Final Project: Grand Prix Ticket System

Mahra Alshamsi - 202312983

Maryam Alraeesi - 202304233

Salama Almogezwi - 202316361

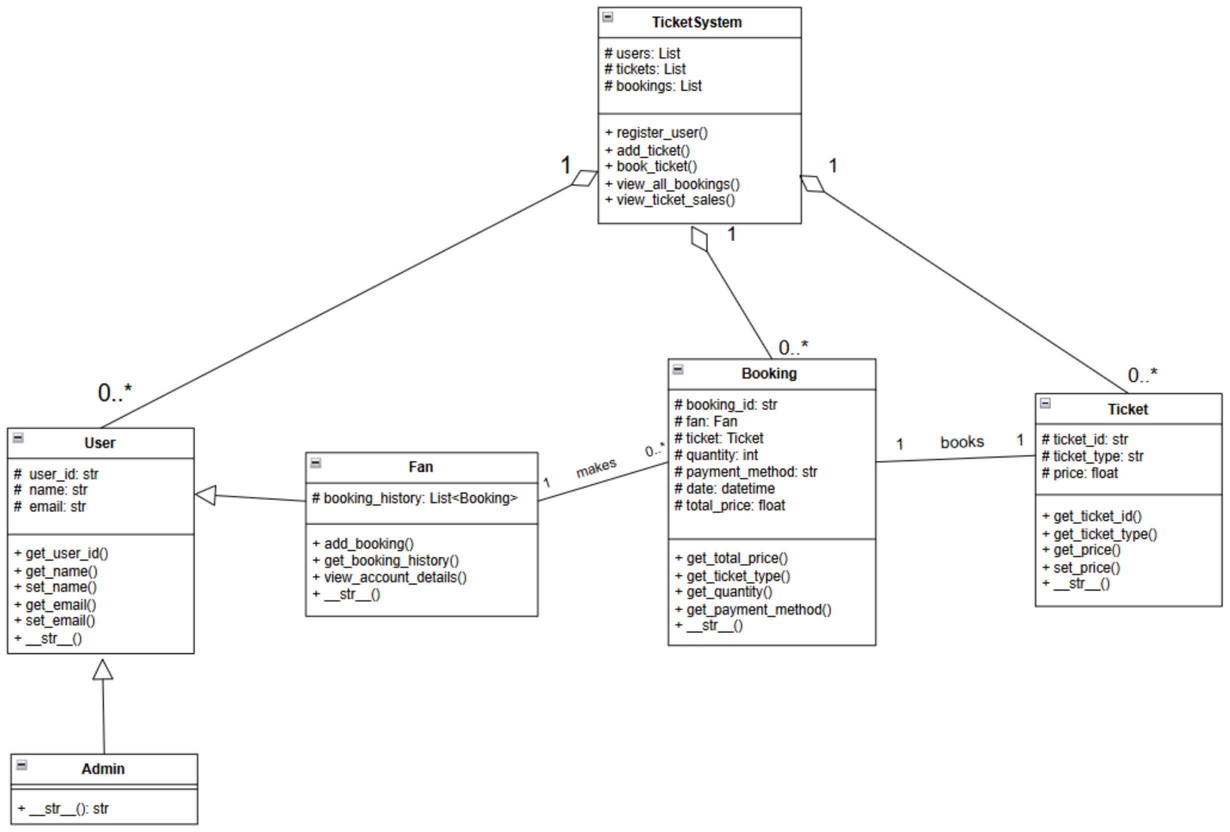
ZAYED UNIVERSITY

ICS220 > 22111 Program. Fund.

Prof Sujith Mathew

May 13, 2025

UML Class Diagram and Description



Classes:

1. TicketSystem:

Attributes: users, tickets, bookings (all protected).
Methods: register_user(), add_ticket(), book_ticket(), view_all_bookings(), and view_ticket_sales().
This class operates as the system's controller, granting access to register users, perform bookings, and manage inventory (tickets).

2. User:

Attributes: user_id, name, and email (all protected)..
Methods: Setters and getters for encapsulations.
Any user of the system is represented abstractly.

3. Fan:

Attributes: booking_history (protected)..
Methods: add_booking(), get_booking_history(), and view_account_details().
Bookings and history viewing are only available to fans, not admins. The base user does not have access to booking features.

4. Admin:

Attributes/Methods: Only implements __str__().

5. Booking:

Attributes: booking_id, fan, ticket, quantity, payment_method, date, and total_price (all protected).
Methods: get_total_price(), get_ticket_type(), get_quantity(), get_payment_method().
Every reservation is for one particular type of ticket in a specific quantity. Every Booking object associates a single ticket type with a single fan.

6. Ticket:

Attributes: ticket_id, ticket_type, price (all protected)..
Methods: get_ticket_id(), get_ticket_type(), get_price(), set_price()
Ticket type refers to the different types of tickets, such as VIP, Standard, Early Bird, and other types. Each ticket type has a different price. Each ticket has a ticket ID.

Relationships:

Association (has-a relationship):

1. **Fan class - Booking class:**
   a. Multiplicity
      i. Fan: 1
      ii. Booking: 0..*
   b. A fan may make multiple reservations, but only one fan makes each reservation.

2. **Booking class - Ticket class:**
   a. Multiplicity
      i. Booking: 1
      ii. Ticket: 1
   b. Each booking is associated with a single ticket type (e.g., VIP or General).

Aggregation (whole-part):

1. **TicketSystem class - User class:**
   a. Multiplicity
      i. TicketSystem: 1
      ii. User: 0..*
   b. Users can exist independently of the system.

2. **TicketSystem class - Ticket class:**
   a. Multiplicity
      i. TicketSystem: 1
      ii. Ticket: 0..*

b. The system manages tickets; however, they remain in existence in the case of a system failure.

3. **TicketSystem class - Booking class:**

    a. Multiplicity

        i. TicketSystem: 1

        ii. Booking: 0..*

    b. Although they are logically a part of the system, bookings can exist independently.

Inheritance (is-a relationship):

1. **Fan class and Admin class inherit from User class.**

    a. All admins and fans are users.

    b. User defines common features like name, email, and user_id.

Python classes

1. user.py (User classes)

```
class User:
   """Base class for all users in the system."""


   def __init__(self, user_id, name, email):
       # Initialize user attributes
       self._user_id = user_id
       self._name = name
```

```
        self._email = email

    def get_user_id(self):
        # Return the user ID
        return self._user_id

    def get_name(self):
        # Return the name of the user
        return self._name

    def set_name(self, name):
        # Set a new name for the user
        self._name = name

    def get_email(self):
        # Return the email address of the user
        return self._email

    def set_email(self, email):
        # Set a new email address for the user
        self._email = email

    def __str__(self):
        # Return a string representation of the User object
        return f"User ID: {self._user_id}, Name: {self._name}, Email:
{self._email}"


class Fan(User):
    """A fan who can purchase tickets and manage bookings."""

    def __init__(self, user_id, name, email):
        # Initialize fan attributes, including booking history
        super().__init__(user_id, name, email)
        self._booking_history = []  # Stores the booking history for the fan

    def add_booking(self, booking):
        # Add a booking to the fan's booking history
        self._booking_history.append(booking)

    def get_booking_history(self):
        # Return the fan's booking history as a list
        return self._booking_history

    def view_account_details(self):
        # Return a summary of the fan's account details
        return f"Fan Account - Name: {self._name}, Email: {self._email},
 Bookings: {len(self._booking_history)}"

    def __str__(self):
        # Return a string representation of the Fan object, including booking
 count
        return f"Fan: {super().__str__()} | Bookings:
 {len(self._booking_history)}"


 class Admin(User):
    """Admin user who can view ticket sales."""
```

```
def __init__(self, user_id, name, email):

    # Initialize admin attributes

    super().__init__(user_id, name, email)


def __str__(self):

    # Return a string representation of the Admin object

    return f"Admin: {super().__str__()}"
```

2. ticket.py (Ticket class only)

```
class Ticket:
   """Represents a ticket type for the Grand Prix."""


def __init__(self, ticket_id, ticket_type, price):

    # Initialize ticket attributes

    self._ticket_id = ticket_id

    self._ticket_type = ticket_type

    self._price = price


def get_ticket_id(self):

    # Return the unique ID of the ticket

    return self._ticket_id


def get_ticket_type(self):

    # Return the type of the ticket (e.g., VIP, General, Student)

    return self._ticket_type
    def get_price(self):

        # Return the price of the ticket

        return self._price


    def set_price(self, price):

        # Update the price of the ticket

        self._price = price


    def __str__(self):

        # Return a string representation of the Ticket object

        return f"Ticket[{self._ticket_id}] {self._ticket_type} - AED
{self._price}"
```

3. booking.py (Booking class only)

```
from datetime import datetime


class Booking:

    """Manages a ticket booking by a fan."""


    def __init__(self, booking_id, fan, ticket, quantity, payment_method):

        # Initialize booking attributes

        self._booking_id = booking_id

        self._fan = fan

        self._ticket = ticket

        self._quantity = quantity

        self._payment_method = payment_method

        self._date = datetime.now()

        self._total_price = self._calculate_total()
        # Automatically add this booking to the fan's booking history
        fan.add_booking(self)

    def _calculate_total(self):
        # Calculate the total price, applying a discount for bulk purchases (5
or more tickets)
        base_total = self._ticket.get_price() * self._quantity
        if self._quantity >= 5:
            return base_total * 0.9  # 10% discount for bulk purchases
        return base_total

    def get_total_price(self):
        # Return the total price of the booking
        return self._total_price

    def get_ticket_type(self):
        # Return the type of ticket booked
        return self._ticket.get_ticket_type()

    def get_quantity(self):
        # Return the quantity of tickets booked
        return self._quantity

    def get_payment_method(self):
        # Return the payment method used for this booking
        return self._payment_method

    def __str__(self):
        # Return a string representation of the Booking object

        return (f"Booking[{self._booking_id}] -

 {self._ticket.get_ticket_type()} x {self._quantity} "

                f"on {self._date.strftime('%Y-%m-%d')} | Payment:

 {self._payment_method} | "

                f"Total: AED {self._total_price:.2f}")
```

4. ticket_system.py

```
from user import Fan, Admin

from ticket import Ticket

from booking import Booking


class TicketSystem:

    """Manages users, tickets, and bookings."""


    def __init__(self):

        # Initialize lists for users, tickets, and bookings

        self._users = []

        self._tickets = []

        self._bookings = []


    def register_user(self, user):

        # Add a user to the system

        self._users.append(user)


    def add_ticket(self, ticket):

        # Add a ticket to the system
        self._tickets.append(ticket)


    def book_ticket(self, fan, ticket_id, quantity, payment_method):
        # Find the ticket by ID and create a booking if it exists
        ticket = next((t for t in self._tickets if t.get_ticket_id() ==
ticket_id), None)
        if ticket:
            # Generate a unique booking ID
            booking_id = f"B{len(self._bookings) + 1}"
            # Create a new booking and add it to the bookings list
            booking = Booking(booking_id, fan, ticket, quantity,
payment_method)
            self._bookings.append(booking)
            return booking
        else:
            # Raise an error if the ticket ID is invalid
            raise ValueError("Invalid Ticket ID")


    def view_all_bookings(self):
        # Return the list of all bookings
        return self._bookings


    def view_ticket_sales(self):
        # Calculate total tickets sold for each ticket type
        sales = {}
        for b in self._bookings:
            ticket_type = b.get_ticket_type()
            sales[ticket_type] = sales.get(ticket_type, 0) + b.get_quantity()
        return sales
```

5. Test_ticket_system.py

```
from user import Fan, Admin

from ticket import Ticket

from ticket_system import TicketSystem


# --- Testing code ---

if __name__ == "__main__":

    # Create a new ticket system instance

    system = TicketSystem()


    # Create a fan and admin user

    fan1 = Fan("F001", "Maryam", "maryam@example.com")

    admin = Admin("A001", "AdminUser", "admin@example.com")

    # Register the users

    system.register_user(fan1)

    system.register_user(admin)


    # Create tickets for different types

    ticket1 = Ticket("T001", "Single Race", 350)

    ticket2 = Ticket("T002", "Weekend Package", 900)

    ticket3 = Ticket("T003", "Season Membership", 3000)

    # Add tickets to the system

    system.add_ticket(ticket1)

    system.add_ticket(ticket2)

    system.add_ticket(ticket3)
    # Make bookings for the fan
    booking1 = system.book_ticket(fan1, "T001", 2, "Credit Card")
    booking2 = system.book_ticket(fan1, "T002", 5, "Digital Wallet")

    # Print the booking history of the fan
    print("\nFan Booking History:")
    for b in fan1.get_booking_history():
        print(b)

    # Print the ticket sales summary for the admin
    print("\nAdmin View - Ticket Sales Summary:")
    sales = system.view_ticket_sales()
    for ticket_type, count in sales.items():
        print(f"{ticket_type}: {count} tickets sold")
```

The output:

```
Fan Booking History:
Booking[B1] - Single Race x 2 on 2025-05-08 | Payment: Credit Card | Total: AED 700.00
Booking[B2] - Weekend Package x 5 on 2025-05-08 | Payment: Digital Wallet | Total: AED 4050.00

Admin View - Ticket Sales Summary:
Single Race: 2 tickets sold
Weekend Package: 5 tickets sold
```

Graphical User Interface (GUI)

1. load_default_tickets.py

Purpose:

This script creates a set of default ticket types and saves them into a file called tickets.pkl using Python's pickle module.

Why it was needed:

- When you first run the system, tickets.pkl is empty or doesn't exist.
- Without predefined tickets (like "Single Race Pass" or "Weekend Package"), the user interface has nothing to display for booking.
- This file ensures that fans always see a list of realistic, usable ticket options with prices and types, right from the start.

How it works:

- Defines several Ticket objects with IDs, types, and prices.
- Serializes them into tickets.pkl using pickle.dump(), making them available when the app runs.

```
# Import the 'pickle' module to serialize and save Python objects to a file

import pickle



# Import the Ticket class from the ticket module

from ticket import Ticket
```

```
# Create a list of default ticket types using the Ticket class

default_tickets = [

   Ticket("T001", "Single Race Pass", 350),       # A single race ticket priced
at 350

   Ticket("T002", "Weekend Package", 900),        # A weekend access package
ticket priced at 900

   Ticket("T003", "Season Membership", 3000),     # A season-long membership
ticket priced at 3000

   Ticket("T004", "Group Discount (5+)", 320)     # A group ticket (5 or more
people) priced at 320 each

]



# Open a file called 'tickets.pkl' in binary write mode ('wb')

with open("tickets.pkl", "wb") as f:

   # Serialize and save the list of default_tickets to the file

   pickle.dump(default_tickets, f)



# Print a confirmation message indicating success

print("Default tickets loaded into tickets.pkl")
```

2. load_default_admin.py

Purpose:
This script creates a default `Admin` user and saves it into `users.pkl`.

Why it was needed:

- The admin dashboard (`admin_gui.py`) requires a valid Admin ID to log in.
- Without any admin user in `users.pkl`, login attempts will always fail with "Admin not found."
- This script solves that by adding a sample admin (e.g., ID: `1234`) so we can access and test the admin dashboard.

How it works:

- Loads existing users from `users.pkl`.
- Checks if the admin already exists.
- If not, creates a new `Admin` object and adds it.
- Saves the updated list back to the file using `pickle`.

```
import pickle
from user import Admin

# Create a sample admin user
admin = Admin("1234", "AdminUser", "admin@example.com")

# Load existing users or create new list
try:
   with open("users.pkl", "rb") as f:
       users = pickle.load(f)
except:
   users = []

# Add admin only if not already present
if not any(u.get_user_id() == "1234" for u in users):
   users.append(admin)

# Save back to file
with open("users.pkl", "wb") as f:
   pickle.dump(users, f)

print("Admin user '1234' added.")
```

3. account_gui.py

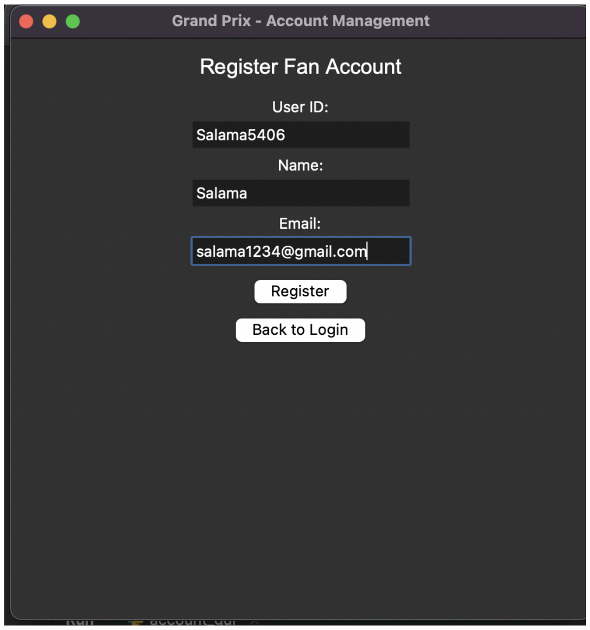
```
# Import the required modules for GUI, data storage, and file handling
import tkinter as tk
from tkinter import messagebox
import pickle
import os

# Import the Fan and Admin classes from the user module
from user import Fan, Admin

# Function to save data (e.g., user list) to a file using pickle
def save_data(filename, data):
  with open(filename, 'wb') as f:
      pickle.dump(data, f)

# Function to load data from a file using pickle, returns an empty list if
file doesn't exist
def load_data(filename):
  if os.path.exists(filename):
      with open(filename, 'rb') as f:
          return pickle.load(f)
  return []

# Load the user data from 'users.pkl' when the program starts
users = load_data('users.pkl')

# Define the main GUI class for account management, inheriting from Tkinter's
Tk class
class AccountGUI(tk.Tk):
  def __init__(self):
      super().__init__()  # Initialize the parent class
      self.title("Grand Prix - Account Management")  # Set window title
      self.geometry("500x500")  # Set window size
      self.current_user = None  # Placeholder for the currently logged-in user
      self.init_login_screen()  # Load the login screen on start

  # Function to initialize the login screen interface
  def init_login_screen(self):
      self.clear_widgets()  # Clear previous widgets (if any)

      tk.Label(self, text="Login", font=("Arial", 18)).pack(pady=10)

      tk.Label(self, text="User ID:").pack()
      self.user_id_entry = tk.Entry(self)  # Input for user ID
      self.user_id_entry.pack()

      tk.Button(self, text="Login", command=self.login_user).pack(pady=5)
      tk.Button(self, text="Create New Fan Account",
command=self.init_register_screen).pack()

  # Function to initialize the registration screen interface
  def init_register_screen(self):
      self.clear_widgets()  # Clear previous widgets

      tk.Label(self, text="Register Fan Account", font=("Arial",
18)).pack(pady=10)

      tk.Label(self, text="User ID:").pack()
      self.reg_id_entry = tk.Entry(self)  # Input for new user ID
```

```
        self.reg_id_entry.pack()

        tk.Label(self, text="Name:").pack()
        self.reg_name_entry = tk.Entry(self)  # Input for name
        self.reg_name_entry.pack()

        tk.Label(self, text="Email:").pack()
        self.reg_email_entry = tk.Entry(self)  # Input for email
        self.reg_email_entry.pack()

        tk.Button(self, text="Register", command=self.register_fan).pack(pady=5)
        tk.Button(self, text="Back to Login",
command=self.init_login_screen).pack()

    # Function to register a new fan account
    def register_fan(self):
        uid = self.reg_id_entry.get()
        name = self.reg_name_entry.get()
        email = self.reg_email_entry.get()

        # Check if user ID already exists
        if any(u.get_user_id() == uid for u in users):
            messagebox.showerror("Error", "User ID already exists.")
            return

        # Create a new fan object and save it
        new_fan = Fan(uid, name, email)
        users.append(new_fan)
        save_data('users.pkl', users)
        messagebox.showinfo("Success", "Fan account created.")
        self.init_login_screen()  # Go back to login after registration

    # Function to log in a user
    def login_user(self):
        uid = self.user_id_entry.get()
        # Search for the user by ID
        matched = next((u for u in users if u.get_user_id() == uid), None)

        if matched:
            self.current_user = matched  # Set current user
            self.init_account_dashboard()  # Load account dashboard
        else:
            messagebox.showerror("Error", "User ID not found.")

    # Function to display account dashboard after login
    def init_account_dashboard(self):
        self.clear_widgets()
        # Welcome message
        tk.Label(self, text=f"Welcome {self.current_user.get_name()}",
font=("Arial", 16)).pack(pady=10)

        # Show user account details
        tk.Label(self, text=self.current_user.view_account_details(),
font=("Arial", 12)).pack(pady=5)

        # Show ticket booking history only for Fan users
        if isinstance(self.current_user, Fan):
            bookings = self.current_user.get_booking_history()
```

```
          if bookings:
              tk.Label(self, text="Your Booked Tickets:", font=("Arial",
14)).pack(pady=10)
              for booking in bookings:
                  tk.Label(self, text=str(booking), wraplength=480,
justify="left", anchor="w").pack(anchor="w", padx=15, pady=5)
          else:
              tk.Label(self, text="No tickets booked yet.").pack(pady=10)

      # Provide options to delete account or logout
      tk.Button(self, text="Delete Account", fg="red",
command=self.delete_account).pack(pady=5)
      tk.Button(self, text="Logout", command=self.init_login_screen).pack()

  # Function to delete the current user account
  def delete_account(self):
      users.remove(self.current_user)  # Remove user from list
      save_data('users.pkl', users)  # Save updated list
      messagebox.showinfo("Deleted", "Your account has been deleted.")
      self.current_user = None
      self.init_login_screen()  # Return to login screen

  # Helper function to clear all widgets from the screen
  def clear_widgets(self):
      for widget in self.winfo_children():
          widget.destroy()

# Run the application if this file is executed directly
if __name__ == "__main__":
```

```
app = AccountGUI()

app.mainloop()
```

First account:



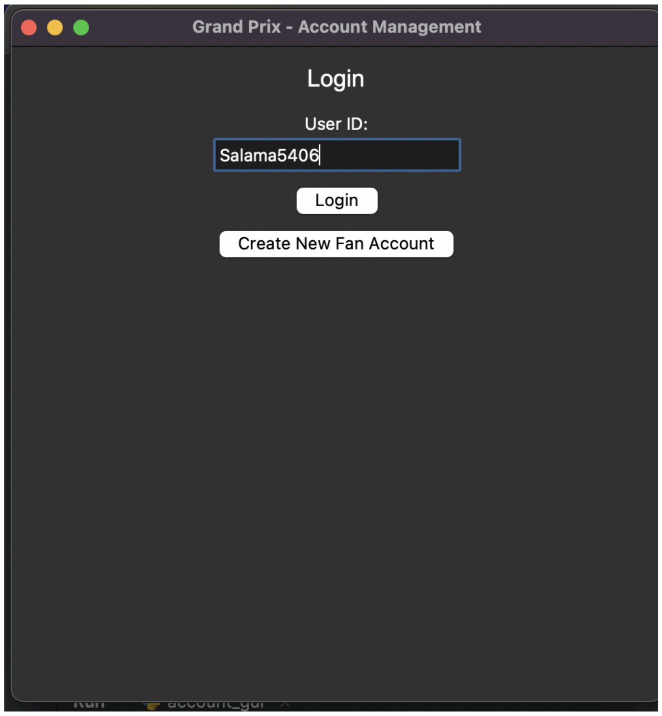
Grand Prix - Account Management

Register Fan Account

User ID: Salama5406

Name: Salama

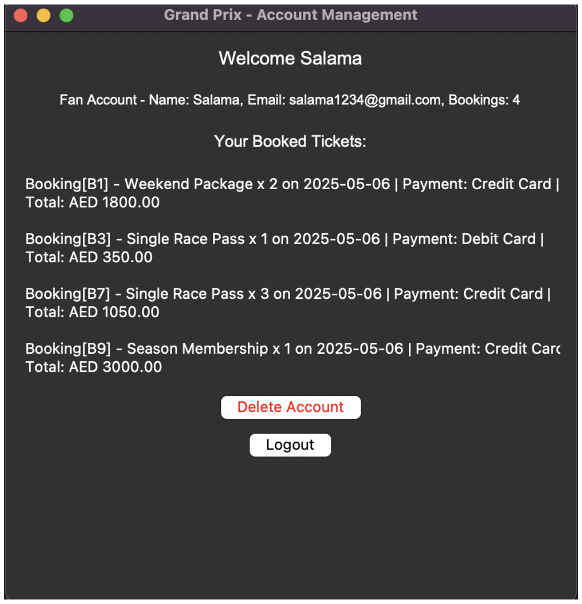
Email: salama1234@gmail.com



Grand Prix - Account Management

Login

User ID: Salama5406



Grand Prix - Account Management

Welcome Salama

Fan Account - Name: Salama, Email: salama1234@gmail.com, Bookings: 4

Your Booked Tickets:

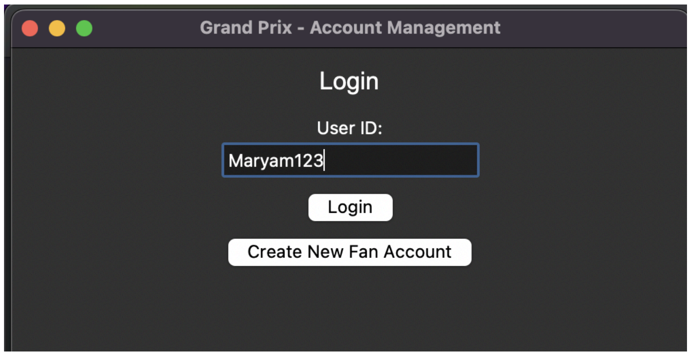
Booking[B1] - Weekend Package x 2 on 2025-05-06 | Payment: Credit Card | Total: AED 1800.00

Booking[B3] - Single Race Pass x 1 on 2025-05-06 | Payment: Debit Card | Total: AED 350.00

Booking[B7] - Single Race Pass x 3 on 2025-05-06 | Payment: Credit Card | Total: AED 1050.00

Booking[B9] - Season Membership x 1 on 2025-05-06 | Payment: Credit Carc Total: AED 3000.00

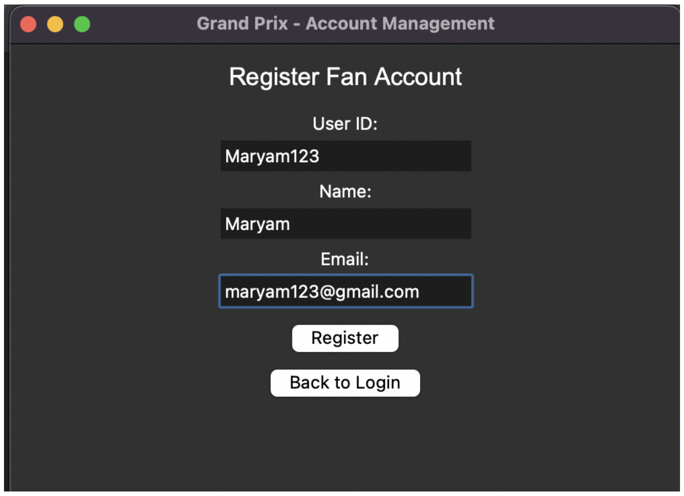
Second Account:


Grand Prix - Account Management

Login

User ID:

Maryam123


Grand Prix - Account Management

Register Fan Account

User ID:

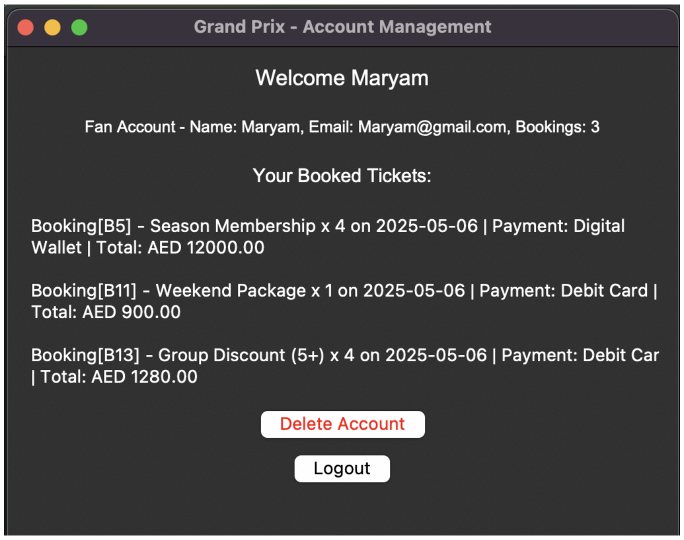
Maryam123

Name:

Maryam

Email:

maryam123@gmail.com


Grand Prix - Account Management

Welcome Maryam

Fan Account - Name: Maryam, Email: Maryam@gmail.com, Bookings: 3

Your Booked Tickets:

Booking[B5] - Season Membership x 4 on 2025-05-06 | Payment: Digital Wallet | Total: AED 12000.00

Booking[B11] - Weekend Package x 1 on 2025-05-06 | Payment: Debit Card | Total: AED 900.00

Booking[B13] - Group Discount (5+) x 4 on 2025-05-06 | Payment: Debit Car | Total: AED 1280.00

4. ticket_gui.py

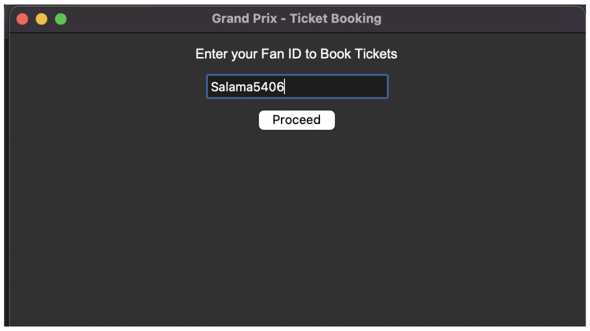
```
# ---------- Imports ----------

import tkinter as tk  # GUI framework
```

```
from tkinter import messagebox  # For pop-up error/info dialogs

import pickle  # For saving/loading data

import os  # For file existence checking


# Import user and system-related classes

from user import Fan

from ticket import Ticket

from ticket_system import TicketSystem




# ---------- Utility Functions ----------

# Save data to a file using pickle

def save_data(filename, data):

  with open(filename, 'wb') as f:

      pickle.dump(data, f)


# Load data from a file if it exists, otherwise return an empty list

def load_data(filename):

  if os.path.exists(filename):

      with open(filename, 'rb') as f:

          return pickle.load(f)

  return []




# ---------- Load Existing Data ----------

# Load users, tickets, and bookings data from files

users = load_data('users.pkl')

tickets = load_data('tickets.pkl')

bookings = load_data('bookings.pkl')
```

```
# ---------- Setup Ticket System ----------
# Create an instance of the TicketSystem and assign the loaded data
system = TicketSystem()
system._users = users
system._tickets = tickets
system._bookings = bookings


# ---------- GUI Class ----------
class TicketGUI(tk.Tk):
  def __init__(self):
      super().__init__()
      self.title("Grand Prix - Ticket Booking")  # Set window title
      self.geometry("600x650")  # Set window size
      self.current_fan = None  # Currently logged in Fan
      self.init_login_screen()  # Start with the login screen

  # Login screen to enter Fan ID
  def init_login_screen(self):
      self.clear_widgets()  # Clear any existing widgets
      tk.Label(self, text="Enter your Fan ID to Book Tickets", font=("Arial",
14)).pack(pady=10)
      self.fan_id_entry = tk.Entry(self)  # Entry field for Fan ID
      self.fan_id_entry.pack()
      tk.Button(self, text="Proceed", command=self.load_fan).pack(pady=5)  #
Proceed button
```

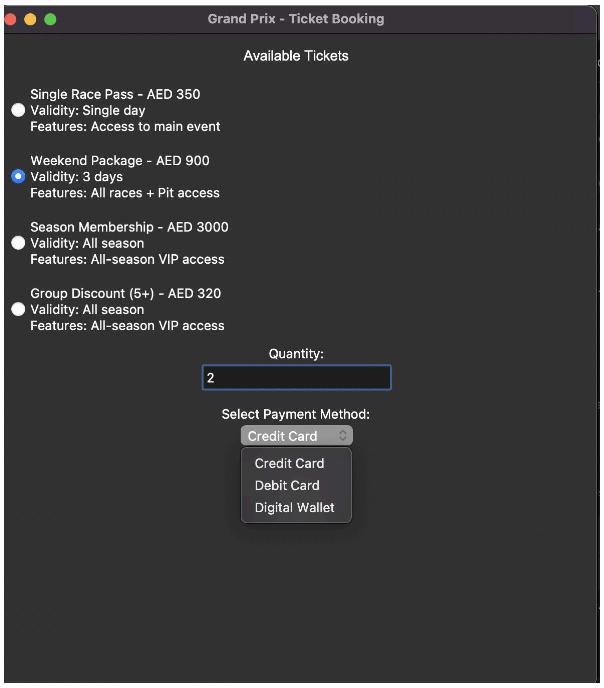
```
  # Load fan details based on entered ID
  def load_fan(self):
      uid = self.fan_id_entry.get()
      # Search for a matching Fan object in the users list
      matched = next((u for u in users if isinstance(u, Fan) and
u.get_user_id() == uid), None)
      if matched:
          self.current_fan = matched  # Save the fan object
          self.init_ticket_booking_screen()  # Proceed to ticket booking
      else:
          messagebox.showerror("Error", "Fan not found.")  # Show error if not
found

  # Show screen with available tickets and booking form
  def init_ticket_booking_screen(self):
      self.clear_widgets()
      tk.Label(self, text="Available Tickets", font=("Arial",
14)).pack(pady=10)

      self.ticket_var = tk.StringVar()  # Variable to hold selected ticket ID

      if not tickets:
          tk.Label(self, text="No tickets available.").pack()
          return

      # Display each ticket as a radio button with details
      for ticket in tickets:
          desc = (
              f"{ticket.get_ticket_type()} - AED {ticket.get_price()}\n"
              f"Validity: {'Single day' if 'Single' in
ticket.get_ticket_type() else '3 days' if 'Weekend' in
ticket.get_ticket_type() else 'All season'}\n"
              f"Features: {'Access to main event' if 'Single' in
ticket.get_ticket_type() else 'All races + Pit access' if 'Weekend' in
ticket.get_ticket_type() else 'All-season VIP access'}"
          )
          tk.Radiobutton(
              self,
              text=desc,
              variable=self.ticket_var,
              value=ticket.get_ticket_id(),
              justify="left",
              anchor="w",
              wraplength=500
          ).pack(anchor="w", padx=10, pady=5)

      # Quantity input
      tk.Label(self, text="Quantity:").pack()
      self.quantity_entry = tk.Entry(self)
      self.quantity_entry.pack()

      # Payment method selection
      tk.Label(self, text="Select Payment Method:").pack(pady=(10, 0))
      self.payment_var = tk.StringVar()
      self.payment_dropdown = tk.OptionMenu(self, self.payment_var, "Credit
Card", "Debit Card", "Digital Wallet")
      self.payment_dropdown.pack()
      # Book and back buttons
      tk.Button(self, text="Book Ticket",
command=self.book_ticket).pack(pady=10)
      tk.Button(self, text="Back", command=self.init_login_screen).pack()

  # Process ticket booking
  def book_ticket(self):
      ticket_id = self.ticket_var.get()
      try:
          quantity = int(self.quantity_entry.get())  # Convert quantity to int
          payment_method = self.payment_var.get()
          if not payment_method:
              messagebox.showerror("Error", "Please select a payment method.")
              return
          # Book ticket through the system
          booking = system.book_ticket(self.current_fan, ticket_id, quantity,
payment_method)
          bookings.append(booking)  # Add booking to list
          save_data('bookings.pkl', bookings)  # Save bookings
          save_data('users.pkl', users)  # Update users with booking info
          self.show_confirmation(booking)  # Show confirmation screen
      except ValueError as e:
          messagebox.showerror("Error", str(e))  # Catch invalid input errors

  # Show confirmation after booking
  def show_confirmation(self, booking):
      self.clear_widgets()
      tk.Label(self, text="Booking Confirmed!", font=("Arial", 18),
fg="green").pack(pady=15)
```

```
      tk.Label(self, text=str(booking), wraplength=500,
justify="left").pack(pady=10)
      tk.Button(self, text="Book Another Ticket",
command=self.init_ticket_booking_screen).pack(pady=5)
      tk.Button(self, text="Back to Start",
command=self.init_login_screen).pack(pady=5)

  # Utility function to remove all widgets from current screen
  def clear_widgets(self):
      for widget in self.winfo_children():
          widget.destroy()



# ---------- Run the Application ----------
if __name__ == "__main__":
  app = TicketGUI()  # Create an instance of the TicketGUI
  app.mainloop()  # Start the Tkinter main event loop
```


Grand Prix - Ticket Booking

Enter your Fan ID to Book Tickets

Salama5406


Grand Prix - Ticket Booking

Available Tickets

○ Single Race Pass - AED 350
Validity: Single day
Features: Access to main event

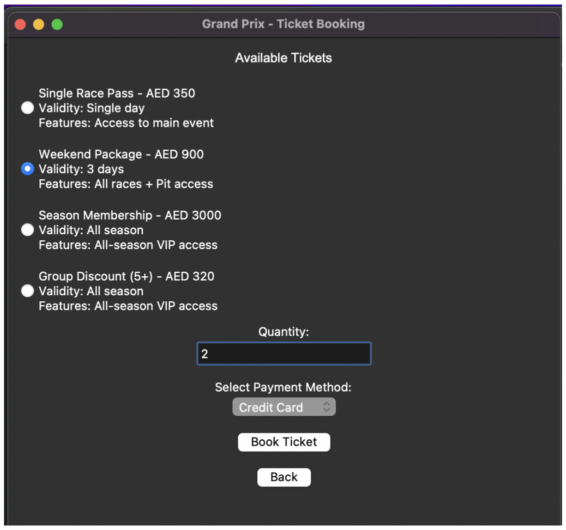
● Weekend Package - AED 900
Validity: 3 days
Features: All races + Pit access

○ Season Membership - AED 3000
Validity: All season
Features: All-season VIP access

○ Group Discount (5+) - AED 320
Validity: All season
Features: All-season VIP access

Quantity:
2

Select Payment Method:
Credit Card


Grand Prix - Ticket Booking

Available Tickets

○ Single Race Pass - AED 350
Validity: Single day
Features: Access to main event

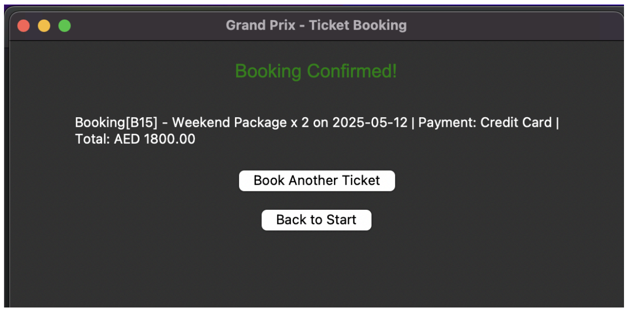
● Weekend Package - AED 900
Validity: 3 days
Features: All races + Pit access

○ Season Membership - AED 3000
Validity: All season
Features: All-season VIP access

○ Group Discount (5+) - AED 320
Validity: All season
Features: All-season VIP access

Quantity:
2

Select Payment Method:
Credit Card


Grand Prix - Ticket Booking

Booking Confirmed!

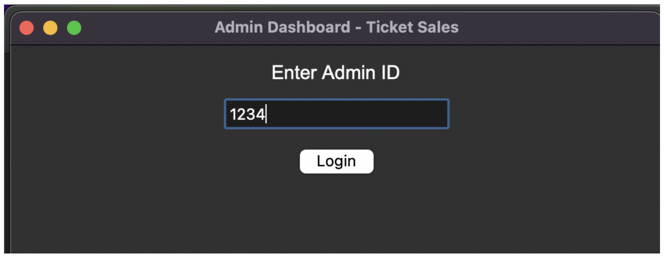
Booking[B15] - Weekend Package x 2 on 2025-05-12 | Payment: Credit Card |
Total: AED 1800.00

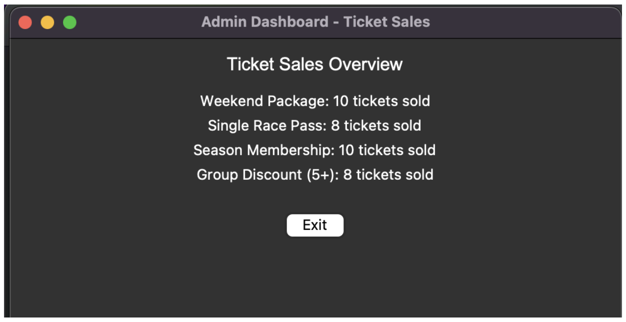
5. admin_gui.py

```
# ---------- Imports ----------

import tkinter as tk  # Tkinter for GUI

from tkinter import messagebox  # For displaying error/info pop-ups

import pickle  # For saving/loading Python objects

import os  # For checking file existence
```

```python
# Import classes needed for admin and ticket system functionality
from user import Admin
from ticket_system import TicketSystem


# ---------- Utility Functions ----------
# Function to load data from a file if it exists
def load_data(filename):
  if os.path.exists(filename):
      with open(filename, 'rb') as f:
          return pickle.load(f)
  return []  # Return empty list if file doesn't exist


# ---------- Load Existing Data ----------
users = load_data('users.pkl')      # Load user data (including Admins)
bookings = load_data('bookings.pkl')  # Load bookings
tickets = load_data('tickets.pkl')    # Load available tickets


# ---------- Setup Ticket System ----------
# Create and configure the system with loaded data
system = TicketSystem()
system._users = users
system._bookings = bookings
system._tickets = tickets


# ---------- Admin GUI Class ----------
```

```python
class AdminGUI(tk.Tk):
  def __init__(self):
      super().__init__()  # Initialize parent class
      self.title("Admin Dashboard - Ticket Sales")  # Window title
      self.geometry("550x400")  # Window size
      self.init_login_screen()  # Show login screen initially

  # Display login screen for admin
  def init_login_screen(self):
      self.clear_widgets()  # Clear old widgets
      tk.Label(self, text="Enter Admin ID", font=("Arial", 16)).pack(pady=10)
      self.admin_id_entry = tk.Entry(self)  # Entry field for admin ID
      self.admin_id_entry.pack()
      tk.Button(self, text="Login", command=self.validate_admin).pack(pady=10)

  # Validate entered admin ID
  def validate_admin(self):
      aid = self.admin_id_entry.get()
      # Find matching Admin object by user ID
      matched = next((a for a in users if isinstance(a, Admin) and
a.get_user_id() == aid), None)
      if matched:
          self.init_dashboard()  # If valid, go to dashboard
      else:
          messagebox.showerror("Error", "Admin not found.")  # Show error if
invalid

  # Display admin dashboard with ticket sales stats
  def init_dashboard(self):
      self.clear_widgets()  # Clear old screen
      tk.Label(self, text="Ticket Sales Overview", font=("Arial",
16)).pack(pady=10)

      sales = system.view_ticket_sales()  # Get ticket sales summary from
system

      # Display sales summary by ticket type
      for ticket_type, count in sales.items():
          tk.Label(self, text=f"{ticket_type}: {count} tickets sold").pack()

      tk.Button(self, text="Exit", command=self.quit).pack(pady=20)  # Exit
button

  # Utility function to clear the current window widgets
  def clear_widgets(self):
      for widget in self.winfo_children():
          widget.destroy()


# ---------- Run the Application ----------
if __name__ == "__main__":
  app = AdminGUI()  # Create instance of the admin GUI
  app.mainloop()  # Start the Tkinter event loop
```





Ticket Sales Overview

Weekend Package: 10 tickets sold
Single Race Pass: 8 tickets sold
Season Membership: 10 tickets sold
Group Discount (5+): 8 tickets sold

Github repository link

https://github.com/MaryamAlraeesi14/Final-Project_Team3

Summary of learnings

Maryam:

Developing the Grand Prix Ticket Management System provided valuable hands-on experience with key object-oriented programming (OOP) concepts in Python. I learned the importance of designing clear class structures and using inheritance to reduce code duplication. The project included creating a User-based class with specialized Fan and Admin subclasses, demonstrating the power of inheritance for organized and reusable code. I also practiced composition by linking Fan and Booking classes, reflecting real-world relationships, and used method overriding to customize __str__() methods for more meaningful outputs. Additionally, I focused on input validation and error handling to ensure system reliability, like handling invalid ticket IDs. Overall, this project improved my understanding of OOP principles and prepared me for building more complex, maintainable systems in the future.

Mahra:

I helped design and create a full ticket booking system for the Grand Prix Experience as part of this project, with the focus on improving user engagement and expediting ticket administration. Using object-oriented principles like inheritance (Fan and Admin inherit from User) and class associations to depict real-world relationships, I helped develop the main classes, which include User, Fan, Admin, Ticket, Booking, and TicketSystem. Important features, including user registration, ticket registration, discount calculations, and booking history management, were also defined. Through the process, I visualize system structure using UML diagrams and distinguish between relationships to write clean, functional Python code that reflects those relationships in real-world scenarios.

Salama:

I've gained a deeper understanding of object-oriented programming, Pickle file handling, and Python GUI development with Tkinter thanks to this project. I gained knowledge on how to create an interactive, multi-window application that enables various user roles, such as administrators and fans, to carry out particular functions like managing accounts, viewing sales, and making bookings. Putting classes like Fan, Admin, Ticket, and TicketSystem into practice made it easier for me to understand how modular design and encapsulation improve program maintainability. Additionally, I learned how to process user input, display dynamic data, validate entries, and connect backend functionality to frontend interfaces. All things considered, this experience equipped me with useful skills for creating data-driven, user-friendly applications.