OD REQUEST AUTOMATION

A MINI-PROJECT REPORT

Submitted by

THARUN R L 220701302

UDHAYA SHANKAR J 220701306

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI-602105

FEB-JUNE

2024

BONAFIDE CERTIFICATE

Certified that this project **"OD REQUEST AUTOMATION"** is the bonafide work of **" THARUN R L (220701302), UDHAYA SHANKAR J (220701306)"** who carried out the project work under my supervision.

SIGNATURE	SIGNATURE
Dr.R.SABITHA	Ms.V.JANANEE
Professor and Academic Head,	Assistant Professor(SG)
Computer Science and Engineering,	Computer Science and Engineering,
Rajalakshmi Engineering College	Rajalakshmi Engineering College,
(Autonomous),	(Autonomous),
Thandalam,Chennai-602 105	Thandalam,Chennai-602 105

Submitted for the Practical examination to be Held on _____

INTERNAL EXAMINER EXTERNAL EXAMINER

ABSTRACT

The OD request automation is an application for assisting a class incharge in managing OD list of students of their class. The system would provide basic set of feature to request OD for a student, decline/approve OD request of a student for a class incharge, counselor and Head of department and view list of student who got OD on particular day.

OD request automation is a typical management information system (MIS), its development include the establishment and maintenance of back-end database and front-end application development aspects. For the former require the establishment of data consistency and integrity of the strong data security and good libraries. As for the latter requires the application full functional, easy to use and so on.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION:

The OD (On-Duty) Request Automation project is designed to streamline the process of submitting, approving, and tracking OD requests within an educational institution. The system caters to four types of users: Students, Class Incharge, Counselors, and Head of Department (HOD). Using a combination of Python with Tkinter for the graphical user interface and PostgreSQL for the database management, this project aims to enhance efficiency, reduce paperwork, and ensure transparency in the OD request process.

1.2 OBJECTIVES:

- Automate the OD request submission and approval process.
- Ensure transparency and traceability of requests and their statuses.
- Minimize manual intervention and paperwork.
- Provide a clear and user-friendly interface for all users.
- Enable Class Incharge to view students on OD on a particular day.

1.3 MODULES

- **Student Module:** Allows students to submit OD requests and view their status.
- **Class Incharge Module:** Enables class incharges to approve or decline OD requests and view students on OD.

- **Counselor Module:** Facilitates counselors to approve or decline OD requests forwarded by class incharges.
- **Head of Department Module:** Allows HOD to approve or decline OD requests forwarded by counselors.

CHAPTER 2

SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION:

- **Python:** A versatile programming language used for backend logic and handling the Tkinter GUI.
- **Tkinter:** A standard GUI library for Python, used to create the graphical user interface.
- **PostgreSQL:** A powerful, open-source relational database system used to manage and store application data.

2.2 LANGUAGES:

2.2.1 SQL:

SQL (Structured Query Language) is used to interact with the PostgreSQL database. It allows for querying, updating, and managing the data within the database.

2.2.2 PYTHON:

Python is the main programming language used in this project. It provides the backend logic, database interactions, and the GUI through Tkinter.

CHAPTER 3

REQUIREMENTS AND ANALYSIS

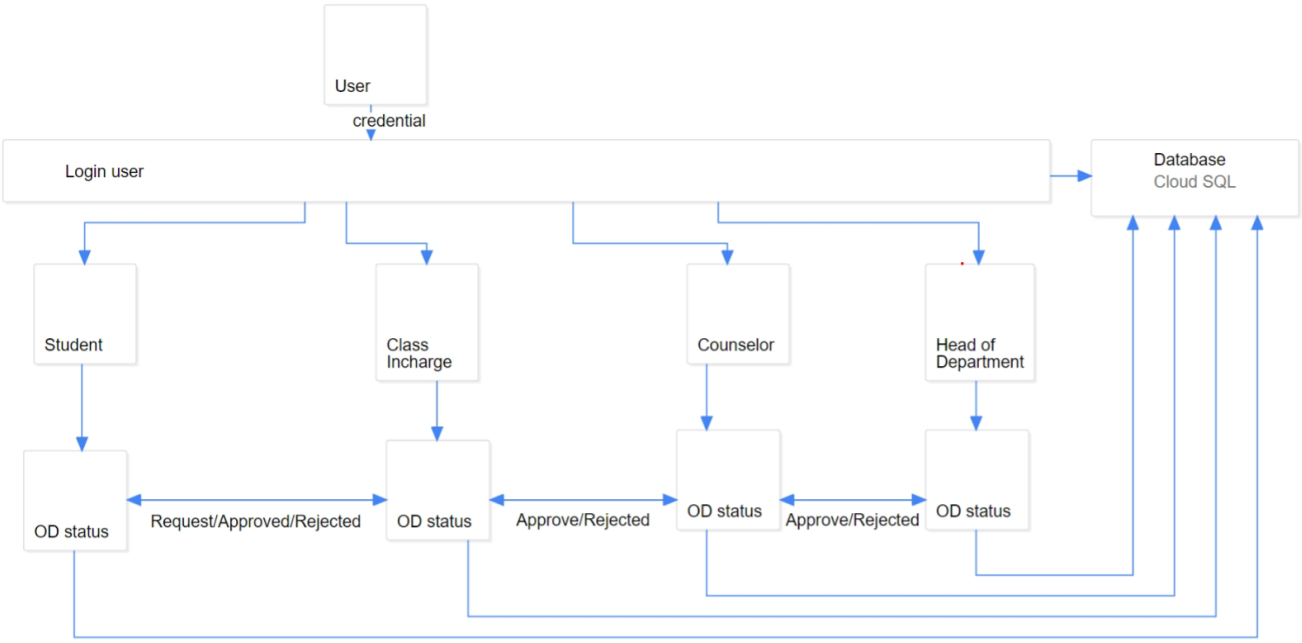
3.1 REQUIREMENT SPECIFICATION

- **Functional Requirements:**
  - User authentication and role-based access.
  - OD request submission by students.
  - Approval/Decline functionality for Class Incharge, Counselor, and HOD.
  - Notification system for request status changes.
  - Viewing OD status and history.
- **Non-Functional Requirements:**
  - System should be reliable and secure.
  - The interface should be user-friendly.
  - The application should be scalable and maintainable.

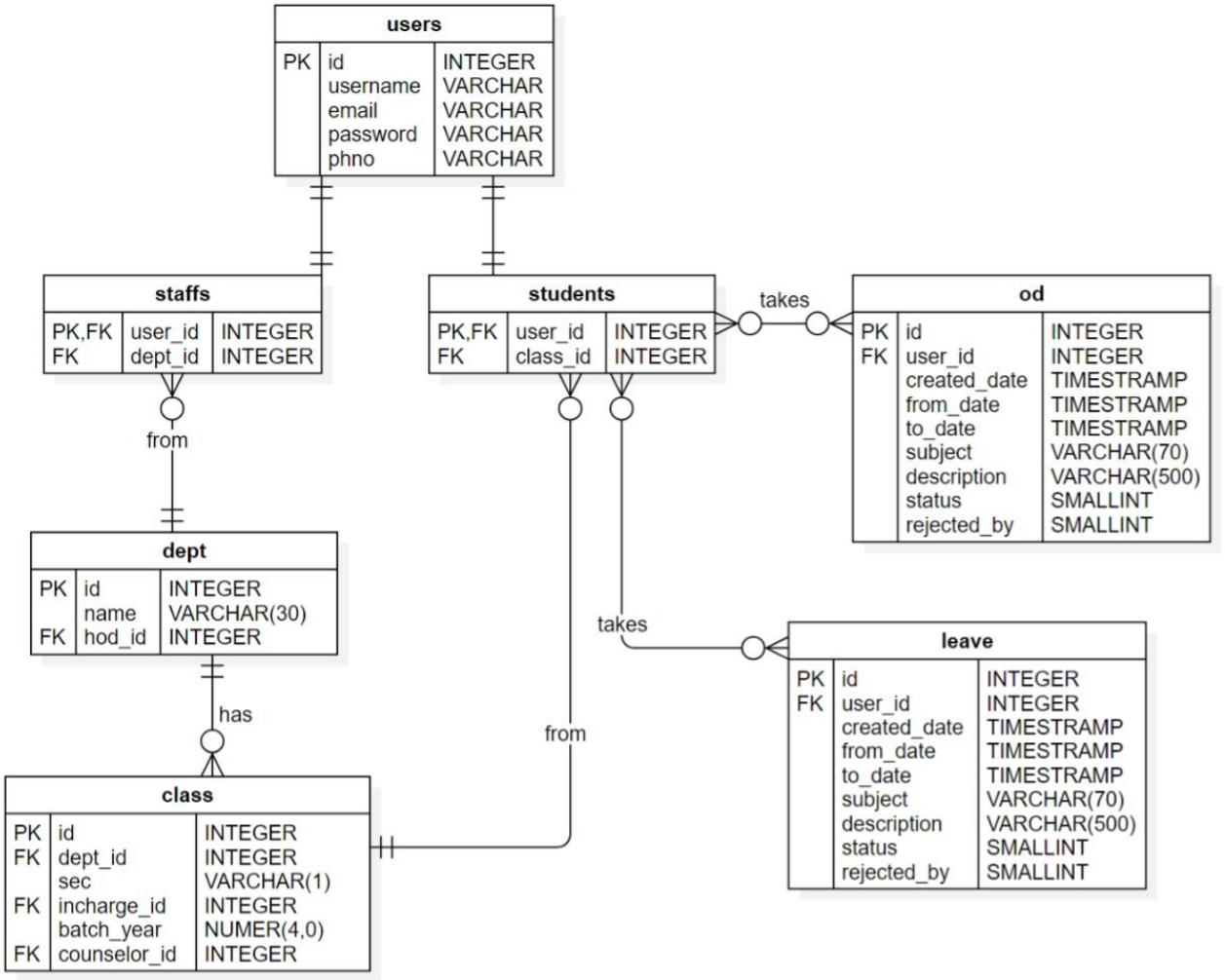
3.2 HARDWARE AND SOFTWARE REQUIREMENTS

- **Hardware Requirements:**
  - A computer with a modern processor (Intel i5 or equivalent).
  - Minimum 4GB RAM.
  - Minimum 500GB HDD/SSD.
- **Software Requirements:**
  - Python 3.x
  - Tkinter library
  - PostgreSQL
  - psycopg2 (PostgreSQL adapter for Python)

3.3 ARCHITECTURE DIAGRAM



3.4 ER DIAGRAM



3.5 NORMALIZATION

Raw database

Column Name	Data Type	Key Constraints
User_id	Integer	Primary key NOT NULL
Username	Varchar(30)	NOT NULL
Email	Varchar(40)	NOT NULL
Password	Varchar(70)	NOT NULL
Phno	Varchar(10)	NOT NULL

Class_id	Integer	Primary key NOT NULL
Dept_id	Integer	Primary key NOT NULL
Sec	Varchar(1)	NOT NULL
Incharge_id	Integer	Primary key NOT NULL
Batch_year	Number(4,0)	NOT NULL
Counselor_id	Integer	Primary key NOT NULL
Dept_name	Varchar(30)	NOT NULL
Hod_id	Integer	Primary key NOT NULL
Od_id	Integer	Primary key NOT NULL
Od_created_date	Timestramp	NOT NULL
Od_from_date	Timestramp	NOT NULL
Od_to_date	Timestramp	NOT NULL
Od_subject	Varchar(70)	NOT NULL
Od_description	Varchar(500)	NOT NULL
Od_status	Integer	NOT NULL
Od_rejected_by	Integer	NOT NULL
Leave_id	Integer	Primary key NOT NULL
Leave_created_date	Timestramp	NOT NULL
Leave_from_date	Timestramp	NOT NULL
Leave_to_date	Timestramp	NOT NULL
Leave_subject	Varchar(70)	NOT NULL
Leave_description	Varchar(500)	NOT NULL
Leave_status	Integer	NOT NULL
Leave_rejected_by	Integer	NOT NULL

- **1NF (First Normal Form):** Ensured that all columns in each table are atomic and contain unique values.

USER TABLE

Column Name	Data Type	Key Constraints
User_id	Integer	Primary key NOT NULL
Username	Varchar(30)	NOT NULL
Email	Varchar(40)	NOT NULL
Password	Varchar(70)	NOT NULL
Phno	Varchar(10)	NOT NULL
Class_id	Integer	Primary key NOT NULL
Dept_id	Integer	Primary key NOT NULL
Sec	Varchar(1)	NOT NULL
Incharge_id	Integer	Primary key NOT NULL
Batch_year	Number(4,0)	NOT NULL
Counselor_id	Integer	Primary key NOT NULL
Dept_name	Varchar(30)	NOT NULL
Hod_id	Integer	Primary key NOT NULL

OD TABLE

Column Name	Data Type	Key Constraints
Od_id	Integer	Primary key NOT NULL
Od_created_date	Timestramp	NOT NULL
Od_from_date	Timestramp	NOT NULL
Od_to_date	Timestramp	NOT NULL
Od_subject	Varchar(70)	NOT NULL
Od_description	Varchar(500)	NOT NULL
Od_stats	Integer	NOT NULL
Od_rejected_by	Integer	NOT NULL
Od_status_id	Integer	Primary key NOT NULL

LEAVE TABLE

Column Name	Data Type	Key Constraints
Leave_id	Integer	Primary key NOT NULL
Leave_created_date	Timestramp	NOT NULL
Leave_from_date	Timestramp	NOT NULL
Leave_to_date	Timestramp	NOT NULL
Leave_subject	Varchar(70)	NOT NULL
Leave_description	Varchar(500)	NOT NULL
Leave_status	Integer	NOT NULL
Leave_rejected_by	Integer	NOT NULL

- **2NF (Second Normal Form):** Ensured that all non-key attributes are fully functionally dependent on the primary key.

USER TABLE

Column Name	Data Type	Key Constraints
User_id	Integer	Primary key NOT NULL
Username	Varchar(30)	NOT NULL
Email	Varchar(40)	NOT NULL
Password	Varchar(70)	NOT NULL
Phno	Varchar(10)	NOT NULL

CLASS TBLE

Column Name	Data Type	Key Constraints
User_id	Integer	Primary key NOT NULL
Class_id	Integer	Primary key NOT NULL
Dept_id	Integer	Primary key NOT NULL
Sec	Varchar(1)	NOT NULL
Incharge_id	Integer	Primary key NOT NULL
Batch_year	Number(4,0)	NOT NULL
Counselor_id	Integer	Primary key NOT NULL
Dept_name	Varchar(30)	NOT NULL
Hod_id	Integer	Primary key NOT NULL

OD TABLE

Column Name	Data Type	Key Constraints
Od_id	Integer	Primary key NOT NULL
Od_created_date	Timestramp	NOT NULL
Od_from_date	Timestramp	NOT NULL
Od_to_date	Timestramp	NOT NULL
Od_subject	Varchar(70)	NOT NULL
Od_description	Varchar(500)	NOT NULL
Od_stats	Integer	NOT NULL
Od_rejected_by	Integer	NOT NULL
Od_status_id	Integer	Primary key NOT NULL

LEAVE TABLE

Column Name	Data Type	Key Constraints
Leave_id	Integer	Primary key NOT NULL
Leave_created_date	Timestramp	NOT NULL
Leave_from_date	Timestramp	NOT NULL
Leave_to_date	Timestramp	NOT NULL
Leave_subject	Varchar(70)	NOT NULL
Leave_description	Varchar(500)	NOT NULL
Leave_status	Integer	NOT NULL
Leave_rejected_by	Integer	NOT NULL

- **3NF (Third Normal Form):** Ensured that all attributes are not only fully functionally dependent on the primary key but are also non-transitively dependent.

USER TABLE

Column Name	Data Type	Key Constraints
User_id	Integer	Primary key NOT NULL
Username	Varchar(30)	NOT NULL
Email	Varchar(40)	NOT NULL
Password	Varchar(70)	NOT NULL
Phno	Varchar(10)	NOT NULL

DEPT TABLE

Column Name	Data Type	Key Constraints
Dept_id	Integer	Primary key NOT NULL
Dept_name	Varchar(30)	NOT NULL
Hod_id	Integer	Primary key NOT NULL

CLASS TBLE

Column Name	Data Type	Key Constraints
User_id	Integer	Primary key NOT NULL
Class_id	Integer	Primary key NOT NULL
Dept_id	Integer	Primary key NOT NULL
Sec	Varchar(1)	NOT NULL
Incharge_id	Integer	Primary key NOT NULL
Batch_year	Number(4,0)	NOT NULL
Counselor_id	Integer	Primary key NOT NULL

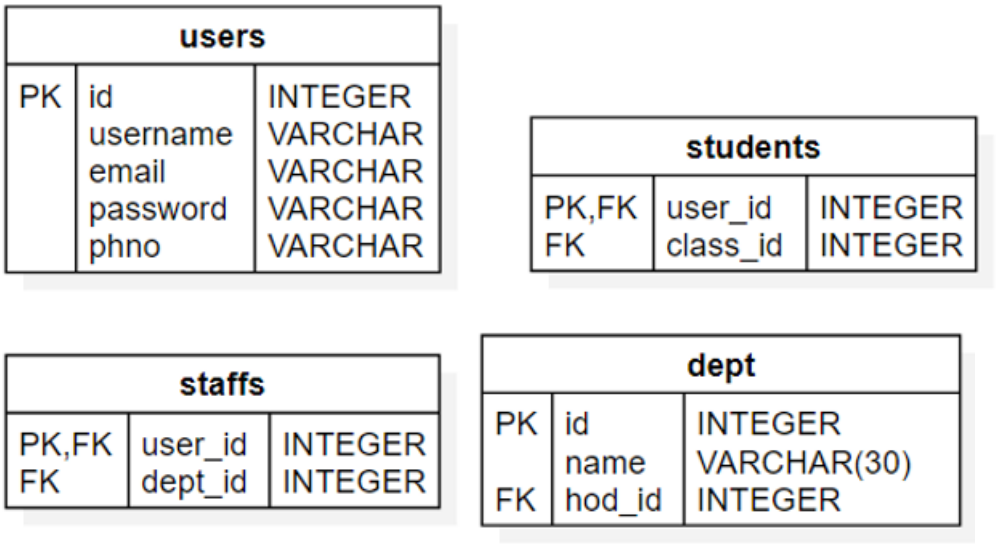
OD TABLE

Column Name	Data Type	Key Constraints
Od_id	Integer	Primary key NOT NULL
Od_created_date	Timestramp	NOT NULL
Od_from_date	Timestramp	NOT NULL
Od_to_date	Timestramp	NOT NULL
Od_subject	Varchar(70)	NOT NULL
Od_description	Varchar(500)	NOT NULL
Od_stats	Integer	NOT NULL
Od_rejected_by	Integer	NOT NULL
Od_status_id	Integer	Primary key NOT NULL

LEAVE TABLE

Column Name	Data Type	Key Constraints
Leave_id	Integer	Primary key NOT NULL
Leave_created_date	Timestramp	NOT NULL
Leave_from_date	Timestramp	NOT NULL
Leave_to_date	Timestramp	NOT NULL
Leave_subject	Varchar(70)	NOT NULL
Leave_description	Varchar(500)	NOT NULL
Leave_status	Integer	NOT NULL
Leave_rejected_by	Integer	NOT NULL

- **BCNF (Boycee Codd Normal Form):** A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- We have followed normalization up to BCNF below are the relations which we have used to implement our mini project.



class		
PK	id	INTEGER
FK	dept_id	INTEGER
	sec	VARCHAR(1)
FK	incharge_id	INTEGER
	batch_year	NUMER(4,0)
FK	counselor_id	INTEGER

leave		
PK	id	INTEGER
FK	user_id	INTEGER
	created_date	TIMESTRAMP
	from_date	TIMESTRAMP
	to_date	TIMESTRAMP
	subject	VARCHAR(70)
	description	VARCHAR(500)
	status	SMALLINT
	rejected_by	SMALLINT

od		
PK	id	INTEGER
FK	user_id	INTEGER
	created_date	TIMESTRAMP
	from_date	TIMESTRAMP
	to_date	TIMESTRAMP
	subject	VARCHAR(70)
	description	VARCHAR(500)
	status	SMALLINT
	rejected_by	SMALLINT

CHAPTER 4

PROGRAM CODE

configs.py

```
import psycopg2
def config():
    try:
        return psycopg2.connect(database="postgres",
                                host="aws-0-ap-southeast-1.pooler.supabase.com",
                                user="postgres.gcurjgyrqycujoxguhwl",
                                password="Tarl@576193",
                                port="5432")
    except:
        print("Database Error!!!")
```

login.py

```
import tkinter
from tkinter import messagebox, ttk
from hashlib import sha256
from configs import config
import sv_ttk
from home import home
import threading

class User:
    def __init__(self, id, name, email, phno, role=None):
        self.id = id
        self.name = name
        self.email = email
        self.phno = phno
        self.role = role

curr_user = None
def connect():
```

```
    global con
    print(1111)
    con=config()
    print(2222)
def passhash(password):
    return sha256(password.encode('utf-8')).hexdigest()

def authentication(email, password, parent):
    def connect_and_authenticate():
        print("before")
        cur = con.cursor()
        print("after")
        cur.execute(f"SELECT id, username, email, phno FROM users WHERE email='{email}'
AND password='{passhash(password)}'")
        user = cur.fetchone()
        print(user)
        if user:
            user_id = user[0]
            role = 0
            cur.execute(f"""
                        SELECT -1 AS role FROM students WHERE user_id = {user_id}
                        UNION
                        SELECT 1 AS role FROM class WHERE incharge_id = {user_id}
                        UNION
                        SELECT 2 AS role FROM class WHERE counselor_id = {user_id}
                        UNION
                        SELECT 3 AS role FROM dept WHERE hod_id = {user_id}
                    """)
            role_result = cur.fetchone()
            print(role_result)
            if role_result:
                role = role_result[0]

            global curr_user
            curr_user = User(user[0], user[1], user[2], user[3], role)
            if role == -1:
                print("Student logged in")
            else:
                print("Staff logged in")
```

```
            parent.destroy()
            home(root, curr_user, con)

        else:
            print("logging failed")
            messagebox.showerror("Login Failed", "Invalid username or password")

    threading.Thread(target=connect_and_authenticate).start()

def on_focus_in(entry, placeholder):
    if entry.get() == placeholder:
        entry.delete(0, 'end')

def on_focus_out(entry, placeholder):
    if entry.get() == "":
        entry.insert(0, placeholder)
        if placeholder == 'Password\t':
            entry.configure(show="")
    else:
        if placeholder == 'Password\t':
            entry.configure(show="*")

class Login(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent, style="Card.TFrame", padding=15)

        self.columnconfigure(0, weight=1)
        threading.Thread(target=connect).start()
        self.add_widgets(parent)

    def add_widgets(self, parent):
        self.email = ttk.Entry(self)
        self.email.insert(0, "Email\t")
        self.email.grid(row=0, column=0, padx=5, pady=(0, 10), sticky="ew")

        self.password = ttk.Entry(self)
        self.password.insert(0, "Password\t")
        self.password.grid(row=2, column=0, padx=5, pady=(0, 10), sticky="ew")
```

```
        self.email.bind('<Button-1>', lambda x: on_focus_in(self.email, "Email\t"))
        self.email.bind('<FocusOut>', lambda x: on_focus_out(self.email, "Email\t"))

        self.password.bind('<Button-1>', lambda x: on_focus_in(self.password, "Password\t"))
        self.password.bind('<FocusOut>', lambda x: on_focus_out(self.password, "Password\t"))

        self.separator = ttk.Separator(self)
        self.separator.grid(row=5, column=0, pady=10, sticky="ew")

        self.login = ttk.Button(self, text="LOG IN", style="Accent.TButton", command=lambda:
authentication(self.email.get(), self.password.get(), parent))
        self.login.grid(row=7, column=0, padx=5, pady=10, sticky="ew")

class App(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent, padding=15)

        label = ttk.Label(self, text="")
        label.grid(row=0, column=0)
        big_font_label = ttk.Label(self, text="REC", font=("Arial", 30, "bold"),
foreground="#56C8FF")
        big_font_label.grid(row=1, column=1)
        Login(self).grid(row=2, column=1, padx=10, pady=(10, 0), sticky="nsew")
        label = ttk.Label(self, text="")
        label.grid(row=3, column=2)
        self.grid_rowconfigure(0, weight=1)
        self.grid_rowconfigure(2, weight=0, minsize=200)
        self.grid_rowconfigure(3, weight=1)
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=0, minsize=400)
        self.grid_columnconfigure(2, weight=1)

root = tkinter.Tk()
root.title("Login")

sv_ttk.set_theme("dark")
App(root).pack(expand=True, fill="both")

root.mainloop()
```

home.py

```
"""A demo script to showcase the Sun Valley ttk theme."""

import tkinter
from tkinter import ttk
from configs import config
from addod import addwindow
from tkcalendar import DateEntry
import threading

class ScrollableFrame(ttk.Frame):
    def __init__(self, container, *args, **kwargs):
        super().__init__(container, *args, **kwargs)
        canvas = tkinter.Canvas(self)
        scrollbar = ttk.Scrollbar(self, orient="vertical", command=canvas.yview)
        self.scrollable_frame = ttk.Frame(canvas)

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

        canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")

        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")
class Table:
    def __init__(self, tab,lst,n,m):
        for i in range(n):
            for j in range(m):
                self.e = ttk.Entry(tab, width=20,font=('Arial', 16, 'bold'))
                self.e.grid(row=i+1, column=j)
                self.e.insert('end', lst[i][j])
```

```
            self.e.config(state='disabled')

def approval(id,i,window,notebook,curr_user,tbl):
    def approv():
        con=config()
        cur=con.cursor()
        if i==-1:
            cur.execute(f"update {tbl} set status=-1,rejected_by={curr_user.role} where id={id}")
        else:
            cur.execute(f"update {tbl} set status=status+{i} where id={id}")
        con.commit()
        window.destroy()
        notebook.refresh_data(curr_user,con)
        print("approved")
    threading.Thread(target=approv).start()

def delete(id,tbl,window,notebook,curr_user):
    def delete1():
        con=config()
        cur=con.cursor()
        cur.execute(f"Delete from {tbl} where id={id}")
        con.commit()
        window.destroy()
        notebook.refresh_data(curr_user, con)
        print("Canceled")
    threading.Thread(target=delete1).start()

def show_record_info(tree,root,curr_user,notebook,tbl):
    selected_item = tree.focus()
    print(1)
    if selected_item:
        record = tree.item(selected_item)
        info_window = tkinter.Toplevel(root)
        info_window.title("Record Information")
        info_frame = ttk.Frame(info_window, padding=20)
        info_frame.pack(fill=tkinter.BOTH, expand=True)
        rn,dept,sec,name1=record['text'].split(' - ')

        name = ttk.Label(info_frame, text="Name:", style="email.TLabel")
```

```
    name.grid(row=0, column=0, sticky='nw')
    name_value = ttk.Label(info_frame, text=name1, style="email.TLabel")
    name_value.grid(row=0, column=1, sticky='w')

    rollno = ttk.Label(info_frame, text="College ID:", style="email.TLabel")
    rollno.grid(row=1, column=0, sticky='nw')
    rollno_values = ttk.Label(info_frame, text=rn, style="email.TLabel")
    rollno_values.grid(row=1, column=1, sticky='w')

    department = ttk.Label(info_frame, text="Department:", style="email.TLabel")
    department.grid(row=2, column=0, sticky='nw')
    department_value = ttk.Label(info_frame, text=dept, wraplength=400,
style="email.TLabel")
    department_value.grid(row=2, column=1, sticky='w')

    section = ttk.Label(info_frame, text="Section:", style="email.TLabel")
    section.grid(row=3, column=0, sticky='nw')
    section_value = ttk.Label(info_frame, text=sec, wraplength=400, style="email.TLabel")
    section_value.grid(row=3, column=1, sticky='w')

    created = ttk.Label(info_frame, text="Created on:", style="email.TLabel")
    created.grid(row=4, column=0, sticky='nw')
    created_value = ttk.Label(info_frame, text=record['values'][3], wraplength=400,
style="email.TLabel")
    created_value.grid(row=4, column=1, sticky='w')

    from_date = ttk.Label(info_frame, text="From :", style="email.TLabel")
    from_date.grid(row=5, column=0, sticky='nw')
    from_date_value = ttk.Label(info_frame, text=record['values'][1], wraplength=400,
style="email.TLabel")
    from_date_value.grid(row=5, column=1, sticky='w')

    to_date = ttk.Label(info_frame, text="To :", style="email.TLabel")
    to_date.grid(row=6, column=0, sticky='nw')
    to_date_value = ttk.Label(info_frame, text=record['values'][2], wraplength=400,
style="email.TLabel")
    to_date_value.grid(row=6, column=1, sticky='w')

    email = ttk.Label(info_frame, text="Event :", style="email.TLabel")
```

```
    email.grid(row=7, column=0, sticky='nw')
    email_value = ttk.Label(info_frame, text=record['values'][0], wraplength=400,
style="email.TLabel")
    email_value.grid(row=7, column=1, sticky='w')

    desc = ttk.Label(info_frame, text="Description :", style="email.TLabel")
    desc.grid(row=8, column=0, sticky='nw')
    desc_value = ttk.Label(info_frame, text=record['values'][5], wraplength=400,
style="email.TLabel")
    desc_value.grid(row=8, column=1, sticky='w')
    role={1:"Incharge",2:"Counselor",3:"HoD"}
    stats=""
    if(record['values'][6]==-1):
        stats="Rejected by "+role[record['values'][7]]
    else:
        for i in (1,2,3):
            if(record['values'][6]<i):
                stats=stats+role[i]+" : "+"Waiting for approval\n"
            else:
                stats=stats+role[i]+" : "+"Approved\n"
    status = ttk.Label(info_frame, text="Status :", style="email.TLabel")
    status.grid(row=9, column=0, sticky='nw')
    status_value = ttk.Label(info_frame, text=stats, wraplength=400, style="email.TLabel")
    status_value.grid(row=9, column=1, sticky='w')
    if (record['values'][6] == 0 and curr_user.role==1) or (record['values'][6] == 1 and
curr_user.role==2) or (record['values'][6] == 2 and curr_user.role==3):
        reject = ttk.Button(info_window, text="Reject",style="Accent.TButton",
command=lambda: approval(selected_item,-1,info_window,notebook,curr_user,tbl))
        reject.pack()
        approve = ttk.Button(info_window, text="Approve",style="Accent.TButton",
command=lambda: approval(selected_item,1,info_window,notebook,curr_user,tbl))
        approve.pack()
    if(record['values'][6]>=0 and record['values'][6]<=2 and curr_user.role==-1):
        cancel = ttk.Button(info_window, text="Cancel",
style="Accent.TButton",command=lambda:
delete(selected_item,tbl,info_window,notebook,curr_user))
        cancel.pack()
    print(5)
```

```
class PanedDemo(ttk.PanedWindow):
    def __init__(self, parent,root,curr_user,con):
        super().__init__(parent)


        self.var = tkinter.IntVar(self, 47)

        self.add_widgets(root,curr_user,parent,con)

    def settabs(self,root,curr_user,tab,tbl,sbar,tree):
        sbar.pack(side="right", fill="y")

        sbar.config(command=tree.yview)

        tree.pack(fill="both", expand=True)
        tree.column("#0", anchor='w', width=200, stretch='no')
        tree.column("1", anchor='n')
        tree.column("2", anchor='w', width=100, stretch='no')
        tree.column("3", anchor='w', width=100, stretch='no')
        tree.column("4", anchor='w', width=100, stretch='no')
        tree.column("5", anchor='w', width=120, stretch='no')
        tree.heading("#0", text="Student")
        tree.heading(1, text="Event")
        tree.heading(2, text="From")
        tree.heading(3, text="To")
        tree.heading(4, text="Created on")
        tree.heading(5, text="Status")
        tree.bind("<Double-1>", lambda event: show_record_info(tree, root, curr_user, self,tbl))
        if curr_user.role == -1:
            self.addbutton = ttk.Button(tab, text="Create",
style="Accent.TButton",command=lambda: addwindow(root, curr_user, self,tbl))
            self.addbutton.pack()
    def settable(self,sframe,date,tbl,curr_user):
        con=config()
        cur=con.cursor()
        if(curr_user.role==1):
            cur.execute(f"select {tbl}.user_id,users.username,dept.name||'-'||class.sec,subject from
{tbl} join students s on {tbl}.user_id=s.user_id join class on class.id=s.class_id join dept on
```

```
dept.id=class.dept_id join users on users.id={tbl}.user_id where '{date}'>= from_date AND
'{date}' <= to_date AND class.incharge_id={curr_user.id} AND status=3")
        elif(curr_user.role == 2):
            cur.execute(
                f"select {tbl}.user_id,users.username,dept.name||'-'||class.sec,subject from {tbl} join
students s on {tbl}.user_id=s.user_id join class on class.id=s.class_id join dept on
dept.id=class.dept_id join users on users.id={tbl}.user_id where '{date}'>= from_date AND
'{date}' <= to_date AND class.counselor_id={curr_user.id} AND status=3")
        elif(curr_user.role == 3):
            cur.execute(
                f"select {tbl}.user_id,users.username,dept.name||'-'||class.sec,subject from {tbl} join
students s on {tbl}.user_id=s.user_id join class on class.id=s.class_id join dept on
dept.id=class.dept_id join users on users.id={tbl}.user_id where '{date}'>= from_date AND
'{date}' <= to_date AND dept.hod_id={curr_user.id} AND status=3")
        data=cur.fetchall()
        for widgets in sframe.winfo_children()[3:]:
            widgets.destroy()
        print(data)
        if(data):
            Table(sframe.scrollable_frame,data,len(data),len(data[0]))
        count=ttk.Label(sframe.scrollable_frame,text=f"The Total number of students :
{len(data)}")
        count.grid(row=len(data)+1,column=0)
        sframe.grid(row=1,column=0,columnspan=2,sticky="nsew")
        con.close()

    def add_widgets(self,root,curr_user,parent,con):
        self.notebook = ttk.Notebook(root)
        self.notebook.pack(expand=True, fill="both")
        self.tab_1=ttk.Frame(self.notebook)
        self.notebook.add(self.tab_1,text="OD")
        self.tab_2 = ttk.Frame(self.notebook)
        self.notebook.add(self.tab_2, text="LEAVE")
        if(curr_user.role>0):
            self.tab_3 = tkinter.Frame(self.notebook)
            self.notebook.add(self.tab_3, text="OD STATS")
            self.tab_4 = ttk.Frame(self.notebook)
            self.notebook.add(self.tab_4, text="LEAVE STATS")
            self.od_date=DateEntry(self.tab_3)
```

```
        self.od_date.grid(row=0, column=0, sticky="w", padx=5, pady=(0, 10))
        self.od_date_button = ttk.Button(self.tab_3, text="Search",
style="Accent.TButton",command=lambda:
self.settable(ScrollableFrame(self.tab_3),self.od_date.get(),'od',curr_user))
        self.od_date_button.grid(row=0,column=1, sticky="w", padx=5, pady=(0, 10))
        self.tab_3.columnconfigure(1, weight=1)
        self.tab_3.rowconfigure(1, weight=1)
        self.tab_4.columnconfigure(1, weight=1)
        self.tab_4.rowconfigure(1, weight=1)
        self.leave_date = DateEntry(self.tab_4)
        self.leave_date.grid(row=0, column=0, sticky="w", padx=5, pady=(0, 10))
        self.leave_date_button = ttk.Button(self.tab_4, text="Search",
style="Accent.TButton",command=lambda:
self.settable(ScrollableFrame(self.tab_4),self.leave_date.get(),'leave',curr_user))
        self.leave_date_button.grid(row=0,column=1, sticky="w", padx=5, pady=(0, 10))

    self.scrollbar = ttk.Scrollbar(self.tab_1)
    self.tree = ttk.Treeview(
        self.tab_1,
        columns=(1, 2, 3, 4, 5),
        height=11,
        selectmode="browse",
        yscrollcommand=self.scrollbar.set,
    )
    self.scrollbar1 = ttk.Scrollbar(self.tab_2)
    self.tree1 = ttk.Treeview(
        self.tab_2,
        columns=(1, 2, 3, 4, 5),
        height=11,
        selectmode="browse",
        yscrollcommand=self.scrollbar.set
    )
    self.settabs(root, curr_user,self.tab_1, 'OD',self.scrollbar,self.tree)
    self.settabs(root, curr_user,self.tab_2, 'LEAVE',self.scrollbar1,self.tree1)
    self.refreshButton = ttk.Button(root, text="Refresh",style="Accent.TButton",
command=lambda :self.refresh_data(curr_user,config()))
    self.refreshButton.pack()
    self.refresh_data(curr_user,con)
```

```
    def refresh_data(self, curr_user, con):
        print("refresh clicked")
        cur = con.cursor()
        user_id=curr_user.id
        # Combined query for both 'od' and 'leave'
        queries = {
            -1: f"""
                SELECT 'od' AS type, od.id, od.user_id||' - '||d.name||' - '||class.sec||' - '||u.username,
subject,
                    TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                    TO_CHAR(created_date,'YYYY-MM-DD'),
                    CASE WHEN status=-1 THEN 'Rejected'
                        WHEN status=3 THEN 'Approved'
                        ELSE 'Pending'
                    END, description, status,rejected_by
                FROM od
                JOIN students on od.user_id = students.user_id
                JOIN class ON class.id = students.class_id
                JOIN dept d ON class.dept_id = d.id
                JOIN users u ON od.user_id=u.id
                WHERE od.user_id = {user_id}

                UNION ALL

                SELECT 'leave' AS type, leave.id, leave.user_id||' - '||d.name||' - '||class.sec||' -
'||u.username, subject,
                    TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                    TO_CHAR(created_date,'YYYY-MM-DD'),
                    CASE WHEN status=-1 THEN 'Rejected'
                        WHEN status=3 THEN 'Approved'
                        ELSE 'Pending'
                    END, description, status,rejected_by
                FROM leave
                JOIN students on leave.user_id = students.user_id
                JOIN class ON class.id = students.class_id
                JOIN dept d ON class.dept_id = d.id
                JOIN users u ON leave.user_id=u.id
```

```
            WHERE leave.user_id = {user_id}
        """,
        1:f"""
            SELECT 'od' AS type, od.id, od.user_id||' - '||d.name||' - '||class.sec||' - '||u.username,
subject,
                TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                TO_CHAR(created_date,'YYYY-MM-DD'),
                CASE WHEN status=-1 THEN 'Rejected'
                    WHEN status=3 THEN 'Approved'
                    WHEN status=0 THEN 'Requested'
                    ELSE 'Approved by Me'
                END, description, status,rejected_by
            FROM od
            JOIN students on od.user_id = students.user_id
            JOIN class ON class.id = students.class_id
            JOIN dept d ON d.id = class.dept_id
            JOIN users u ON od.user_id=u.id
            WHERE class.incharge_id = {user_id} AND (status>=0 OR rejected_by >= 1)

            UNION ALL

            SELECT 'leave' AS type, leave.id, leave.user_id||' - '||d.name||' - '||class.sec||' -
'||u.username, subject,
                TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                TO_CHAR(created_date,'YYYY-MM-DD'),
                CASE WHEN status=-1 THEN 'Rejected'
                    WHEN status=3 THEN 'Approved'
                    WHEN status=0 THEN 'Requested'
                    ELSE 'Approved by Me'
                END, description, status,rejected_by
            FROM leave
            JOIN students on leave.user_id = students.user_id
            JOIN class ON class.id = students.class_id
            JOIN dept d ON d.id = class.dept_id
            JOIN users u ON leave.user_id=u.id
            WHERE class.incharge_id = {user_id} AND (status>=0 OR rejected_by >= 1)
        """,
```

```
        2: f"""
            SELECT 'od' AS type, od.id, od.user_id||' - '||d.name||' - '||class.sec||' - '||u.username,
subject,
                TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                TO_CHAR(created_date,'YYYY-MM-DD'),
          CASE WHEN status=-1 THEN 'Rejected'
              WHEN status=3 THEN 'Approved'
              WHEN status=1 THEN 'Requested'
              ELSE 'Approved by Me'
          END, description, status,rejected_by
      FROM od
            JOIN students on od.user_id = students.user_id
            JOIN class ON class.id = students.class_id
            JOIN dept d ON d.id = class.dept_id
            JOIN users u ON od.user_id=u.id
            WHERE class.counselor_id = {user_id} AND (status >= 1 OR rejected_by >= 2)

            UNION ALL

            SELECT 'leave' AS type, leave.id, leave.user_id||' - '||d.name||' - '||class.sec||' -
'||u.username, subject,
                TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                TO_CHAR(created_date,'YYYY-MM-DD'),
                CASE WHEN status=-1 THEN 'Rejected'
                    WHEN status=3 THEN 'Approved'
                    WHEN status=1 THEN 'Requested'
                    ELSE 'Approved by Me'
                END, description, status,rejected_by
            FROM leave
            JOIN students on leave.user_id = students.user_id
            JOIN class ON class.id = students.class_id
            JOIN dept d ON d.id = class.dept_id
            JOIN users u ON leave.user_id=u.id
            WHERE class.counselor_id = {user_id} AND (status >= 1 OR rejected_by >= 2)
        """,
        3: f"""
```

```
            SELECT 'od' AS type, od.id, od.user_id||' - '||d.name||' - '||class.sec||' - '||u.username,
subject,
                TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                TO_CHAR(created_date,'YYYY-MM-DD'),
                CASE WHEN status=-1 THEN 'Rejected'
                    WHEN status=3 THEN 'Approved'
                    WHEN status=2 THEN 'Requested'
                    ELSE 'Approved by Me'
                END, description, status,rejected_by
            FROM od
            JOIN students on od.user_id = students.user_id
            JOIN class ON class.id = students.class_id
            JOIN dept d ON d.id = class.dept_id
            JOIN users u ON od.user_id=u.id
            WHERE d.hod_id = {user_id} AND (status >= 2 OR rejected_by = 3)

            UNION ALL

            SELECT 'leave' AS type, leave.id, leave.user_id||' - '||d.name||' - '||class.sec||' -
'||u.username, subject,
                TO_CHAR(from_date,'YYYY-MM-DD'), TO_CHAR(to_date,'YYYY-MM-
DD'),
                TO_CHAR(created_date,'YYYY-MM-DD'),
                CASE WHEN status=-1 THEN 'Rejected'
                    WHEN status=3 THEN 'Approved'
                    WHEN status=2 THEN 'Requested'
                    ELSE 'Approved by Me'
                END, description, status,rejected_by
            FROM leave
            JOIN students on leave.user_id = students.user_id
            JOIN class ON class.id = students.class_id
            JOIN dept d ON d.id = class.dept_id
            JOIN users u ON leave.user_id=u.id
            WHERE d.hod_id = {user_id} AND (status >= 2 OR rejected_by = 3)
        """,
        0:f"SELECT NULL Where False"
    }
```

```
        cur.execute(queries[curr_user.role])
        data = cur.fetchall()
        con.close()

        print("Data fetched, updating trees")

        # Batch update for both tree views
        tree_data = {
            'od': [],
            'leave': []
        }
        for row in data:
            type, *values = row
            tree_data[type].append(("", values[0], values[1], values[2:10]))
        print(tree_data)
        if self.tree.get_children():
            self.tree.delete(*self.tree.get_children())
        for item in tree_data['od']:
            parnt, iid, text, values = item
            tag = self.get_tag(values[6], curr_user.role)
            self.tree.insert(parent=parnt, index="end", iid=iid, text=text, values=values, tags=[tag])

        if self.tree1.get_children():
            self.tree1.delete(*self.tree1.get_children())
        for item in tree_data['leave']:
            parnt, iid, text, values = item
            tag = self.get_tag(values[6], curr_user.role)
            self.tree1.insert(parent=parnt, index="end", iid=iid, text=text, values=values,
tags=[tag])

        # Tag configuration
        self.configure_tags(self.tree)
        self.configure_tags(self.tree1)
        self.after(60000, self.refresh_data, curr_user,config())

    def get_tag(self, status, role):
        if status == (role - 1):
            return 'unapproved'
        elif status == 3:
```

```
            return 'approved'
        elif status == -1:
            return 'rejected'
        else:
            return ''

    def configure_tags(self, tree):
        tree.tag_configure('unapproved', background='#57C7FF', foreground='#000000')
        tree.tag_configure('approved', background='#1CD760', foreground='#000000')
        tree.tag_configure('rejected', background='#EA4938', foreground='#000000')


class App(ttk.Frame):
    def __init__(self, parent,curr_user,con):
        super().__init__(parent, padding=15)

        for index in range(2):
            self.columnconfigure(index, weight=1)
            self.rowconfigure(index, weight=1)

        PanedDemo(self,parent,curr_user,con).grid(row=0, column=1, padx=10, pady=(10, 0),
sticky="nsew")


def home(root,curr_user,con):
    root.title(curr_user.name)
    App(root,curr_user,con).pack(expand=True, fill="both")
```

addod.py

```
import tkinter
from tkinter import ttk
from tkcalendar import DateEntry
from configs import config
import threading

class OdAdd(ttk.Frame):
    def __init__(self, parent,curr_user,info,notebook,tbl):
```

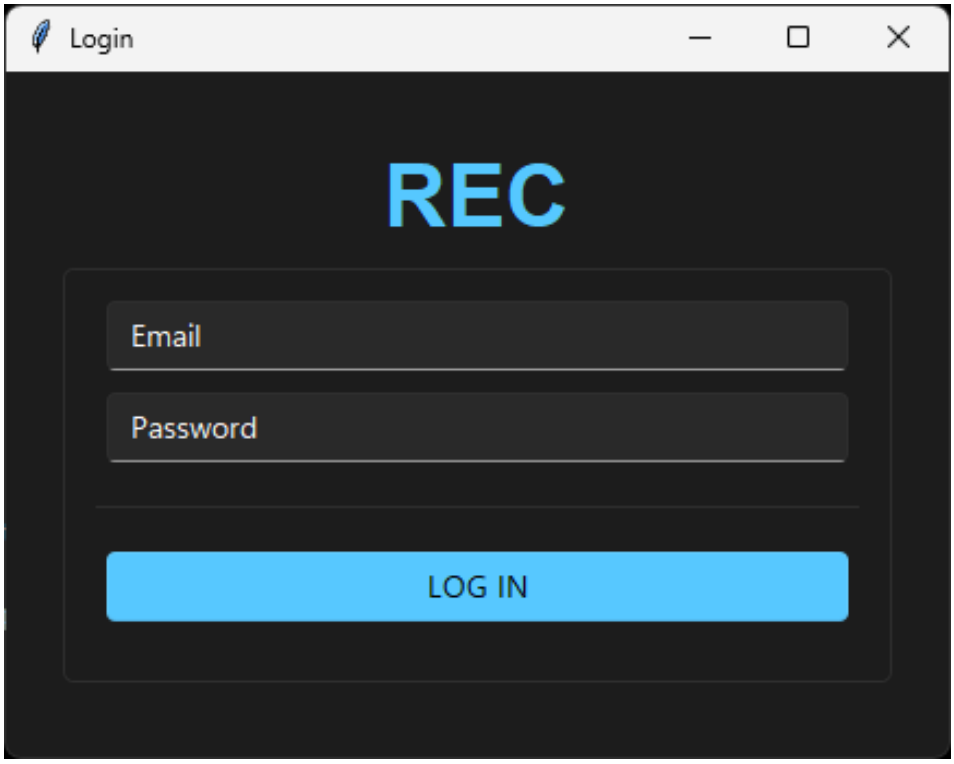
```
        super().__init__(parent, style="Card.TFrame", padding=15)

        self.columnconfigure(0, weight=1)

        self.add_widgets(parent,curr_user,info,notebook,tbl)

    def add_od_leave(self,todate,fromdate,subject,desc,curr_user,info,notebook,tbl):
        def add_odleave():
            info.destroy()
            print(subject)
            t = todate.split('/')
            f = fromdate.split('/')
            to_date = '20' + t[2] + '-' + t[0] + '-' + t[1]
            from_date = '20' + f[2] + '-' + f[0] + '-' + f[1]
            con = config()
            cur = con.cursor()
            cur.execute(
                f"insert into {tbl}
values(default,{curr_user.id},default,'{from_date}','{to_date}','{subject}','{desc}',default,null);
")
            con.commit()
            notebook.refresh_data(curr_user, con)
        threading.Thread(target=add_odleave).start()

    def add_widgets(self,parent,curr_user,info,notebook,tbl):

        self.fromdate = ttk.Label(self, text="From:")
        self.fromdate.grid(row=0, column=0, sticky="ew", padx=5, pady=(0, 10))

        self.from_entry=DateEntry(self,width=50)
        self.from_entry.grid(row=0, column=1, sticky="ew", padx=5, pady=(0, 10))

        self.to = ttk.Label(self, text="To:")
        self.to.grid(row=1, column=0, sticky="ew",  padx=5, pady=(0, 10))

        self.subject_label = ttk.Label(self, text="Subject:")
        self.subject_label.grid(row=2, column=0, sticky="ew",  padx=5, pady=(0, 10))

        # Entry widgets
```

```
        self.to_entry = DateEntry(self,width=50)
        self.to_entry.grid(row=1, column=1, padx=5, pady=(0, 10), sticky="ew")

        self.subject_entry = ttk.Entry(self, width=50)
        self.subject_entry.grid(row=2, column=1, padx=5, pady=(0, 10), sticky="ew")

        self.body_text = tkinter.Text(self, width=80,
height=20,highlightbackground="#8A8A8A",highlightcolor='#56C8FF',highlightthickness=1)
        self.body_text.grid(row=3, column=0,columnspan=2, padx=5, pady=(0, 10),sticky="ew")
        self.separator = ttk.Separator(self)
        self.separator.grid(row=5, column=0,columnspan=2, pady=10, sticky="ew")
        self.send_button = ttk.Button(self,
text="Send",style="Accent.TButton",command=lambda
:self.add_od_leave(self.to_entry.get(),self.from_entry.get(),self.subject_entry.get(),self.body_te
xt.get("1.0", "end-1c"),curr_user,info,notebook,tbl))
        self.send_button.grid(row=7, column=0,columnspan=2, padx=5, pady=10, sticky="ew")
class App(ttk.Frame):
    def __init__(self, parent,curr_user,notebook,tbl):
        super().__init__(parent, padding=15)
        label = ttk.Label(self, text="")
        label.grid(row=0, column=0)
        big_font_label = ttk.Label( self,text=f"APPLY {tbl}", font=("Arial", 30, "bold"),
foreground="#56C8FF")
        big_font_label.grid(row=1,column=1)
        OdAdd(self,curr_user,parent,notebook,tbl).grid(
            row=2, column=1, padx=10, pady=(10, 0), sticky="nsew",)
        label = ttk.Label(self, text="")
        label.grid(row=3, column=2)
        self.grid_rowconfigure(0, weight=1)
        self.grid_rowconfigure(2, weight=0,minsize=200)
        self.grid_rowconfigure(3, weight=1)
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=0,minsize=400)
        self.grid_columnconfigure(2, weight=1)

def addwindow(root,curr_user,notebook,tbl):
    info = tkinter.Toplevel(root)
    info.title(f"Create {tbl}")
    App(info,curr_user,notebook,tbl).pack(expand=True, fill="both")
```

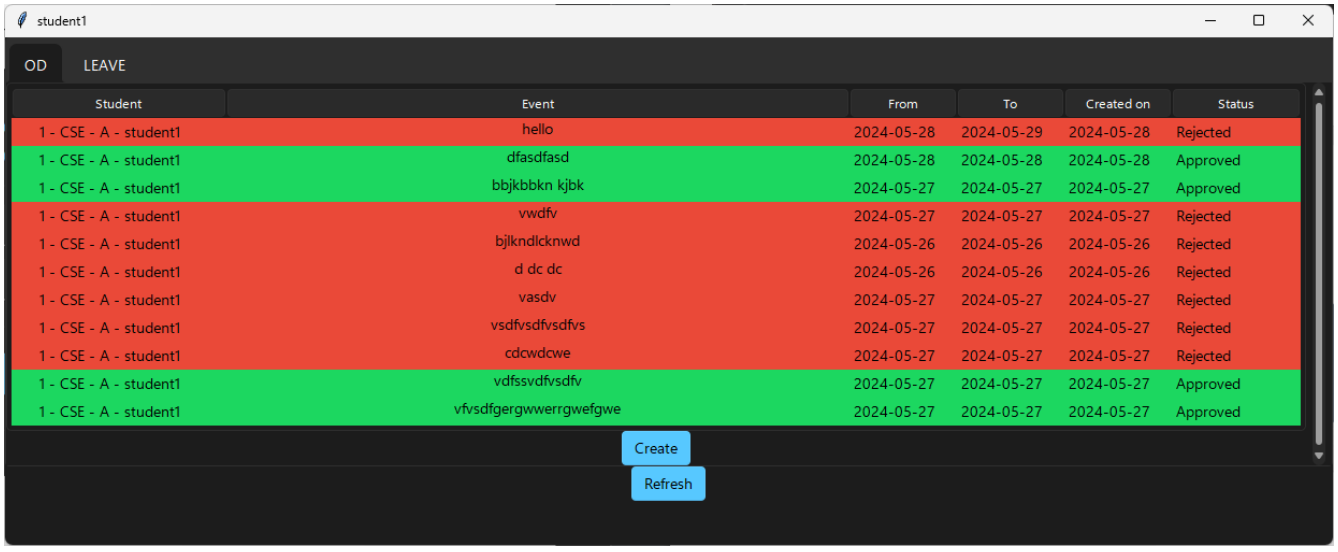
CHAPTER 5

RESULTS AND DISCUSSION

LOGIN PAGE

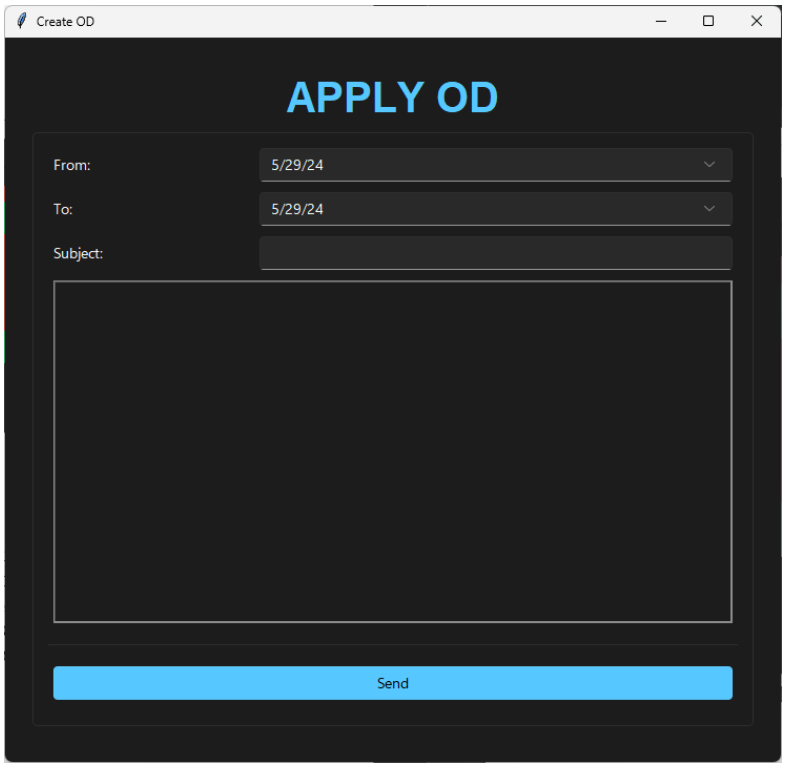


STUDENT STATUS

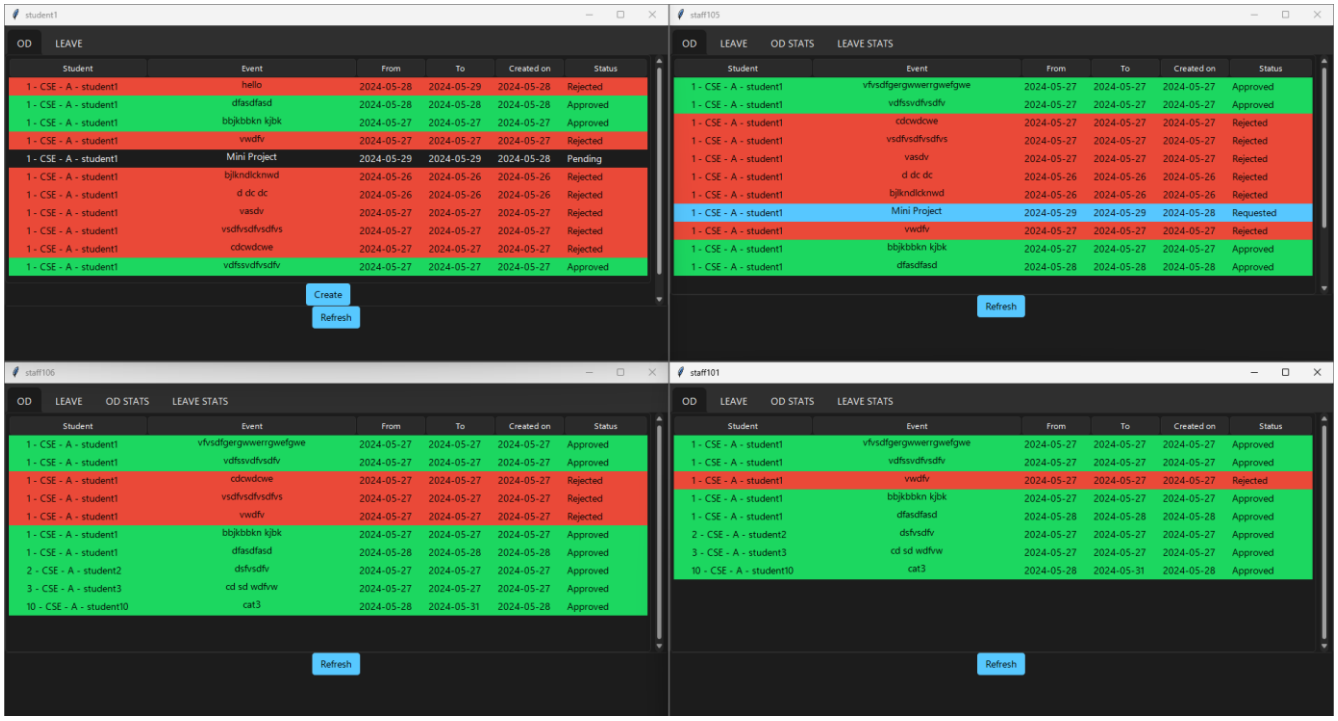


Student	Event	From	To	Created on	Status
1 - CSE - A - student1	hello	2024-05-28	2024-05-29	2024-05-28	Rejected
1 - CSE - A - student1	dfasdfasd	2024-05-28	2024-05-28	2024-05-28	Approved
1 - CSE - A - student1	bbjkbbkn kjbk	2024-05-27	2024-05-27	2024-05-27	Approved
1 - CSE - A - student1	vwdfv	2024-05-27	2024-05-27	2024-05-27	Rejected
1 - CSE - A - student1	bjlkndlcknwd	2024-05-26	2024-05-26	2024-05-26	Rejected
1 - CSE - A - student1	d dc dc	2024-05-26	2024-05-26	2024-05-26	Rejected
1 - CSE - A - student1	vasdv	2024-05-27	2024-05-27	2024-05-27	Rejected
1 - CSE - A - student1	vsdfvsdfvsdfvs	2024-05-27	2024-05-27	2024-05-27	Rejected
1 - CSE - A - student1	cdcwdcwe	2024-05-27	2024-05-27	2024-05-27	Rejected
1 - CSE - A - student1	vdfssvdfvsdfv	2024-05-27	2024-05-27	2024-05-27	Approved
1 - CSE - A - student1	vfvsdfgergwwerrgwefgwe	2024-05-27	2024-05-27	2024-05-27	Approved

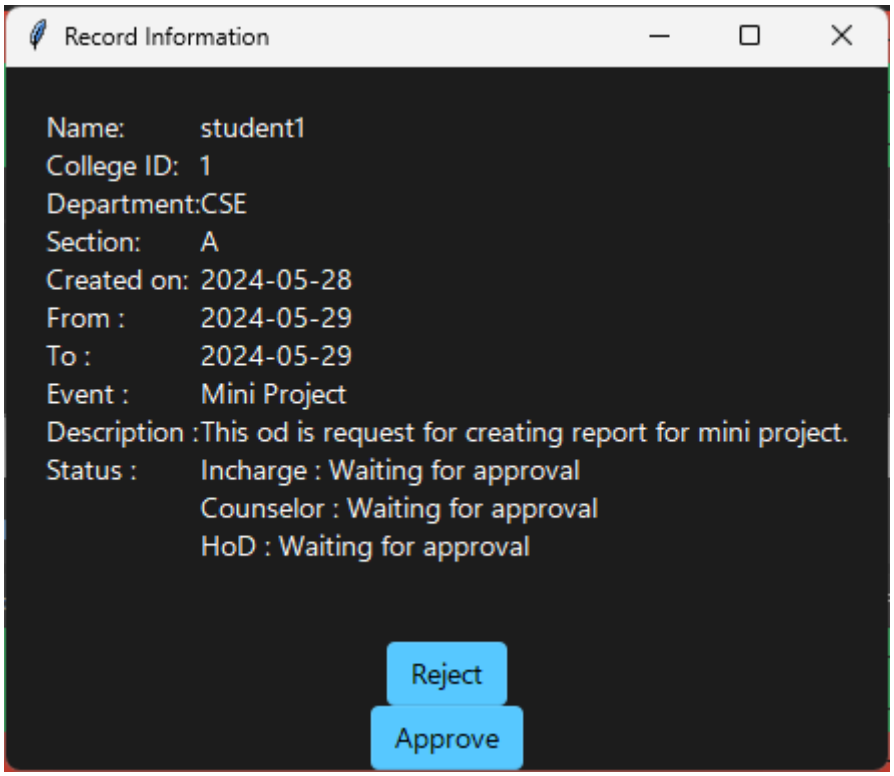
CREATING OD



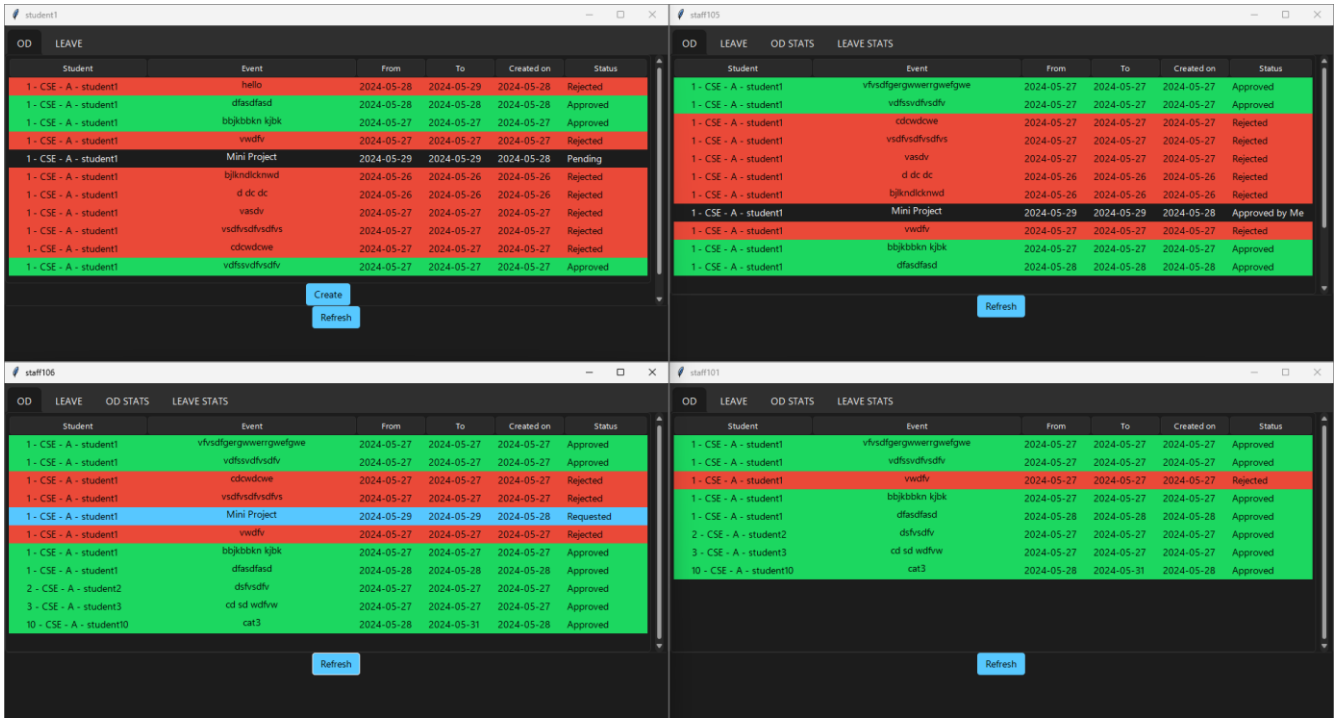
PENDING OD



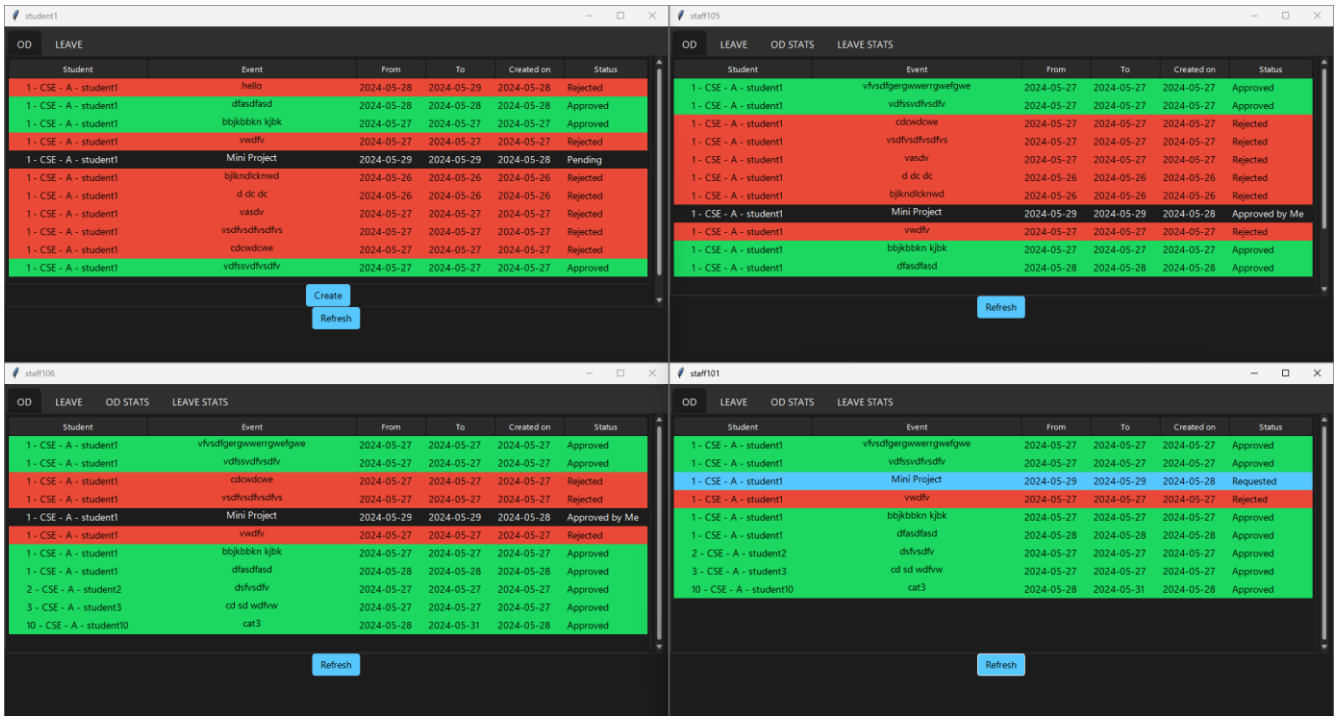
ACCEPT/DECLINE



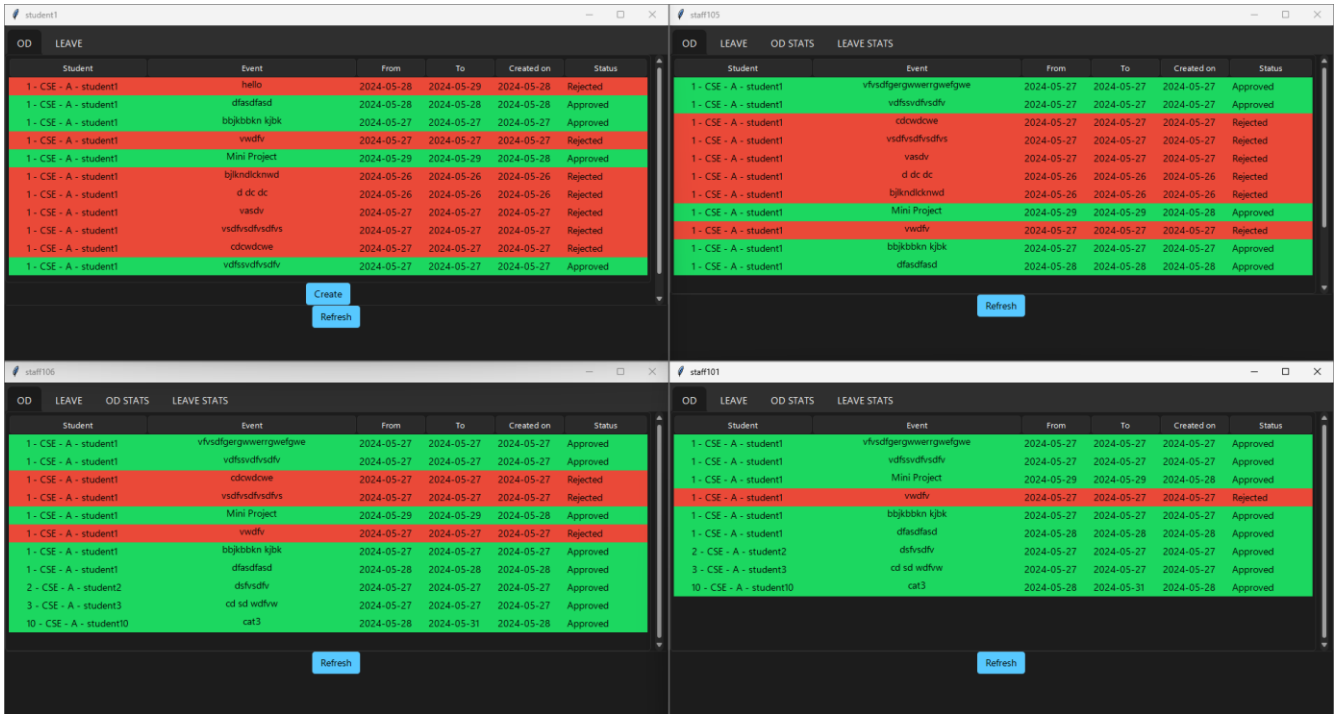
FORWARDING APPROVAL/DECLINE



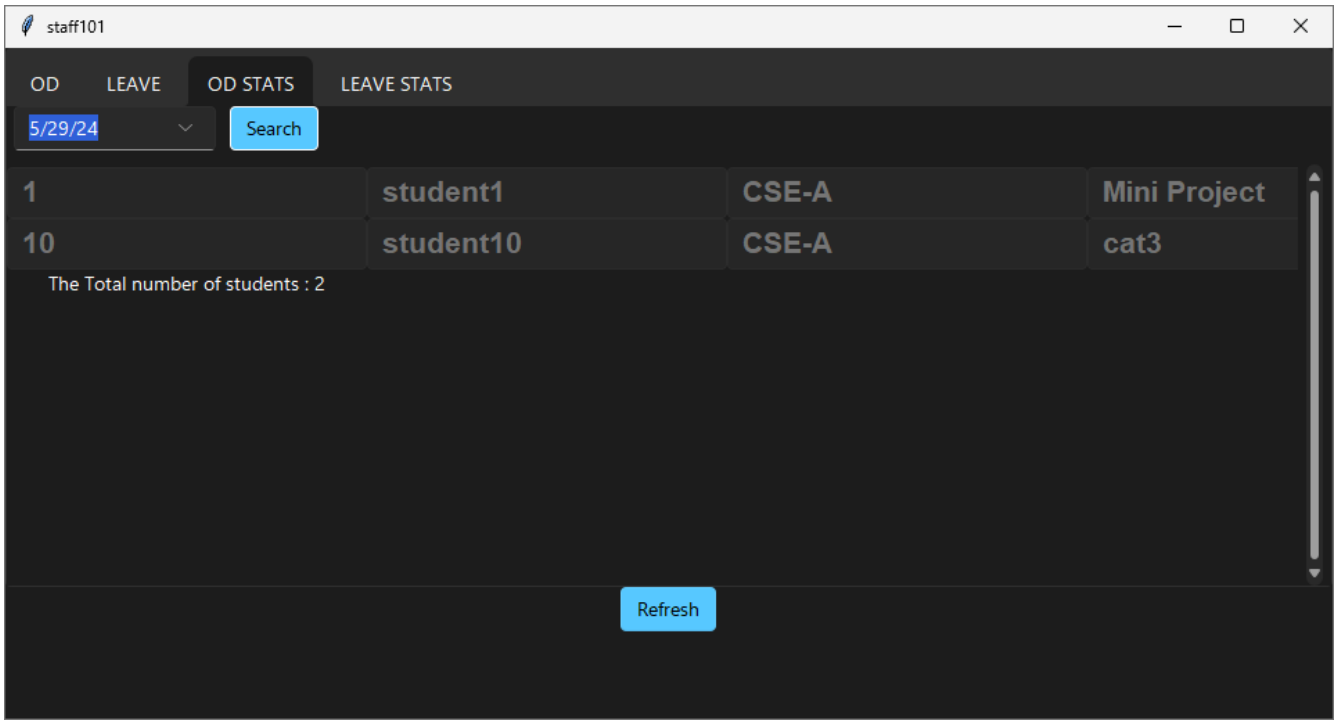
FORWARDING APPROVAL/DECLINE



FINAL APPROVAL



LIST OF STUDENTS OBTAINED OD



The OD Request Automation system successfully automates the request and tracking of OD requests to Class Incharge, Counsellor and Head of department. The system improves efficiency by reducing manual paperwork and streamlines communication between students and faculty. Each user role has been carefully integrated to ensure that the process flows smoothly from request submission to final approval or rejection. The system's GUI, built with Tkinter, provides an intuitive interface, making it easy for users to navigate and perform necessary actions.

CHAPTER 6

CONCLUSION

The OD Request Automation project achieves its objective of streamlining the OD request process within an educational institution. By utilizing Python for the backend and Tkinter for the GUI, along with PostgreSQL for robust data management.

CHAPTER 7

REFERENCES

- https://realpython.com/python-gui-tkinter/
- https://www.w3schools.com/postgresql/index.php
- https://github.com/rdbende/Sun-Valley-ttk-theme