**Assignment 3: Software Implementation - OO Project with GUI and Data storage**

1. UML Class Diagram:

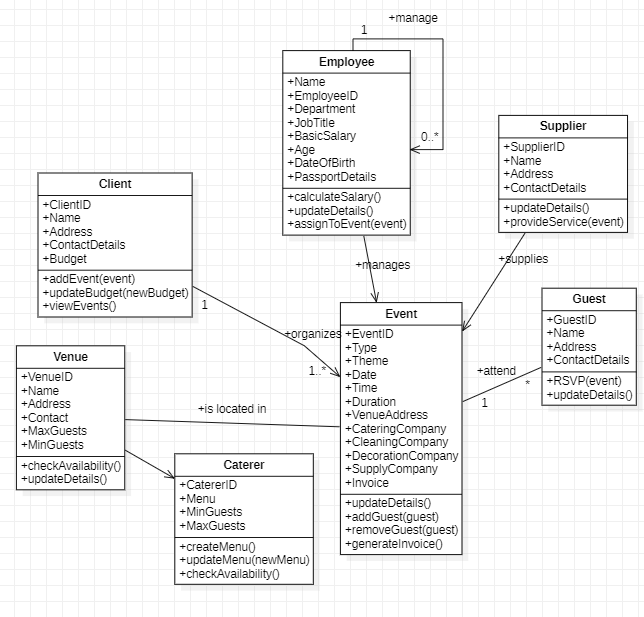
The class diagram represents the structure and relationships within the software system designed for The Best Events Company. At the core of the system are various entities represented as classes, each encapsulating relevant attributes and behaviors.

Employees play pivotal roles within the company, with distinct roles such as Sales Managers, Salespersons, Marketing Managers, Marketers, Accountants, Designers, and Handymen. They are managed by other employees, forming hierarchical relationships essential for organizational functioning.

Clients, essential for business, organize events and are central to the system. Guests attend these events, forming many-to-many relationships with events. Events, categorized into weddings, birthdays, themed parties, and graduations, encapsulate essential event details such as type, theme, date, and venue.

Suppliers provide crucial services such as catering, cleaning, decorations, and furniture supply. These services are linked to events, enabling seamless management of event logistics. Venues, where events are hosted, impose constraints such as capacity, further shaping event planning.

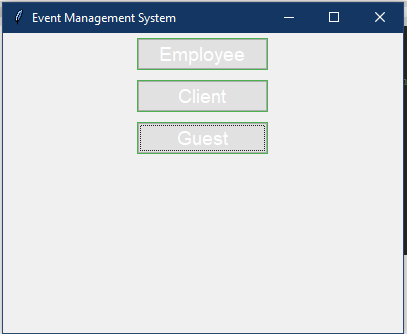
Overall, the class diagram provides a structured overview of the system, delineating entities, their attributes, behaviors, and interrelationships. It serves as a blueprint for developing a comprehensive software solution tailored to the needs of The Best Events Company, streamlining employee management, client interactions, event planning, and supplier coordination.



1. **Class Code:**
2. class Employee:  
    def \_\_init\_\_(self, name, employee\_id, department, job\_title, basic\_salary, age, date\_of\_birth, passport\_details):  
    self.\_name = name  
    self.\_employee\_id = employee\_id  
    self.\_department = department  
    self.\_job\_title = job\_title  
    self.\_basic\_salary = basic\_salary  
    self.\_age = age  
    self.\_date\_of\_birth = date\_of\_birth  
    self.\_passport\_details = passport\_details  
     
    # Getters  
    def get\_name(self):  
    return self.\_name  
     
    def get\_employee\_id(self):  
    return self.\_employee\_id  
     
    def get\_department(self):  
    return self.\_department  
     
    def get\_job\_title(self):  
    return self.\_job\_title  
     
    def get\_basic\_salary(self):  
    return self.\_basic\_salary  
     
    def get\_age(self):  
    return self.\_age  
     
    def get\_date\_of\_birth(self):  
    return self.\_date\_of\_birth  
     
    def get\_passport\_details(self):  
    return self.\_passport\_details  
     
    # Setters  
    def set\_department(self, department):  
    self.\_department = department  
     
    def set\_job\_title(self, job\_title):  
    self.\_job\_title = job\_title  
     
    def set\_basic\_salary(self, basic\_salary):  
    self.\_basic\_salary = basic\_salary  
     
    def set\_age(self, age):  
    self.\_age = age  
     
    def set\_date\_of\_birth(self, date\_of\_birth):  
    self.\_date\_of\_birth = date\_of\_birth  
     
    def set\_passport\_details(self, passport\_details):  
    self.\_passport\_details = passport\_details  
     
     
   class Client:  
    def \_\_init\_\_(self, client\_id, name, address, contact\_details, budget):  
    self.\_client\_id = client\_id  
    self.\_name = name  
    self.\_address = address  
    self.\_contact\_details = contact\_details  
    self.\_budget = budget  
     
    # Getters  
    def get\_client\_id(self):  
    return self.\_client\_id  
     
    def get\_name(self):  
    return self.\_name  
     
    def get\_address(self):  
    return self.\_address  
     
    def get\_contact\_details(self):  
    return self.\_contact\_details  
     
    def get\_budget(self):  
    return self.\_budget  
     
    # Setters  
    def set\_address(self, address):  
    self.\_address = address  
     
    def set\_contact\_details(self, contact\_details):  
    self.\_contact\_details = contact\_details  
     
    def set\_budget(self, budget):  
    self.\_budget = budget  
     
     
   class Guest:  
    def \_\_init\_\_(self, guest\_id, name, address, contact\_details):  
    self.\_guest\_id = guest\_id  
    self.\_name = name  
    self.\_address = address  
    self.\_contact\_details = contact\_details  
     
    # Getters  
    def get\_guest\_id(self):  
    return self.\_guest\_id  
     
    def get\_name(self):  
    return self.\_name  
     
    def get\_address(self):  
    return self.\_address  
     
    def get\_contact\_details(self):  
    return self.\_contact\_details  
     
    # Setters  
    def set\_address(self, address):  
    self.\_address = address  
     
    def set\_contact\_details(self, contact\_details):  
    self.\_contact\_details = contact\_details  
     
     
   class Event:  
    def \_\_init\_\_(self, event\_id, event\_type, theme, date, time, duration, venue\_address, catering\_company,  
    cleaning\_company, decorations\_company, entertainment\_company, furniture\_supply\_company, invoice):  
    self.\_event\_id = event\_id  
    self.\_event\_type = event\_type  
    self.\_theme = theme  
    self.\_date = date  
    self.\_time = time  
    self.\_duration = duration  
    self.\_venue\_address = venue\_address  
    self.\_catering\_company = catering\_company  
    self.\_cleaning\_company = cleaning\_company  
    self.\_decorations\_company = decorations\_company  
    self.\_entertainment\_company = entertainment\_company  
    self.\_furniture\_supply\_company = furniture\_supply\_company  
    self.\_invoice = invoice  
     
    # Getters  
    def get\_event\_id(self):  
    return self.\_event\_id  
     
    def get\_event\_type(self):  
    return self.\_event\_type  
     
    def get\_theme(self):  
    return self.\_theme  
     
    def get\_date(self):  
    return self.\_date  
     
    def get\_time(self):  
    return self.\_time  
     
    def get\_duration(self):  
    return self.\_duration  
     
    def get\_venue\_address(self):  
    return self.\_venue\_address  
     
    def get\_catering\_company(self):  
    return self.\_catering\_company  
     
    def get\_cleaning\_company(self):  
    return self.\_cleaning\_company  
     
    def get\_decorations\_company(self):  
    return self.\_decorations\_company  
     
    def get\_entertainment\_company(self):  
    return self.\_entertainment\_company  
     
    def get\_furniture\_supply\_company(self):  
    return self.\_furniture\_supply\_company  
     
    def get\_invoice(self):  
    return self.\_invoice  
     
    # Setters  
    def set\_theme(self, theme):  
    self.\_theme = theme  
     
    def set\_date(self, date):  
    self.\_date = date  
     
    def set\_time(self, time):  
    self.\_time = time  
     
    def set\_duration(self, duration):  
    self.\_duration = duration  
     
    def set\_venue\_address(self, venue\_address):  
    self.\_venue\_address = venue\_address  
     
    def set\_invoice(self, invoice):  
    self.\_invoice = invoice  
     
     
   class Venue:  
    def \_\_init\_\_(self, venue\_id, name, address, contact, min\_guests, max\_guests):  
    self.\_venue\_id = venue\_id  
    self.\_name = name  
    self.\_address = address  
    self.\_contact = contact  
    self.\_min\_guests = min\_guests  
    self.\_max\_guests = max\_guests  
     
    # Getters  
    def get\_venue\_id(self):  
    return self.\_venue\_id  
     
    def get\_name(self):  
    return self.\_name  
     
    def get\_address(self):  
    return self.\_address  
     
    def get\_contact(self):  
    return self.\_contact  
     
    def get\_min\_guests(self):  
    return self.\_min\_guests  
     
    def get\_max\_guests(self):  
    return self.\_max\_guests  
     
    # Setters  
    def set\_address(self, address):  
    self.\_address = address  
     
    def set\_contact(self, contact):  
    self.\_contact = contact  
     
     
   class Supplier:  
    def \_\_init\_\_(self, supplier\_id, name, address, contact\_details):  
    self.\_supplier\_id = supplier\_id  
    self.\_name = name  
    self.\_address = address  
    self.\_contact\_details = contact\_details  
     
    # Getters  
    def get\_supplier\_id(self):  
    return self.\_supplier\_id  
     
    def get\_name(self):  
    return self.\_name  
     
    def get\_address(self):  
    return self.\_address  
     
    def get\_contact\_details(self):  
    return self.\_contact\_details  
     
    # Setters  
    def set\_address(self, address):  
    self.\_address = address  
     
    def set\_contact\_details(self, contact\_details):  
    self.\_contact\_details = contact\_details  
     
     
   class Caterer:  
    def \_\_init\_\_(self, caterer\_id, name, address, contact\_details, menu, min\_guests, max\_guests):  
    self.\_caterer\_id = caterer\_id  
    self.\_name = name  
    self.\_address = address  
    self.\_contact\_details = contact\_details  
    self.\_menu = menu  
    self.\_min\_guests = min\_guests  
    self.\_max\_guests = max\_guests  
     
    # Getters  
    def get\_caterer\_id(self):  
    return self.\_caterer\_id  
     
    def get\_name(self):  
    return self.\_name  
     
    def get\_address(self):  
    return self.\_address  
     
    def get\_contact\_details(self):  
    return self.\_contact\_details  
     
    def get\_menu(self):  
    return self.\_menu  
     
    def get\_min\_guests(self):  
    return self.\_min\_guests  
     
    def get\_max\_guests(self):  
    return self.\_max\_guests  
     
    # Setters  
    def set\_address(self, address):  
    self.\_address = address  
     
    def set\_contact\_details(self, contact\_details):  
    self.\_contact\_details = contact\_details  
     
    def set\_menu(self, menu):  
    self.\_menu = menu  
     
   import tkinter as tk  
   from tkinter import ttk, messagebox  
   import pickle  
     
     
   class EmployeeManagement:  
    def \_\_init\_\_(self, master):  
    self.master = master  
    self.master.title("Employee Management System")  
     
    self.create\_employee\_frame = ttk.Frame(master)  
    self.create\_employee\_frame.pack(pady=10)  
     
    self.search\_employee\_frame = ttk.Frame(master)  
    self.search\_employee\_frame.pack(pady=10)  
     
    ttk.Label(self.create\_employee\_frame, text="Name:").grid(row=0, column=0, padx=5, pady=5)  
    self.name\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.name\_entry.grid(row=0, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_employee\_frame, text="Employee ID:").grid(row=1, column=0, padx=5, pady=5)  
    self.emp\_id\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.emp\_id\_entry.grid(row=1, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_employee\_frame, text="Department:").grid(row=2, column=0, padx=5, pady=5)  
    self.department\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.department\_entry.grid(row=2, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_employee\_frame, text="Job Title:").grid(row=3, column=0, padx=5, pady=5)  
    self.job\_title\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.job\_title\_entry.grid(row=3, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_employee\_frame, text="Basic Salary:").grid(row=4, column=0, padx=5, pady=5)  
    self.basic\_salary\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.basic\_salary\_entry.grid(row=4, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_employee\_frame, text="Age:").grid(row=5, column=0, padx=5, pady=5)  
    self.age\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.age\_entry.grid(row=5, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_employee\_frame, text="Date of Birth:").grid(row=6, column=0, padx=5, pady=5)  
    self.dob\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.dob\_entry.grid(row=6, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_employee\_frame, text="Passport Details:").grid(row=7, column=0, padx=5, pady=5)  
    self.passport\_entry = ttk.Entry(self.create\_employee\_frame)  
    self.passport\_entry.grid(row=7, column=1, padx=5, pady=5)  
     
    self.create\_employee\_button = ttk.Button(self.create\_employee\_frame, text="Create Employee", command=self.create\_employee)  
    self.create\_employee\_button.grid(row=8, columnspan=2, padx=5, pady=5)  
     
    ttk.Label(self.search\_employee\_frame, text="Search Employee by ID:").grid(row=0, column=0, padx=5, pady=5)  
    self.search\_emp\_id\_entry = ttk.Entry(self.search\_employee\_frame)  
    self.search\_emp\_id\_entry.grid(row=0, column=1, padx=5, pady=5)  
     
    self.search\_employee\_button = ttk.Button(self.search\_employee\_frame, text="Search", command=self.search\_employee)  
    self.search\_employee\_button.grid(row=1, columnspan=2, padx=5, pady=5)  
     
    def create\_employee(self):  
    name = self.name\_entry.get()  
    employee\_id = self.emp\_id\_entry.get()  
    department = self.department\_entry.get()  
    job\_title = self.job\_title\_entry.get()  
    basic\_salary = self.basic\_salary\_entry.get()  
    age = self.age\_entry.get()  
    date\_of\_birth = self.dob\_entry.get()  
    passport\_details = self.passport\_entry.get()  
     
    employee = Employee(name, employee\_id, department, job\_title, basic\_salary, age, date\_of\_birth, passport\_details)  
    self.save\_employee(employee)  
     
    def save\_employee(self, employee):  
    try:  
    with open('employees.pkl', 'ab') as file:  
    pickle.dump(employee, file)  
    print("Employee saved successfully!")  
    except Exception as e:  
    print("Error saving employee:", e)  
     
    def search\_employee(self):  
    employee\_id = self.search\_emp\_id\_entry.get()  
    try:  
    with open('employees.pkl', 'rb') as file:  
    while True:  
    try:  
    employee = pickle.load(file)  
    if employee.get\_employee\_id() == employee\_id:  
    message = "Employee found!\n"  
    message += f"Name: {employee.get\_name()}\n"  
    message += f"Employee ID: {employee.get\_employee\_id()}\n"  
    message += f"Department: {employee.get\_department()}\n"  
    message += f"Job Title: {employee.get\_job\_title()}\n"  
    message += f"Basic Salary: {employee.get\_basic\_salary()}\n"  
    message += f"Age: {employee.get\_age()}\n"  
    message += f"Date of Birth: {employee.get\_date\_of\_birth()}\n"  
    message += f"Passport Details: {employee.get\_passport\_details()}"  
    messagebox.showinfo("Employee Details", message)  
    return  
    except EOFError:  
    break  
    messagebox.showerror("Employee not found!")  
    except Exception as e:  
    messagebox.showerror("Error searching for employee:", e)  
   class ClientManagementApp:  
    def \_\_init\_\_(self, master):  
    self.master = master  
    self.master.title("Client Management System")  
     
    self.create\_client\_frame = ttk.Frame(master)  
    self.create\_client\_frame.pack(pady=10)  
     
    self.search\_client\_frame = ttk.Frame(master)  
    self.search\_client\_frame.pack(pady=10)  
     
    ttk.Label(self.create\_client\_frame, text="Client ID:").grid(row=0, column=0, padx=5, pady=5)  
    self.client\_id\_entry = ttk.Entry(self.create\_client\_frame)  
    self.client\_id\_entry.grid(row=0, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_client\_frame, text="Name:").grid(row=1, column=0, padx=5, pady=5)  
    self.name\_entry = ttk.Entry(self.create\_client\_frame)  
    self.name\_entry.grid(row=1, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_client\_frame, text="Address:").grid(row=2, column=0, padx=5, pady=5)  
    self.address\_entry = ttk.Entry(self.create\_client\_frame)  
    self.address\_entry.grid(row=2, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_client\_frame, text="Contact Details:").grid(row=3, column=0, padx=5, pady=5)  
    self.contact\_entry = ttk.Entry(self.create\_client\_frame)  
    self.contact\_entry.grid(row=3, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_client\_frame, text="Budget:").grid(row=4, column=0, padx=5, pady=5)  
    self.budget\_entry = ttk.Entry(self.create\_client\_frame)  
    self.budget\_entry.grid(row=4, column=1, padx=5, pady=5)  
     
    self.create\_client\_button = ttk.Button(self.create\_client\_frame, text="Create Client", command=self.create\_client)  
    self.create\_client\_button.grid(row=5, columnspan=2, padx=5, pady=5)  
     
    ttk.Label(self.search\_client\_frame, text="Search Client by ID:").grid(row=0, column=0, padx=5, pady=5)  
    self.search\_client\_id\_entry = ttk.Entry(self.search\_client\_frame)  
    self.search\_client\_id\_entry.grid(row=0, column=1, padx=5, pady=5)  
     
    self.search\_client\_button = ttk.Button(self.search\_client\_frame, text="Search", command=self.search\_client)  
    self.search\_client\_button.grid(row=1, columnspan=2, padx=5, pady=5)  
     
    def create\_client(self):  
    client\_id = self.client\_id\_entry.get()  
    name = self.name\_entry.get()  
    address = self.address\_entry.get()  
    contact\_details = self.contact\_entry.get()  
    budget = self.budget\_entry.get()  
     
    client = Client(client\_id, name, address, contact\_details, budget)  
    self.save\_client(client)  
     
    def save\_client(self, client):  
    try:  
    with open('clients.pkl', 'ab') as file:  
    pickle.dump(client, file)  
    messagebox.showinfo("Success", "Client saved successfully!")  
    except Exception as e:  
    messagebox.showerror("Error", f"Error saving client: {e}")  
     
    def search\_client(self):  
    client\_id = self.search\_client\_id\_entry.get()  
    try:  
    with open('clients.pkl', 'rb') as file:  
    while True:  
    try:  
    client = pickle.load(file)  
    if client.get\_client\_id() == client\_id:  
    message = f"Client found!\nName: {client.get\_name()}\nClient ID: {client.get\_client\_id()}\nAddress: {client.get\_address()}\nContact Details: {client.get\_contact\_details()}\nBudget: {client.get\_budget()}"  
    messagebox.showinfo("Client Details", message)  
    return  
    except EOFError:  
    break  
    messagebox.showinfo("Client Details", "Client not found!")  
    except Exception as e:  
    messagebox.showerror("Error", f"Error searching for client: {e}")  
     
     
   class GuestManagementApp:  
    def \_\_init\_\_(self, master):  
    self.master = master  
    self.master.title("Guest Management System")  
     
    self.create\_guest\_frame = ttk.Frame(master)  
    self.create\_guest\_frame.pack(pady=10)  
     
    self.search\_guest\_frame = ttk.Frame(master)  
    self.search\_guest\_frame.pack(pady=10)  
     
    ttk.Label(self.create\_guest\_frame, text="Guest ID:").grid(row=0, column=0, padx=5, pady=5)  
    self.guest\_id\_entry = ttk.Entry(self.create\_guest\_frame)  
    self.guest\_id\_entry.grid(row=0, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_guest\_frame, text="Name:").grid(row=1, column=0, padx=5, pady=5)  
    self.name\_entry = ttk.Entry(self.create\_guest\_frame)  
    self.name\_entry.grid(row=1, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_guest\_frame, text="Address:").grid(row=2, column=0, padx=5, pady=5)  
    self.address\_entry = ttk.Entry(self.create\_guest\_frame)  
    self.address\_entry.grid(row=2, column=1, padx=5, pady=5)  
     
    ttk.Label(self.create\_guest\_frame, text="Contact Details:").grid(row=3, column=0, padx=5, pady=5)  
    self.contact\_entry = ttk.Entry(self.create\_guest\_frame)  
    self.contact\_entry.grid(row=3, column=1, padx=5, pady=5)  
     
    self.create\_guest\_button = ttk.Button(self.create\_guest\_frame, text="Create Guest", command=self.create\_guest)  
    self.create\_guest\_button.grid(row=4, columnspan=2, padx=5, pady=5)  
     
    ttk.Label(self.search\_guest\_frame, text="Search Guest by ID:").grid(row=0, column=0, padx=5, pady=5)  
    self.search\_guest\_id\_entry = ttk.Entry(self.search\_guest\_frame)  
    self.search\_guest\_id\_entry.grid(row=0, column=1, padx=5, pady=5)  
     
    self.search\_guest\_button = ttk.Button(self.search\_guest\_frame, text="Search", command=self.search\_guest)  
    self.search\_guest\_button.grid(row=1, columnspan=2, padx=5, pady=5)  
     
    def create\_guest(self):  
    guest\_id = self.guest\_id\_entry.get()  
    name = self.name\_entry.get()  
    address = self.address\_entry.get()  
    contact\_details = self.contact\_entry.get()  
     
    guest = Guest(guest\_id, name, address, contact\_details)  
    self.save\_guest(guest)  
     
    def save\_guest(self, guest):  
    try:  
    with open('guests.pkl', 'ab') as file:  
    pickle.dump(guest, file)  
    messagebox.showinfo("Success", "Guest saved successfully!")  
    except Exception as e:  
    messagebox.showerror("Error", f"Error saving guest: {e}")  
     
    def search\_guest(self):  
    guest\_id = self.search\_guest\_id\_entry.get()  
    try:  
    with open('guests.pkl', 'rb') as file:  
    while True:  
    try:  
    guest = pickle.load(file)  
    if guest.get\_guest\_id() == guest\_id:  
    message = f"Guest found!\nName: {guest.get\_name()}\nGuest ID: {guest.get\_guest\_id()}\nAddress: {guest.get\_address()}\nContact Details: {guest.get\_contact\_details()}"  
    messagebox.showinfo("Guest Details", message)  
    return  
    except EOFError:  
    break  
    messagebox.showinfo("Guest Details", "Guest not found!")  
    except Exception as e:  
    messagebox.showerror("Error", f"Error searching for guest: {e}")  
     
   class EventManagementApp:  
    def \_\_init\_\_(self, master):  
    self.master = master  
    self.master.title("Event Management System")  
    # Create style for buttons  
    self.style = ttk.Style()  
    self.style.configure('TButton', font=('Helvetica', 14), background='#4CAF50', foreground='white')  
     
    # Create buttons for each option  
    self.employee\_button = ttk.Button(master, text="Employee", command=self.show\_employee\_options)  
    self.employee\_button.pack(pady=5)  
     
     
    self.client\_button = ttk.Button(master, text="Client", command=self.show\_client\_options)  
    self.client\_button.pack(pady=5)  
     
     
    self.guest\_button = ttk.Button(master, text="Guest", command=self.show\_guest\_options)  
    self.guest\_button.pack(pady=5)  
     
     
    def show\_employee\_options(self):  
    root = tk.Tk()  
    app = EmployeeManagement(root)  
    root.mainloop()  
    def show\_client\_options(self):  
    root = tk.Tk()  
    app = ClientManagementApp(root)  
    root.mainloop()  
     
    def show\_guest\_options(self):  
    root = tk.Tk()  
    root.title("Guest Management System")  
    app = GuestManagementApp(root)  
    root.mainloop()  
     
   def main():  
    root = tk.Tk()  
    root.geometry("400x300")  
    app = EventManagementApp(root)  
    root.mainloop()  
     
   if \_\_name\_\_ == "\_\_main\_\_":  
    main()
3. **Evaluation**

Below are the screenshots of different runs of the program:

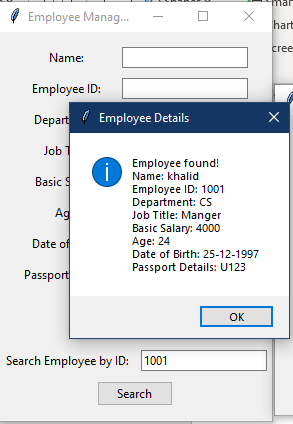
* This is the main screen where the program starts:



* This is to create and search the employee:

A screenshot of a computer

Description automatically generated



* Screen to create and Search Client

A screenshot of a computer

Description automatically generated

* Below is the screen to create and search the guests:

A screenshot of a computer

Description automatically generated

1. **Summary of Learning**

This project provided valuable learning outcomes in GUI development using Tkinter, object-oriented programming principles, data management with Pickle, user interaction, error handling, problem-solving, documentation, and project management. Through designing intuitive interfaces, encapsulating data and functionality within classes, and managing data persistence, I gained practical skills applicable to software development. This project highlighted the importance of user feedback, input validation, and error handling for robust applications. Effective documentation and project management were essential for organizing code and managing the development process. Overall, this project enhanced my skills in various aspects of software development within a practical context.