import tkinter as tk
import itertools
import random

# Initialize a dictionary to store employees and their constraints
employee_constraints = {}

# Create the main GUI window
window = tk.Tk()
window.title("Employee Schedule")

def add_employee_and_constraint():
    employee_name = entry_name.get()
    employee_constraint = entry_constraint.get()

    if employee_name:
        if employee_name in employee_constraints:
            employee_constraints[employee_name].append(employee_constraint)
        else:
            employee_constraints[employee_name] = [employee_constraint]

        if f"Random Day Off for {employee_name}" not in random_day_buttons:
            row = 12
            random_day_button = tk.Button(
                window,
                text=f"Random Day Off for {employee_name}",
                command=lambda name=employee_name: set_random_day_off(name)
            )
            random_day_buttons[f"Random Day Off for {employee_name}"] = random_day_button
            random_day_button.grid(row=25+ len(random_day_buttons), column=0)

        remove_constraint_button = tk.Button(
            window,
            text=f"Remove Constraint for {employee_name}",
            command=lambda name=employee_name: remove_constraint(name)
        )
        remove_constraint_button.grid(row=25+ len(random_day_buttons), column=1)
        remove_constraint_buttons[f"Remove Constraint for {employee_name}"] = remove_constraint_button

    update_employee_list()

    entry_name.delete(0, tk.END)
    entry_constraint.delete(0, tk.END)

def removeconstraint(employee_name):
    if employee_name in employee_constraints and employee_constraints[employee_name]:
        employee_constraints[employee_name].pop()
    update_employee_list()

def set_random_day_off(employee_name):
    days_of_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

    if employee_name in employee_constraints:
        constraints = employee_constraints[employee_name]
    else:
        constraints = []

    # filter out days already in constraints
    available_days = [day for day in days_of_week if day not in constraints]

    # ...
if available_days:
    random_day_off = random.choice(available_days)

if employee_name in employee_constraints:
    employee_constraints[employee_name].append(random_day_off)
else:
    employee_constraints[employee_name] = [random_day_off]

update_employee_list()

# update the list of employees and constraints displayed in the GUI
def update_employee_list():
    employee_list.config(state=tk.NORMAL)
    employee_list.delete("1.0", tk.END)

    for employee, constraint in employee_constraints.items():
        employee_list.insert(tk.END, "Employee: {employee}, Constraints: {', '.join(constraint)}\n")

        employee_list.config(state=tk.DISABLED)

# Function to generate the schedule
def generate_schedule():
    employees = list(employee_constraints.keys())
    constraints = employee_constraints
    num_locations = int(entry_num_locations.get())  # gets the number of locations from the entry field.

    people_needed = {}
    for i in range(1, num_locations + 1):
        location_key = f'Location{i}'
        people_needed[location_key] = int(location_entries[location_key].get())

    # Call the code to generate the schedule
    schedule = generate_employee_schedule(employees, constraints, people_needed)

    # Display the schedule in a text box
    result_text.config(state=tk.NORMAL)
    result_text.delete("1.0", tk.END)  # Clear the previous result
    result_text.insert(tk.END, "Generated Schedule:\n")
    for day in schedule:
        for location, employees in schedule[day].items():
            result_text.insert(tk.END, f"{day}: {', '.join(employees)}\n")

    result_text.config(state=tk.DISABLED)

# Existing code for generating the employee schedule
def generate_employee_schedule(employees, constraints, people_needed):
    days_of_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
    locations = ['Location' + str(i) for i in range(1, len(people_needed) + 1)]

    # Initialize a dictionary to store the schedule
    schedule = {day: [location] for location in locations for day in days_of_week}

    # Shuffle employees for randomness
    random.shuffle(employees)

    # Make the schedule while considering constraints
    for day in days_of_week:
        day_constraints = [constraints.get(employee, []) for employee in employees]
available_employees = [employee for employee, constraint in zip(employees, day_constraints) if day not in constraint]

for location in locations:
    employees_needed = people_needed[location]

for _ in range(employees_needed):
    if available_employees:
        employee = available_employees.pop()
        schedule[day][location].append(employee)

return schedule

# Frame for columns 0-1
frame_left = tk.Frame(window)
frame_left.grid(row=0, column=0, columnspan=2, rowspan=25)

# Frame for columns 2-3
frame_right = tk.Frame(window)
frame_right.grid(row=0, column=2, columnspan=2, rowspan=25)

# input field for the number of locations in frame_left
num_locations_label = tk.Label(frame_left, text="Number of Locations:"
num_locations_label.grid(row=0, column=0)
entry_num_locations = tk.Entry(frame_left, width=10)
entry_num_locations.grid(row=0, column=1)

# Create entry fields for the number of people needed at each location in frame_left
location_entries = {}
for i in range(1, 3):  # Adjust the range based on your maximum number of locations
    location_label = tk.Label(frame_left, text=f"L {i} Personnel Needs:"
location_label.grid(row=i, column=0)
location_entry = tk.Entry(frame_left, width=10)
location_entry.grid(row=i, column=1)
location_entries[f"Location{i}"] = location_entry

# input field for employee name in frame_left
entry_name_label = tk.Label(frame_left, text="Employee Name:"
entry_name_label.grid(row=8, column=0)
entry_name = tk.Entry(frame_left, width=20)
entry_name.grid(row=8, column=1)

# input field for employee constraint in frame_left
entry_constraint_label = tk.Label(frame_left, text="Constraint:"
entry_constraint_label.grid(row=9, column=0)
entry_constraint = tk.Entry(frame_left, width=20)
entry_constraint.grid(row=9, column=1)

# Create a button to add employees and constraints in frame_left
add_button = tk.Button(frame_left, text="Add Employee and Constraint", command=add_employee_and_constraint)
add_button.grid(row=10, column=0, columnspan=2)

# Create a label for the list of employees and constraints in frame_left
employee_list_label = tk.Label(frame_left, text="Employees and Constraints:"
employee_list_label.grid(row=11, column=0, columnspan=2)

employee_list = tk.Text(frame_left, height=10, width=50, state=tk.DISABLED)
employee_list.grid(row=12, column=0, columnspan=2)

def clear_employee_list():

employee_constraints.clear()
update_employee_list()

# Create a "Clear" button in frame_left
clear_button = tk.Button(frame_left, text="Clear List", command=clear_employee_list)
clear_button.grid(row=13, column=0, columnspan=2)

# Create a button to generate schedule in frame_right
generate_button = tk.Button(frame_right, text="Generate Schedule", command=generate_schedule)
generate_button.grid(row=0, column=0, columnspan=2)

result_text = tk.Text(frame_right, height=20, width=50, state=tk.DISABLED)
result_text.grid(row=1, column=0, columnspan=2)

# A dictionary to store random day off buttons
random_day_buttons = {}

remove_constraint_buttons = {}

# Start the GUI event loop
window.mainloop()