Assignment 3 - Final Project

Abdullah M. Awad Aljeaidi - 202112794

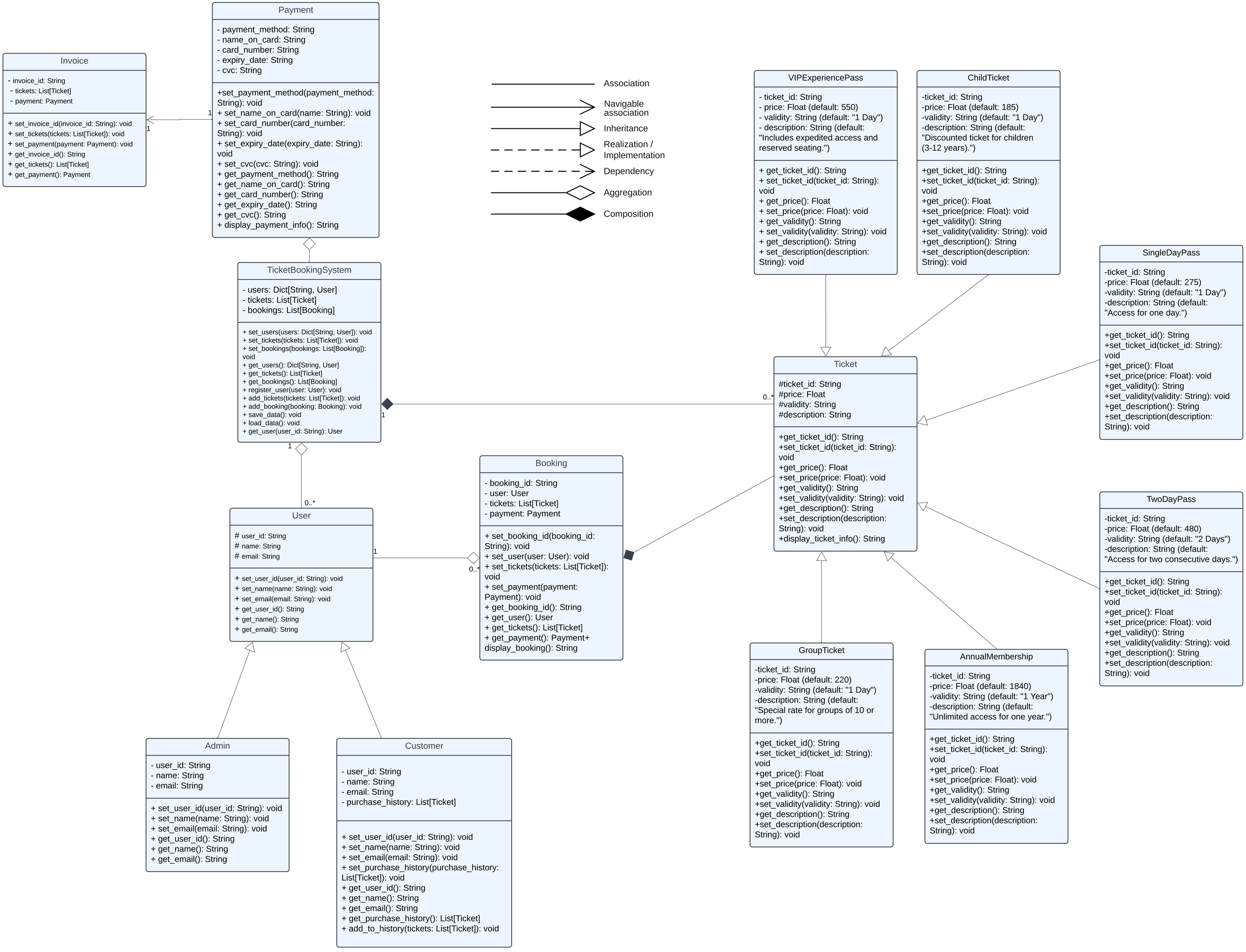
Saif Abdulla Alhammadi - 202220746

College of Interdisciplinary Studies

ICS220: Programming Fundamentals

Prof. Afshan Parkar

December 5, 2024


Payment
- payment_method: String
- name_on_card: String
- card_number: String
- expiry_date: String
- cvc: String
+set_payment_method(payment_method: String): void
+ set_name_on_card(name: String): void
+ set_card_number(card_number: String): void
+ set_expiry_date(expiry_date: String): void
+ set_cvc(cvc: String): void
+ get_payment_method(): String
+ get_name_on_card(): String
+ get_card_number(): String
+ get_expiry_date(): String
+ get_cvc(): String
+ display_payment_info(): String
Invoice
- invoice_id: String
- tickets: List[Ticket]
- payment: Payment
+ set_invoice_id(invoice_id: String): void
+ set_tickets(tickets: List[Ticket]): void
+ set_payment(payment: Payment): void
+ get_invoice_id(): String
+ get_tickets(): List[Ticket]
+ get_payment(): Payment
1
1
Association
Navigable association
Inheritance
Realization / Implementation
Dependency
Aggregation
Composition
VIPExperiencePass
- ticket_id: String
- price: Float (default: 550)
- validity: String (default: "1 Day")
- description: String (default: "Includes expedited access and reserved seating.")
+ get_ticket_id(): String
+ set_ticket_id(ticket_id: String): void
+ get_price(): Float
+ set_price(price: Float): void
+ get_validity(): String
+ set_validity(validity: String): void
+ get_description(): String
+ set_description(description: String): void
ChildTicket
-ticket_id: String
-price: Float (default: 185)
-validity: String (default: "1 Day")
-description: String (default: "Discounted ticket for children (3-12 years).")
+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void
SingleDayPass
-ticket_id: String
-price: Float (default: 275)
-validity: String (default: "1 Day")
-description: String (default: "Access for one day.")
+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void
TicketBookingSystem
- users: Dict[String, User]
- tickets: List[Ticket]
- bookings: List[Booking]
+ set_users(users: Dict[String, User]): void
+ set_tickets(tickets: List[Ticket]): void
+ set_bookings(bookings: List[Booking]): void
+ get_users(): Dict[String, User]
+ get_tickets(): List[Ticket]
+ get_bookings(): List[Booking]
+ register_user(user: User): void
+ add_tickets(tickets: List[Ticket]): void
+ add_booking(booking: Booking): void
+ save_data(): void
+ load_data(): void
+ get_user(user_id: String): User
1
0..*
1
1
0..*
Ticket
#ticket_id: String
#price: Float
#validity: String
#description: String
+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void
+display_ticket_info(): String
Booking
- booking_id: String
- user: User
- tickets: List[Ticket]
- payment: Payment
+ set_booking_id(booking_id: String): void
+ set_user(user: User): void
+ set_tickets(tickets: List[Ticket]): void
+ set_payment(payment: Payment): void
+ get_booking_id(): String
+ get_user(): User
+ get_tickets(): List[Ticket]
+ get_payment(): Payment+ display_booking(): String
User
user_id: String
name: String
email: String
+ set_user_id(user_id: String): void
+ set_name(name: String): void
+ set_email(email: String): void
+ get_user_id(): String
+ get_name(): String
+ get_email(): String
TwoDayPass
-ticket_id: String
-price: Float (default: 480)
-validity: String (default: "2 Days")
-description: String (default: "Access for two consecutive days.")
+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void
GroupTicket
-ticket_id: String
-price: Float (default: 220)
-validity: String (default: "1 Day")
-description: String (default: "Special rate for groups of 10 or more.")
+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void
AnnualMembership
-ticket_id: String
-price: Float (default: 1840)
-validity: String (default: "1 Year")
-description: String (default: "Unlimited access for one year.")
+get_ticket_id(): String
+set_ticket_id(ticket_id: String): void
+get_price(): Float
+set_price(price: Float): void
+get_validity(): String
+set_validity(validity: String): void
+get_description(): String
+set_description(description: String): void
Admin
- user_id: String
- name: String
- email: String
+ set_user_id(user_id: String): void
+ set_name(name: String): void
+ set_email(email: String): void
+ get_user_id(): String
+ get_name(): String
+ get_email(): String
Customer
- user_id: String
- name: String
- email: String
- purchase_history: List[Ticket]
+ set_user_id(user_id: String): void
+ set_name(name: String): void
+ set_email(email: String): void
+ set_purchase_history(purchase_history: List[Ticket]): void
+ get_user_id(): String
+ get_name(): String
+ get_email(): String
+ get_purchase_history(): List[Ticket]
+ add_to_history(tickets: List[Ticket]): void

Class 1	Class 2	Association Type	Cardinality	Relationship Description
Payment	Invoice	Unary Association	1 (Payment) : 1 (Invoice)	Each Payment is directly linked to one Invoice.
TicketBookingSystem	Payment	Aggregation	1 (TicketBookingSystem) : * (Payment)	The TicketBookingSystem manages multiple Payments for ticket purchases.
User	TicketBookingSystem	Aggregation	1 (User) : 1 (TicketBookingSystem)	A User is associated with a TicketBookingSystem, which enables ticket management.
Admin	User	Inheritance	1 (Admin) : 1 (User)	Admin inherits from the User class, with added administrative attributes and methods.
Customer	User	Inheritance	1 (Customer) : 1 (Admin)	Customer inherits from the User class, with attributes specific to ticket purchasing.
User	Booking	Aggregation	1 (User) : * (Booking)	A User can have multiple Bookings in their purchase history.
Ticket	Booking	Composition	1 (Booking) : * (Ticket)	A Booking includes multiple Tickets that are tied to the booking process.
Ticket	TicketBookingSystem	Composition	1 (TicketBookingSystem) : * (Ticket)	The TicketBookingSystem directly manages a collection of Tickets.
Ticket	SingleDayPass	Inheritance	1 (SingleDayPass) : 1 (Ticket)	SingleDayPass inherits from the Ticket class.
Ticket	TwoDayPass	Inheritance	1 (Ticket) : 1 (TwoDayPass)	TwoDayPass inherits from the Ticket class.
Ticket	AnnualMembership	Inheritance	1 (Ticket) : 1 (AnnualMembership)	AnnualMembership inherits from the Ticket class.
Ticket	ChildTicket	Inheritance	1 (Ticket) : 1 (ChildTicket)	ChildTicket inherits from the Ticket class.
Ticket	GroupTicket	Inheritance	1 (Ticket) : 1 (GroupTicket)	GroupTicket inherits from the Ticket class.
Ticket	VIPExperiencePass	Inheritance	1 (Ticket) : 1 (VIPExperiencePass)	VIPExperiencePass inherits from the Ticket class.

ticket_system.py:

```
import pickle

# Base Ticket class
class Ticket:
    def __init__(self, ticket_id, price, validity, description):
        # Initialize ticket with ID, price, validity, and description
        self._ticket_id = ticket_id
        self._price = price
        self._validity = validity
        self._description = description

    # Getter methods
    def get_ticket_id(self):
        return self._ticket_id

    def get_price(self):
        return self._price

    def get_validity(self):
        return self._validity

    def get_description(self):
        return self._description

    # Setter methods
    def set_ticket_id(self, ticket_id):
        self._ticket_id = ticket_id

    def set_price(self, price):
        self._price = price

    def set_validity(self, validity):
        self._validity = validity

    def set_description(self, description):
```

```
        self._description = description

    # Display ticket details
    def display_ticket_info(self):
        return f"ID: {self._ticket_id}, Price: {self._price} DHS, Validity:
{self._validity}, Description: {self._description}"

# Subclasses for specific ticket types
class SingleDayPass(Ticket):
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 275, "1 Day", "Access for one day.")

class TwoDayPass(Ticket):
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 480, "2 Days", "Access for two
consecutive days.")

class AnnualMembership(Ticket):
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 1840, "1 Year", "Unlimited access for
one year.")

class ChildTicket(Ticket):
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 185, "1 Day", "Discounted ticket for
children (3-12 years).")

class GroupTicket(Ticket):
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 220, "1 Day", "Special rate for groups
of 10 or more.")

class VIPExperiencePass(Ticket):
    def __init__(self, ticket_id):
        super().__init__(ticket_id, 550, "1 Day", "Includes expedited
access and reserved seating.")
```

```python
# Payment class for managing payment details
class Payment:
   def __init__(self, payment_method, name_on_card, card_number,
expiry_date, cvc):
       # Initialize payment attributes
       self._payment_method = payment_method
       self._name_on_card = name_on_card
       self._card_number = card_number
       self._expiry_date = expiry_date
       self._cvc = cvc

   # Getter methods
   def get_payment_method(self):
       return self._payment_method

   def get_name_on_card(self):
       return self._name_on_card

   def get_card_number(self):
       return self._card_number

   def get_expiry_date(self):
       return self._expiry_date

   def get_cvc(self):
       return self._cvc

   # Setter methods
   def set_payment_method(self, payment_method):
       self._payment_method = payment_method

   def set_name_on_card(self, name_on_card):
       self._name_on_card = name_on_card

   def set_card_number(self, card_number):
```

```
        self._card_number = card_number

    def set_expiry_date(self, expiry_date):
        self._expiry_date = expiry_date

    def set_cvc(self, cvc):
        self._cvc = cvc

    # Display payment details (excluding sensitive information)
    def display_payment_info(self):
        return f"Payment Method: {self._payment_method}, Name on Card:
{self._name_on_card}"

# Invoice class for generating invoices
class Invoice:
    def __init__(self, invoice_id, tickets, payment):
        # Initialize invoice with ID, tickets, and payment details
        self._invoice_id = invoice_id
        self._tickets = tickets
        self._payment = payment

    # Getter methods
    def get_invoice_id(self):
        return self._invoice_id

    def get_tickets(self):
        return self._tickets

    def get_payment(self):
        return self._payment

    # Setter methods
    def set_invoice_id(self, invoice_id):
        self._invoice_id = invoice_id

    def set_tickets(self, tickets):
```

```
        self._tickets = tickets

    def set_payment(self, payment):
        self._payment = payment

    # Display invoice details
    def display_invoice(self):
        tickets_info = "\n".join([ticket.display_ticket_info() for ticket
in self._tickets])
        payment_info = self._payment.display_payment_info()
        return f"Invoice ID:
{self._invoice_id}\nTickets:\n{tickets_info}\n{payment_info}"

# Booking class for managing user bookings
class Booking:
    def __init__(self, booking_id, user, tickets, payment=None):
        # Initialize booking with ID, user, tickets, and optional payment
        self._booking_id = booking_id
        self._user = user
        self._tickets = tickets
        self._payment = payment

    # Getter methods
    def get_booking_id(self):
        return self._booking_id

    def get_user(self):
        return self._user

    def get_tickets(self):
        return self._tickets

    def get_payment(self):
        return self._payment

    # Setter methods
```

```
    def set_booking_id(self, booking_id):
        self._booking_id = booking_id

    def set_user(self, user):
        self._user = user

    def set_tickets(self, tickets):
        self._tickets = tickets

    def set_payment(self, payment):
        self._payment = payment

    # Display booking details
    def display_booking(self):
        tickets_info = "\n".join([ticket.display_ticket_info() for ticket
in self._tickets])
        payment_info = self._payment.display_payment_info() if
self._payment else "No payment details available."
        return f"Booking ID: {self._booking_id}\nUser:
{self._user.get_name()}\nTickets:\n{tickets_info}\nPayment:\n{payment_info
}"

# Base User class
class User:
    def __init__(self, user_id, name, email):
        # Initialize user with ID, name, and email
        self._user_id = user_id
        self._name = name
        self._email = email

    # Getter methods
    def get_user_id(self):
        return self._user_id

    def get_name(self):
        return self._name
```

```python
    def get_email(self):
        return self._email

    # Setter methods
    def set_user_id(self, user_id):
        self._user_id = user_id

    def set_name(self, name):
        self._name = name

    def set_email(self, email):
        self._email = email

# Admin subclass for system administrators
class Admin(User):
    def __init__(self, user_id, name, email):
        super().__init__(user_id, name, email)
        # Additional admin-specific methods can be added here

# Customer subclass for ticket purchasers
class Customer(User):
    def __init__(self, user_id, name, email):
        super().__init__(user_id, name, email)
        self._purchase_history = []

    # Getter methods
    def get_purchase_history(self):
        return self._purchase_history

    # Setter methods
    def set_purchase_history(self, purchase_history):
        self._purchase_history = purchase_history

    # Add tickets to purchase history
    def add_to_history(self, tickets):
```

```
        self._purchase_history.extend(tickets)

# TicketBookingSystem class for managing the entire system
class TicketBookingSystem:
    def __init__(self):
        # Initialize system with users, tickets, and bookings
        self._users = {}
        self._tickets = []
        self._bookings = []
        self.load_data()  # Load system data from file

    # Getter methods
    def get_users(self):
        return self._users

    def get_tickets(self):
        return self._tickets

    def get_bookings(self):
        return self._bookings

    # Setter methods
    def set_users(self, users):
        self._users = users

    def set_tickets(self, tickets):
        self._tickets = tickets

    def set_bookings(self, bookings):
        self._bookings = bookings

    # Register a new user
    def register_user(self, user):
        self._users[user.get_user_id()] = user
        self.save_data()
```

```python
    # Add tickets to the system
    def add_tickets(self, tickets):
        self._tickets.extend(tickets)
        self.save_data()

    # Add a booking to the system
    def add_booking(self, booking):
        self._bookings.append(booking)
        self.save_data()

    # Save system data to file
    def save_data(self):
        with open("data.pkl", "wb") as file:
            pickle.dump({"users": self._users, "tickets": self._tickets,
"bookings": self._bookings}, file)

    # Load system data from file
    def load_data(self):
        try:
            with open("data.pkl", "rb") as file:
                data = pickle.load(file)
                self._users = data.get("users", {})
                self._tickets = data.get("tickets", [])
                self._bookings = data.get("bookings", [])
        except FileNotFoundError:
            self._users = {}
            self._tickets = []
            self._bookings = []

    # Get a user by ID
    def get_user(self, user_id):
        return self._users.get(user_id)
```

gui.py:

```
import tkinter as tk
from tkinter import ttk, messagebox
from ticket_system import TicketBookingSystem, Customer, SingleDayPass,
TwoDayPass, AnnualMembership, ChildTicket, GroupTicket, VIPExperiencePass,
Payment, Booking

class TicketBookingGUI:
    def __init__(self, master):
        # Initialize the GUI and Ticket Booking System
        self.system = TicketBookingSystem()
        self.master = master
        self.master.title("Adventure Land Ticket Booking")
        self.frame = tk.Frame(self.master)
        self.frame.pack()

        self.user = None  # Current logged-in user
        self.selected_tickets = []  # Store tickets selected for purchase

        self.login_screen()  # Start with the login screen

    def login_screen(self):
        # Login or Registration screen
        self.clear_frame()

        tk.Label(self.frame, text="Adventure Land Ticket
Booking").pack(pady=10)
        tk.Label(self.frame, text="Name:").pack()
        self.name_entry = tk.Entry(self.frame)  # Entry for name
        self.name_entry.pack()
        tk.Label(self.frame, text="Email:").pack()
        self.email_entry = tk.Entry(self.frame)  # Entry for email
        self.email_entry.pack()
        tk.Button(self.frame, text="Register/Login",
command=self.register_or_login).pack(pady=10)
```

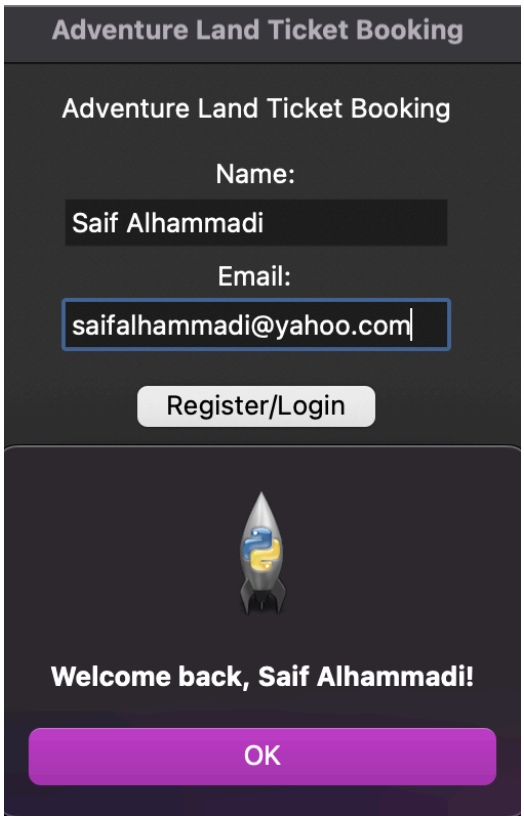
```
    def register_or_login(self):
        # Handle user registration or login
        name = self.name_entry.get()
        email = self.email_entry.get()

        if not name or not email:
            # Show error if fields are empty
            messagebox.showerror("Error", "Both fields are required!")
            return

        user_id = f"{name.lower().replace(' ', '')}_{email.split('@')[0]}"
        user = self.system.get_user(user_id)

        if user:
            # User exists, log in
            self.user = user
            messagebox.showinfo("Welcome Back", f"Welcome back,
{self.user.get_name()}!")
        else:
            # Register new user
            self.user = Customer(user_id, name, email)
            self.system.register_user(self.user)
            messagebox.showinfo("Account Created", f"Account created for
{self.user.get_name()}!")

        self.ticket_selection_screen()

    def ticket_selection_screen(self):
        # Screen for selecting tickets
        self.clear_frame()

        tk.Label(self.frame, text="Select Tickets").pack(pady=10)

        self.ticket_quantities = {}  # Dictionary to store ticket type and
quantities
```

```
        for ticket_class in [SingleDayPass, TwoDayPass, AnnualMembership,
ChildTicket, GroupTicket, VIPExperiencePass]:
            ticket = ticket_class("temp_id")  # Create a temporary ticket
to display info
            quantity_var = tk.IntVar(value=0)
            self.ticket_quantities[ticket_class] = quantity_var
            tk.Label(self.frame, text=f"{ticket.get_description()} -
{ticket.get_price()} DHS").pack(anchor="w")
            tk.Spinbox(self.frame, from_=0, to=10,
textvariable=quantity_var, width=5).pack(anchor="w")

        tk.Button(self.frame, text="Proceed to Payment",
command=self.payment_page).pack(pady=10)

    def payment_page(self):
        # Screen for entering payment details
        self.selected_tickets = []

        # Collect selected tickets
        for ticket_class, quantity_var in self.ticket_quantities.items():
            quantity = quantity_var.get()
            for _ in range(quantity):
                ticket_id = f"T{len(self.system.get_tickets()) +
len(self.selected_tickets) + 1}"
                ticket = ticket_class(ticket_id)
                self.selected_tickets.append(ticket)

        if not self.selected_tickets:
            # Show error if no tickets selected
            messagebox.showerror("Error", "Please select at least one
ticket!")
            return

        self.clear_frame()
        tk.Label(self.frame, text="Payment Page").pack(pady=10)
```

```
        # Payment form fields
        tk.Label(self.frame, text="Payment Method:").pack()
        self.payment_method = ttk.Combobox(self.frame, values=["Credit
Card", "Debit Card", "PayPal", "Apple Pay"])
        self.payment_method.pack()
        tk.Label(self.frame, text="Name on Card:").pack()
        self.name_on_card_entry = tk.Entry(self.frame)
        self.name_on_card_entry.pack()
        tk.Label(self.frame, text="Card Number:").pack()
        self.card_number_entry = tk.Entry(self.frame)
        self.card_number_entry.pack()
        tk.Label(self.frame, text="Expiry Date:").pack()
        self.expiry_date_entry = tk.Entry(self.frame)
        self.expiry_date_entry.pack()
        tk.Label(self.frame, text="CVC:").pack()
        self.cvc_entry = tk.Entry(self.frame)
        self.cvc_entry.pack()
        tk.Button(self.frame, text="Submit Payment",
command=self.process_payment).pack(pady=10)

    def process_payment(self):
        # Process payment and complete booking
        payment_method = self.payment_method.get()
        name_on_card = self.name_on_card_entry.get()
        card_number = self.card_number_entry.get()
        expiry_date = self.expiry_date_entry.get()
        cvc = self.cvc_entry.get()

        if not all([payment_method, name_on_card, card_number, expiry_date,
cvc]):
            # Show error if any field is empty
            messagebox.showerror("Error", "All payment fields are
required!")
            return

        # Create payment object
```

```
        payment = Payment(payment_method, name_on_card, card_number,
expiry_date, cvc)
        booking_id = f"B{len(self.system.get_bookings()) + 1}"
        booking = Booking(booking_id, self.user, self.selected_tickets,
payment)

        # Save booking and tickets
        self.user.add_to_history(self.selected_tickets)
        self.system.add_tickets(self.selected_tickets)
        self.system.add_booking(booking)

        # Display invoice
        self.invoice_screen(booking)

    def invoice_screen(self, booking):
        # Screen to display the invoice
        self.clear_frame()

        tk.Label(self.frame, text="Invoice").pack(pady=10)
        tk.Label(self.frame,
text=booking.display_booking()).pack(anchor="w")
        tk.Button(self.frame, text="Logout",
command=self.login_screen).pack(pady=10)

    def clear_frame(self):
        # Clear all widgets from the current frame
        for widget in self.frame.winfo_children():
            widget.destroy()

# Main entry point for the GUI application
if __name__ == "__main__":
    root = tk.Tk()  # Create the main application window
    app = TicketBookingGUI(root)  # Initialize the GUI
    root.mainloop()  # Start the Tkinter event loop
```

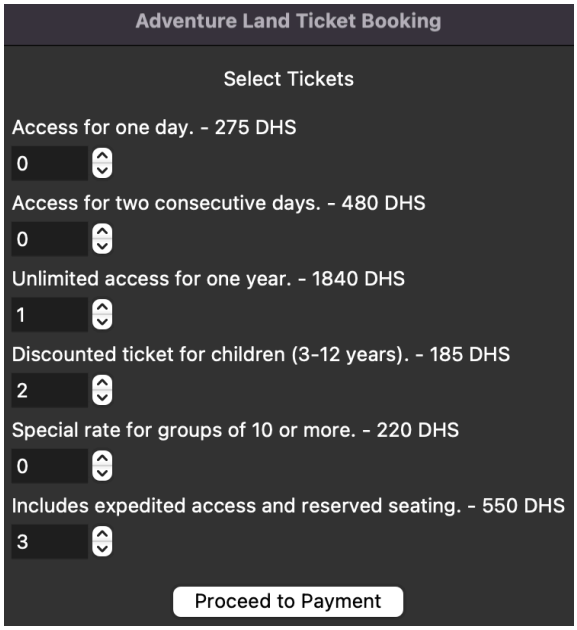
GUI Implementation:



Explanation Here:

Login/Registration Screen:

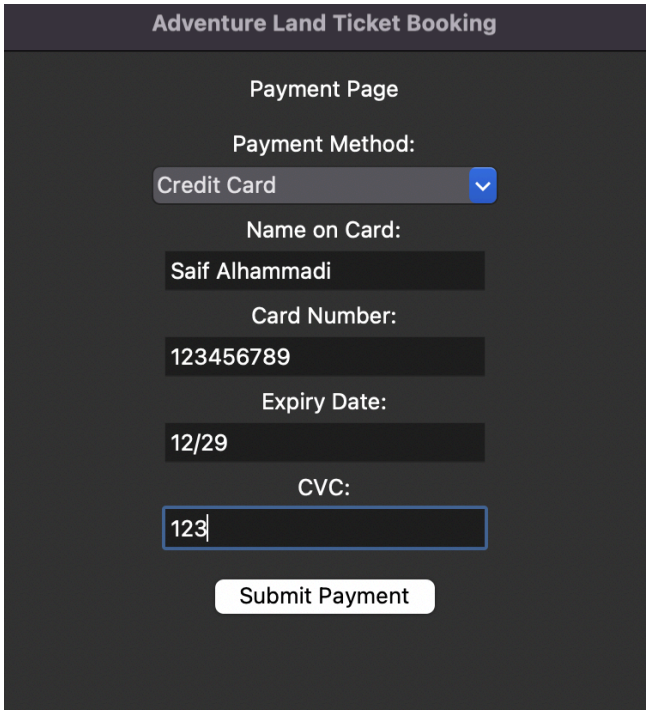
- This is the initial screen where the user enters their name and email.
- If the user exists in the system, they are logged in with a welcome message.
- If not, they are registered and greeted with an account creation confirmation.



Explanation here:

Ticket Selection Screen:

- Here, users can select the type and quantity of tickets they wish to purchase.
- Each ticket type is displayed with its price and a spinbox to choose the desired quantity.
- Once the selection is complete, the user proceeds to the payment page.

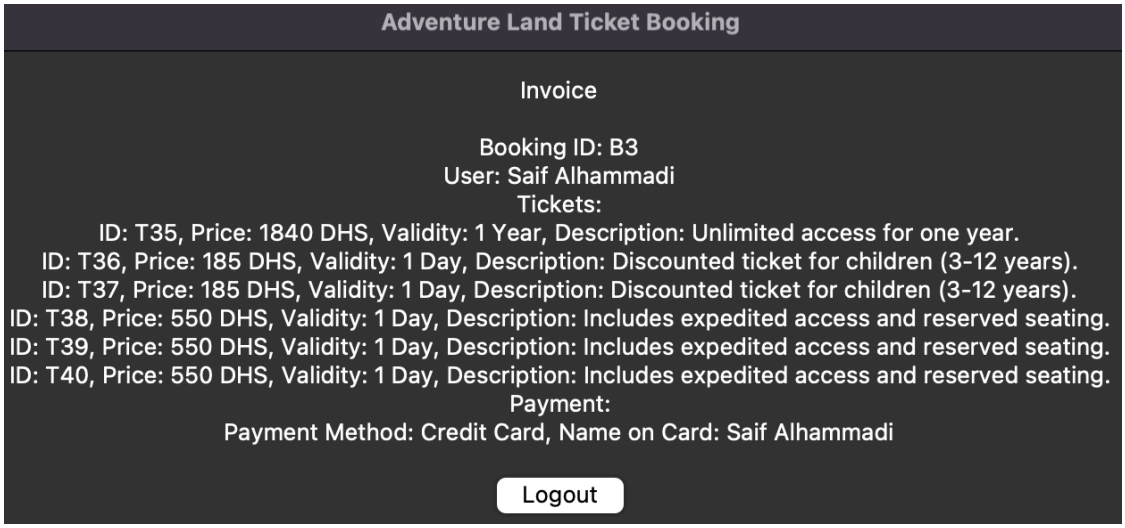


Explanation here:

This screen allows users to enter payment details, including:

- Payment method (e.g., Credit Card, PayPal).
- Name on the card.
- Card number, expiry date, and CVC.

After completing the form, users submit their payment to finalize the booking.



Explanation Here:

Invoice Screen:

- The invoice page displays a summary of the booking:
  - Booking ID, user details, and ticket information (type, quantity, price, and validity).
  - Payment details, such as the method and cardholder's name.
- The user can log out after reviewing the invoice.

GitHub Link:

https://github.com/saif-alh/ICS220-Final-Project-Saif-and-Abdulla.git

LucidChart Link:

https://lucid.app/lucidchart/71486605-e2ed-43d8-a510-acfadb9a7da2/edit?viewport_loc=-1978%2C1178%2C1925%2C2651%2C0_0&invitationId=inv_8782a98a-b6f5-44e5-aa07-2c95f758f8d3

Summary of learnings:

This final assignment provided a deep insight into designing and developing a real-world software application using Object-Oriented principles:

#LO1_OOAD: Using UML diagrams, we learned how to analyze real-world problems and represent them visually. We identified the entities, attributes, and relationships, such as inheritance, aggregation, and composition, to create a clear system blueprint.

#LO2_OOProgramming: To implement the UML design, we learned to write clean, object-oriented Python programs. By creating classes, attributes, and methods, we transformed the diagram into functional code that could solve problems and handle errors easily.

#LO3_SWImplementation: We developed a multi-layered program with a user-friendly GUI that allows users to create, read, update, and delete data. We ensured the software was interactive and met all functional requirements, such as managing tickets, accounts, and sales.

#LO4_SWDocumentation: We improved our abilities to write clear documentation and explanations and ensured they were understandable. This included the UML diagram, the code structure, the GUI features, and the testing methods. This made sure everything was easy to understand and maintain.

Overall, this project helped us understand how to design and build a real-world system using UML diagrams, Python programming, and a user-friendly GUI. We learned to turn ideas into functional software while ensuring it is well-documented and meets all user needs. This final assignment improved our skills in problem-solving, teamwork, and creating clear and easy solutions.