Programming Final Assignment

Shahad Marwan Alkhlaifi (202316653)

Reem Fahad Alhamami (202302133)

Fatima Obaid Alqaoud (202306391)

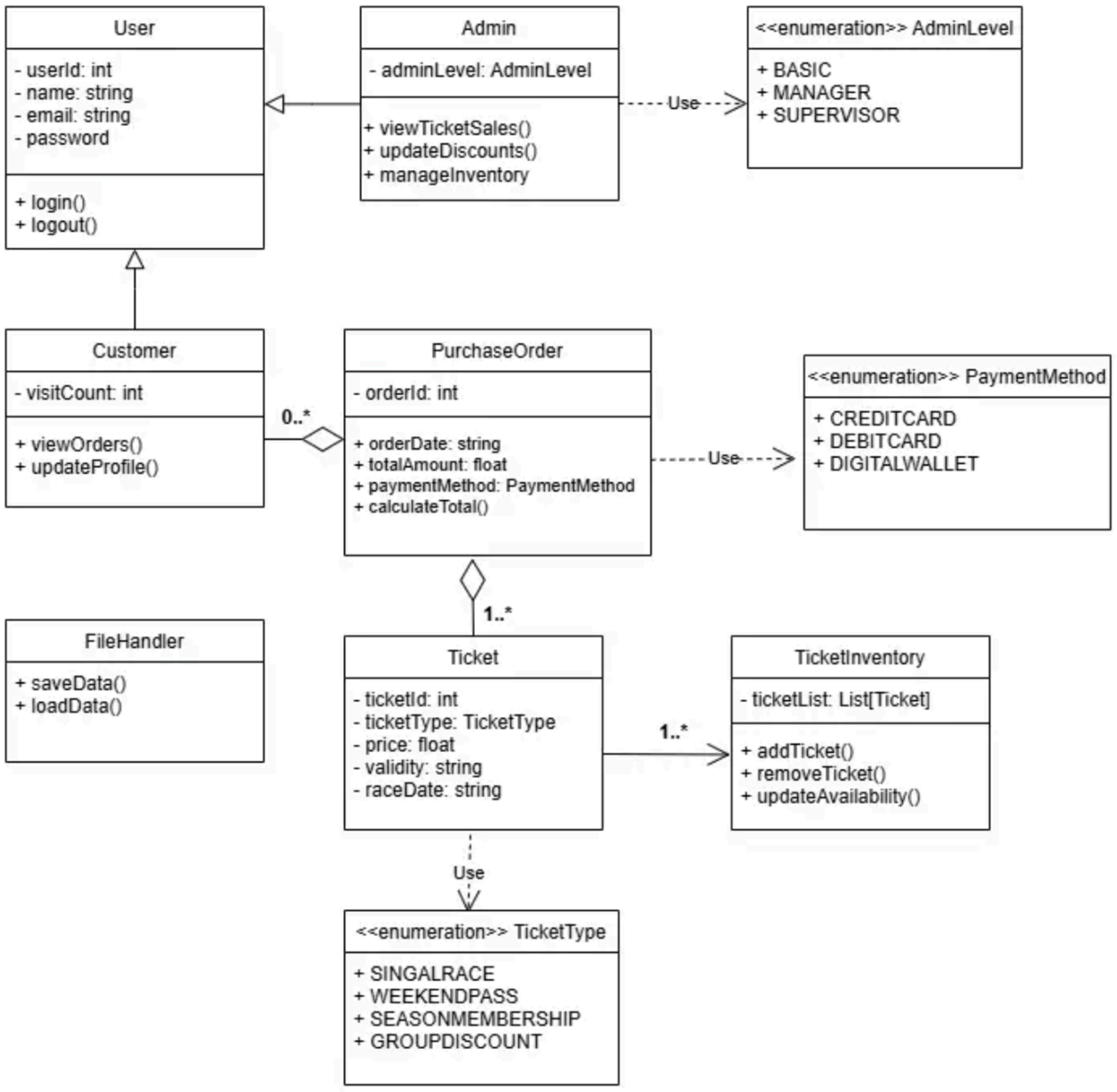
ICS220 - 23018 Program. Fund.

9 - May - 2025

Prof Areej Abdulfattah

PLANNING & UML DESIGN

UML Class Diagram



UML Class Diagram explanation:

This is the UML diagram for the structure of the Grand Prix ticket booking system. It consists of various components used to manage users, ticket orders, and data saving. The central class is User, which contains common information such as user ID, name, email, and password. There are two types of users: administrators and consumers. Administrators can view ticket sales, apply discounts and manage system tickets. Consumers can see only their orders and update their profile. Depending on what permissions they have as an administrator, they can be BASIC, MANAGER, or SUPERVISOR. When the tickets are purchased by the customer, the order is saved in a PurchaseOrder class. The class saves the order date, total cost, and payment method, i.e., credit card or electronic wallet. An order can consist of one or more tickets. The Ticket class keeps track of information including ticket type, price, and race date. Tickets could come in many forms including single race, weekend pass, season membership, and group discount. A TicketInventory class stores tickets with a list of active tickets and add, removes, and updates capabilities for them. There is also a FileHandler class used to save and retrieve data to and from files. This is to protect the system data. All things considered, the diagram shows how every system component interacts with every other to create a simple and organized way to purchase and control tickets.

OBJECT-ORIENTED IMPLEMENTATION

PYTHON CODE -

User & Customer Classes

```
#USER & CUSTOMER CLASSES

class User:

   """

   Base class for a system user (can be Customer or Admin).

   """

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

       # Private attributes for user info

       self.__user_id = user_id

       self.__name = name
```

```
        self.__email = email
        self.__password = password
    def login(self, password):
        # Check if the password matches
        return self.__password == password
    def logout(self):
        # Print logout message
        print(f"{self.__name} has logged out.")
    def __str__(self):
        # Return user as string
        return f"User: {self.__name} ({self.__email})"
    # Getter methods to access private attributes
    def get_user_id(self):
        return self.__user_id
    def get_name(self):
        return self.__name
    def get_email(self):
        return self.__email
    # Setter method to change name
    def set_name(self, name):
        if not name:
            raise ValueError("Name cannot be empty.")
        self.__name = name
    # Setter method to change email
    def set_email(self, email):
        if "@" not in email:
            raise ValueError("Invalid email address.")
        self.__email = emai
class Customer(User):
```

```python
"""
Represents a Customer.
Inherits from User and adds more features.
"""
def __init__(self, user_id, name, email, password, visit_count=0):
    # Call parent constructor
    super().__init__(user_id, name, email, password)
    # Customer-specific attributes
    self.__visit_count = visit_count
    self.__orders = []  # List to store past orders
def view_orders(self):
    # Show all orders
    if not self.__orders:
        print("No orders placed yet.")
    for order in self.__orders:
        print(order)
def update_profile(self, name, email):
    # Update name and email
    self.set_name(name)
    self.set_email(email)
    print("Profile updated.")
def add_order(self, order):
    # Add an order to history
    self.__orders.append(order)
def get_visit_count(self):
    # Return visit count
    return self.__visit_coun
def __str__(self):
    # Return customer as string
```

```
        return f"Customer: {self.get_name()} ({self.get_email()})"
```

Ticket and Ticket Inventory Classes

```
#Ticket and Ticket Inventory Classes

from enum import Enum  # Import Enum for ticket types

class TicketType(Enum):
   """
   Enum for different types of tickets.
   """
   SINGLERACE = 1
   WEEKENDPASS = 2
   SEASONMEMBERSHIP = 3
   GROUPDISCOUNT = 4

class Ticket:
   """
   A class representing a ticket with various attributes.
   """

   def __init__(self, ticket_id, ticket_type, price, validity,
race_date):
       """
       Initializes a Ticket instance.

       :param ticket_id: Unique identifier for the ticket
       :param ticket_type: Type of the ticket (from TicketType enum)
       :param price: Price of the ticket
       :param validity: Validity of the ticket
       :param race_date: Date of the race
       """
       self.ticket_id = ticket_id  # Store ticket ID
       self.ticket_type = ticket_type  # Store ticket type (enum)
       self.price = price  # Store price
       self.validity = validity  # Store validity
       self.race_date = race_date  # Store race date

   def __str__(self):
       """
       Return a string representation of the Ticket.
```

```
        """
        return f"Ticket ID: {self.ticket_id}, Type: 
{self.ticket_type.name}, Price: {self.price}, Date: {self.race_date}"

class TicketInventory:
    """
    A class representing the inventory of tickets available.
    """

    def __init__(self):
        self.ticket_list = []  # Initialize an empty list of tickets

    def add_ticket(self, ticket):
        """
        Add a ticket to the inventory.

        :param ticket: Ticket object to add
        """
        self.ticket_list.append(ticket)  # Add the ticket to the list

    def remove_ticket(self, ticket_id):
        """
        Remove a ticket from the inventory based on its ID.

        :param ticket_id: ID of the ticket to remove
        """
        # Filter out the ticket with the given ID
        self.ticket_list = [ticket for ticket in self.ticket_list if 
ticket.ticket_id != ticket_id]

    def update_availability(self):
        """
        Check the number of available tickets.

        :return: Number of tickets left in inventory
        """
        return len(self.ticket_list)  # Return the count of available 
tickets

    def __str__(self):
        """
        Return the status of the ticket inventory.
        """
```

```
       return f"Inventory: {len(self.ticket_list)} tickets available"
```

PurchaseOrder Class

```
#PurchaseOrder Class
from enum import Enum

class PaymentMethod(Enum):
   """
   Enum for different payment methods.
   """
   CREDITCARD = 1
   DEBITCARD = 2
   DIGITALWALLET = 3

class PurchaseOrder:
   """
   A class representing a customer's purchase order.
   """
   def _init_(self, order_id, order_date, total_amount,
payment_method):
       """
       Initializes a PurchaseOrder instance.

       :param order_id: Unique identifier for the order
       :param order_date: Date when the order was placed
       :param total_amount: Total cost of the order
       :param payment_method: Payment method used (from PaymentMethod
enum)
       """
       self.order_id = order_id
       self.order_date = order_date
       self.total_amount = total_amount
       self.payment_method = payment_method

   def calculate_total(self):
       """Calculate the total price (include discount logic if
needed)."""
       return self.total_amount

   def _str_(self):
       """Return a string representation of the PurchaseOrder."""
```

```
        return f"Order ID: {self.order_id}, Date: {self.order_date}, 
Total: {self.total_amount}, Payment: {self.payment_method.name}"
```

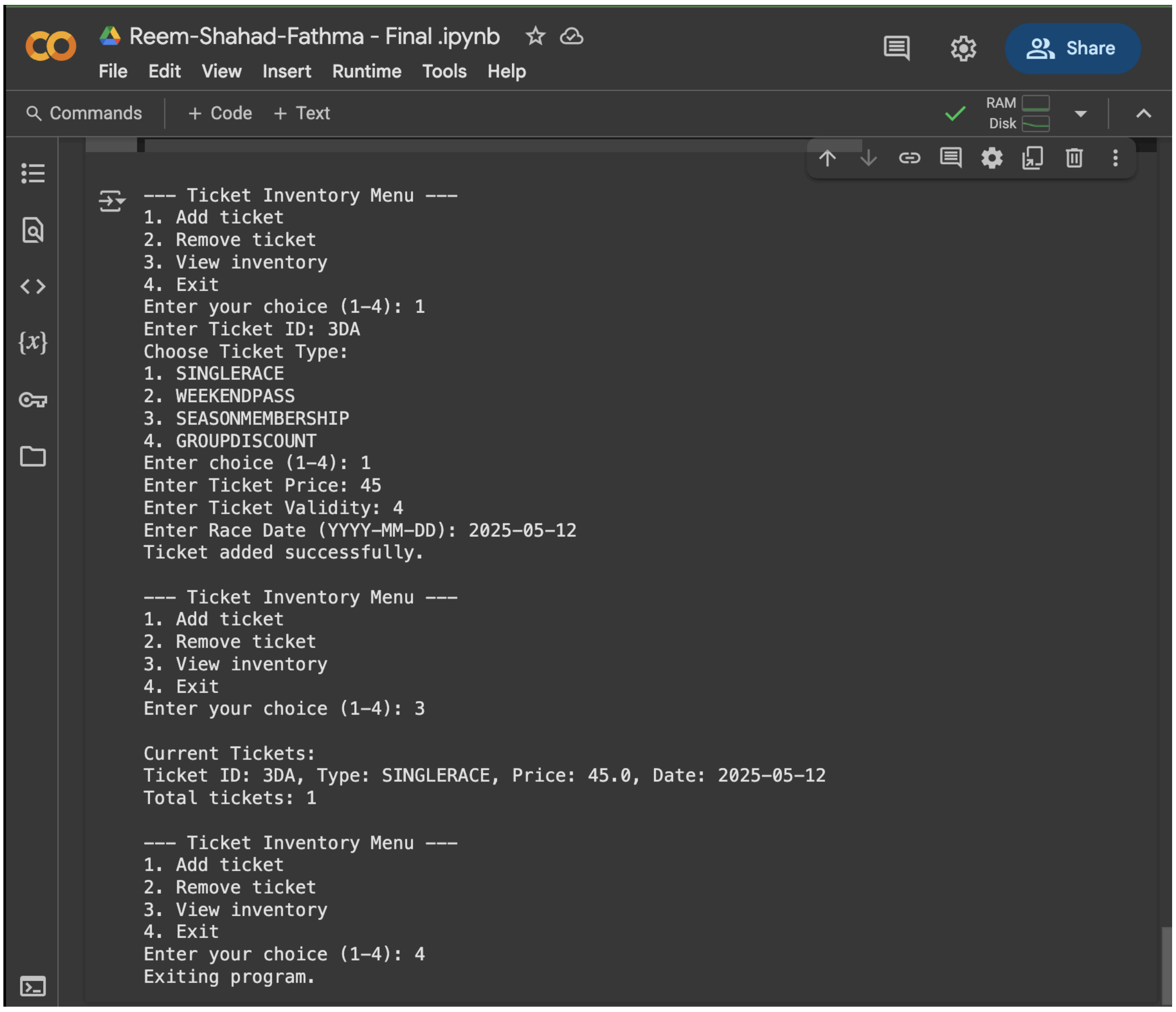
Pickle for Saving and Loading Data

```
#Pickle for Saving and Loading Data

import pickle

class FileHandler:
    """
    Class to handle saving and loading data using Pickle.
    """
    def save_data(self, data, filename):
        """Save data to a binary file using Pickle."""
        with open(filename, 'wb') as f:
            pickle.dump(data, f)

    def load_data(self, filename):
        """Load data from a Pickle file."""
        try:
            with open(filename, 'rb') as f:
                return pickle.load(f)
        except FileNotFoundError:
            return []  # Return an empty list if the file is not found
```

Testing The System

```

#Testing the System
from enum import Enum

class TicketType(Enum):
    SINGLERACE = 1
    WEEKENDPASS = 2
    SEASONMEMBERSHIP = 3
    GROUPDISCOUNT = 4

class Ticket:
    def __init__(self, ticket_id, ticket_type, price, validity, 
race_date):
        self.ticket_id = ticket_id
        self.ticket_type = ticket_type
        self.price = price
        self.validity = validity
        self.race_date = race_date

    def __str__(self):
        return f"Ticket ID: {self.ticket_id}, Type:
{self.ticket_type.name}, Price: {self.price}, Date: {self.race_date}"

class TicketInventory:
    def __init__(self):
        self.ticket_list = []

    def add_ticket(self, ticket):
        self.ticket_list.append(ticket)

    def remove_ticket(self, ticket_id):
        self.ticket_list = [ticket for ticket in self.ticket_list if
ticket.ticket_id != ticket_id]

    def update_availability(self):
        return len(self.ticket_list)

    def display_all_tickets(self):
        if not self.ticket_list:
            print("No tickets available.")
        else:
            for ticket in self.ticket_list:
                print(ticket)

# Helper to display menu
def show_menu():
    print("\n--- Ticket Inventory Menu ---")
    print("1. Add ticket")
    print("2. Remove ticket")
    print("3. View inventory")
    print("4. Exit")

# Main program
if __name__ == "__main__":
    inventory = TicketInventory()
    while True:
        show_menu()
        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            # Add a ticket
            ticket_id = input("Enter Ticket ID: ")
            print("Choose Ticket Type:")
            for t in TicketType:
                print(f"{t.value}. {t.name}")
            type_choice = int(input("Enter choice (1-4): "))
            ticket_type = TicketType(type_choice)

            price = float(input("Enter Ticket Price: "))
            validity = input("Enter Ticket Validity: ")
            race_date = input("Enter Race Date (YYYY-MM-DD): ")

            ticket = Ticket(ticket_id, ticket_type, price, validity,
race_date)
            inventory.add_ticket(ticket)
            print("Ticket added successfully.")

        elif choice == "2":
            # Remove a ticket
            ticket_id = input("Enter Ticket ID to remove: ")
            inventory.remove_ticket(ticket_id)
            print("Ticket removed (if it existed).")

        elif choice == "3":
            # View tickets
            print("\nCurrent Tickets:")
            inventory.display_all_tickets()
            print(f"Total tickets: {inventory.update_availability()}")

        elif choice == "4":
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please try again.")
```

Screenshot of Testing The System


Reem-Shahad-Fathma - Final .ipynb
File Edit View Insert Runtime Tools Help
Share
Commands
+ Code + Text
RAM
Disk

```
--- Ticket Inventory Menu ---
1. Add ticket
2. Remove ticket
3. View inventory
4. Exit
Enter your choice (1-4): 1
Enter Ticket ID: 3DA
Choose Ticket Type:
1. SINGLERACE
2. WEEKENDPASS
3. SEASONMEMBERSHIP
4. GROUPDISCOUNT
Enter choice (1-4): 1
Enter Ticket Price: 45
Enter Ticket Validity: 4
Enter Race Date (YYYY-MM-DD): 2025-05-12
Ticket added successfully.

--- Ticket Inventory Menu ---
1. Add ticket
2. Remove ticket
3. View inventory
4. Exit
Enter your choice (1-4): 3

Current Tickets:
Ticket ID: 3DA, Type: SINGLERACE, Price: 45.0, Date: 2025-05-12
Total tickets: 1

--- Ticket Inventory Menu ---
1. Add ticket
2. Remove ticket
3. View inventory
4. Exit
Enter your choice (1-4): 4
Exiting program.
```

Explanation:

In this part of the project, we turned our UML class diagram into working Python code using object-oriented programming (OOP). We created different classes like User, Ticket, and Order, each with their own properties and functions. This helped us organize the code better and make it easier to understand. We used inheritance so that common features (like login details) didn’t need to be repeated. For example, the Admin class inherited from the User class but had extra features like managing tickets. We also used encapsulation to keep some data private and only accessible through methods. This made our code more secure and professional. Overall, OOP helped us build a clean and reusable structure for our ticketing system.

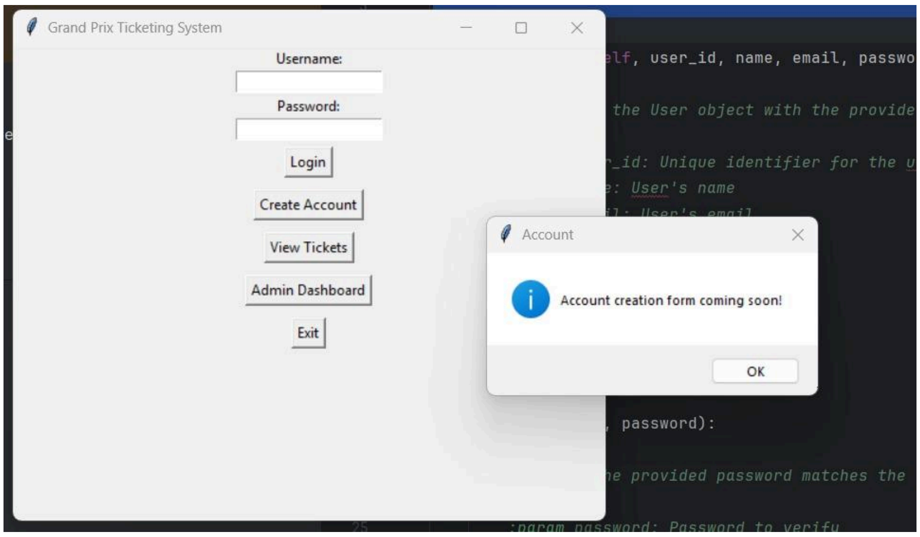
GUI WITH TKINTER

```
import tkinter as tk
from tkinter import messagebox

# Global user store
users = {
   "admin": "admin123",
   "user1": "pass1"
}

# Main app window
root = tk.Tk()
root.title("Grand Prix Ticketing System")
root.geometry("800x800")

# Functions


def login():
   username = username_entry.get()
   password = password_entry.get()

   if username in users and users[username] == password:
       messagebox.showinfo("Login Success", f"Welcome, {username}!")
   else:
       messagebox.showerror("Login Failed", "Invalid username or
password.")

def open_register_window():
   def register_user():
       new_user = entry_username.get()
       new_pass = entry_password.get()

       if new_user in users:
           messagebox.showerror("Error", "Username already exists.")
       elif not new_user or not new_pass:
           messagebox.showerror("Error", "Fields cannot be empty.")
       else:
           users[new_user] = new_pass
           messagebox.showinfo("Success", f"Account created for
{new_user}.")
           reg_window.destroy()
```

```
    reg_window = tk.Toplevel(root)
    reg_window.title("Create Account")
    reg_window.geometry("300x200")

    tk.Label(reg_window, text="New Username:").pack()
    entry_username = tk.Entry(reg_window)
    entry_username.pack()

    tk.Label(reg_window, text="New Password:").pack()
    entry_password = tk.Entry(reg_window, show="*")
    entry_password.pack()

    tk.Button(reg_window, text="Register",
command=register_user).pack(pady=10)

def open_tickets_window():
    ticket_window = tk.Toplevel(root)
    ticket_window.title("View Tickets")
    ticket_window.geometry("300x250")

    tk.Label(ticket_window, text="Available Tickets", font=("Arial",
14)).pack(pady=10)

    tickets = [
        "1. Single Race - 100 AED",
        "2. Weekend Pass - 250 AED",
        "3. Season Membership - 800 AED",
        "4. Group Discount - 20% off (min. 4 people)"
    ]

    for t in tickets:
        tk.Label(ticket_window, text=t).pack(anchor="w", padx=20)

def open_admin_dashboard():
    admin_window = tk.Toplevel(root)
    admin_window.title("Admin Dashboard")
    admin_window.geometry("300x200")

    tk.Label(admin_window, text="Admin Controls", font=("Arial",
14)).pack(pady=10)
    tk.Label(admin_window, text="- Track Sales").pack(anchor="w",
padx=20)
```

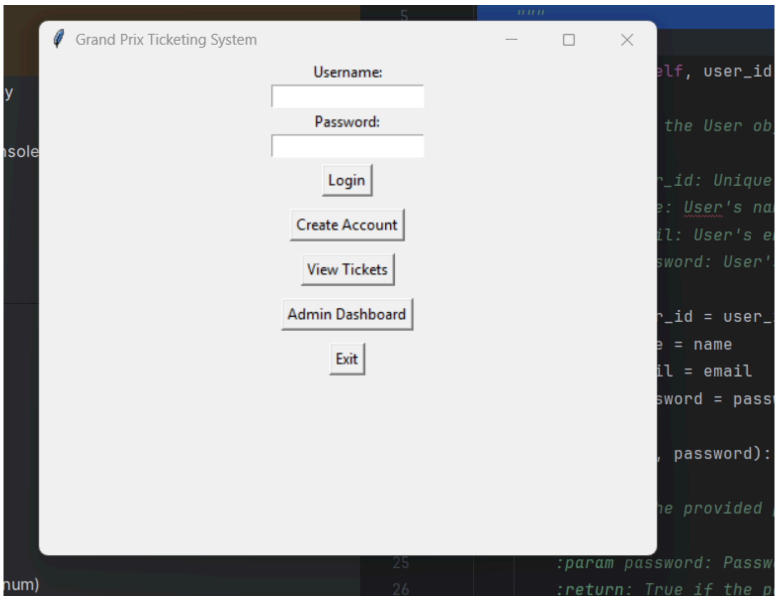
```
    tk.Label(admin_window, text="- Modify Discounts").pack(anchor="w",
padx=20)
    tk.Label(admin_window, text="- Manage Inventory").pack(anchor="w",
padx=20)

def exit_app():
    if messagebox.askokcancel("Exit", "Do you want to exit?"):
        root.destroy()

# UI Layout


tk.Label(root, text="Username:").pack()
username_entry = tk.Entry(root)
username_entry.pack()

tk.Label(root, text="Password:").pack()
password_entry = tk.Entry(root, show="*")
password_entry.pack()

tk.Button(root, text="Login", width=20, command=login).pack(pady=5)
tk.Button(root, text="Create Account", width=20,
command=open_register_window).pack(pady=5)
tk.Button(root, text="View Tickets", width=20,
command=open_tickets_window).pack(pady=5)
tk.Button(root, text="Admin Dashboard", width=20,
command=open_admin_dashboard).pack(pady=5)
tk.Button(root, text="Exit", width=20, command=exit_app).pack(pady=10)

root.mainloop()
```

Screenshot of Testing

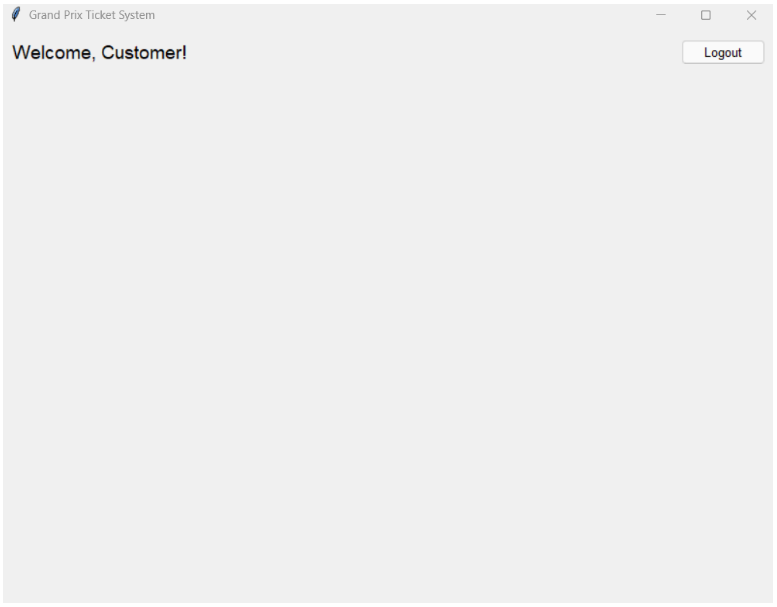


Account creation form coming soon!

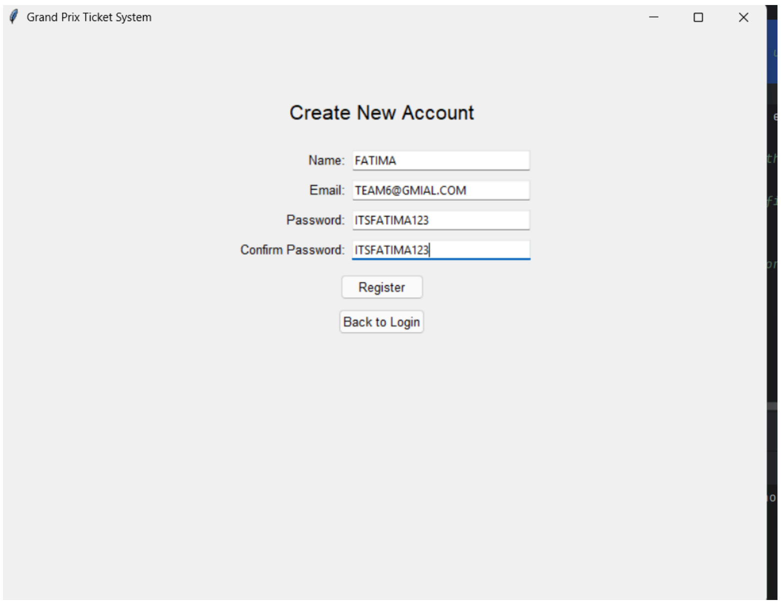
```
elf, user_id, name, email, passwo
 the User object with the provide
_id: Unique identifier for the u
: User's name
l: User's email
, password):
e provided password matches the
:param password: Password to verify
```



```
elf, user_id
 the User ob
_id: Unique
: User's na
il: User's e
sword: User'
_id = user_
e = name
il = email
sword = pass
, password):
e provided
:param password: Passw
:return: True if the p
```



Welcome, Customer!



Name: FATIMA
Email: TEAM6@GMIAL.COM
Password: ITSFATIMA123
Confirm Password: ITSFATIMA123

Explanation:

We used the tkinter library in Python to build a simple and user-friendly graphical interface. This allowed users to interact with the system using buttons and forms instead of typing commands. The GUI had a main screen with options to log in, create an account, view tickets, and open the admin dashboard. Each button opened a new window where users could do tasks like register or browse ticket types. We also added pop-up messages to show success or error messages, which made the system more helpful and clear. By using tkinter, we made our system easy to use for everyone, even those who don’t know programming.

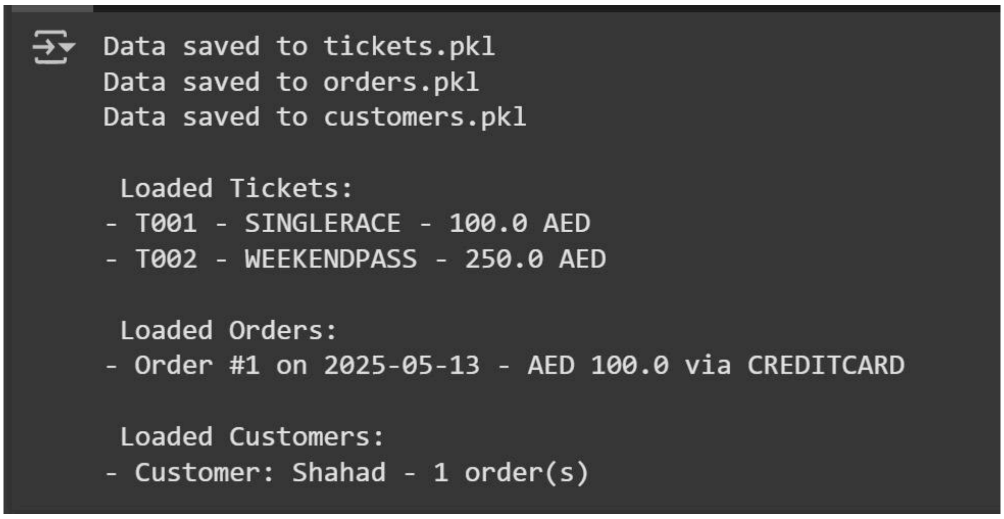
DATA STORAGE WITH PICKLE

```
import pickle
import os
from enum import Enum

#FileHandler Class
class FileHandler:
   """
   Handles saving and loading data using Python's pickle module.
   """
   @staticmethod
```

```python
    def save_data(data, filename):
        try:
            with open(filename, 'wb') as f:
                pickle.dump(data, f)
            print(f"Data saved to {filename}")
        except Exception as e:
            print(f"[Error] Could not save {filename}: {e}")

    @staticmethod
    def load_data(filename):
        if not os.path.exists(filename):
            print(f"[Info] {filename} not found. Returning empty data.")
            return {} if "user" in filename else []
        try:
            with open(filename, 'rb') as f:
                return pickle.load(f)
        except EOFError:
            print(f"[Info] {filename} is empty. Returning empty data.")
            return {} if "user" in filename else []
        except Exception as e:
            print(f"[Error] Could not load {filename}: {e}")
            return {} if "user" in filename else []

#Project Classes
class TicketType(Enum):
    SINGLERACE = 1
    WEEKENDPASS = 2
    SEASONMEMBERSHIP = 3
    GROUPDISCOUNT = 4

class Ticket:
    def __init__(self, ticket_id, ticket_type, price, validity,
race_date):
        self.ticket_id = ticket_id
        self.ticket_type = ticket_type
        self.price = price
        self.validity = validity
        self.race_date = race_date

    def __str__(self):
        return f"{self.ticket_id} - {self.ticket_type.name} -
{self.price} AED"
```

```python
class PaymentMethod(Enum):
    CREDITCARD = 1
    DEBITCARD = 2
    DIGITALWALLET = 3

class PurchaseOrder:
    def __init__(self, order_id, order_date, total_amount,
payment_method):
        self.order_id = order_id
        self.order_date = order_date
        self.total_amount = total_amount
        self.payment_method = payment_method

    def calculate_total(self):
        return self.total_amount

    def __str__(self):
        return f"Order #{self.order_id} on {self.order_date} - AED
{self.total_amount} via {self.payment_method.name}"

class User:
    def __init__(self, user_id, name, email, password):
        self.__user_id = user_id
        self.__name = name
        self.__email = email
        self.__password = password

    def get_name(self):
        return self.__name

    def get_email(self):
        return self.__email

    def __str__(self):
        return f"{self.__name} ({self.__email})"

class Customer(User):
    def __init__(self, user_id, name, email, password, visit_count=0):
        super().__init__(user_id, name, email, password)
        self.__visit_count = visit_count
        self.__orders = []

    def add_order(self, order):
        self.__orders.append(order)

    def __str__(self):
        return f"Customer: {self.get_name()} - {len(self.__orders)}
 order(s)"

 #Testing Pickle Saving & Loading

 # Create ticket objects
 ticket1 = Ticket("T001", TicketType.SINGLERACE, 100.0, "1 day",
 "2025-12-01")
 ticket2 = Ticket("T002", TicketType.WEEKENDPASS, 250.0, "3 days",
 "2025-12-05")

 # Create a purchase order
 order1 = PurchaseOrder(1, "2025-05-13", 100.0,
 PaymentMethod.CREDITCARD)

 # Create a customer and add the order
 customer = Customer(1, "Shahad", "shahad@example.com", "1234")
 customer.add_order(order1)

 # Save data using FileHandler
 FileHandler.save_data([ticket1, ticket2], "tickets.pkl")
 FileHandler.save_data([order1], "orders.pkl")
 FileHandler.save_data([customer], "customers.pkl")

 # Load and print to confirm persistence
 loaded_tickets = FileHandler.load_data("tickets.pkl")
 loaded_orders = FileHandler.load_data("orders.pkl")
 loaded_customers = FileHandler.load_data("customers.pkl")

 # Display loaded results
 print("\n Loaded Tickets:")
 for t in loaded_tickets:
    print("-", t)

 print("\n Loaded Orders:")
 for o in loaded_orders:
    print("-", o)

 print("\n Loaded Customers:")
 for c in loaded_customers:
    print("-", c)
```

Screenshot of Testing



```
Data saved to tickets.pkl
Data saved to orders.pkl
Data saved to customers.pkl

 Loaded Tickets:
- T001 - SINGLERACE - 100.0 AED
- T002 - WEEKENDPASS - 250.0 AED

 Loaded Orders:
- Order #1 on 2025-05-13 - AED 100.0 via CREDITCARD

 Loaded Customers:
- Customer: Shahad - 1 order(s)
```

Explanation:

To save user accounts, tickets, and orders, we used the pickle module to store data in binary files like users.pkl and orders.pkl. When the program starts, it loads data from these files so users can continue where they left off. When users register or buy tickets, the data is saved again to update the files. We also created helper functions to load and save data safely, and we handled errors in case files are missing or empty. This way, our system can remember data between sessions. Using pickle made it easier to store and retrieve complex data like user objects without needing a database.

REPORT & DOCUMENTATION

Summary

In this course, we learned the fundamentals of object-oriented programming (OOP) and software modelling, focusing on designing and implementing real-world systems using Python. Key concepts included encapsulation, inheritance, and polymorphism, which helped us build reusable and organized code. We also explored Unified Modeling Language (UML) diagrams to plan system structure before development. Through the Grand Prix Ticketing System assignment, we applied these principles by designing classes for users, tickets, and orders, and building a graphical user interface (GUI) using tkinter for user interaction. A major focus was on data persistence, where we used the pickle module to store and retrieve user and ticket data in binary files such as users.pkl, tickets.pkl, and orders.pkl. We handled common file exceptions to ensure the application remained stable and demonstrated that the data persisted across multiple sessions. This assignment allowed us to integrate our knowledge of software design, GUI development, and file handling to create a functional, user-friendly ticket booking system.

Group Roles

Shahad's Contribution

Shahad was responsible for planning and designing the UML class diagram, which served as the blueprint for how our ticketing system would be structured. She used object-oriented concepts to map out the relationships between the main components, like Users, Tickets, Orders, and Admin. This helped the team understand how the classes would work together before starting the coding process. Shahad also worked on the Pickle data storage testing. She created test files for storing user and ticket data in binary format using Python's pickle module. She wrote and ran small scripts to test if data could be successfully saved and loaded across different program runs. This helped ensure that our system could support persistent storage, a key requirement of the project.

Fathma's Contribution

Fathma focused mainly on the object-oriented implementation (OOP) of the project. She was in charge of turning the UML diagram into actual Python classes, like the User, Ticket, and

Order classes. She made sure that the principles of encapsulation, inheritance, and polymorphism were applied correctly throughout the code. She also created example objects and tested their behavior to make sure they were working as expected. In addition, Fathma took the lead in GUI testing. After Reem completed the GUI interface, Fathma tested the login, registration, and ticket viewing windows. She clicked through the buttons, entered different types of data (including invalid input), and reported any bugs or errors to the group for fixing. Her testing made sure the GUI was user-friendly and bug-free.

Reem’s Contribution

Reem played a major role in the GUI development using tkinter. She created the visual parts of the program, including the main login window, registration form, ticket display screen, and admin dashboard. She made sure the layout was organized and that each button and label was properly connected to a function. Reem also worked on integrating the pickle file storage into the system. She created helper functions to save and load user data to and from users.pkl and tested these with Shahad. She added exception handling to make sure the app didn’t crash if the file was missing or empty. Reem also helped write the project explanations and final report, especially for the GUI and data storage parts. Her documentation explained how each part of the code worked in simple language, making it easier for the reader to understand.

GitHub

https://github.com/reem-alhamami/Assignment-3