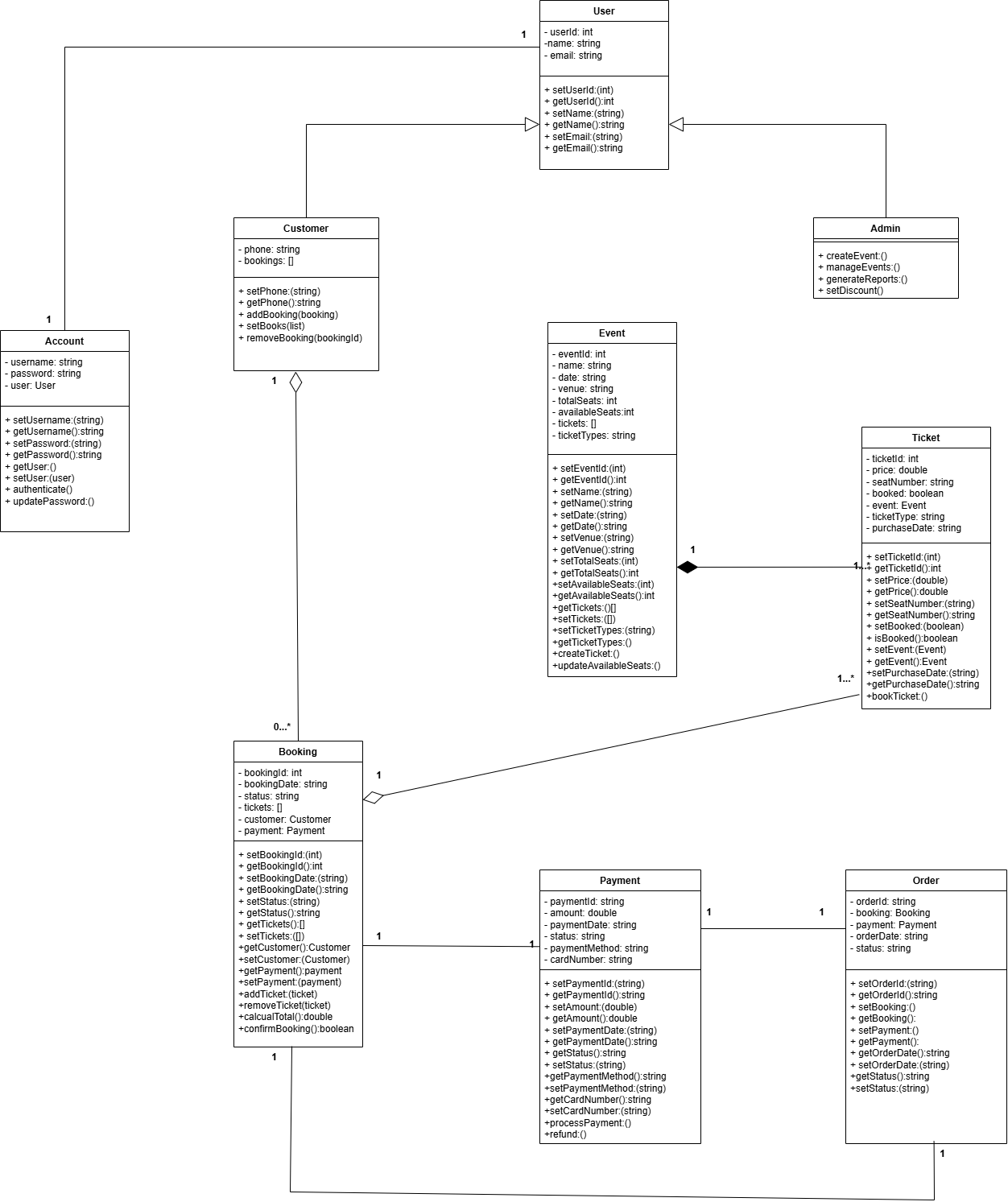
Grand Prix Experience

**UML Class Diagram:**



**Description:**

This is a UML class diagram for an event ticketing system that allows customers to book tickets for events, make payments, and create orders. Here's a breakdown of the classes and their relationships:

**Classes and Their Attributes/Methods:**

1. User
   * Base class with basic user information
   * Attributes: userid, name, email
   * Methods: getters and setters for all attributes
2. Customer (extends User)
   * Represents a customer who can make bookings
   * Additional attributes: phone, bookings (array)
   * Methods: getters/setters for phone, methods to add/set bookings
3. Event
   * Represents an event that has tickets
   * Attributes: eventid, name, date, venue, totalSeats, availableSeats, tickets (array), ticketTypes
   * Methods: various getters/setters, createTicket(), updateAvailableSeats()
4. Ticket
   * Represents a ticket for an event
   * Attributes: ticketId, price, seatNumber, booked (boolean), event (reference to Event), ticketType, purchaseDate
   * Methods: getters/setters, bookTicket()
5. Booking
   * Represents a booking made by a customer
   * Attributes: bookingId, bookingDate, status, tickets (array), customer (reference to Customer), payment (reference to Payment)
   * Methods: getters/setters, addTicket(), removeTicket(), confirmBooking(), calculateTotal()
6. Payment
   * Handles payment information
   * Attributes: paymentid, paymentMethod, cardNumber, paymentDate, status, amount
   * Methods: getters/setters, processPayment(), refund()
7. Order
   * Represents an order created from a booking
   * Attributes: orderid, booking (reference to Booking), payment (reference to Payment), orderDate, status
   * Methods: getters/setters
8. Admin (likely extends User, though not explicitly shown)
   * Administrative user with special privileges
   * Methods: createEvent(), manageEvents(), generateReports(), setDiscount()
9. Account
   * Manages user authentication
   * Attributes: username, password, user (reference to User)
   * Methods: getters/setters, authenticate(), updatePassword()

**Complete Code:**

import random  
import time  
import tkinter as tk  
from tkinter import ttk, messagebox, simpledialog  
import pickle  
import os  
import matplotlib.pyplot as plt  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
  
# Ensure data directory exists  
if not os.path.exists("data"):  
 os.makedirs("data")  
  
  
# Base User class  
class User:  
 def \_\_init\_\_(self, name, email):  
 self.userId = str(random.randint(1000, 9999))  
 self.name = name  
 self.email = email  
  
 def getUserId(self):  
 return self.userId  
  
 def setUserId(self, userId):  
 self.userId = userId  
  
 def getName(self):  
 return self.name  
  
 def setName(self, name):  
 self.name = name  
  
 def getEmail(self):  
 return self.email  
  
 def setEmail(self, email):  
 self.email = email  
  
  
class Customer(User):  
 def \_\_init\_\_(self, name, email, phone):  
 User.\_\_init\_\_(self, name, email)  
 self.phone = phone  
 self.bookings = []  
  
 def getPhone(self):  
 return self.phone  
  
 def setPhone(self, phone):  
 self.phone = phone  
  
 def addBooking(self, booking):  
 self.bookings.append(booking)  
  
 def getBookings(self):  
 return self.bookings  
  
 def setBookings(self, bookings):  
 self.bookings = bookings  
  
 def removeBooking(self, booking\_id):  
 for i, booking in enumerate(self.bookings):  
 if booking.getBookingId() == booking\_id:  
 del self.bookings[i]  
 return True  
 return False  
  
  
class Admin(User):  
 def \_\_init\_\_(self, name, email):  
 User.\_\_init\_\_(self, name, email)  
  
 def createEvent(self, name, date, venue, total\_seats):  
 return Event(name, date, venue, total\_seats)  
  
 def manageEvents(self):  
 # Implementation for managing events  
 pass  
  
 def generateReports(self):  
 # Implementation for generating reports  
 pass  
  
 def setDiscount(self, event, discount\_percentage):  
 if discount\_percentage < 0 or discount\_percentage > 100:  
 raise ValueError("Discount percentage must be between 0 and 100")  
  
 for ticket in event.getTickets():  
 if not ticket.isBooked():  
 original\_price = ticket.getPrice()  
 discounted\_price = original\_price \* (1 - discount\_percentage / 100)  
 ticket.setPrice(discounted\_price)  
  
  
class Account:  
 def \_\_init\_\_(self, username, password, user):  
 self.username = username  
 self.password = password  
 self.user = user  
  
 def getUsername(self):  
 return self.username  
  
 def setUsername(self, username):  
 self.username = username  
  
 def getPassword(self):  
 return self.password  
  
 def setPassword(self, password):  
 self.password = password  
  
 def getUser(self):  
 return self.user  
  
 def setUser(self, user):  
 self.user = user  
  
 def authenticate(self, username, password):  
 return self.username == username and self.password == password  
  
 def updatePassword(self, old\_password, new\_password):  
 if self.password == old\_password:  
 self.password = new\_password  
 return True  
 return False  
  
  
class Event:  
 def \_\_init\_\_(self, name, date, venue, total\_seats):  
 self.eventId = "E" + str(random.randint(100, 999))  
 self.name = name  
 self.date = date  
 self.venue = venue  
 self.totalSeats = total\_seats  
 self.availableSeats = total\_seats  
 self.tickets = []  
 self.ticketTypes = {  
 "Single Day": {"price": 150.0, "features": "Access to race day only"},  
 "Weekend": {"price": 300.0, "features": "Access to qualifying and race day"},  
 "VIP": {"price": 500.0, "features": "Premium seating, paddock access, and refreshments"}  
 }  
  
 def getEventId(self):  
 return self.eventId  
  
 def setEventId(self, eventId):  
 self.eventId = eventId  
  
 def getName(self):  
 return self.name  
  
 def setName(self, name):  
 self.name = name  
  
 def getDate(self):  
 return self.date  
  
 def setDate(self, date):  
 self.date = date  
  
 def getVenue(self):  
 return self.venue  
  
 def setVenue(self, venue):  
 self.venue = venue  
  
 def getTotalSeats(self):  
 return self.totalSeats  
  
 def setTotalSeats(self, totalSeats):  
 self.totalSeats = totalSeats  
  
 def getAvailableSeats(self):  
 return self.availableSeats  
  
 def setAvailableSeats(self, availableSeats):  
 self.availableSeats = availableSeats  
  
 def getTickets(self):  
 return self.tickets  
  
 def setTickets(self, tickets):  
 self.tickets = tickets  
  
 def getTicketTypes(self):  
 return self.ticketTypes  
  
 def setTicketTypes(self, ticketTypes):  
 self.ticketTypes = ticketTypes  
  
 def createTicket(self, seat\_number, price, ticket\_type="Single Day"):  
 ticket = Ticket(self, seat\_number, price, ticket\_type)  
 self.tickets.append(ticket)  
 return ticket  
  
 def updateAvailableSeats(self):  
 booked = 0  
 for ticket in self.tickets:  
 if ticket.isBooked():  
 booked = booked + 1  
 self.availableSeats = self.totalSeats - booked  
  
  
class Ticket:  
 def \_\_init\_\_(self, event, seat\_number, price, ticket\_type="Single Day"):  
 self.ticketId = "T" + str(random.randint(1000, 9999))  
 self.event = event  
 self.price = price  
 self.seatNumber = seat\_number  
 self.booked = False  
 self.ticketType = ticket\_type  
 self.purchaseDate = None  
  
 def getTicketId(self):  
 return self.ticketId  
  
 def setTicketId(self, ticketId):  
 self.ticketId = ticketId  
  
 def getPrice(self):  
 return self.price  
  
 def setPrice(self, price):  
 self.price = price  
  
 def getSeatNumber(self):  
 return self.seatNumber  
  
 def setSeatNumber(self, seatNumber):  
 self.seatNumber = seatNumber  
  
 def isBooked(self):  
 return self.booked  
  
 def setBooked(self, booked):  
 self.booked = booked  
  
 def getEvent(self):  
 return self.event  
  
 def setEvent(self, event):  
 self.event = event  
  
 def getTicketType(self):  
 return self.ticketType  
  
 def setTicketType(self, ticketType):  
 self.ticketType = ticketType  
  
 def getPurchaseDate(self):  
 return self.purchaseDate  
  
 def setPurchaseDate(self, purchaseDate):  
 self.purchaseDate = purchaseDate  
  
 def bookTicket(self):  
 if not self.booked:  
 self.booked = True  
 self.purchaseDate = time.strftime("%Y-%m-%d")  
 self.event.updateAvailableSeats()  
 return True  
 return False  
  
  
class Booking:  
 def \_\_init\_\_(self, customer):  
 self.bookingId = "B" + str(random.randint(1000, 9999))  
 self.bookingDate = time.strftime("%Y-%m-%d")  
 self.status = "Pending"  
 self.tickets = []  
 self.customer = customer  
 self.payment = None  
  
 def getBookingId(self):  
 return self.bookingId  
  
 def setBookingId(self, bookingId):  
 self.bookingId = bookingId  
  
 def getBookingDate(self):  
 return self.bookingDate  
  
 def setBookingDate(self, bookingDate):  
 self.bookingDate = bookingDate  
  
 def getStatus(self):  
 return self.status  
  
 def setStatus(self, status):  
 self.status = status  
  
 def getTickets(self):  
 return self.tickets  
  
 def setTickets(self, tickets):  
 self.tickets = tickets  
  
 def getCustomer(self):  
 return self.customer  
  
 def setCustomer(self, customer):  
 self.customer = customer  
  
 def getPayment(self):  
 return self.payment  
  
 def setPayment(self, payment):  
 self.payment = payment  
  
 def addTicket(self, ticket):  
 if ticket.bookTicket():  
 self.tickets.append(ticket)  
  
 def removeTicket(self, ticket):  
 if ticket in self.tickets:  
 self.tickets.remove(ticket)  
 ticket.setBooked(False)  
 ticket.getEvent().updateAvailableSeats()  
  
 def calculateTotal(self):  
 total = 0  
 for ticket in self.tickets:  
 total = total + ticket.getPrice()  
 return total  
  
 def confirmBooking(self):  
 if len(self.tickets) > 0 and self.payment and self.payment.getStatus() == "Completed":  
 self.status = "Confirmed"  
 return True  
 return False  
  
  
class Payment:  
 def \_\_init\_\_(self, amount):  
 self.paymentId = "P" + str(random.randint(1000, 9999))  
 self.amount = amount  
 self.paymentDate = None  
 self.status = "Pending"  
 self.paymentMethod = None  
 self.cardNumber = None  
  
 def getPaymentId(self):  
 return self.paymentId  
  
 def setPaymentId(self, paymentId):  
 self.paymentId = paymentId  
  
 def getAmount(self):  
 return self.amount  
  
 def setAmount(self, amount):  
 self.amount = amount  
  
 def getPaymentDate(self):  
 return self.paymentDate  
  
 def setPaymentDate(self, paymentDate):  
 self.paymentDate = paymentDate  
  
 def getStatus(self):  
 return self.status  
  
 def setStatus(self, status):  
 self.status = status  
  
 def getPaymentMethod(self):  
 return self.paymentMethod  
  
 def setPaymentMethod(self, paymentMethod):  
 self.paymentMethod = paymentMethod  
  
 def getCardNumber(self):  
 return self.cardNumber  
  
 def setCardNumber(self, cardNumber):  
 self.cardNumber = cardNumber  
  
 def processPayment(self, payment\_method, card\_number):  
 try:  
 # Validate card number (simplified)  
 if len(card\_number) < 13 or len(card\_number) > 19:  
 raise ValueError("Invalid card number length")  
  
 if not card\_number.isdigit():  
 raise ValueError("Card number must contain only digits")  
  
 self.paymentMethod = payment\_method  
 self.cardNumber = card\_number[-4:] # Store only last 4 digits for security  
 self.paymentDate = time.strftime("%Y-%m-%d")  
 self.status = "Completed"  
 return True  
 except ValueError as e:  
 messagebox.showerror("Payment Error", str(e))  
 return False  
  
 def refund(self):  
 try:  
 if self.status != "Completed":  
 raise ValueError("Cannot refund a payment that is not completed")  
 self.status = "Refunded"  
 return True  
 except ValueError as e:  
 messagebox.showerror("Refund Error", str(e))  
 return False  
  
  
class Order:  
 def \_\_init\_\_(self, booking, payment):  
 self.orderId = "O" + str(random.randint(1000, 9999))  
 self.booking = booking  
 self.payment = payment  
 self.orderDate = time.strftime("%Y-%m-%d")  
 self.status = "Processed"  
  
 def getOrderId(self):  
 return self.orderId  
  
 def setOrderId(self, orderId):  
 self.orderId = orderId  
  
 def getBooking(self):  
 return self.booking  
  
 def setBooking(self, booking):  
 self.booking = booking  
  
 def getPayment(self):  
 return self.payment  
  
 def setPayment(self, payment):  
 self.payment = payment  
  
 def getOrderDate(self):  
 return self.orderDate  
  
 def setOrderDate(self, orderDate):  
 self.orderDate = orderDate  
  
 def getStatus(self):  
 return self.status  
  
 def setStatus(self, status):  
 self.status = status  
  
  
# Data Manager for handling file operations  
class DataManager:  
 def \_\_init\_\_(self):  
 self.accounts\_file = "data/accounts.pkl"  
 self.events\_file = "data/events.pkl"  
 self.orders\_file = "data/orders.pkl"  
  
 def save\_accounts(self, accounts):  
 try:  
 with open(self.accounts\_file, "wb") as file:  
 pickle.dump(accounts, file)  
 except Exception as e:  
 messagebox.showerror("Error", "Failed to save accounts: " + str(e))  
  
 def load\_accounts(self):  
 try:  
 if os.path.exists(self.accounts\_file):  
 with open(self.accounts\_file, "rb") as file:  
 return pickle.load(file)  
 return []  
 except Exception as e:  
 messagebox.showerror("Error", "Failed to load accounts: " + str(e))  
 return []  
  
 def save\_events(self, events):  
 try:  
 with open(self.events\_file, "wb") as file:  
 pickle.dump(events, file)  
 except Exception as e:  
 messagebox.showerror("Error", "Failed to save events: " + str(e))  
  
 def load\_events(self):  
 try:  
 if os.path.exists(self.events\_file):  
 with open(self.events\_file, "rb") as file:  
 return pickle.load(file)  
 return []  
 except Exception as e:  
 messagebox.showerror("Error", "Failed to load events: " + str(e))  
 return []  
  
 def save\_orders(self, orders):  
 try:  
 with open(self.orders\_file, "wb") as file:  
 pickle.dump(orders, file)  
 except Exception as e:  
 messagebox.showerror("Error", "Failed to save orders: " + str(e))  
  
 def load\_orders(self):  
 try:  
 if os.path.exists(self.orders\_file):  
 with open(self.orders\_file, "rb") as file:  
 return pickle.load(file)  
 return []  
 except Exception as e:  
 messagebox.showerror("Error", "Failed to load orders: " + str(e))  
 return []  
  
  
# Main GUI Application  
class TicketBookingApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Ticket Booking System")  
 self.root.geometry("800x600")  
 self.data\_manager = DataManager()  
 self.accounts = self.data\_manager.load\_accounts()  
 self.events = self.data\_manager.load\_events()  
 self.orders = self.data\_manager.load\_orders()  
 self.current\_user = None  
  
 # Welcome Frame  
 self.welcome\_frame = tk.Frame(self.root)  
 self.welcome\_frame.pack(fill="both", expand=True)  
 self.show\_welcome\_screen()  
  
 def show\_welcome\_screen(self):  
 for widget in self.welcome\_frame.winfo\_children():  
 widget.destroy()  
  
 tk.Label(self.welcome\_frame, text="Welcome to Ticket Booking System", font=("Arial", 20)).pack(pady=20)  
 tk.Button(self.welcome\_frame, text="Login", width=20, command=self.show\_login\_screen).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Create Account", width=20, command=self.show\_account\_creation\_screen).pack(pady=10)  
  
 def show\_login\_screen(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Login", font=("Arial", 18)).pack(pady=20)  
  
 tk.Label(self.welcome\_frame, text="Username").pack()  
 username\_entry = tk.Entry(self.welcome\_frame)  
 username\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Password").pack()  
 password\_entry = tk.Entry(self.welcome\_frame, show="\*")  
 password\_entry.pack()  
  
 def login():  
 username = username\_entry.get()  
 password = password\_entry.get()  
 for account in self.accounts:  
 if account.authenticate(username, password):  
 self.current\_user = account.getUser()  
 if isinstance(self.current\_user, Admin):  
 self.show\_admin\_dashboard()  
 elif isinstance(self.current\_user, Customer):  
 self.show\_customer\_dashboard()  
 return  
 messagebox.showerror("Login Failed", "Invalid username or password")  
  
 tk.Button(self.welcome\_frame, text="Login", command=login).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_welcome\_screen).pack(pady=10)  
  
 def show\_account\_creation\_screen(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Create Account", font=("Arial", 18)).pack(pady=20)  
  
 tk.Label(self.welcome\_frame, text="Name").pack()  
 name\_entry = tk.Entry(self.welcome\_frame)  
 name\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Email").pack()  
 email\_entry = tk.Entry(self.welcome\_frame)  
 email\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Phone").pack()  
 phone\_entry = tk.Entry(self.welcome\_frame)  
 phone\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Username").pack()  
 username\_entry = tk.Entry(self.welcome\_frame)  
 username\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Password").pack()  
 password\_entry = tk.Entry(self.welcome\_frame, show="\*")  
 password\_entry.pack()  
  
 def create\_account():  
 name = name\_entry.get()  
 email = email\_entry.get()  
 phone = phone\_entry.get()  
 username = username\_entry.get()  
 password = password\_entry.get()  
  
 if not name or not email or not phone or not username or not password:  
 messagebox.showerror("Error", "All fields are required")  
 return  
  
 # Create a new customer and account  
 customer = Customer(name, email, phone)  
 new\_account = Account(username, password, customer)  
  
 # Check if username is unique  
 for account in self.accounts:  
 if account.getUsername() == username:  
 messagebox.showerror("Error", "Username already exists")  
 return  
  
 self.accounts.append(new\_account)  
 self.data\_manager.save\_accounts(self.accounts)  
 messagebox.showinfo("Success", "Account created successfully")  
 self.show\_welcome\_screen()  
  
 tk.Button(self.welcome\_frame, text="Create Account", command=create\_account).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_welcome\_screen).pack(pady=10)  
  
 def show\_customer\_dashboard(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Welcome, " + self.current\_user.getName(), font=("Arial", 18)).pack(pady=20)  
  
 tk.Button(self.welcome\_frame, text="View Events", width=20, command=self.show\_event\_list).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="View Bookings", width=20, command=self.show\_booking\_list).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Logout", width=20, command=self.logout).pack(pady=10)  
  
 def show\_admin\_dashboard(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Admin Dashboard", font=("Arial", 18)).pack(pady=20)  
 tk.Button(self.welcome\_frame, text="Manage Events", width=20, command=self.manage\_events).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Manage Orders", width=20, command=self.manage\_orders).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Manage Customers", width=20, command=self.manage\_customers).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="View Reports", width=20, command=self.view\_ticket\_sales).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Logout", width=20, command=self.logout).pack(pady=10)  
  
 def manage\_events(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Manage Events", font=("Arial", 18)).pack(pady=20)  
  
 # Frame for listing events  
 events\_frame = tk.Frame(self.welcome\_frame)  
 events\_frame.pack(fill="both", expand=True)  
  
 # List all events  
 for event in self.events:  
 frame = tk.Frame(events\_frame, relief="ridge", borderwidth=1)  
 frame.pack(fill="x", padx=10, pady=5)  
 tk.Label(frame, text="Event Name: " + event.getName()).pack(anchor="w")  
 tk.Label(frame, text="Date: " + event.getDate()).pack(anchor="w")  
 tk.Label(frame, text="Venue: " + event.getVenue()).pack(anchor="w")  
 tk.Label(frame, text="Total Seats: " + str(event.getTotalSeats()) + ", Available Seats: " + str(event.getAvailableSeats())).pack(anchor="w")  
  
 def update\_discount(event\_obj=event):  
 discount = simpledialog.askinteger("Set Discount", "Enter discount percentage (0-100):")  
 if discount is not None:  
 try:  
 Admin(self.current\_user.getName(), self.current\_user.getEmail()).setDiscount(event\_obj,  
 discount)  
 self.data\_manager.save\_events(self.events)  
 messagebox.showinfo("Success", f"Discount of {discount}% applied successfully!")  
 self.manage\_events()  
 except ValueError as e:  
 messagebox.showerror("Error", str(e))  
 tk.Button(frame, text="Set Discount", command=update\_discount).pack(side="right", padx=5)  
 tk.Button(frame, text="Modify", command=lambda e=event: self.modify\_event(e)).pack(side="right", padx=5)  
 tk.Button(frame, text="Delete", command=lambda e=event: self.delete\_event(e)).pack(side="right", padx=5)  
  
 # Add event button  
 tk.Button(self.welcome\_frame, text="Add Event", command=self.add\_event).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_admin\_dashboard).pack(pady=10)  
  
 def add\_event(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Add Event", font=("Arial", 18)).pack(pady=20)  
  
 tk.Label(self.welcome\_frame, text="Event Name").pack()  
 name\_entry = tk.Entry(self.welcome\_frame)  
 name\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Date (YYYY-MM-DD)").pack()  
 date\_entry = tk.Entry(self.welcome\_frame)  
 date\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Venue").pack()  
 venue\_entry = tk.Entry(self.welcome\_frame)  
 venue\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Total Seats").pack()  
 seats\_entry = tk.Entry(self.welcome\_frame)  
 seats\_entry.pack()  
  
 def save\_new\_event():  
 name = name\_entry.get()  
 date = date\_entry.get()  
 venue = venue\_entry.get()  
 try:  
 total\_seats = int(seats\_entry.get())  
 except ValueError:  
 messagebox.showerror("Error", "Total seats must be a valid number.")  
 return  
  
 if not name or not date or not venue or total\_seats <= 0:  
 messagebox.showerror("Error", "All fields are required, and total seats must be greater than zero.")  
 return  
 # Create the new event  
 new\_event = Event(name, date, venue, total\_seats)  
 self.events.append(new\_event)  
 # Save events to file  
 self.data\_manager.save\_events(self.events)  
 messagebox.showinfo("Success", "Event added successfully!")  
 self.manage\_events()  
  
 tk.Button(self.welcome\_frame, text="Save", command=save\_new\_event).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.manage\_events).pack(pady=10)  
  
 def modify\_event(self, event):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Modify Event", font=("Arial", 18)).pack(pady=20)  
  
 tk.Label(self.welcome\_frame, text="Event Name").pack()  
 name\_entry = tk.Entry(self.welcome\_frame)  
 name\_entry.insert(0, event.getName())  
 name\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Date (YYYY-MM-DD)").pack()  
 date\_entry = tk.Entry(self.welcome\_frame)  
 date\_entry.insert(0, event.getDate())  
 date\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Venue").pack()  
 venue\_entry = tk.Entry(self.welcome\_frame)  
 venue\_entry.insert(0, event.getVenue())  
 venue\_entry.pack()  
  
 tk.Label(self.welcome\_frame, text="Total Seats").pack()  
 seats\_entry = tk.Entry(self.welcome\_frame)  
 seats\_entry.insert(0, event.getTotalSeats())  
 seats\_entry.pack()  
  
 def save\_modified\_event():  
 name = name\_entry.get()  
 date = date\_entry.get()  
 venue = venue\_entry.get()  
 try:  
 total\_seats = int(seats\_entry.get())  
 except ValueError:  
 messagebox.showerror("Error", "Total seats must be a valid number.")  
 return  
 if not name or not date or not venue or total\_seats <= 0:  
 messagebox.showerror("Error", "All fields are required, and total seats must be greater than zero.")  
 return  
  
 # Update the event details  
 event.setName(name)  
 event.setDate(date)  
 event.setVenue(venue)  
 event.setTotalSeats(total\_seats)  
 event.updateAvailableSeats()  
  
 # Save events to file  
 self.data\_manager.save\_events(self.events)  
 messagebox.showinfo("Success", "Event details updated successfully!")  
 self.manage\_events()  
  
 tk.Button(self.welcome\_frame, text="Save", command=save\_modified\_event).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.manage\_events).pack(pady=10)  
  
 def delete\_event(self, event):  
 confirm = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete the event '" + event.getName() + "'?")  
 if confirm:  
 try:  
 # Remove the event from the list  
 self.events.remove(event)  
 self.data\_manager.save\_events(self.events)  
 messagebox.showinfo("Success", "Event deleted successfully!")  
 self.manage\_events()  
 except Exception as e:  
 messagebox.showerror("Error", f"Failed to delete event: {str(e)}")  
  
 def manage\_customers(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Manage Customers", font=("Arial", 18)).pack(pady=20)  
 # Frame for listing customers  
 customer\_list\_frame = tk.Frame(self.welcome\_frame)  
 customer\_list\_frame.pack(fill="both", expand=True)  
  
 # List all customers  
 for account in self.accounts:  
 if isinstance(account.getUser(), Customer):  
 customer = account.getUser()  
 frame = tk.Frame(customer\_list\_frame, relief="ridge", borderwidth=1)  
 frame.pack(fill="x", padx=10, pady=5)  
 tk.Label(frame, text="Customer Name: " + customer.getName()).pack(anchor="w")  
 tk.Label(frame, text="Email: " + customer.getEmail()).pack(anchor="w")  
 tk.Label(frame, text="Phone: " + customer.getPhone()).pack(anchor="w")  
 tk.Button(frame, text="Modify", command=lambda c=customer, a=account: self.modify\_customer(c, a)).pack(side="right", padx=5)  
 tk.Button(frame, text="Delete", command=lambda a=account: self.delete\_customer(a)).pack(side="right", padx=5)  
 # Add customer button  
 tk.Button(self.welcome\_frame, text="Add Customer", command=self.add\_customer).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_admin\_dashboard).pack(pady=10)  
 #Method where we will be adding customer  
 def add\_customer(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Add Customer", font=("Arial", 18)).pack(pady=20)  
 tk.Label(self.welcome\_frame, text="Name").pack()  
 name\_entry = tk.Entry(self.welcome\_frame)  
 name\_entry.pack()  
 tk.Label(self.welcome\_frame, text="Email").pack()  
 email\_entry = tk.Entry(self.welcome\_frame)  
 email\_entry.pack()  
 tk.Label(self.welcome\_frame, text="Phone").pack()  
 phone\_entry = tk.Entry(self.welcome\_frame)  
 phone\_entry.pack()  
 tk.Label(self.welcome\_frame, text="Username").pack()  
 username\_entry = tk.Entry(self.welcome\_frame)  
 username\_entry.pack()  
 tk.Label(self.welcome\_frame, text="Password").pack()  
 password\_entry = tk.Entry(self.welcome\_frame, show="\*")  
 password\_entry.pack()  
  
 def save\_new\_customer():  
 name = name\_entry.get()  
 email = email\_entry.get()  
 phone = phone\_entry.get()  
 username = username\_entry.get()  
 password = password\_entry.get()  
 if not name or not email or not phone or not username or not password:  
 messagebox.showerror("Error", "All fields are required")  
 return  
 # Ensure username is unique  
 for account in self.accounts:  
 if account.getUsername() == username:  
 messagebox.showerror("Error", "Username already exists")  
 return  
 # Create customer and account  
 new\_customer = Customer(name, email, phone)  
 new\_account = Account(username, password, new\_customer)  
 self.accounts.append(new\_account)  
 self.data\_manager.save\_accounts(self.accounts)  
 messagebox.showinfo("Success", "Customer added successfully!")  
 self.manage\_customers()  
 tk.Button(self.welcome\_frame, text="Save", command=save\_new\_customer).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.manage\_customers).pack(pady=10)  
  
 def modify\_customer(self, customer, account):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Modify Customer", font=("Arial", 18)).pack(pady=20)  
 tk.Label(self.welcome\_frame, text="Name").pack()  
 name\_entry = tk.Entry(self.welcome\_frame)  
 name\_entry.insert(0, customer.getName())  
 name\_entry.pack()  
 tk.Label(self.welcome\_frame, text="Email").pack()  
 email\_entry = tk.Entry(self.welcome\_frame)  
 email\_entry.insert(0, customer.getEmail())  
 email\_entry.pack()  
 tk.Label(self.welcome\_frame, text="Phone").pack()  
 phone\_entry = tk.Entry(self.welcome\_frame)  
 phone\_entry.insert(0, customer.getPhone())  
 phone\_entry.pack()  
 tk.Label(self.welcome\_frame, text="Username").pack()  
 username\_entry = tk.Entry(self.welcome\_frame)  
 username\_entry.insert(0, account.getUsername())  
 username\_entry.pack()  
 #Method to save modifiefed customer  
 def save\_modified\_customer():  
 name = name\_entry.get()  
 email = email\_entry.get()  
 phone = phone\_entry.get()  
 username = username\_entry.get()  
 if not name or not email or not phone or not username:  
 messagebox.showerror("Error", "All fields are required")  
 return  
 # Ensure username is unique if changed  
 if username != account.getUsername():  
 for acc in self.accounts:  
 if acc.getUsername() == username:  
 messagebox.showerror("Error", "Username already exists")  
 return  
 # Update customer and account details  
 customer.setName(name)  
 customer.setEmail(email)  
 customer.setPhone(phone)  
 account.setUsername(username)  
 # Save accounts to file  
 self.data\_manager.save\_accounts(self.accounts)  
 messagebox.showinfo("Success", "Customer details updated successfully!")  
 self.manage\_customers()  
  
 tk.Button(self.welcome\_frame, text="Save", command=save\_modified\_customer).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.manage\_customers).pack(pady=10)  
  
 def delete\_customer(self, account):  
 confirm = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete this customer?")  
 if confirm:  
 try:  
 # Remove the customer account from the accounts list  
 self.accounts.remove(account)  
 self.data\_manager.save\_accounts(self.accounts)  
 messagebox.showinfo("Success", "Customer deleted successfully!")  
 self.manage\_customers()  
 except Exception as e:  
 messagebox.showerror("Error", f"Failed to delete customer: {str(e)}")  
  
 def show\_event\_list(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Available Events", font=("Arial", 18)).pack(pady=20)  
 for event in self.events:  
 event\_frame = tk.Frame(self.welcome\_frame, relief="ridge", borderwidth=1)  
 event\_frame.pack(fill="x", padx=10, pady=5)  
  
 tk.Label(event\_frame, text="Event: " + event.getName()).pack(anchor="w")  
 tk.Label(event\_frame, text="Date: " + event.getDate()).pack(anchor="w")  
 tk.Label(event\_frame, text="Venue: " + event.getVenue()).pack(anchor="w")  
 tk.Label(event\_frame, text="Available Seats: " + str(event.getAvailableSeats())).pack(anchor="w")  
 tk.Button(event\_frame, text="Book Tickets", command=lambda e=event: self.book\_tickets(e)).pack(anchor="e",pady=5)  
  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_customer\_dashboard).pack(pady=10)  
 #Method to book tickets  
 def book\_tickets(self, event):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Book Tickets for " + event.getName(), font=("Arial", 18)).pack(pady=20)  
 ticket\_types = event.getTicketTypes()  
 ticket\_selections = {}  
  
 for ticket\_type, details in ticket\_types.items():  
 frame = tk.Frame(self.welcome\_frame)  
 frame.pack(pady=5)  
 tk.Label(frame, text=ticket\_type + " - AED" + str(details['price'])).pack(side="left", padx=10)  
 tk.Label(frame, text=details["features"]).pack(side="left", padx=10)  
 quantity\_var = tk.IntVar(value=0)  
 ticket\_selections[ticket\_type] = quantity\_var  
 tk.Spinbox(frame, from\_=0, to=event.getAvailableSeats(), textvariable=quantity\_var, width=5).pack(  
 side="right")  
  
 def confirm\_booking():  
 total\_tickets = 0  
 selected\_tickets = []  
 for ticket\_type, quantity\_var in ticket\_selections.items():  
 quantity = quantity\_var.get()  
 if quantity > 0:  
 total\_tickets += quantity  
 for i in range(quantity):  
 ticket = event.createTicket(  
 seat\_number=random.randint(1, 1000),  
 price=ticket\_types[ticket\_type]["price"],  
 ticket\_type=ticket\_type  
 )  
 selected\_tickets.append(ticket)  
  
 if total\_tickets == 0:  
 messagebox.showerror("Error", "Please select at least one ticket")  
 return  
  
 # Create booking and add to current\_user  
 booking = Booking(self.current\_user)  
 for ticket in selected\_tickets:  
 booking.addTicket(ticket)  
  
 self.current\_user.addBooking(booking)  
 self.data\_manager.save\_accounts(self.accounts)  
 self.data\_manager.save\_events(self.events)  
 self.show\_payment\_screen(booking)  
  
 tk.Button(self.welcome\_frame, text="Confirm Booking", command=confirm\_booking).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_event\_list).pack(pady=10)  
 #Method to manage orderss  
 def manage\_orders(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Manage Orders", font=("Arial", 18)).pack(pady=20)  
  
 # Frame for listing orders  
 orders\_frame = tk.Frame(self.welcome\_frame)  
 orders\_frame.pack(fill="both", expand=True)  
 # List all orders  
 for order in self.orders:  
 frame = tk.Frame(orders\_frame, relief="ridge", borderwidth=1)  
 frame.pack(fill="x", padx=10, pady=5)  
 tk.Label(frame, text="Order ID: " + order.getOrderId()).pack(anchor="w")  
 tk.Label(frame, text="Booking ID: " + order.getBooking().getBookingId()).pack(anchor="w")  
 tk.Label(frame, text="Order Date: " + order.getOrderDate()).pack(anchor="w")  
 tk.Label(frame, text="Order Status: " + order.getStatus()).pack(anchor="w")  
 tk.Button(frame, text="Modify", command=lambda o=order: self.modify\_order(o)).pack(side="right", padx=5)  
 tk.Button(frame, text="Delete", command=lambda o=order: self.delete\_order(o)).pack(side="right", padx=5)  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_admin\_dashboard).pack(pady=10)  
  
 def modify\_order(self, order):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Modify Order", font=("Arial", 18)).pack(pady=20)  
  
 tk.Label(self.welcome\_frame, text="Order ID: " + order.getOrderId()).pack(pady=5)  
 tk.Label(self.welcome\_frame, text="Booking ID: " + order.getBooking().getBookingId()).pack(pady=5)  
  
 tk.Label(self.welcome\_frame, text="Order Status").pack()  
 status\_entry = tk.Entry(self.welcome\_frame)  
 status\_entry.insert(0, order.getStatus())  
 status\_entry.pack()  
  
 def save\_modified\_order():  
 status = status\_entry.get()  
 if not status:  
 messagebox.showerror("Error", "Order status cannot be empty.")  
 return  
 order.setStatus(status)  
 self.data\_manager.save\_orders(self.orders)  
 messagebox.showinfo("Success", "Order updated successfully!")  
 self.manage\_orders()  
  
 tk.Button(self.welcome\_frame, text="Save", command=save\_modified\_order).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back", command=self.manage\_orders).pack(pady=10)  
  
 def delete\_order(self, order):  
 confirm = messagebox.askyesno("Confirm Delete","Are you sure you want to delete the order '" + order.getOrderId() + "'?")  
  
 if confirm:  
 try:  
 # Remove the order from the list  
 self.orders.remove(order)  
 self.data\_manager.save\_orders(self.orders)  
 messagebox.showinfo("Success", "Order deleted successfully!")  
 self.manage\_orders()  
 # try carch is used to catch any unexpected exception  
 except Exception as e:  
 messagebox.showerror("Error", "Failed to delete order: " + str(e))  
 # Method to show payment screen  
 def show\_payment\_screen(self, booking):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Payment", font=("Arial", 18)).pack(pady=20)  
  
 tk.Label(self.welcome\_frame, text=f"Total Amount: AED{booking.calculateTotal()}").pack(pady=10)  
  
 tk.Label(self.welcome\_frame, text="Payment Method").pack()  
 payment\_method\_var = tk.StringVar(value="Credit/Debit Card")  
 tk.Radiobutton(self.welcome\_frame, text="Credit/Debit Card", variable=payment\_method\_var,value="Credit/Debit Card").pack()  
 tk.Radiobutton(self.welcome\_frame, text="PayPal", variable=payment\_method\_var, value="PayPal").pack()  
  
 tk.Label(self.welcome\_frame, text="Card Number").pack()  
 card\_number\_entry = tk.Entry(self.welcome\_frame)  
 card\_number\_entry.pack()  
 # Processing payment here  
 def process\_payment():  
 payment\_method = payment\_method\_var.get()  
 card\_number = card\_number\_entry.get()  
 # check cardnumber  
 if not card\_number:  
 messagebox.showerror("Error", "Card number is required")  
 return  
  
 payment = Payment(booking.calculateTotal())  
 if payment.processPayment(payment\_method, card\_number):  
 booking.setPayment(payment)  
 if booking.confirmBooking():  
 self.orders.append(Order(booking, payment))  
 self.data\_manager.save\_orders(self.orders)  
 messagebox.showinfo("Success", "Payment successful and booking confirmed!")  
 self.show\_customer\_dashboard()  
 else:  
 messagebox.showerror("Error", "Booking confirmation failed")  
 else:  
 messagebox.showerror("Error", "Payment processing failed")  
  
 tk.Button(self.welcome\_frame, text="Pay Now", command=process\_payment).pack(pady=10)  
 tk.Button(self.welcome\_frame, text="Back",command=lambda: self.book\_tickets(booking.getTickets()[0].getEvent())).pack(pady=10)  
 # Method to should the list of bookings with a method to show ticket details  
 def show\_booking\_list(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Your Bookings", font=("Arial", 18)).pack(pady=20)  
  
 if len(self.current\_user.getBookings()) == 0:  
 tk.Label(self.welcome\_frame, text="No bookings found").pack(pady=10)  
 else:  
 for booking in self.current\_user.getBookings():  
 booking\_frame = tk.Frame(self.welcome\_frame, relief="ridge", borderwidth=1)  
 booking\_frame.pack(fill="x", padx=10, pady=5)  
  
 tk.Label(booking\_frame, text="Booking ID: " + booking.getBookingId()).pack(anchor="w")  
 tk.Label(booking\_frame, text="Booking Date: " + booking.getBookingDate()).pack(anchor="w")  
 tk.Label(booking\_frame, text="Status: " + booking.getStatus()).pack(anchor="w")  
 tk.Label(booking\_frame, text="Total Tickets: " + str(len(booking.getTickets()))).pack(anchor="w")  
  
 def show\_ticket\_details(booking\_obj=booking):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Tickets for Booking ID: " + str(booking\_obj.getBookingId()),  
 font=("Arial", 18)).pack(pady=20)  
 for ticket in booking\_obj.getTickets():  
 ticket\_frame = tk.Frame(self.welcome\_frame, relief="ridge", borderwidth=1)  
 ticket\_frame.pack(fill="x", padx=10, pady=5)  
  
 tk.Label(ticket\_frame, text="Ticket ID: " + str(ticket.getTicketId())).pack(anchor="w")  
 tk.Label(ticket\_frame, text="Type: " + str(ticket.getTicketType())).pack(anchor="w")  
 tk.Label(ticket\_frame, text="Price: AED" + str(ticket.getPrice())).pack(anchor="w")  
 tk.Label(ticket\_frame, text="Seat Number: " + str(ticket.getSeatNumber())).pack(anchor="w")  
 tk.Label(ticket\_frame, text="Purchase Date: " + str(ticket.getPurchaseDate())).pack(anchor="w")  
  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_booking\_list).pack(pady=10)  
  
 tk.Button(booking\_frame, text="View Tickets", command=show\_ticket\_details).pack(anchor="e", pady=5)  
  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_customer\_dashboard).pack(pady=10)  
  
 def view\_ticket\_sales(self):  
 self.clear\_screen()  
 tk.Label(self.welcome\_frame, text="Ticket Sales Report", font=("Arial", 18)).pack(pady=20)  
  
 sales\_data = {}  
 for order in self.orders:  
 order\_date = order.getOrderDate()  
 if order\_date not in sales\_data:  
 sales\_data[order\_date] = 0  
 sales\_data[order\_date] += len(order.getBooking().getTickets())  
  
 if not sales\_data:  
 tk.Label(self.welcome\_frame, text="No sales data available.").pack(pady=10)  
 else:  
 fig, ax = plt.subplots()  
 ax.bar(sales\_data.keys(), sales\_data.values())  
 ax.set\_title("Ticket Sales by Date")  
 ax.set\_xlabel("Date")  
 ax.set\_ylabel("Number of Tickets Sold")  
  
 canvas = FigureCanvasTkAgg(fig, self.welcome\_frame)  
 canvas.get\_tk\_widget().pack(fill="both", expand=True)  
  
 tk.Button(self.welcome\_frame, text="Back", command=self.show\_admin\_dashboard).pack(pady=10)  
  
 def logout(self):  
 self.current\_user = None  
 self.show\_welcome\_screen()  
 def clear\_screen(self):  
 for widget in self.welcome\_frame.winfo\_children():  
 widget.destroy()  
  
#This is where we will be calling the main application  
root = tk.Tk()  
app = TicketBookingApp(root)  
root.mainloop()

**GUI Screenshots**

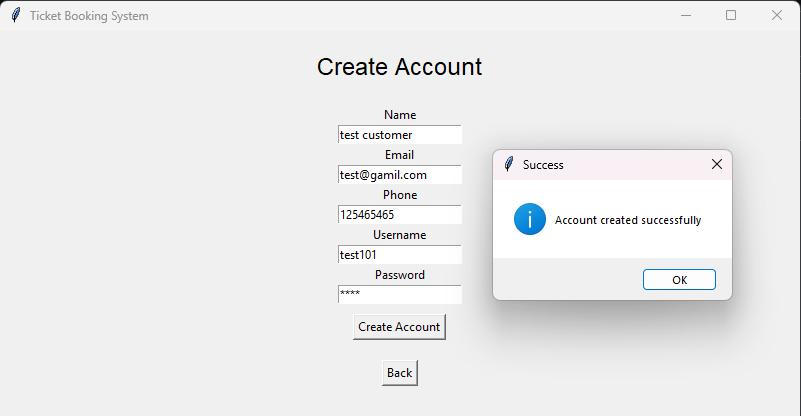
**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a login screen

AI-generated content may be incorrect.**

**Customer Side:**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Admin Side:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer screen

AI-generated content may be incorrect.**