
    def check_empty_fields(self):

        if any(value=="" for value in \
```

```
                self.create_new_user().get_user_data().values()):

            messagebox.showerror("Error", "All fields are required.")
            return True
        else:
            return False
    ## check matching password
    def password_matched(self):
        if self.create_new_user().get_userpassw() != \
            self.create_new_user().get_confirmed_passw():
            messagebox.showerror("Error", "Passwords don't match.")
            return False
        else:
            return True

    ## check for existing enrollment_id
    def enrollment_id_exists(self):
        current_user=  self.create_new_user()
        if LoginDatabase.is_existing_user(current_user):
            messagebox.showerror("Error", "User already exists. Please try
another username.")
            return True
        else:
            return False


    ## register new user to database:
    def register(self):
        try:
            if not self.check_empty_fields() and self.password_matched() and not
self.enrollment_id_exists():

                LoginDatabase.upload_data(self.create_new_user())
                messagebox.showinfo("OK","You are registered.")
                name = self.get_name()
                user_account_type = self.get_account_type()
                enrollment_id=self.get_enrollment_id()
                self.register_screen.destroy()
                Singelton_Studentify_Screen().set_user_label(name,user_account_ty
pe,enrollment_id)


        except Exception as e:
            messagebox.showerror("Error", str(e))
```

#data_base_connect.py

```
import sqlite3
import tensorflow as tf
import numpy as np

from user import User
from user import NewUser


class LoginDatabase:

    @staticmethod
    def _get_connection():
        connection = sqlite3.connect("DEVELOPMENT/database/login_database.db")
        cursor=connection.cursor()
        cursor.execute("CREATE TABLE IF NOT EXISTS user(name VARCHAR,
enrollment_id VARCHAR PRIMARY KEY, account_type VARCHAR, email VARCHAR, password
VARCHAR);")
        return connection, cursor




    @staticmethod
    def is_existing_user(current_user:User):

        enrollment_id = current_user.get_enrollment_id()
        user_passw = current_user.get_userpassw()

        conn, cursor = LoginDatabase._get_connection()
        cursor.execute("SELECT * FROM user WHERE enrollment_id=? AND
password=?",(enrollment_id,user_passw))
        row = cursor.fetchone()
        conn.close()
        return row

    @staticmethod
    def get_user_details(enrollment_id):
        conn, cursor = LoginDatabase._get_connection()
        cursor.execute("SELECT enrollment_id, name, email FROM user WHERE
enrollment_id = ?", (enrollment_id,))
        user = cursor.fetchone()
        conn.close()
```

```
        if user:
            return {"enrollment_id": user[0], "name": user[1], "email": user[2]}
        return None


    @staticmethod
    def upload_data(new_user:NewUser):
        dict=new_user.get_user_data()
        conn, cursor = LoginDatabase._get_connection()
        cursor.execute("INSERT INTO user (name,enrollment_id, account_type,
email, password)VALUES (?,?,?,?,?)",
                                    (dict["enrollment_id"], dict["name"],
dict["account_type"], dict["email"], dict["password"]))

        conn.commit()
        conn.close()


class EventDatabase:

    @staticmethod
    def _get_connection():
        connection = sqlite3.connect("DEVELOPMENT/database/event_database.db")
        cursor = connection.cursor()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS event(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                location VARCHAR,
                event_name VARCHAR,
                date VARCHAR,
                time VARCHAR,
                about TEXT,
                host VARCHAR,
                host_id VARCHAR
            );
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS rsvp(
                event_id INTEGER,
                username VARCHAR,
                rsvp_event VARCHAR,
                FOREIGN KEY(event_id) REFERENCES event(id)
            );
        """)
```

```
        return connection, cursor

    @staticmethod
    def add_event(location, event_name, date, time, about, host,host_id):
        conn, cursor = EventDatabase._get_connection()
        cursor.execute("INSERT INTO event (location, event_name, date, time,
about, host, host_id) VALUES (?, ?, ?, ?, ?, ?,?)",
                       (location, event_name, date, time, about, host, host_id))
        conn.commit()
        conn.close()

    @staticmethod
    def get_events():
        conn, cursor = EventDatabase._get_connection()
        cursor.execute("SELECT * FROM event")
        events = cursor.fetchall()
        conn.close()
        return events

    @staticmethod
    def add_rsvp(event_id, username,rsvp_event):
        conn, cursor = EventDatabase._get_connection()
        cursor.execute("INSERT INTO rsvp (event_id, username, rsvp_event) VALUES
(?, ?, ?)", (event_id, username,rsvp_event))
        conn.commit()
        conn.close()

    @staticmethod
    def get_event_rsvps(event_id):
        conn, cursor = EventDatabase._get_connection()
        cursor.execute("SELECT username,rsvp_event FROM rsvp WHERE event_id=?",
(event_id,))
        rsvps = cursor.fetchall()
        conn.close()
        return [rsvp for rsvp in rsvps]

    @staticmethod
    def get_event_host(event_id):
        conn, cursor = EventDatabase._get_connection()
        cursor.execute("SELECT host_id FROM event WHERE id=?", (event_id,))
        host = cursor.fetchone()
        conn.close()
        return host[0] if host else None
```

```
class InquiryDatabase:

    @staticmethod
    def _get_connection():
        connection = sqlite3.connect("DEVELOPMENT/database/inquiry_database.db")
        cursor = connection.cursor()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS inquiry(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name VARCHAR,
                enrollment_id VARCHAR,
                email VARCHAR,
                department VARCHAR,
                inquiry_type VARCHAR,
                subject VARCHAR,
                details TEXT,
                status VARCHAR DEFAULT 'Open'
            );
        """)
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS reply(
                inquiry_id INTEGER,
                replier_id VARCHAR,
                reply TEXT,
                FOREIGN KEY(inquiry_id) REFERENCES inquiry(id)
            );
        """)
        return connection, cursor

    @staticmethod
    def add_inquiry(name, enrollment_id, email, department, inquiry_type,
subject, details):
        conn, cursor = InquiryDatabase._get_connection()
        cursor.execute("INSERT INTO inquiry (name, enrollment_id, email,
department, inquiry_type, subject, details) VALUES (?, ?, ?, ?, ?, ?, ?)",
                       (name, enrollment_id, email, department, inquiry_type,
subject, details))
        conn.commit()
        conn.close()

    @staticmethod
    def get_inquiries():
        conn, cursor = InquiryDatabase._get_connection()
        cursor.execute("SELECT * FROM inquiry WHERE status = 'Open'")
        inquiries = cursor.fetchall()
```

```
        conn.close()
        return inquiries

    @staticmethod
    def add_reply(inquiry_id, replier_id, reply):
        conn, cursor = InquiryDatabase._get_connection()
        cursor.execute("INSERT INTO reply (inquiry_id, replier_id, reply) VALUES
(?, ?, ?)", (inquiry_id, replier_id, reply))
        cursor.execute("UPDATE inquiry SET status = 'Closed' WHERE id = ?",
(inquiry_id,))
        conn.commit()
        conn.close()

    @staticmethod
    def get_replies(inquiry_id):
        conn, cursor = InquiryDatabase._get_connection()
        cursor.execute("SELECT reply FROM reply WHERE inquiry_id=?",
(inquiry_id,))
        replies = cursor.fetchall()
        conn.close()
        return [reply[0] for reply in replies]

class ForumPostDatabase:

    @staticmethod
    def _get_connection():
        connection = sqlite3.connect("DEVELOPMENT/database/forum_database.db")
        cursor = connection.cursor()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS posts (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title VARCHAR,
                content TEXT,
                enrollment_id VARCHAR
            );
        """)
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS comments (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                post_id INTEGER,
                username VARCHAR,
                content TEXT,
                FOREIGN KEY(post_id) REFERENCES posts(id)
            );
        """)
```

```
        return connection, cursor

    @staticmethod
    def add_post(title, content,enrollment_id):
        conn, cursor = ForumPostDatabase._get_connection()
        cursor.execute("INSERT INTO posts (title, content, enrollment_id) VALUES
(?, ?, ?)", (title, content,enrollment_id))
        conn.commit()
        conn.close()

    @staticmethod
    def get_posts(page=0, per_page=10):
        conn, cursor = ForumPostDatabase._get_connection()
        offset = page * per_page
        cursor.execute("SELECT id, title, content, enrollment_id FROM posts ORDER
BY id DESC LIMIT ? OFFSET ?", (per_page, offset))
        posts = cursor.fetchall()
        result = []
        for post in posts:
            post_id, title, content,enrollment_id = post
            cursor.execute("SELECT username, content FROM comments WHERE post_id
= ?", (post_id,))
            comments = cursor.fetchall()
            result.append({
                'id': post_id,
                'title': title,
                'content': content,
                'enrollment_id' : enrollment_id,
                'comments': [{'username': comment[0], 'content': comment[1]} for
comment in comments]
            })
        conn.close()
        return result

    @staticmethod
    def add_comment(post_id, username, content):
        conn, cursor = ForumPostDatabase._get_connection()
        cursor.execute("INSERT INTO comments (post_id, username, content) VALUES
(?, ?, ?)", (post_id, username, content))
        conn.commit()
        conn.close()


class NotificationDatabase:
    @staticmethod
```

```
    def _get_connection():
        connection =
sqlite3.connect("DEVELOPMENT/database/notification_database.db")
        cursor = connection.cursor()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS notifications(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id VARCHAR,
                content TEXT,
                status VARCHAR DEFAULT 'Unread'
            );
        """)
        return connection, cursor

    @staticmethod
    def add_notification(user_id, content):
        conn, cursor = NotificationDatabase._get_connection()
        cursor.execute("INSERT INTO notifications (user_id, content) VALUES (?,
?)", (user_id, content))
        conn.commit()
        conn.close()

    @staticmethod
    def get_user_notifications(user_id):
        conn, cursor = NotificationDatabase._get_connection()
        cursor.execute("SELECT content FROM notifications WHERE user_id=? AND
status='Unread'", (user_id,))
        notifications = cursor.fetchall()
        conn.close()
        return [notif[0] for notif in notifications]

    @staticmethod
    def mark_as_read(user_id):
        conn, cursor = NotificationDatabase._get_connection()
        cursor.execute("UPDATE notifications SET status='Read' WHERE user_id=?",
(user_id,))
        conn.commit()
        conn.close()
```

#singelton_studentify_screen.py

```
from tkinter import *
from tkinter import messagebox
```

```
from PIL import Image, ImageTk
import os, sys
from singelton_addeventsscreen import Singelton_AddEventsScreen
from singelton_inquiryscreen import Singelton_InquiryScreen
from singelton_forum_post import SingletonForumPost
from singleton_view_event import Singleton_ViewEvent
from singelton_view_inquiries_screen import Singleton_ViewInquiriesScreen
from singleton_shownotification import Singleton_ShowNotification

class Singelton_Studentify_Screen:
    __instance = None

    def __new__(cls):
        if cls.__instance is None:
            cls.__instance = super(Singelton_Studentify_Screen, cls).__new__(cls)
            cls.__instance.create_widget()
        return cls.__instance

    def create_widget(self):
        self.studentify_screen = Tk()
        self.studentify_screen.title("Studentify")
        self.studentify_screen.geometry("1200x750")
        self.studentify_screen.resizable(False, False)

        # Background image
        self.bgimage =
ImageTk.PhotoImage(Image.open('DEVELOPMENT/studentify.jpg'))
        self.bgimage_label = Label(self.studentify_screen, image=self.bgimage)
        self.bgimage_label.place(x=0, y=0, relwidth=1, relheight=1)

        # User label
        self.user_label = Label(self.studentify_screen, font=("times new roman",
20), bg="lightgray")
        self.user_label.grid(row=0, column=0, columnspan=4, sticky="ew", padx=10,
pady=10)

        # Configure grid layout for responsiveness
        self.studentify_screen.grid_columnconfigure((0, 1, 2, 3), weight=1)
        self.studentify_screen.grid_rowconfigure((0,1, 2), weight=1)

        # Buttons
        self.add_events_btn = Button(self.studentify_screen, text="Add Events",
font=("times new roman", 15), command=self.open_add_events_screen)
        self.add_events_btn.grid(row=1, column=0, padx=10, pady=10, sticky="ew")
```

```
        self.view_events_btn = Button(self.studentify_screen, text="View Events",
font=("times new roman", 15), command=self.view_events)
        self.view_events_btn.grid(row=1, column=1, padx=10, pady=10, sticky="ew")

        self.forum_post_btn = Button(self.studentify_screen, text="Forum Post",
font=("times new roman", 15), command=self.forum_post)
        self.forum_post_btn.grid(row=1, column=2, padx=10, pady=10, sticky="ew")

        self.inquiry_btn = Button(self.studentify_screen, text="Inquiry",
font=("times new roman", 15), command=self.inquiry)
        self.inquiry_btn.grid(row=1, column=3, padx=10, pady=10, sticky="ew")

        self.view_inquiries_btn = Button(self.studentify_screen, text="View
Inquiries", font=("times new roman", 15), command=self.view_inquiries)
        self.view_inquiries_btn.grid(row=2, column=0, columnspan=2, padx=10,
pady=10, sticky="ew")

        self.notification_btn = Button(self.studentify_screen,
text="Notification", font=("times new roman", 15), command=self.notification)
        self.notification_btn.grid(row=2, column=2, columnspan=2, padx=10,
pady=10, sticky="ew")

        self.logout_btn = Button(self.studentify_screen, text="Logout",
font=("times new roman", 15), command=self.logout)
        self.logout_btn.grid(row=0, column=3, padx=10, pady=10, sticky="ew")

    def set_user_label(self, username, account_type,enrollment_id):
        self.user_label.configure(text=f"Welcome {username}!\nAccount type:
{account_type}")
        self.user_name = username
        self.account_type = account_type
        self.enrollment_id=enrollment_id

    def open_add_events_screen(self):
        Singelton_AddEventsScreen(self.studentify_screen, self.enrollment_id,
self.user_name)
    def view_events(self):
        Singleton_ViewEvent(self.studentify_screen, self.user_name)

    def forum_post(self):
        SingletonForumPost(self.studentify_screen,self.enrollment_id).pass_name(s
elf.user_name)

    def inquiry(self):
        Singelton_InquiryScreen(self.studentify_screen,self.enrollment_id)
```

```
    def view_inquiries(self):
        if self.account_type == "Student Representative":
            Singleton_ViewInquiriesScreen(self.studentify_screen)
        else:
            messagebox.showwarning("Access Denied", "You are not authorised to
access this.")

    def notification(self):
        Singleton_ShowNotification(self.studentify_screen,self.enrollment_id)

    def logout(self):
        os.execv(sys.executable, [os.path.basename(sys.executable)] + sys.argv)

"""
if __name__ == "__main__":
    root = Tk()
    # Simulate a user login with their enrollment ID
    #user_enrollment_id = "as"  # Replace with the actual user's enrollment ID
after login
    Singelton_Studentify_Screen().set_user_label("akhil","Student","1234")
    root.mainloop()
    """
```

#singelton_addeventsscreen.py

```
from tkinter import *
from tkinter import messagebox
from PIL import Image, ImageTk
from datetime import datetime
from data_base_connect import EventDatabase

class Singelton_AddEventsScreen:
    __instance = None

    def __new__(cls, root,enrollment_ID,user_name):
        if cls.__instance is None:
            cls.__instance = super(Singelton_AddEventsScreen, cls).__new__(cls)
            cls.__instance.create_widget(root,enrollment_ID,user_name)
        return cls.__instance

    def create_widget(self, root, enrollment_ID, user_name):
        self.add_events_window = Toplevel(root)
```

```
        self.add_events_window.title("Add Event")
        self.add_events_window.geometry("600x500")  # Increased height for new
fields
        self.add_events_window.resizable(False, False)

        # Background image
        self.bgimage =
ImageTk.PhotoImage(Image.open('DEVELOPMENT/studentify.jpg'))
        self.bgimage_label = Label(self.add_events_window, image=self.bgimage)
        self.bgimage_label.place(x=0, y=0, relwidth=1, relheight=1)

        # Location
        self.location_label = Label(self.add_events_window, text='Location',
font=("times new roman", 15), bg="white")
        self.location_label.place(x=50, y=50)
        self.location_entry = Entry(self.add_events_window, font=("times new
roman", 15))
        self.location_entry.place(x=200, y=50, width=300)

        # Event Name
        self.event_name_label = Label(self.add_events_window, text='Event Name',
font=("times new roman", 15), bg="white")
        self.event_name_label.place(x=50, y=100)
        self.event_name_entry = Entry(self.add_events_window, font=("times new
roman", 15))
        self.event_name_entry.place(x=200, y=100, width=300)

        # Date (Day, Month, Year)
        self.date_label = Label(self.add_events_window, text='Date', font=("times
new roman", 15), bg="white")
        self.date_label.place(x=50, y=150)

        self.day_var = StringVar()
        self.month_var = StringVar()
        self.year_var = StringVar()

        # Populate dropdowns with relevant data
        current_year = datetime.now().year
        self.day_options = [str(day).zfill(2) for day in range(1, 32)]
        self.month_options = [str(month).zfill(2) for month in range(1, 13)]
        self.year_options = [str(year) for year in range(current_year,
current_year + 5)]

        self.day_menu = OptionMenu(self.add_events_window, self.day_var,
*self.day_options)
```

```
        self.day_menu.place(x=200, y=150, width=80)
        self.day_var.set(self.day_options[0])

        self.month_menu = OptionMenu(self.add_events_window, self.month_var,
*self.month_options)
        self.month_menu.place(x=300, y=150, width=80)
        self.month_var.set(self.month_options[0])

        self.year_menu = OptionMenu(self.add_events_window, self.year_var,
*self.year_options)
        self.year_menu.place(x=400, y=150, width=80)
        self.year_var.set(self.year_options[0])

        # Time
        self.time_label = Label(self.add_events_window, text='Time (HH:MM)',
font=("times new roman", 15), bg="white")
        self.time_label.place(x=50, y=200)
        self.time_entry = Entry(self.add_events_window, font=("times new roman",
15))
        self.time_entry.place(x=200, y=200, width=300)

        # About (Larger Text Area)
        self.about_label = Label(self.add_events_window, text='About',
font=("times new roman", 15), bg="white")
        self.about_label.place(x=50, y=250)
        self.about_entry = Text(self.add_events_window, height=5, font=("times
new roman", 15))
        self.about_entry.place(x=200, y=250, width=300)

        # Submit Button
        # Use a lambda to pass arguments to the submit_event method
        self.submit_btn = Button(self.add_events_window, text="Submit",
font=("times new roman", 15), bg="lightblue", bd=2,
                                 command=lambda: self.submit_event(enrollment_ID,
user_name))
        self.submit_btn.place(x=250, y=420, width=100)

        # Handle window close to reset instance
        self.add_events_window.protocol("WM_DELETE_WINDOW", self.on_close)




    def submit_event(self,enrollment_ID,user_name):
        location = self.location_entry.get()
```

```
        event_name = self.event_name_entry.get()
        date = f"{self.day_var.get()}-{self.month_var.get()}-
{self.year_var.get()}"
        time = self.time_entry.get()
        about = self.about_entry.get("1.0", END).strip()
        host = user_name
        host_id = enrollment_ID

        if not location or not event_name or not about or not time:
            messagebox.showerror("Error", "All fields are required.")
            self.on_close()
        else:
            try:
                # Convert the date and time to datetime object for validation
                datetime.strptime(f"{self.year_var.get()}-{self.month_var.get()}-
{self.day_var.get()} {time}", "%Y-%m-%d %H:%M")
                # Save event data to database
                EventDatabase.add_event(location,event_name,date,time,about,host,
host_id)
                # Implement event submission logic here
                messagebox.showinfo("Success", "Event added successfully.")
                self.on_close()

            except ValueError:
                messagebox.showerror("Error", "Invalid date or time format.")
                self.on_close()

    def on_close(self):
        """ Reset instance and close the window """

        Singelton_AddEventsScreen.__instance = None
        self.add_events_window.destroy()
```

singleton_view_event.py

```
from tkinter import *
from tkinter import messagebox
from data_base_connect import EventDatabase, NotificationDatabase


class Singleton_ViewEvent:
    __instance = None
```

```
    def __new__(cls, root, current_user):
        if cls.__instance is None:
            cls.__instance = super(Singleton_ViewEvent, cls).__new__(cls)
            cls.__instance.create_widget(root, current_user)
        return cls.__instance

    def create_widget(self, root, current_user):
        self.view_events_window = Toplevel(root)
        self.view_events_window.title("View Events")
        self.view_events_window.geometry("800x600")
        self.view_events_window.resizable(False, False)

        self.current_user = current_user

        self.events_listbox = Listbox(self.view_events_window, font=("times new
roman", 15))
        self.events_listbox.place(x=50, y=50, width=700, height=400)
        self.events_listbox.bind("<Double-Button-1>", self.show_event_details)

        self.load_events()

        # Handle window close to reset instance
        self.view_events_window.protocol("WM_DELETE_WINDOW", self.on_close)

    def load_events(self):
        events = EventDatabase.get_events()
        for event in events:
            event_id, location, event_name, date, time, about, host, host_id =
event
            self.events_listbox.insert(END, f"{event_name}")

    def show_event_details(self, event):
        selected_index = self.events_listbox.curselection()
        if selected_index:
            event_name = self.events_listbox.get(selected_index)
            events = EventDatabase.get_events()
            for event in events:
                event_id, location, event_name_db, date, time, about,
host,host_id = event
                if event_name == event_name_db:
                    self.show_event_popup(event_id, location, event_name_db,
date, time, about, host, host_id)
```

```
    def show_event_popup(self, event_id, location, event_name, date, time, about,
host, host_id):
        top = Toplevel(self.view_events_window)
        top.title(event_name)
        top.geometry("400x400")

        Label(top, text=f"Event: {event_name}", font=("times new roman",
15)).pack()
        Label(top, text=f"Location: {location}", font=("times new roman",
15)).pack()
        Label(top, text=f"Date: {date}", font=("times new roman", 15)).pack()
        Label(top, text=f"Time: {time}", font=("times new roman", 15)).pack()
        Label(top, text=f"About: {about}", font=("times new roman", 15)).pack()


        # Get the host details
        event_host = host
        Label(top, text=f"Host: {event_host}", font=("times new roman",
15)).pack()

        # RSVP Dropdown
        self.rsvp_var = StringVar(value="Select RSVP")
        rsvp_menu = OptionMenu(top, self.rsvp_var, "Yes", "No", "Maybe")
        rsvp_menu.pack()

        Button(top, text="Submit RSVP", command=lambda:
self.submit_rsvp(event_id,event_name)).pack()

        rsvp_people = EventDatabase.get_event_rsvps(event_id)
        print(len(rsvp_people))
        if rsvp_people:

            Label(top, text="People response to event:", font=("times new roman",
15)).pack()
            for person in rsvp_people:

                    Label(top, text=person, font=("times new roman", 15)).pack()



    def submit_rsvp(self, event_id,event_name):
        rsvp_option = self.rsvp_var.get()
        if rsvp_option == "Select RSVP":
            messagebox.showerror("Error", "Please select an RSVP option.")
            return
```

```
        # Add the RSVP to the database
        EventDatabase.add_rsvp(event_id, self.current_user,rsvp_option)

        # Notify the host with the RSVP option

        host_id=EventDatabase.get_event_host(event_id)
        NotificationDatabase.add_notification(host_id, f"The student
{self.current_user} said {rsvp_option} for your event {event_name}")
        self.on_close()
        messagebox.showinfo("RSVP", f"You have RSVPed with option:
{rsvp_option}.")

    def on_close(self):
        """ Reset instance and close the window """
        Singleton_ViewEvent.__instance = None
        self.view_events_window.destroy()
```

#singelton_inquiryscreen.py

```
from tkinter import *
from tkinter import messagebox
from PIL import Image, ImageTk
from data_base_connect import InquiryDatabase

class Singelton_InquiryScreen:
    __instance = None

    def __new__(cls, root,enrollment_id):
        if cls.__instance is None:
            cls.__instance = super(Singelton_InquiryScreen, cls).__new__(cls)
            cls.__instance.create_widget(root,enrollment_id)
        return cls.__instance

    def create_widget(self, root,enrollment_id):
        self.inquiry_window = Toplevel(root)
        self.inquiry_window.title("Inquiry Form")
        self.inquiry_window.geometry("600x600")
        self.inquiry_window.resizable(False, False)

        # Background image
```

```
        self.bgimage =
ImageTk.PhotoImage(Image.open('DEVELOPMENT/studentify.jpg'))
        self.bgimage_label = Label(self.inquiry_window, image=self.bgimage)
        self.bgimage_label.place(x=0, y=0, relwidth=1, relheight=1)

        # Name
        self.name_label = Label(self.inquiry_window, text='Name', font=("times
new roman", 15), bg="white")
        self.name_label.place(x=50, y=50)
        self.name_entry = Entry(self.inquiry_window, font=("times new roman",
15))
        self.name_entry.place(x=250, y=50, width=300)

        # Enrollment ID
        self.enrollment_label = Label(self.inquiry_window, text='Enrollment ID',
font=("times new roman", 15), bg="white")
        self.enrollment_label.place(x=50, y=100)
        self.enrollment_entry = Entry(self.inquiry_window, font=("times new
roman", 15))
        self.enrollment_entry.place(x=250, y=100, width=300)
        # Set default value for the entry
        self.enrollment_entry.insert(0, enrollment_id)

        # Email
        self.email_label = Label(self.inquiry_window, text='Email', font=("times
new roman", 15), bg="white")
        self.email_label.place(x=50, y=150)
        self.email_entry = Entry(self.inquiry_window, font=("times new roman",
15))
        self.email_entry.place(x=250, y=150, width=300)

        # Department (Dropdown)
        self.department_label = Label(self.inquiry_window, text='Department',
font=("times new roman", 15), bg="white")
        self.department_label.place(x=50, y=200)
        self.department_var = StringVar()
        self.department_options = [
            "Technology and Bionics", "Life Sciences",
            "Society and Economics", "Communication and Environment"
        ]
        self.department_menu = OptionMenu(self.inquiry_window,
self.department_var, *self.department_options)
        self.department_menu.place(x=250, y=200, width=300)
        self.department_var.set(self.department_options[0])  # Set default value
```

```
        # Type of Inquiry (Dropdown)
        self.inquiry_type_label = Label(self.inquiry_window, text='Type of
Inquiry', font=("times new roman", 15), bg="white")
        self.inquiry_type_label.place(x=50, y=250)
        self.inquiry_type_var = StringVar()
        self.inquiry_type_options = [
            "Examination", "Re-registration", "Studies",
            "Transcript and documents", "Workshops",
            "Co-curricular activities", "Sports"
        ]
        self.inquiry_type_menu = OptionMenu(self.inquiry_window,
self.inquiry_type_var, *self.inquiry_type_options)
        self.inquiry_type_menu.place(x=250, y=250, width=300)
        self.inquiry_type_var.set(self.inquiry_type_options[0])  # Set default
value

        # Subject
        self.subject_label = Label(self.inquiry_window, text='Subject',
font=("times new roman", 15), bg="white")
        self.subject_label.place(x=50, y=300)
        self.subject_entry = Entry(self.inquiry_window, font=("times new roman",
15))
        self.subject_entry.place(x=250, y=300, width=300)

        # Details (Larger Text Area)
        self.details_label = Label(self.inquiry_window, text='Details',
font=("times new roman", 15), bg="white")
        self.details_label.place(x=50, y=350)
        self.details_entry = Text(self.inquiry_window, height=5, font=("times new
roman", 15))
        self.details_entry.place(x=250, y=350, width=300)

        # Submit Button
        self.submit_btn = Button(self.inquiry_window, text="Submit", font=("times
new roman", 15), bg="lightblue", bd=2, command=self.submit_inquiry)
        self.submit_btn.place(x=250, y=500, width=100)

        # Handle window close to reset instance
        self.inquiry_window.protocol("WM_DELETE_WINDOW", self.on_close)

    def submit_inquiry(self):
        name = self.name_entry.get()
        enrollment_id = self.enrollment_entry.get()
        email = self.email_entry.get()
        department = self.department_var.get()
```

```
        inquiry_type = self.inquiry_type_var.get()
        subject = self.subject_entry.get()
        details = self.details_entry.get("1.0", END).strip()

        if not name or not enrollment_id or not email or not subject or not
details:
            messagebox.showerror("Error", "All fields are required.")
            self.on_close()
        else:
            try:
                # Save inquiry data to the database
                InquiryDatabase.add_inquiry(name, enrollment_id, email,
department, inquiry_type, subject, details)
                # Implement inquiry submission logic here
                messagebox.showinfo("Success", "Inquiry submitted successfully.")
                self.on_close()


            except Exception as e:
                print (e)
                messagebox.showerror("Error",f"An error occurred: {e}")

    def on_close(self):
        """ Reset instance and close the window """

        Singelton_InquiryScreen.__instance = None
        self.inquiry_window.destroy()
```

#singelton_forum_post.py

```
from tkinter import *
from tkinter import messagebox, simpledialog
from PIL import Image, ImageTk
from data_base_connect import ForumPostDatabase, NotificationDatabase

class SingletonForumPost:
    __instance = None

    def __new__(cls, root, enrollment_id):
        if cls.__instance is None:
```

```
            cls.__instance = super(SingletonForumPost, cls).__new__(cls)
            cls.__instance.create_widget(root, enrollment_id)

        return cls.__instance

    def create_widget(self, root, enrollment_id):
        self.enrollment_id = enrollment_id
        self.forum_window = Toplevel(root)
        self.forum_window.title("Forum Post")
        self.forum_window.geometry("800x600")
        self.forum_window.resizable(False, False)

        # Background image
        self.bgimage =
ImageTk.PhotoImage(Image.open('DEVELOPMENT/studentify.jpg'))
        self.bgimage_label = Label(self.forum_window, image=self.bgimage)
        self.bgimage_label.place(x=0, y=0, relwidth=1, relheight=1)

        # New Post Section
        self.new_post_label = Label(self.forum_window, text="Create a New Post",
font=("times new roman", 18), bg="white")
        self.new_post_label.place(x=50, y=10)

        self.post_title_label = Label(self.forum_window, text="Title",
font=("times new roman", 15), bg="white")
        self.post_title_label.place(x=50, y=50)
        self.post_title_entry = Entry(self.forum_window, font=("times new roman",
15))
        self.post_title_entry.place(x=150, y=50, width=300)

        self.post_content_label = Label(self.forum_window, text="Content",
font=("times new roman", 15), bg="white")
        self.post_content_label.place(x=50, y=100)
        self.post_content_entry = Text(self.forum_window, font=("times new
roman", 15), height=5)
        self.post_content_entry.place(x=150, y=100, width=300)

        self.submit_post_btn = Button(self.forum_window, text="Submit Post",
font=("times new roman", 15), bg="lightblue", command=lambda: self.submit_post())
        self.submit_post_btn.place(x=500, y=200)

        # Posts Section with Scrollbar
        self.posts_frame = Frame(self.forum_window)
        self.posts_frame.place(x=50, y=250, width=700, height=300)
```

```
        self.canvas = Canvas(self.posts_frame, bg="white")
        self.scrollbar = Scrollbar(self.posts_frame, orient=VERTICAL,
command=self.canvas.yview)
        self.scrollable_frame = Frame(self.canvas, bg="white")

        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: self.canvas.configure(
                scrollregion=self.canvas.bbox("all")
            )
        )

        self.canvas.create_window((0, 0), window=self.scrollable_frame,
anchor="nw")
        self.canvas.configure(yscrollcommand=self.scrollbar.set)

        self.canvas.pack(side=LEFT, fill=BOTH, expand=True)
        self.scrollbar.pack(side=RIGHT, fill=Y)

        self.current_page = 0
        self.posts_per_page = 10

        self.update_posts()

        # Handle window close to reset instance
        self.forum_window.protocol("WM_DELETE_WINDOW", self.on_close)

    def submit_post(self):
        title = self.post_title_entry.get()
        content = self.post_content_entry.get("1.0", END).strip()

        if not title or not content:
            messagebox.showerror("Error", "Title and content cannot be empty.")
        else:
            ForumPostDatabase.add_post(title, content, self.enrollment_id)
            self.clear_new_post()
            self.update_posts()

    def clear_new_post(self):
        self.post_title_entry.delete(0, END)
        self.post_content_entry.delete("1.0", END)

    def pass_name(self, user_name):
        self.user_name = user_name
```

```
    def update_posts(self):
        for widget in self.scrollable_frame.winfo_children():
            widget.destroy()

        posts = ForumPostDatabase.get_posts(self.current_page,
self.posts_per_page)

        for post in posts:
            post_title_label = Label(self.scrollable_frame, text=post['title'],
font=("times new roman", 15, "bold"), bg="white")
            post_title_label.pack(anchor="w", pady=5)

            post_content_label = Label(self.scrollable_frame,
text=post['content'], font=("times new roman", 15), bg="white", wraplength=650,
justify=LEFT)
            post_content_label.pack(anchor="w", pady=5)

            comment_btn = Button(self.scrollable_frame, text="Comment",
font=("times new roman", 12), bg="lightblue", command=lambda post_id=post['id']:
self.add_comment(post_id, post['enrollment_id']))
            comment_btn.pack(anchor="w", pady=5)

            for comment in post['comments']:
                comment_label = Label(self.scrollable_frame,
text=f"{comment['username']}: {comment['content']}", font=("times new roman",
12), bg="white", wraplength=650, justify=LEFT)
                comment_label.pack(anchor="w", pady=5)

        load_more_btn = Button(self.scrollable_frame, text="Load More",
font=("times new roman", 15), bg="lightblue", command=self.load_more_posts)
        load_more_btn.pack(pady=10)

    def load_more_posts(self):
        self.current_page += 1
        self.update_posts()

    def add_comment(self, post_id, enrollment_id):
        name = self.user_name
        if not name:
            messagebox.showerror("Error", "Unable to fetch name.")
            return

        comment = simpledialog.askstring("Add Comment", "Enter your comment:")
        if comment:
            ForumPostDatabase.add_comment(post_id, name, comment)
```

```
            NotificationDatabase.add_notification(enrollment_id, f"Forum Post:
The student {name} commented: {comment}")
            self.update_posts()

    def on_close(self):
        """Reset instance and close the window"""
        SingletonForumPost.__instance = None
        self.forum_window.destroy()

if __name__ == "__main__":
    root = Tk()
    # Simulate a user login with their enrollment ID
    # user_enrollment_id = "as"  # Replace with the actual user's enrollment ID
after login
    # app = SingletonForumPost(root, user_enrollment_id)
    root.mainloop()
```

#singleton_shownotification.py

```
# singleton_shownotification.py

from tkinter import *
from data_base_connect import NotificationDatabase, LoginDatabase

class Singleton_ShowNotification:
    __instance = None

    def __new__(cls, root, enrollment_id):
        if cls.__instance is None:
            cls.__instance = super(Singleton_ShowNotification, cls).__new__(cls)
            cls.__instance.create_widget(root, enrollment_id)
        return cls.__instance

    def create_widget(self, root, enrollment_id):
        self.notification_window = Toplevel(root)
        self.notification_window.title("Notifications")
        self.notification_window.geometry("400x400")
        self.notification_window.resizable(False, False)

        # Notification Listbox
        self.notification_listbox = Listbox(self.notification_window,
font=("times new roman", 12), width=50, height=20)
```

```
        self.notification_listbox.pack(pady=20)

        # Load Notifications
        self.load_notifications(enrollment_id)

        # Handle window close
        self.notification_window.protocol("WM_DELETE_WINDOW", self.on_close)

    def load_notifications(self, enrollment_id):
        notifications =
NotificationDatabase.get_user_notifications(enrollment_id)
        for notification in notifications:
            self.notification_listbox.insert(END, notification)
        # Mark notifications as read after displaying them
        NotificationDatabase.mark_as_read(enrollment_id)

    def on_close(self):
        Singleton_ShowNotification.__instance = None
        self.notification_window.destroy()
```

#singelton_view_inquiries_screen.py

```
from tkinter import *
from tkinter import messagebox
from data_base_connect import InquiryDatabase, NotificationDatabase,
LoginDatabase

class Singleton_ViewInquiriesScreen:
    __instance = None

    def __new__(cls, root):
        if cls.__instance is None:
            cls.__instance = super(Singleton_ViewInquiriesScreen,
cls).__new__(cls)
            cls.__instance.create_widget(root)
        return cls.__instance

    def create_widget(self, root):
        self.view_inquiries_window = Toplevel(root)
        self.view_inquiries_window.title("View Inquiries")
        self.view_inquiries_window.geometry("1200x750")
        self.view_inquiries_window.resizable(True, True)
```

```
        # Inquiries Listbox
        self.inquiries_listbox = Listbox(self.view_inquiries_window, font=("times
new roman", 12), width=100, height=15)
        self.inquiries_listbox.place(x=50, y=50, width=400, height=600)
        self.inquiries_listbox.bind("<Double-Button-1>",
self.show_inquiry_details)

        # Inquiry Details
        self.details_label = Label(self.view_inquiries_window, text="",
font=("times new roman", 14), wraplength=750, anchor="w")
        self.details_label.place(x=500, y=50, width=650, height=400)

        # Reply Section
        self.reply_label = Label(self.view_inquiries_window, text="Reply:",
font=("times new roman", 15))
        self.reply_label.place(x=500, y=460)

        self.reply_text = Text(self.view_inquiries_window, font=("times new
roman", 15), height=8, width=60)
        self.reply_text.place(x=500, y=490)

        self.submit_reply_btn = Button(self.view_inquiries_window, text="Submit
Reply", font=("times new roman", 15), command=self.submit_reply)
        self.submit_reply_btn.place(x=650, y=650)

        # Load Inquiries
        self.load_inquiries()

        # Handle window close
        self.view_inquiries_window.protocol("WM_DELETE_WINDOW", self.on_close)

    def load_inquiries(self):
        inquiries = InquiryDatabase.get_inquiries()
        self.inquiries_listbox.delete(0, END)  # Clear any existing items
        for inquiry in inquiries:
            # Adjust the unpacking logic to match the number of columns in the
database
            inquiry_id, name, enrollment_id, email, department, inquiry_type,
subject, details, status = inquiry
            self.inquiries_listbox.insert(END, f"{subject} - {name}")

    def show_inquiry_details(self, event):
        selected_index = self.inquiries_listbox.curselection()
        if selected_index:
            subject_name = self.inquiries_listbox.get(selected_index)
```

```
            inquiries = InquiryDatabase.get_inquiries()
            for inquiry in inquiries:
                # Adjust the unpacking logic to match the number of columns in
the database
                inquiry_id, name, enrollment_id, email, department, inquiry_type,
subject, details, status = inquiry
                if f"{subject} - {name}" == subject_name:
                    self.details_label.config(text=f"From: {name}
({email})\nDepartment: {department}\nType: {inquiry_type}\nDetails: {details}")
                    self.current_inquiry_id = inquiry_id
                    self.current_inquirer_id = enrollment_id  # Track who asked
the inquiry
                    break

    def submit_reply(self):
        reply_content = self.reply_text.get("1.0", END).strip()
        if reply_content:

            current_user =
LoginDatabase.get_user_details(self.current_inquirer_id)
            print(f"current_user is {self.current_inquirer_id}")
            if current_user:

                InquiryDatabase.add_reply(self.current_inquiry_id,
current_user['enrollment_id'], reply_content)

                NotificationDatabase.add_notification(current_user['enrollment_id
'], f"Your inquiry has been replied: {reply_content}")

                messagebox.showinfo("Success", "Reply submitted and notification
sent successfully.")
                self.reply_text.delete("1.0", END)
                self.load_inquiries()

    def on_close(self):
        Singleton_ViewInquiriesScreen.__instance = None
        self.view_inquiries_window.destroy()
```