Done by
Dhamini machireddy

Ticket Booking System

Control structure

Task 1: Conditional Statements

In a BookingSystem, you have been given the task is to create a program to book tickets. if available tickets more than the number of tickets to book then display the remaining tickets or ticket unavailable:

Tasks:

1. Write a program that takes the availableTicket and noOfBookingTicket as input.

   Code

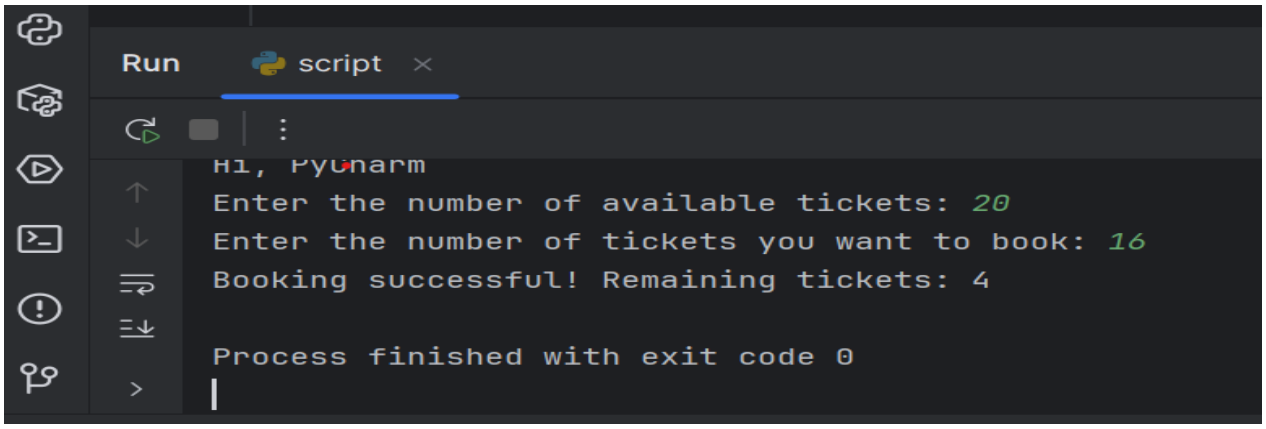
2. Use conditional statements (if-else) to determine if the ticket is available or not.
3. Display an appropriate message based on ticket availability.

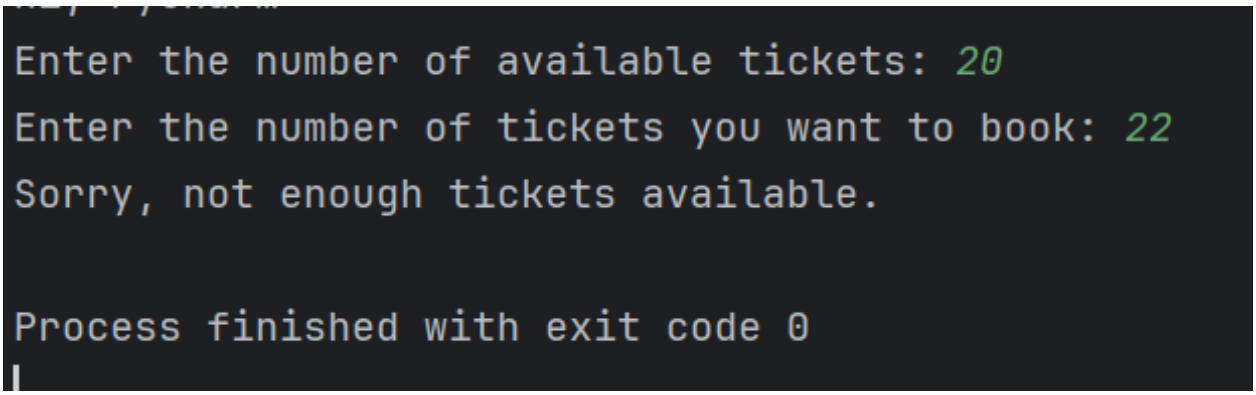
Code :

```
# Step 1: Take input
available_ticket = int(input("Enter the number of available tickets: "))
no_of_booking_ticket = int(input("Enter the number of tickets you want to book: "))

# Step 2: Use conditional statement to check availability
if available_ticket >= no_of_booking_ticket:
   remaining = available_ticket - no_of_booking_ticket
   print(f"Booking successful! Remaining tickets: {remaining}")
else:
   print("Sorry, not enough tickets available.")
```


```python
# Step 1: Take input
available_ticket = int(input("Enter the number of available tickets: "))
no_of_booking_ticket = int(input("Enter the number of tickets you want to book: "))

# Step 2: Use conditional statement to check availability
if available_ticket >= no_of_booking_ticket:
    remaining = available_ticket - no_of_booking_ticket
    print(f"Booking successful! Remaining tickets: {remaining}")
else:
    print("Sorry, not enough tickets available.")
```

Done by
Dhamini machireddy





Task 2: Nested Conditional Statements

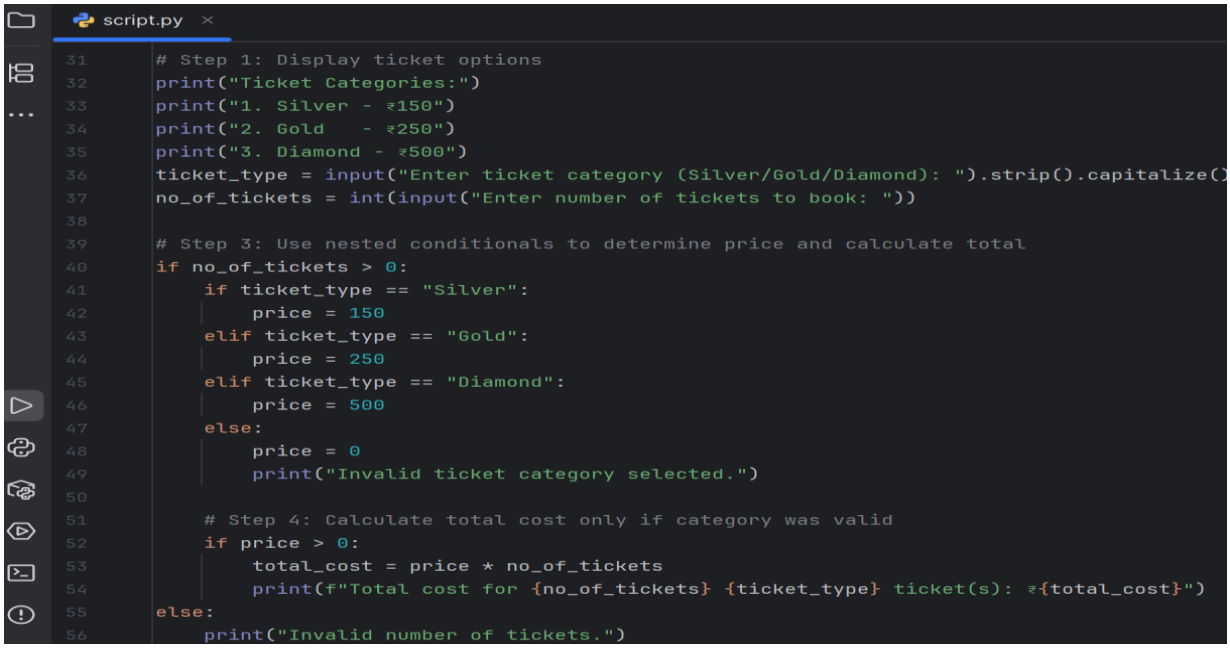
Create a program that simulates a Ticket booking and calculating cost of tickets. Display tickets options such as "Silver", "Gold", "Dimond". Based on ticket category fix the base ticket price and get the user input for ticket type and no of tickets need and calculate the total cost of tickets booked.

Code :

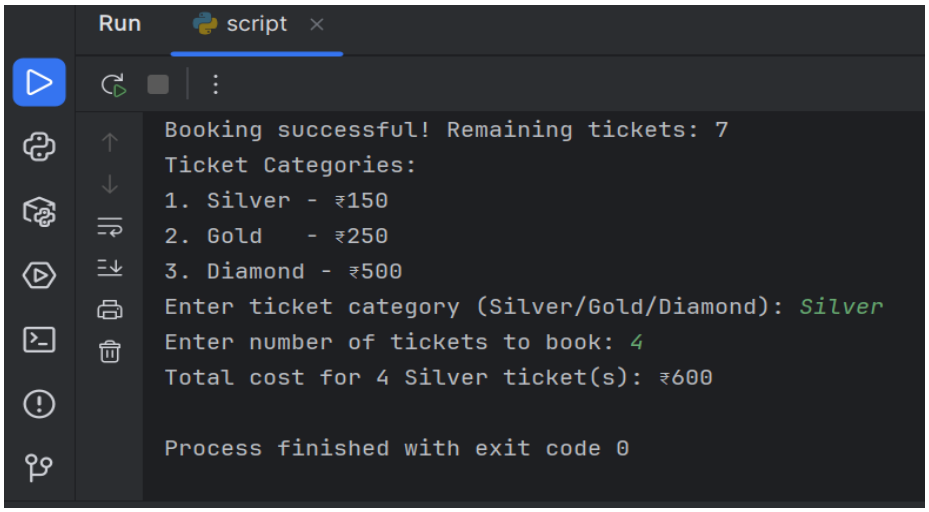
```
# Step 1: Display ticket options
print("Ticket Categories:")
print("1. Silver - ₹150")
print("2. Gold   - ₹250")
print("3. Diamond - ₹500")
ticket_type = input("Enter ticket category (Silver/Gold/Diamond): ").strip().capitalize()
no_of_tickets = int(input("Enter number of tickets to book: "))

# Step 3: Use nested conditionals to determine price and calculate total
if no_of_tickets > 0:
    if ticket_type == "Silver":
        price = 150
    elif ticket_type == "Gold":
        price = 250
    elif ticket_type == "Diamond":
```

Done by
Dhamini machireddy

```
        price = 500
    else:
        price = 0
        print("Invalid ticket category selected.")

    # Step 4: Calculate total cost only if category was valid
    if price > 0:
        total_cost = price * no_of_tickets
        print(f"Total cost for {no_of_tickets} {ticket_type} ticket(s): ₹{total_cost}")
else:
    print("Invalid number of tickets.")
```



Output :



Done by
Dhamini machireddy

Task 3: Looping

From the above task book the tickets for repeatedly until user type "Exit"

Code :

```
print("🎟️ Welcome to the Ticket Booking System 🎟️")
print("Type 'Exit' anytime to stop booking.\n")
while True:
    print("\nTicket Categories:")
    print("1. Silver  - ₹150")
    print("2. Gold    - ₹250")
    print("3. Diamond - ₹500")
    ticket_type = input("Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop: ").strip().capitalize()
    if ticket_type == "Exit":
        print("Thank you for using the Ticket Booking System!")
        break
    try:
        no_of_tickets = int(input("Enter number of tickets to book: "))
    except ValueError:
        print("Invalid input. Please enter a number for tickets.")
        continue
    if no_of_tickets <= 0:
        print("Number of tickets must be greater than 0.")
        continue
    if ticket_type == "Silver":
        price = 150
    elif ticket_type == "Gold":
        price = 250
    elif ticket_type == "Diamond":
        price = 500
    else:
        print("Invalid ticket category selected.")
        continue
    total_cost = price * no_of_tickets
    print(f"☑️ Booking successful! {no_of_tickets} {ticket_type} ticket(s) booked. Total cost: ₹{total_cost}")
```

code ss:

Done by
Dhamini machireddy

```python
print("🎟️ Welcome to the Ticket Booking System 🎟️")
print("Type 'Exit' anytime to stop booking.\n")
while True:
    print("\nTicket Categories:")
    print("1. Silver  - ₹150")
    print("2. Gold    - ₹250")
    print("3. Diamond - ₹500")
    ticket_type = input("Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop: ").strip().capitalize()
    if ticket_type == "Exit":
        print("Thank you for using the Ticket Booking System!")
        break
    try:
        no_of_tickets = int(input("Enter number of tickets to book: "))
    except ValueError:
        print("Invalid input. Please enter a number for tickets.")
        continue
    if no_of_tickets <= 0:
        print("Number of tickets must be greater than 0.")
        continue
    if ticket_type == "Silver":
        price = 150
    elif ticket_type == "Gold":
        price = 250
    elif ticket_type == "Diamond":
        price = 500
    else:
        print("Invalid ticket category selected.")
        continue
    total_cost = price * no_of_tickets
    print(f"✅ Booking successful! {no_of_tickets} {ticket_type} ticket(s) booked. Total cost: ₹{total_cost}")
```

Output :

```
Enter number of tickets to book: 4
Total cost for 4 Gold ticket(s): ₹1000
🎟️ Welcome to the Ticket Booking System 🎟️
Type 'Exit' anytime to stop booking.


Ticket Categories:
1. Silver  - ₹150
2. Gold    - ₹250
3. Diamond - ₹500
Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop: gold
Enter number of tickets to book: 2
✅ Booking successful! 2 Gold ticket(s) booked. Total cost: ₹500

Ticket Categories:
1. Silver  - ₹150
2. Gold    - ₹250
3. Diamond - ₹500
Enter ticket category (Silver/Gold/Diamond) or type 'Exit' to stop: exit
Thank you for using the Ticket Booking System!

Process finished with exit code 0
```

Done by
Dhamini machireddy

Task 4: Class & Object

Create a Following classes with the following attributes and methods:

1. **Event** Class**:**

- **Attributes:**
  - event_name, ○ event_date DATE, ○ event_time TIME, ○ venue_name, ○ total_seats, ○ available_seats, ○ ticket_price DECIMAL,
  - event_type ENUM('Movie', 'Sports', 'Concert')
- **Methods and Constuctors:**
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
  - **calculate_total_revenue()**: Calculate and return the total revenue based on the number of tickets sold.
  - **getBookedNoOfTickets()**: return the total booked tickets
  - **book_tickets(num_tickets)**: Book a specified number of tickets for an event. Initially available seats are equal to the total seats when tickets are booked available seats number should be reduced.
  - **cancel_booking(num_tickets)**: Cancel the booking and update the available seats.
  - **display_event_details():** Display event details, including event name, date time seat availability.

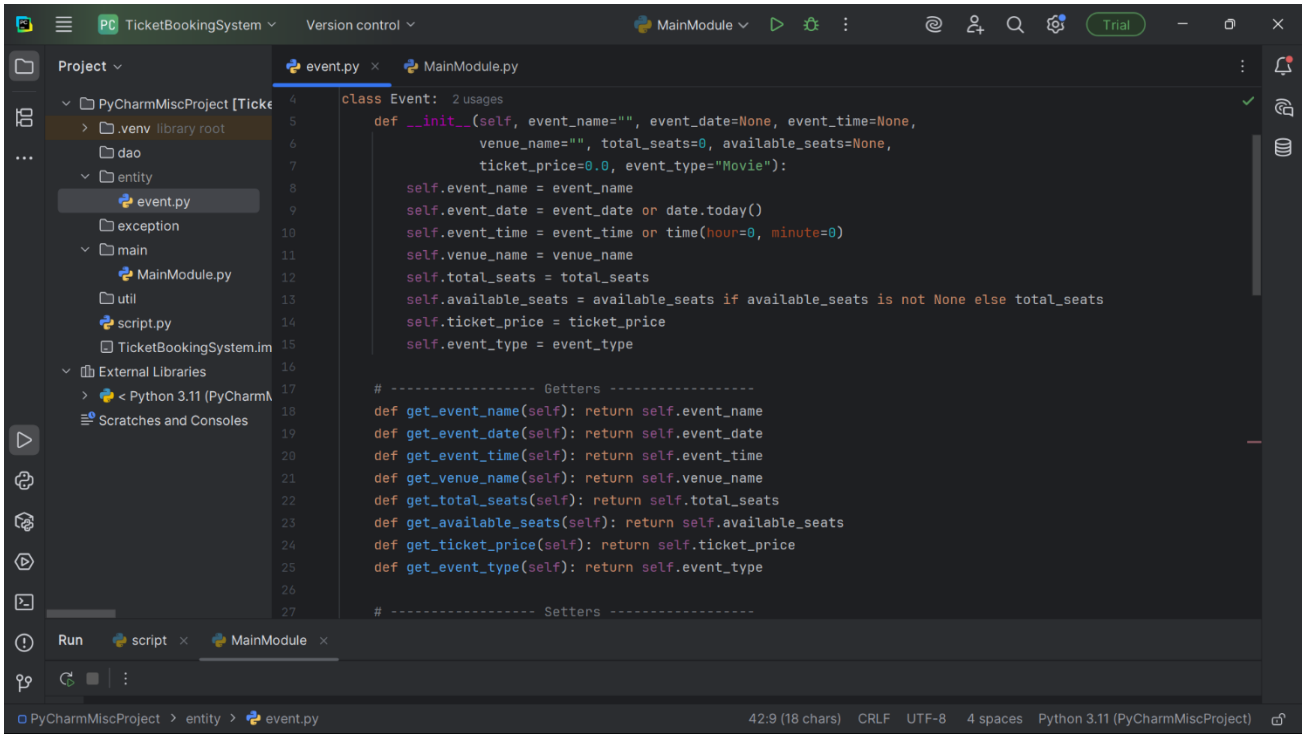
Code :

```
from datetime import date, time

class Event:
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type="Movie"):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # ------------------ Getters ------------------
    def get_event_name(self): return self.event_name
    def get_event_date(self): return self.event_date
    def get_event_time(self): return self.event_time
    def get_venue_name(self): return self.venue_name
    def get_total_seats(self): return self.total_seats
    def get_available_seats(self): return self.available_seats
    def get_ticket_price(self): return self.ticket_price
    def get_event_type(self): return self.event_type
```

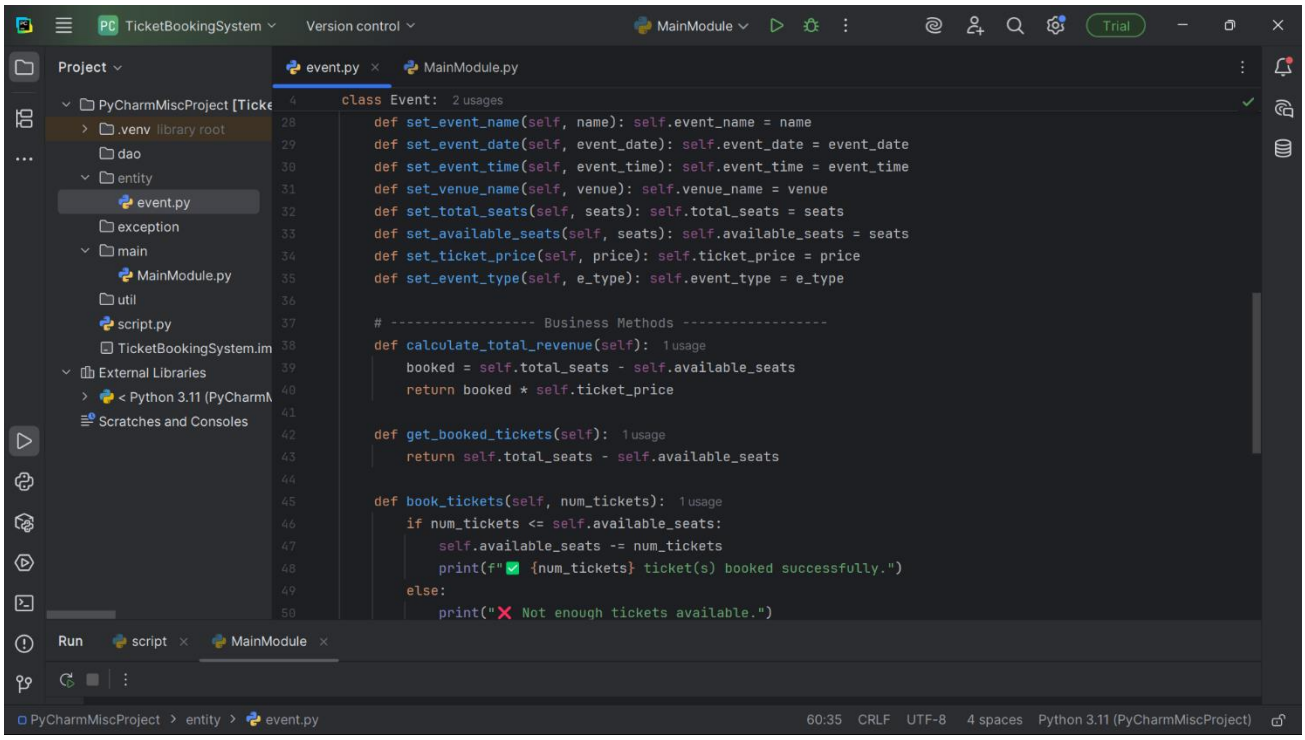
Done by
Dhamini machireddy

```
def set_event_name(self, name): self.event_name = name
def set_event_date(self, event_date): self.event_date = event_date
def set_event_time(self, event_time): self.event_time = event_time
def set_venue_name(self, venue): self.venue_name = venue
def set_total_seats(self, seats): self.total_seats = seats
def set_available_seats(self, seats): self.available_seats = seats
def set_ticket_price(self, price): self.ticket_price = price
def set_event_type(self, e_type): self.event_type = e_type
def calculate_total_revenue(self):
    booked = self.total_seats - self.available_seats
    return booked * self.ticket_price

def get_booked_tickets(self):
    return self.total_seats - self.available_seats

def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"☑ {num_tickets} ticket(s) booked successfully.")
    else:
        print("❌ Not enough tickets available.")

def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"🔄 {num_tickets} ticket(s) cancelled successfully.")
    else:
        print("❌ Cannot cancel more than total booked tickets.")

def display_event_details(self):
    print("\n🧮 Event Details")
    print(f"Event Name      : {self.event_name}")
    print(f"Event Date      : {self.event_date}")
    print(f"Event Time      : {self.event_time}")
    print(f"Venue Name      : {self.venue_name}")
    print(f"Event Type      : {self.event_type}")
    print(f"Total Seats     : {self.total_seats}")
    print(f"Available Seats  : {self.available_seats}")
    print(f"Ticket Price (₹) : {self.ticket_price:.2f}")
```

Done by
Dhamini machireddy

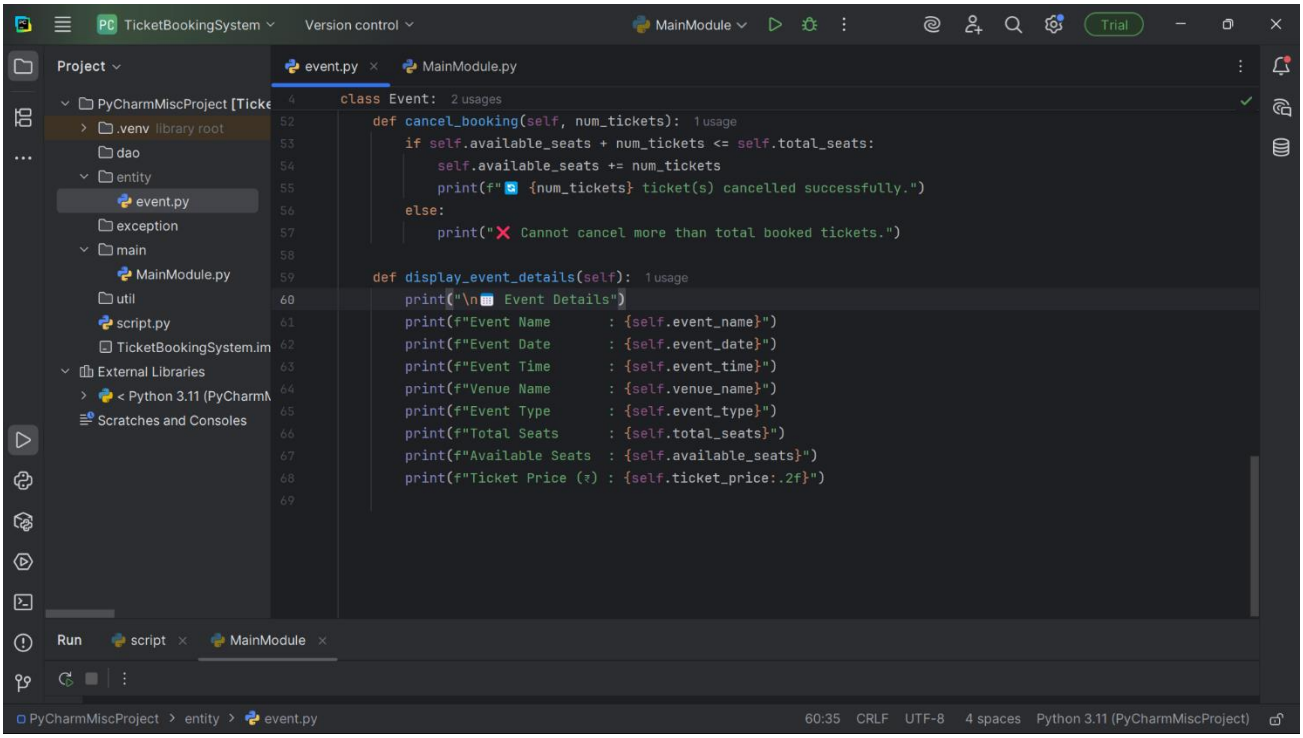


```
class Event:  2 usages
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type="Movie"):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # ------------------ Getters ------------------
    def get_event_name(self): return self.event_name
    def get_event_date(self): return self.event_date
    def get_event_time(self): return self.event_time
    def get_venue_name(self): return self.venue_name
    def get_total_seats(self): return self.total_seats
    def get_available_seats(self): return self.available_seats
    def get_ticket_price(self): return self.ticket_price
    def get_event_type(self): return self.event_type

    # ------------------ Setters ------------------
```

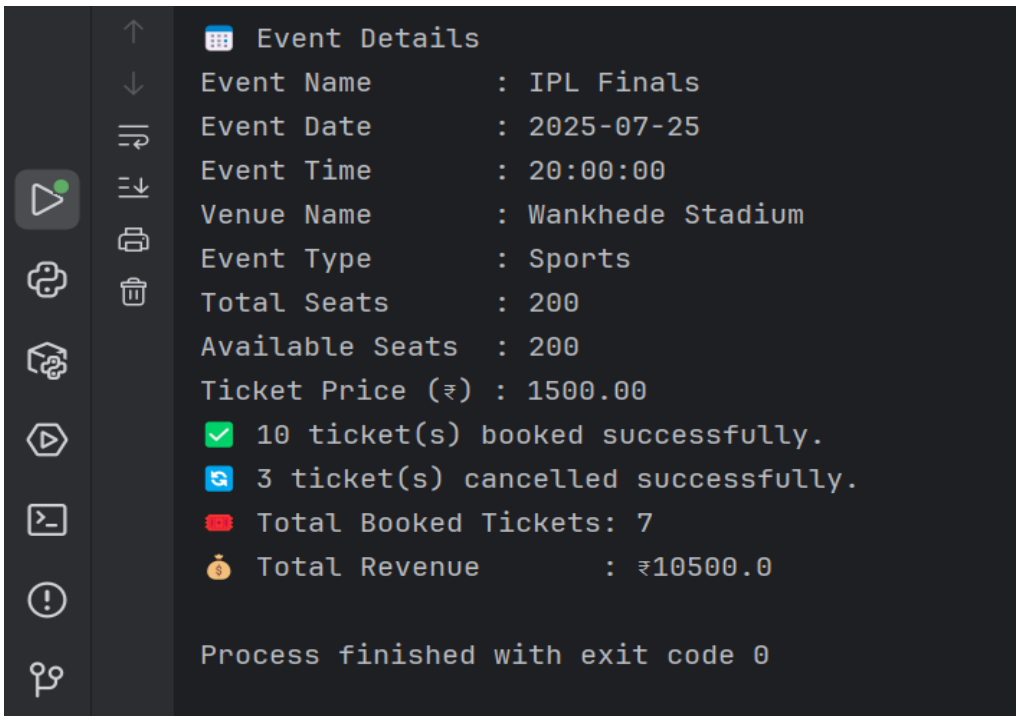


```
class Event:  2 usages
    def set_event_name(self, name): self.event_name = name
    def set_event_date(self, event_date): self.event_date = event_date
    def set_event_time(self, event_time): self.event_time = event_time
    def set_venue_name(self, venue): self.venue_name = venue
    def set_total_seats(self, seats): self.total_seats = seats
    def set_available_seats(self, seats): self.available_seats = seats
    def set_ticket_price(self, price): self.ticket_price = price
    def set_event_type(self, e_type): self.event_type = e_type

    # ------------------ Business Methods ------------------
    def calculate_total_revenue(self):  1 usage
        booked = self.total_seats - self.available_seats
        return booked * self.ticket_price

    def get_booked_tickets(self):  1 usage
        return self.total_seats - self.available_seats

    def book_tickets(self, num_tickets):  1 usage
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✅ {num_tickets} ticket(s) booked successfully.")
        else:
            print("❌ Not enough tickets available.")
```

Done by
Dhamini machireddy



output :



2. **Venue** Class
   - **Attributes**:

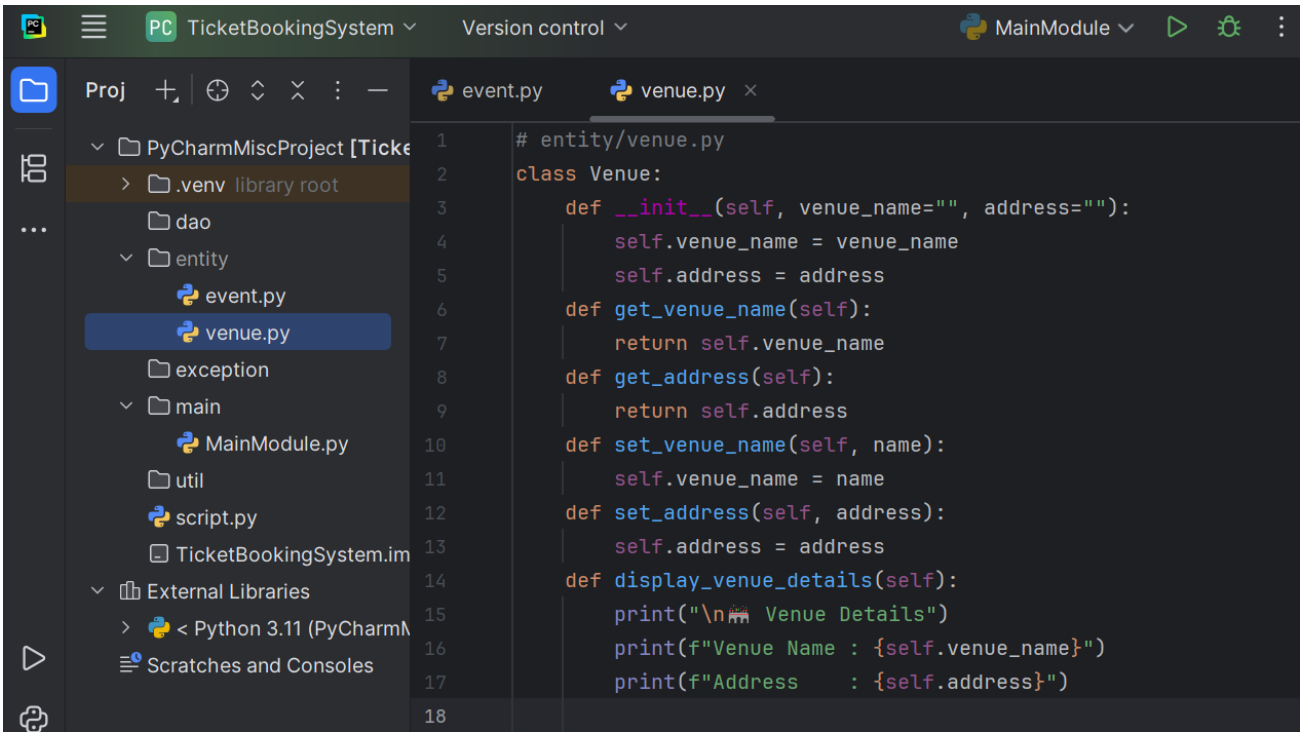
- venue_name, ○ address

- **Methods and Constuctors:**
  - **display_venue_details():** Display venue details.
  - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

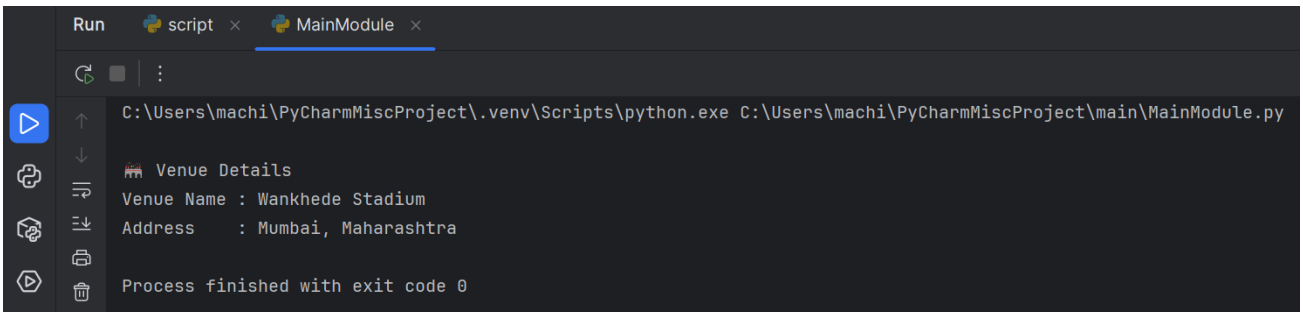
Code:

```
class Venue:
    def __init__(self, venue_name="", address=""):
        self.venue_name = venue_name
        self.address = address

    # ---------- Getter Methods ----------
    def get_venue_name(self):
        return self.venue_name

    def get_address(self):
        return self.address

    # ---------- Setter Methods ----------
    def set_venue_name(self, name):
        self.venue_name = name

    def set_address(self, address):
        self.address = address

    # ---------- Display Method ----------
    def display_venue_details(self):
        print("\n🏟 Venue Details")
        print(f"Venue Name : {self.venue_name}")
        print(f"Address    : {self.address}")
```

Done by
Dhamini machireddy



Output :



3. **Customer** Class
   - **Attributes:**
     - customer_name, email, phone_number,
   - **Methods and Constuctors:**
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
     - **display_customer_details()**: Display customer details.

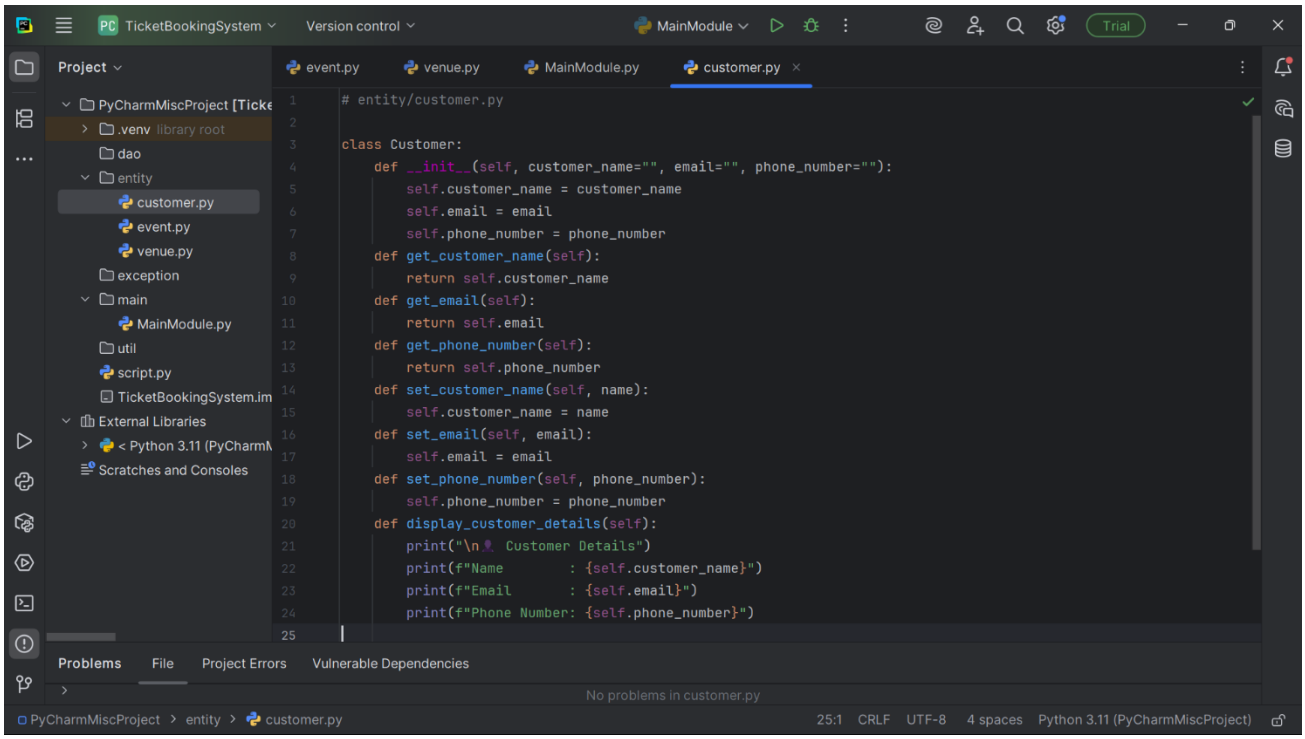
Code :

```
# entity/customer.py

class Customer:
    def __init__(self, customer_name="", email="", phone_number=""):
```

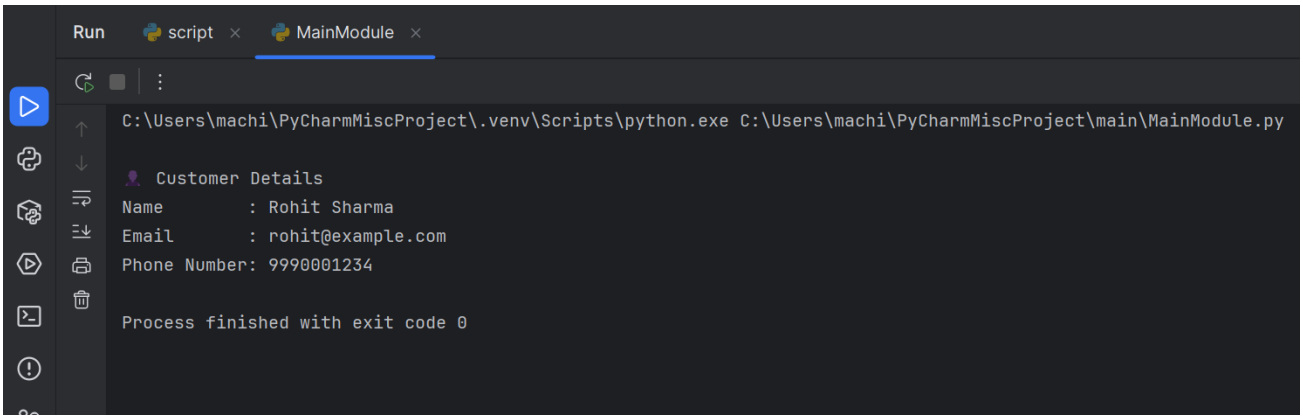
Done by
Dhamini machireddy

```
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    # ---------- Getter Methods ----------
    def get_customer_name(self):
        return self.customer_name

    def get_email(self):
        return self.email

    def get_phone_number(self):
        return self.phone_number

    # ---------- Setter Methods ----------
    def set_customer_name(self, name):
        self.customer_name = name

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

    def set_phone_number(self, phone_number):
        self.phone_number = phone_number

    # ---------- Display Method ----------
    def display_customer_details(self):
        print("\n👤 Customer Details")
        print(f"Name        : {self.customer_name}")
        print(f"Email       : {self.email}")
        print(f"Phone Number: {self.phone_number}")
```

Done by
Dhamini machireddy



output:



4. **Booking** Class to represent the Tiket booking system. Perform the following operation in main method. Note:- Use Event class object for the following operation.

- **Methods and Constuctors:**
  - **calculate_booking_cost(num_tickets)**: Calculate and set the total cost of the booking.
  - **book_tickets(num_tickets)**: Book a specified number of tickets for an event. ○ **cancel_booking(num_tickets)**: Cancel the booking and update the available seats.
  - **getAvailableNoOfTickets()**: return the total available tickets ○ **getEventDetails()**: return event details from the event class

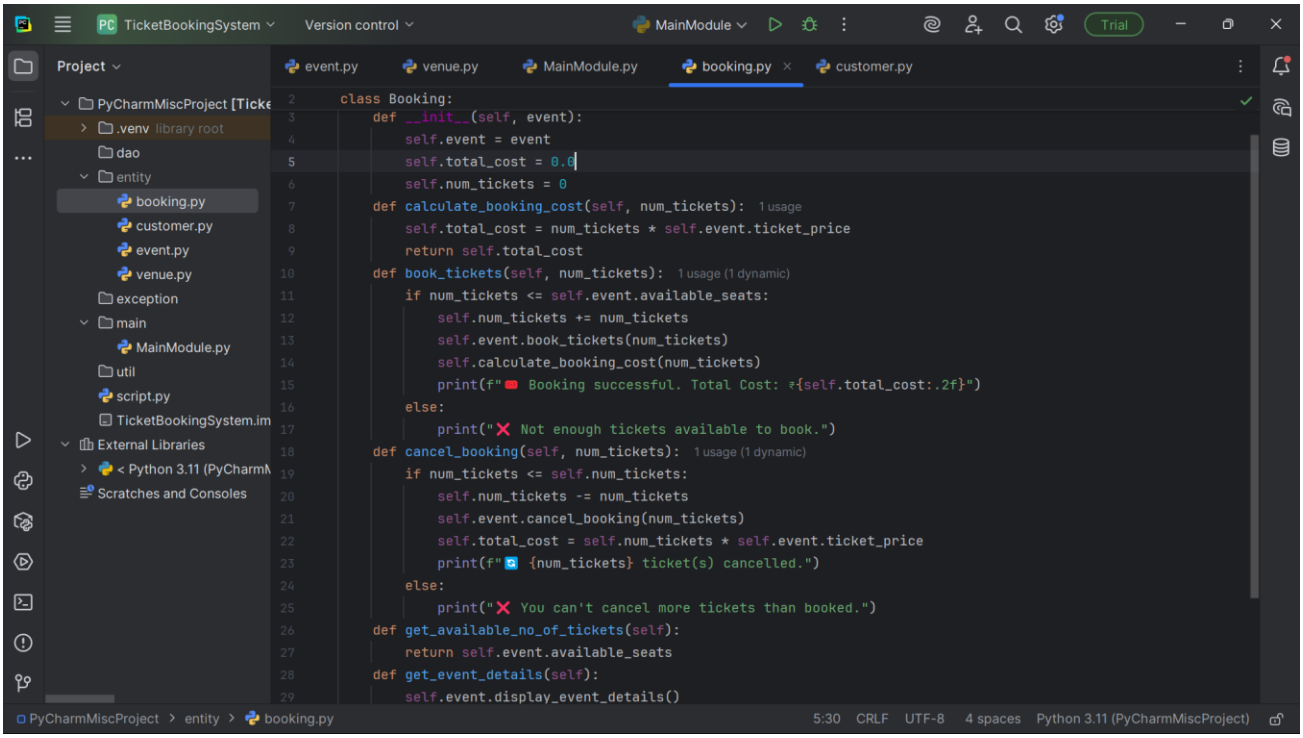
code:

```
# entity/booking.py
```

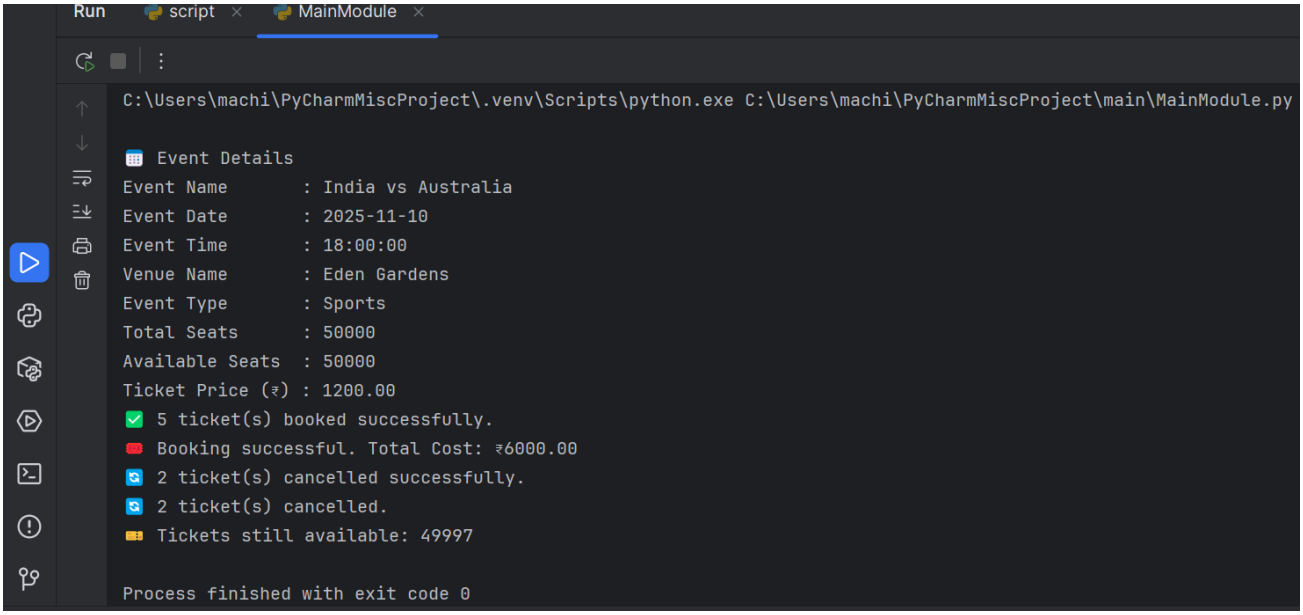
Done by
Dhamini machireddy

```
class Booking:
    def __init__(self, event):
        self.event = event  # reference to Event object
        self.total_cost = 0.0
        self.num_tickets = 0

    def calculate_booking_cost(self, num_tickets):
        self.total_cost = num_tickets * self.event.ticket_price
        return self.total_cost

    def book_tickets(self, num_tickets):
        if num_tickets <= self.event.available_seats:
            self.num_tickets += num_tickets
            self.event.book_tickets(num_tickets)
            self.calculate_booking_cost(num_tickets)
            print(f"🎟️ Booking successful. Total Cost: ₹{self.total_cost:.2f}")
        else:
            print("❌ Not enough tickets available to book.")

    def cancel_booking(self, num_tickets):
        if num_tickets <= self.num_tickets:
            self.num_tickets -= num_tickets
            self.event.cancel_booking(num_tickets)
            self.total_cost = self.num_tickets * self.event.ticket_price
            print(f"🔄 {num_tickets} ticket(s) cancelled.")
        else:
            print("❌ You can't cancel more tickets than booked.")

    def get_available_no_of_tickets(self):
        return self.event.available_seats

    def get_event_details(self):
        self.event.display_event_details()
```

Done by
Dhamini machireddy



Output :



Task 5: Inheritance and polymorphism

1. Inheritance

- Create a subclass **Movie** that inherits from **Event**. Add the following attributes and methods:
  - **Attributes:**
    1. genre: Genre of the movie (e.g., Action, Comedy, Horror).
    2. ActorName
    3. ActresName

Done by
Dhamini machireddy

- **Methods:**

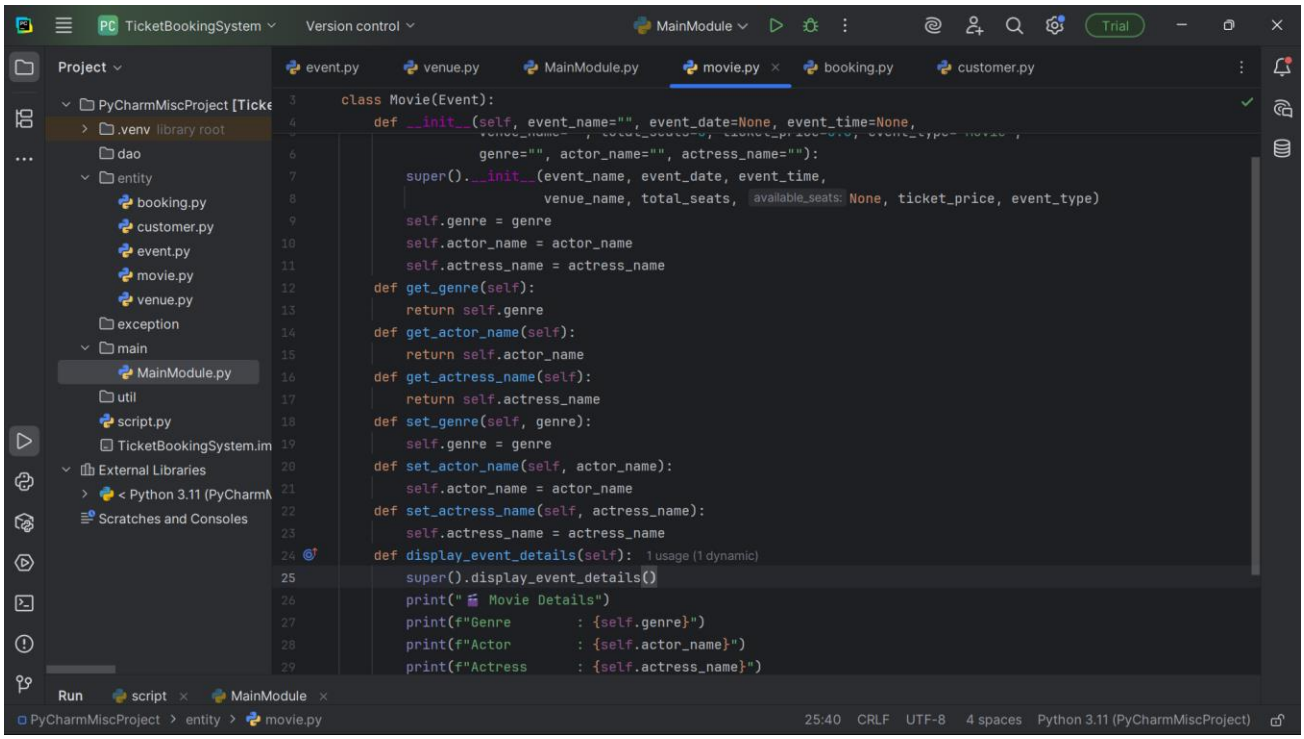
1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

2. **display_event_details():** Display movie details, including genre.

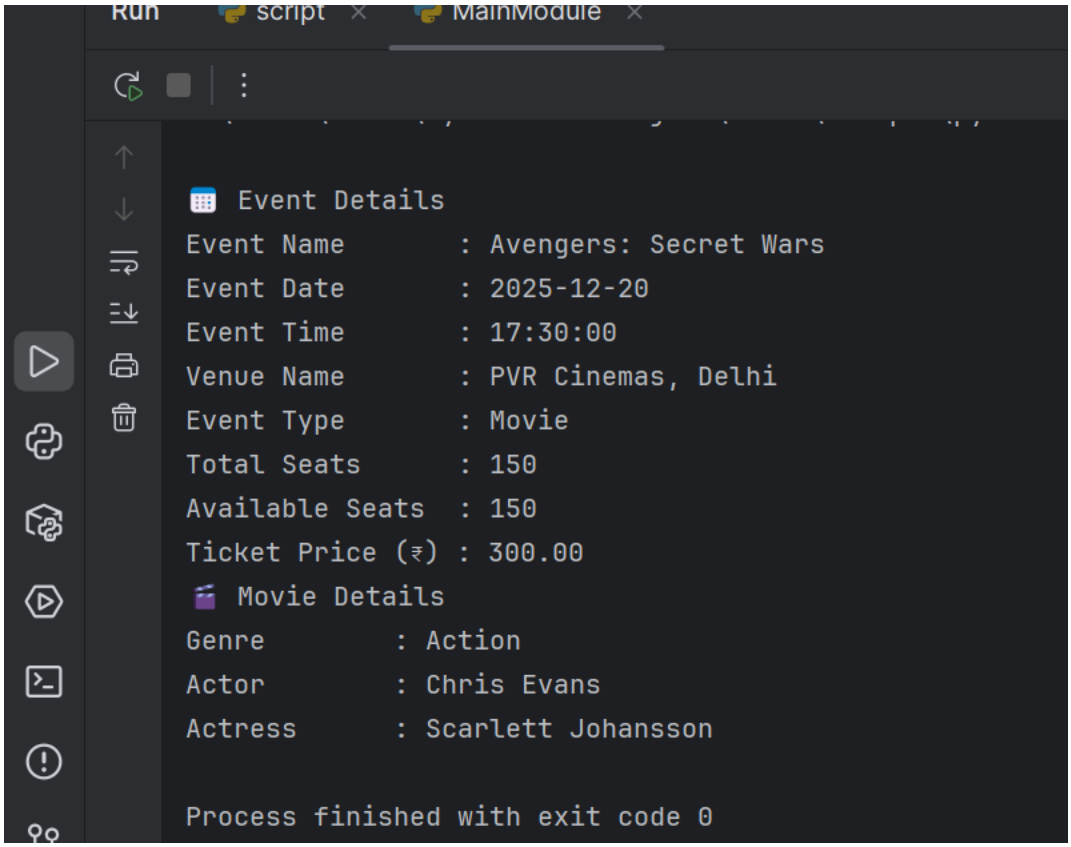
Code :

```
# entity/movie.py
from entity.event import Event
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue_name, total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name
    def get_genre(self):
        return self.genre
    def get_actor_name(self):
        return self.actor_name
    def get_actress_name(self):
        return self.actress_name
    def set_genre(self, genre):
        self.genre = genre
    def set_actor_name(self, actor_name):
        self.actor_name = actor_name
    def set_actress_name(self, actress_name):
        self.actress_name = actress_name
    def display_event_details(self):
        super().display_event_details()
        print(" 🎬 Movie Details")
        print(f"Genre       : {self.genre}")
        print(f"Actor       : {self.actor_name}")
        print(f"Actress     : {self.actress_name}")
```

Done by
Dhamini machireddy



Output :



- Create another subclass **Concert** that inherits from **Event**.

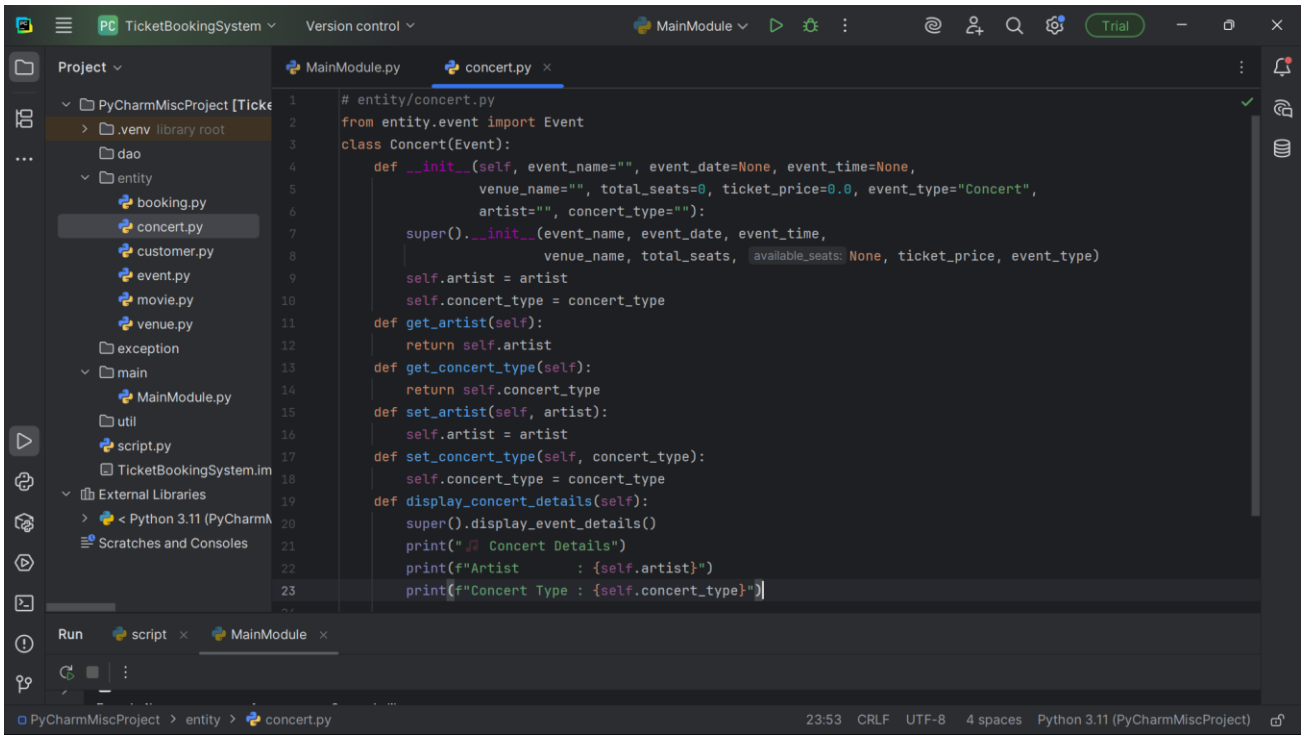
Done by
Dhamini machireddy

- Add the following attributes and methods:
  - **Attributes:**
    1. artist: Name of the performing artist or band.
    2. type: (Theatrical, Classical, Rock, Recital) ○ **Methods:**
       1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
       2. **display_concert_details():** Display concert details, including the artist.

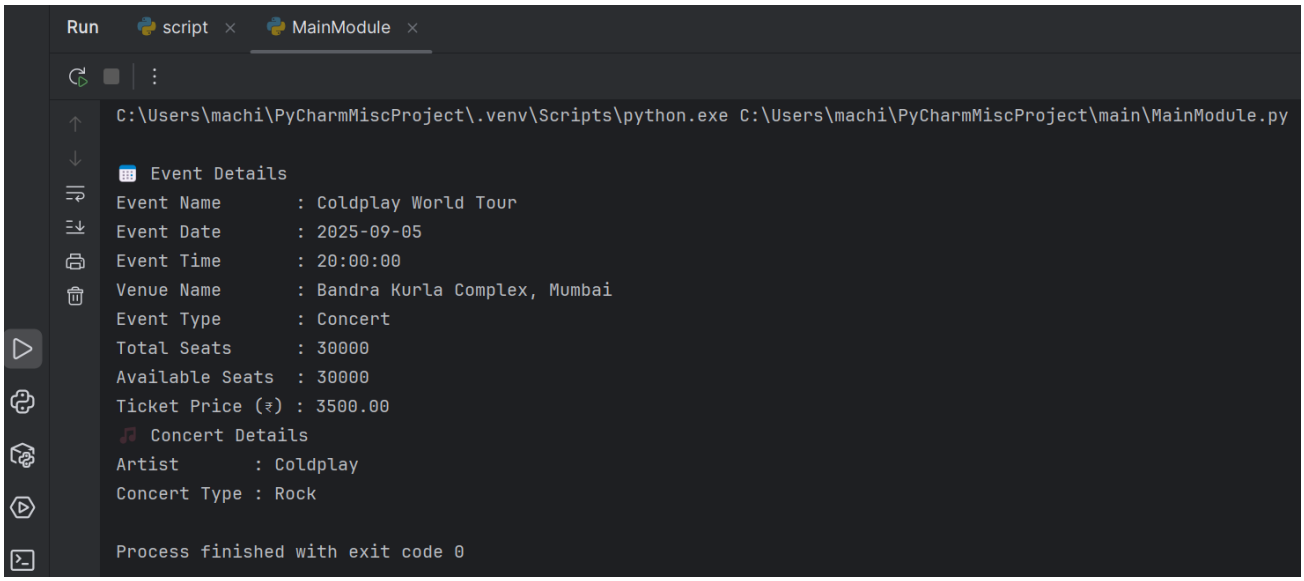
Code :

```
# entity/concert.py

from entity.event import Event

class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time,
                         venue_name, total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

    # ---------- Getter Methods ----------
    def get_artist(self):
        return self.artist

    def get_concert_type(self):
        return self.concert_type
    # ---------- Setter Methods ----------
    def set_artist(self, artist):
        self.artist = artist

    def set_concert_type(self, concert_type):
        self.concert_type = concert_type

    # ---------- Display Method ----------
    def display_concert_details(self):
        super().display_event_details()
        print("🎵 Concert Details")
        print(f"Artist       : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
```

Done by
Dhamini machireddy



Output :



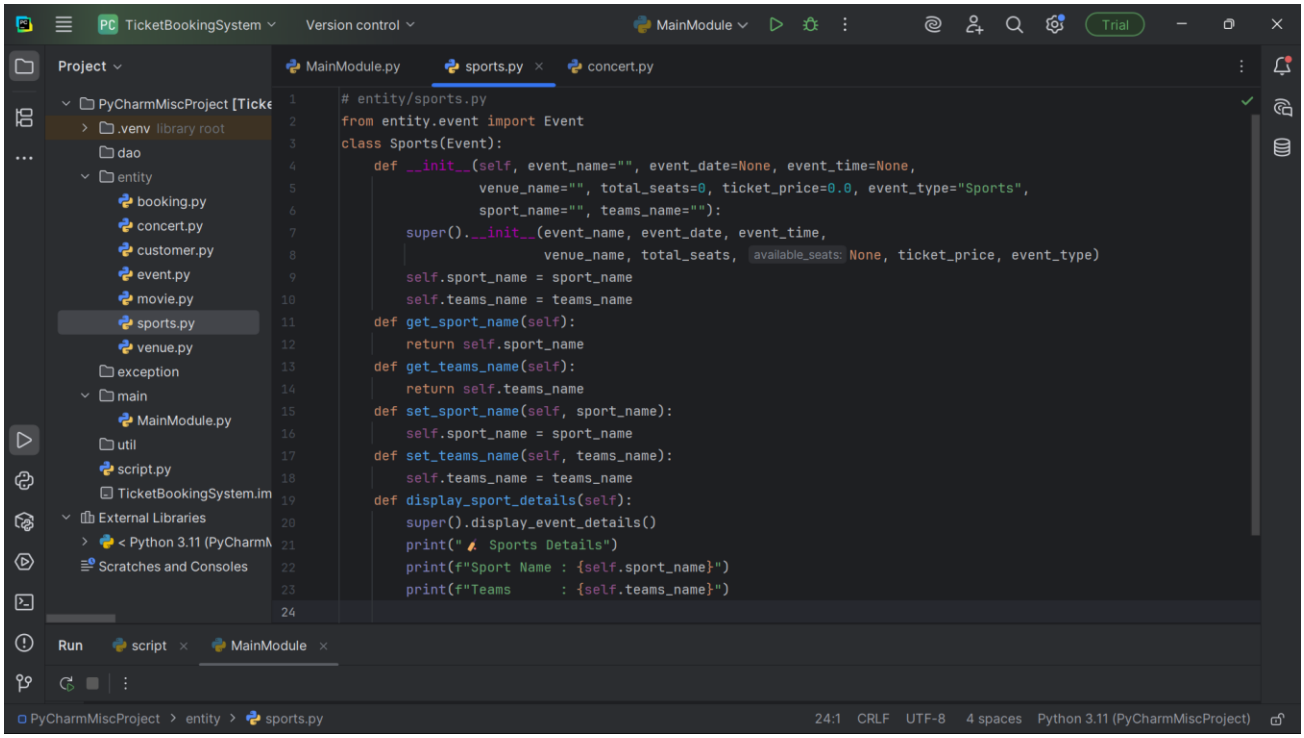
- Create another subclass **Sports** that inherits from **Event**. Add the following attributes and methods:
  - **Attributes:**
    1. sportName: Name of the game.
    2. teamsName: (India vs Pakistan)
  - **Methods:**
    1. Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
    2. **display_sport_details():** Display concert details, including the artist.

Done by
Dhamini machireddy

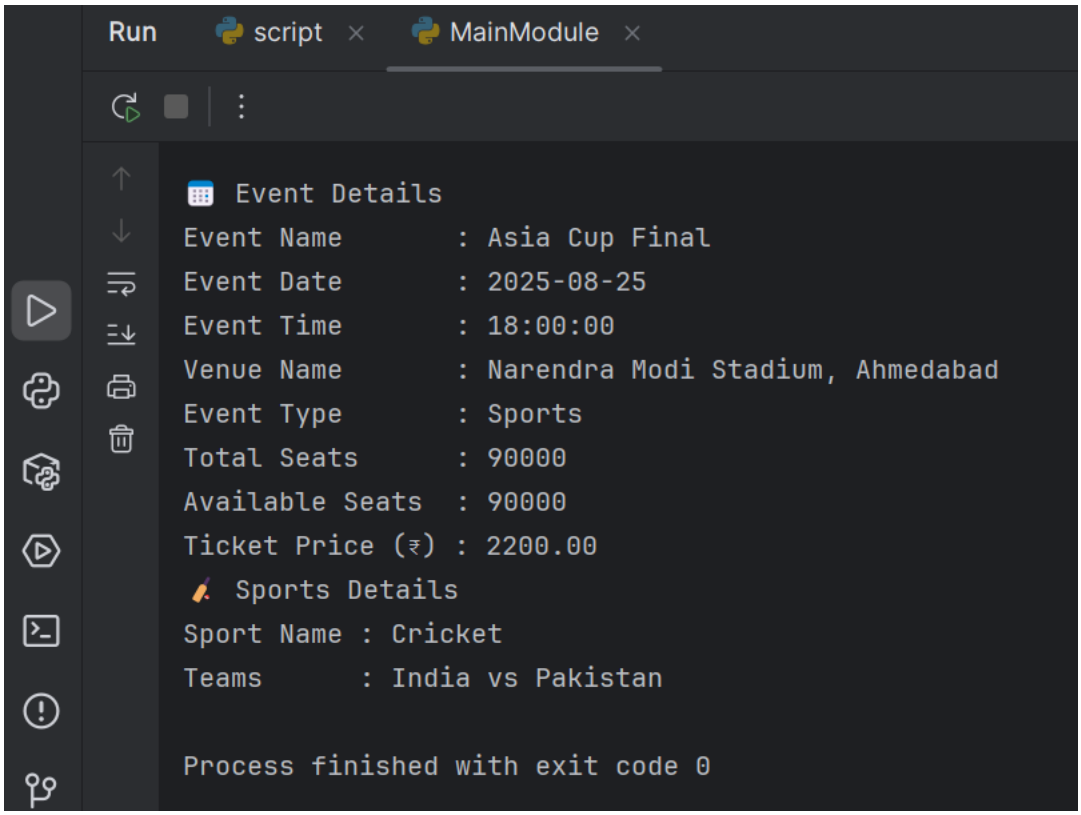
Code:

```
# entity/sports.py

from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue_name, total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name


    def get_sport_name(self):
        return self.sport_name

    def get_teams_name(self):
        return self.teams_name

    def set_sport_name(self, sport_name):
        self.sport_name = sport_name

    def set_teams_name(self, teams_name):
        self.teams_name = teams_name


    def display_sport_details(self):
        super().display_event_details()
        print(" 🏏 Sports Details")
        print(f"Sport Name : {self.sport_name}")
        print(f"Teams      : {self.teams_name}")
```

Done by
Dhamini machireddy



Output :



- Create a class **TicketBookingSystem** with the following methods:

Done by
Dhamini machireddy

- **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu_name:str):**

  Create a new event with the specified details and event type (movie, sport or concert) and return event object.

  - **display_event_details(event: Event)**: Accepts an event object and calls its **display_event_details()** method to display event details.
  - **book_tickets(event: Event, num_tickets: int):**
    1. Accepts an event object and the number of tickets to be booked.
    2. Checks if there are enough available seats for the booking.
    3. If seats are available, updates the available seats and returns the total cost of the booking.
    4. If seats are not available, displays a message indicating that the event is sold out.
  - **cancel_tickets(event: Event, num_tickets)**: cancel a specified number of tickets for an event.
  - **main():** simulates the ticket booking system
    1. User can book tickets and view the event details as per their choice in menu (movies, sports, concerts).
    2. Display event details using the display_event_details() method without knowing the specific event type (demonstrate polymorphism).
    3. Make bookings using the book_tickets() and cancel tickets cancel_tickets() method.

Code :

```
# main/ticket_booking_system.py

from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime


class TicketBookingSystem:

    def create_event(self, event_name: str, date_str: str, time_str: str,
                     total_seats: int, ticket_price: float, event_type: str,
                     venue_name: str):
        event_date = datetime.strptime(date_str, "%Y-%m-%d").date()
        event_time = datetime.strptime(time_str, "%H:%M").time()

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            return Movie(event_name, event_date, event_time, venue_name,
                         total_seats, ticket_price, "Movie", genre, actor, actress)
```

Done by
Dhamini machireddy

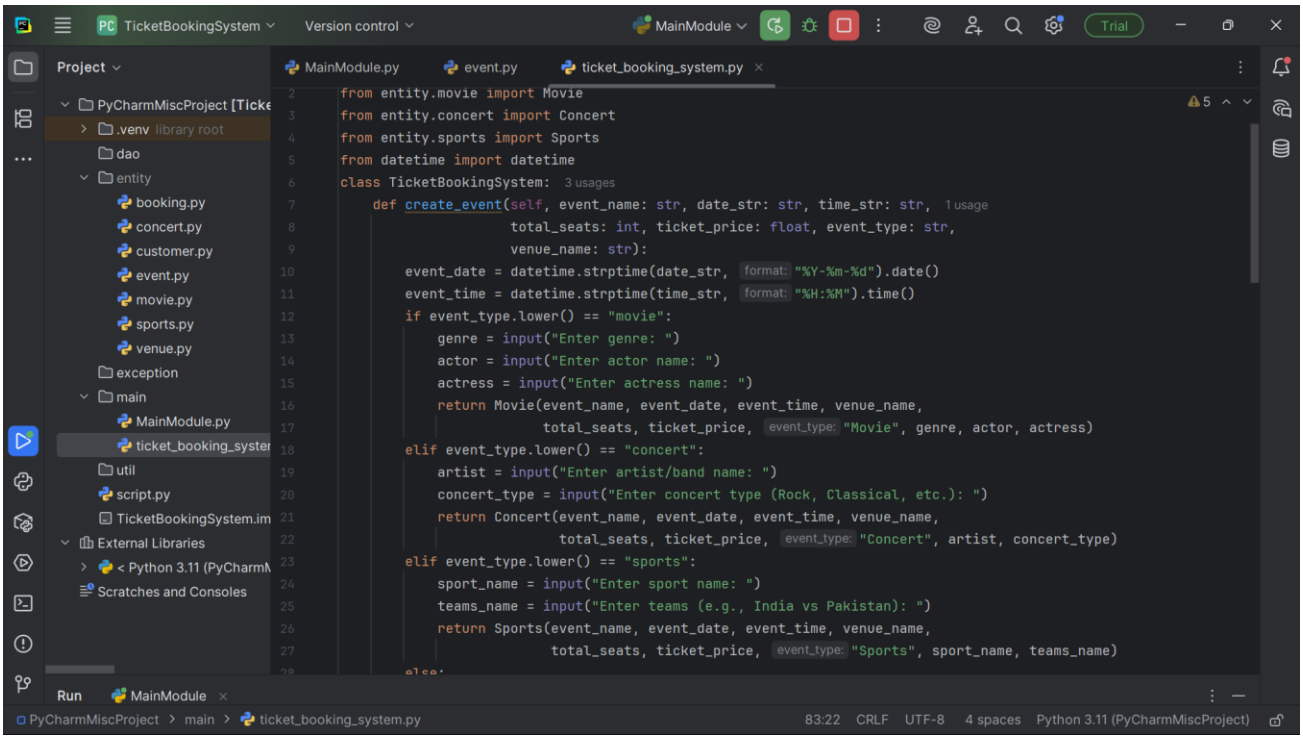
```
    elif event_type.lower() == "concert":
        artist = input("Enter artist/band name: ")
        concert_type = input("Enter concert type (Rock, Classical, etc.): ")
        return Concert(event_name, event_date, event_time, venue_name,
                       total_seats, ticket_price, "Concert", artist, concert_type)

    elif event_type.lower() == "sports":
        sport_name = input("Enter sport name: ")
        teams_name = input("Enter teams (e.g., India vs Pakistan): ")
        return Sports(event_name, event_date, event_time, venue_name,
                      total_seats, ticket_price, "Sports", sport_name, teams_name)

    else:
        print("❌ Invalid event type!")
        return None

def display_event_details(self, event):
    event.display_event_details()  # Polymorphism

def book_tickets(self, event, num_tickets: int):
    if num_tickets <= event.available_seats:
        event.book_tickets(num_tickets)
        cost = num_tickets * event.ticket_price
        print(f"🎟️ Total cost: ₹{cost}")
        return cost
    else:
        print("❌ Event is sold out or not enough tickets.")
        return 0

def cancel_tickets(self, event, num_tickets: int):
    event.cancel_booking(num_tickets)

def main(self):
    system = TicketBookingSystem()
    current_event = None

    while True:
        print("\n📌 Ticket Booking Menu")
        print("1. Create New Event")
        print("2. View Event Details")
```

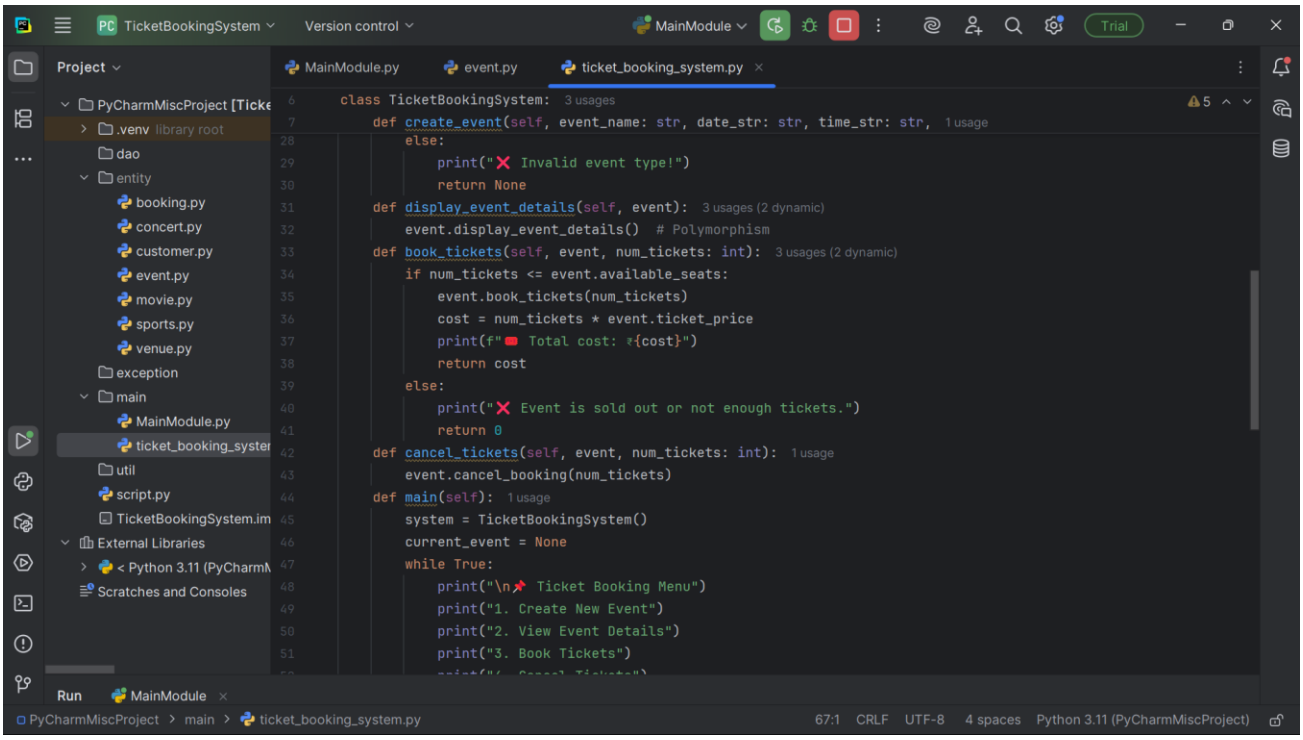
Done by
Dhamini machireddy

```
print("3. Book Tickets")
print("4. Cancel Tickets")
print("5. Exit")

choice = input("Enter your choice (1-5): ")

if choice == "1":
    name = input("Enter event name: ")
    date_str = input("Enter event date (YYYY-MM-DD): ")
    time_str = input("Enter event time (HH:MM): ")
    venue = input("Enter venue name: ")
    seats = int(input("Enter total seats: "))
    price = float(input("Enter ticket price: "))
    type_str = input("Enter event type (Movie/Sports/Concert): ")
    current_event = system.create_event(name, date_str, time_str, seats, price, type_str, venue)

elif choice == "2":
    if current_event:
        system.display_event_details(current_event)
    else:
        print("⚠ No event created yet.")

elif choice == "3":
    if current_event:
        num = int(input("Enter number of tickets to book: "))
        system.book_tickets(current_event, num)
    else:
        print("⚠ No event available for booking.")

elif choice == "4":
    if current_event:
        num = int(input("Enter number of tickets to cancel: "))
        system.cancel_tickets(current_event, num)
    else:
        print("⚠ No event available for cancellation.")

elif choice == "5":
    print("👋 Thank you for using the Ticket Booking System.")
    break

else:
```

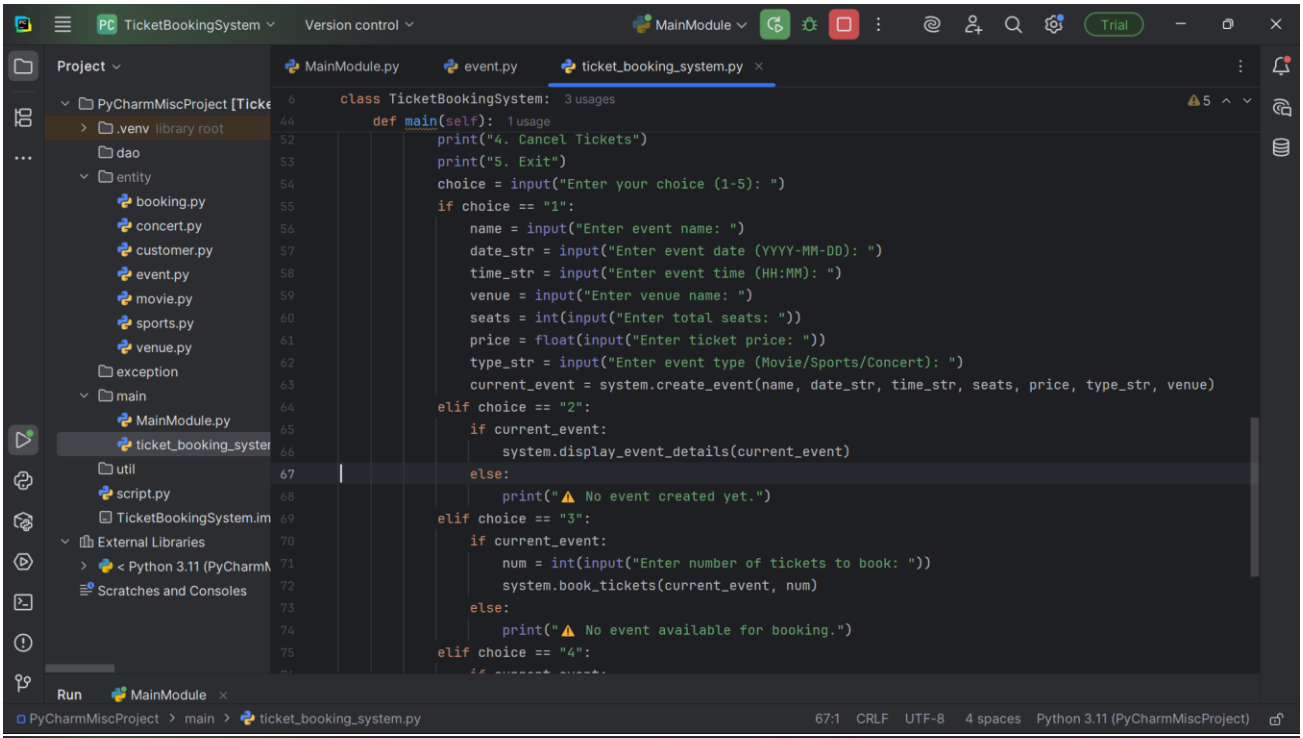
Done by
Dhamini machireddy

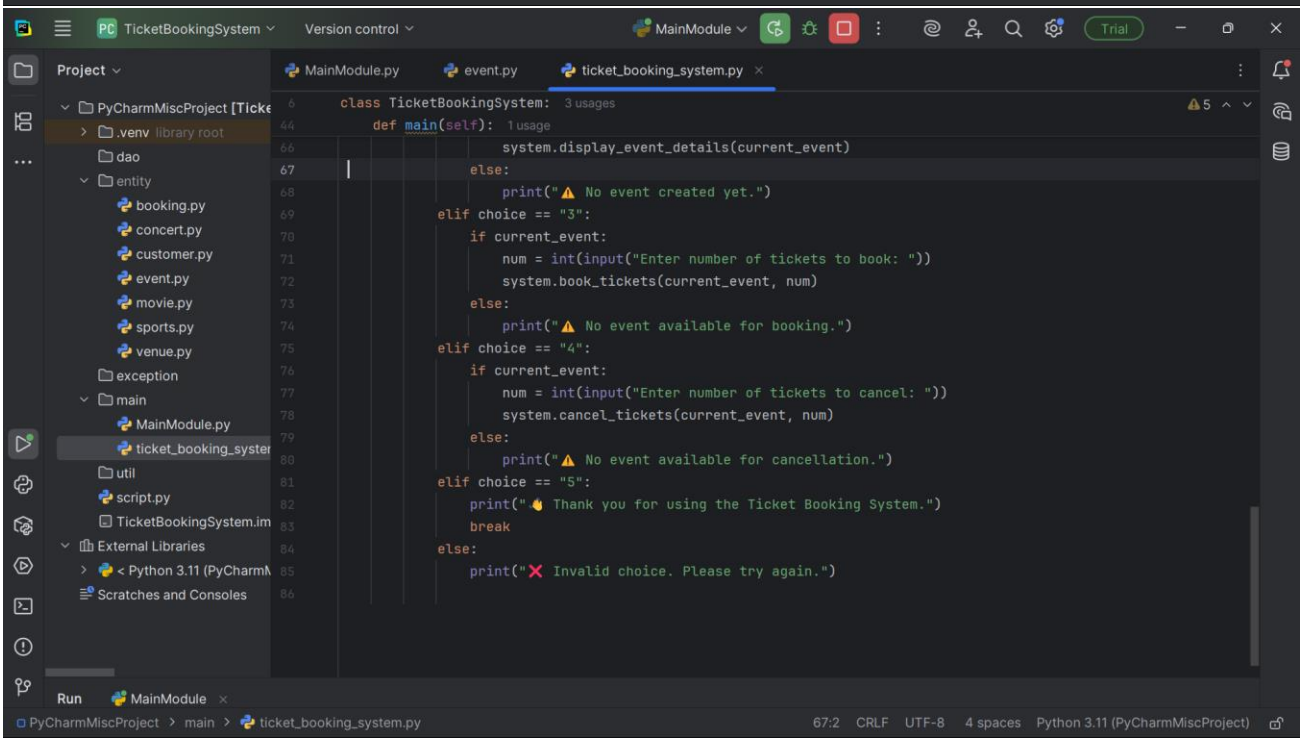
print("❌ Invalid choice. Please try again.")





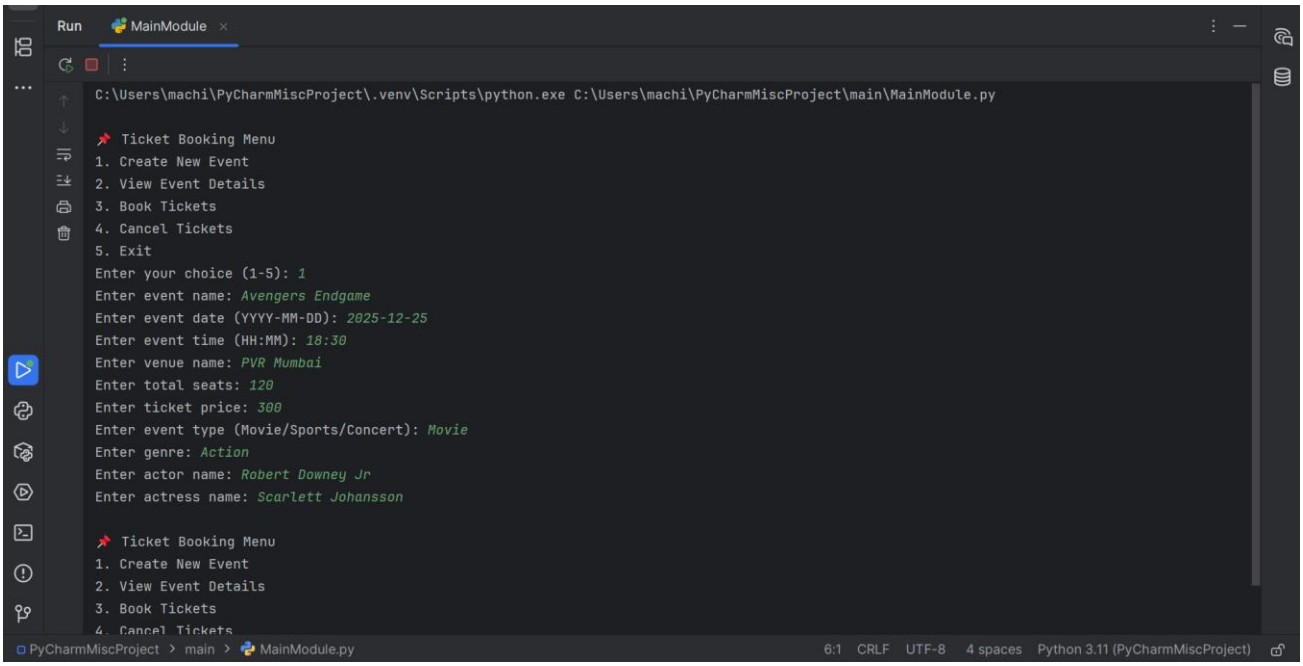
Done by
Dhamini machireddy

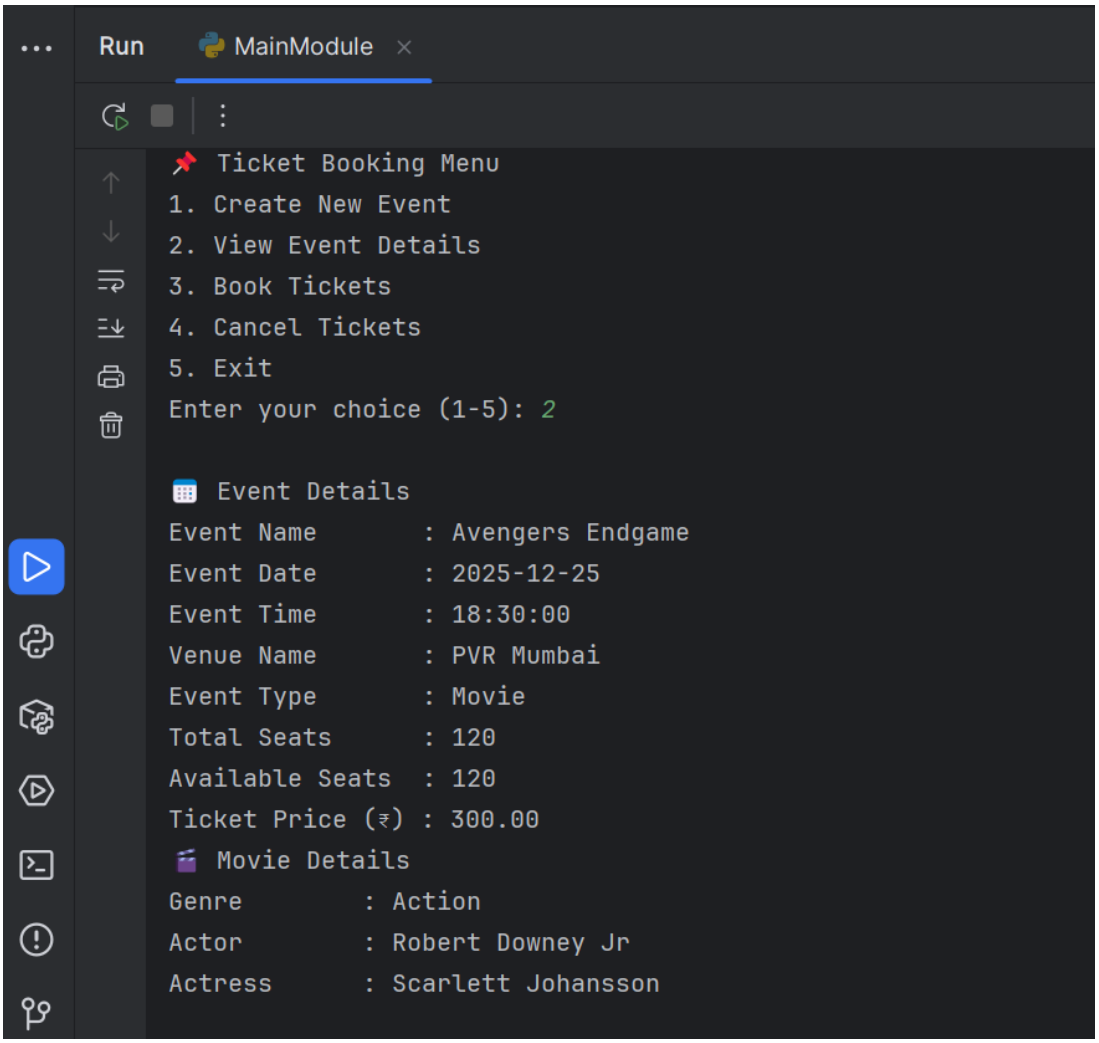




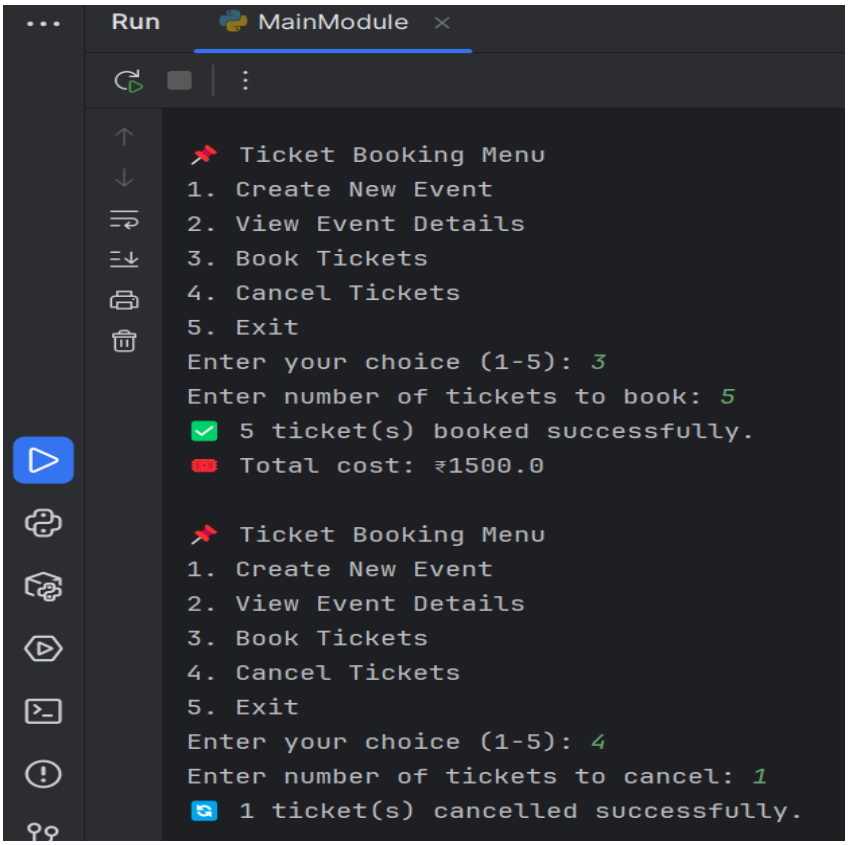
Output:

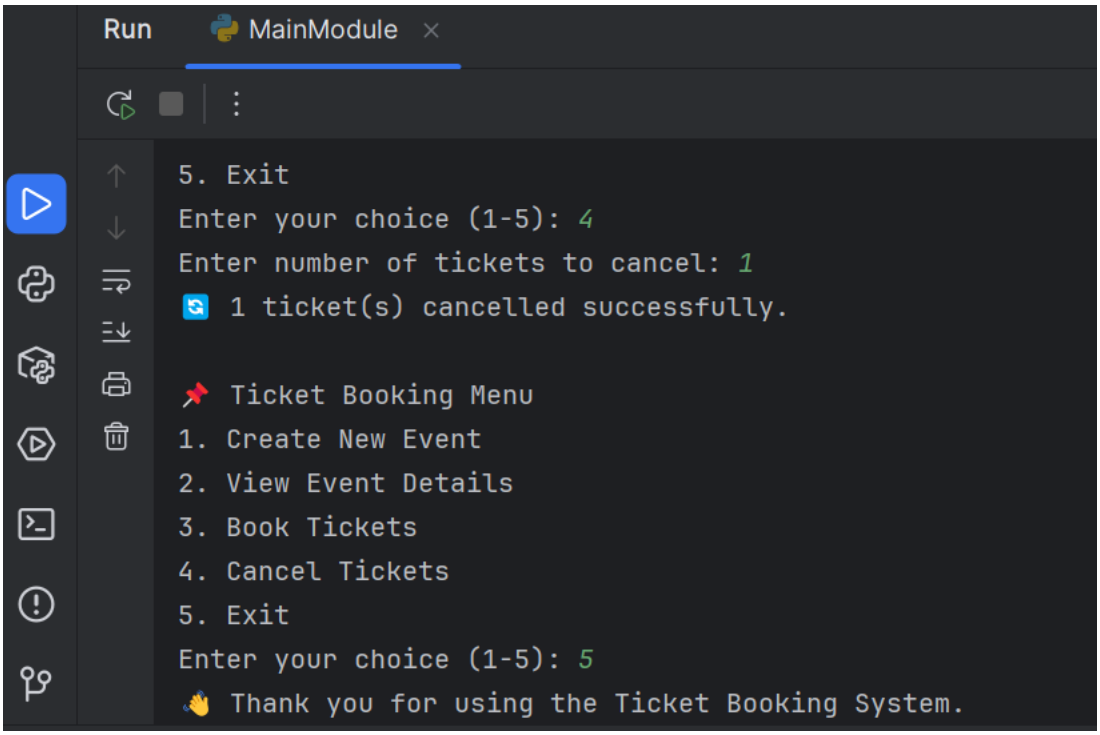
Done by
Dhamini machireddy





Done by
Dhamini machireddy





Done by
Dhamini machireddy

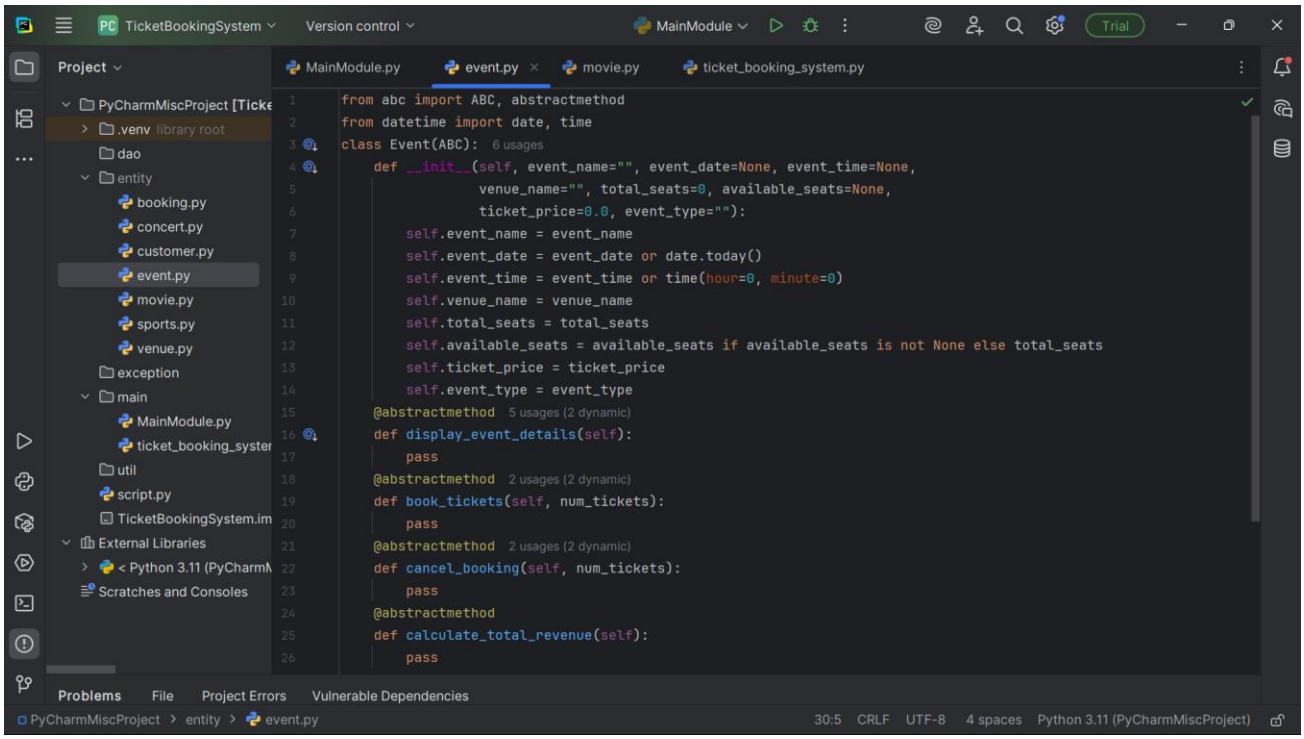
Task 6: Abstraction Requirements:

1. **Event Abstraction:**
    - Create an abstract class **Event** that represents a generic event. It should include the following attributes and methods as mentioned in *TASK 1*:

Code :

```
from abc import ABC, abstractmethod
from datetime import date, time

class Event(ABC):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type=""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue_name = venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass

    @abstractmethod
    def calculate_total_revenue(self):
        pass

    @abstractmethod
    def get_booked_tickets(self):
        pass
```

Done by
Dhamini machireddy



2. Concrete **Event** Classes**:**

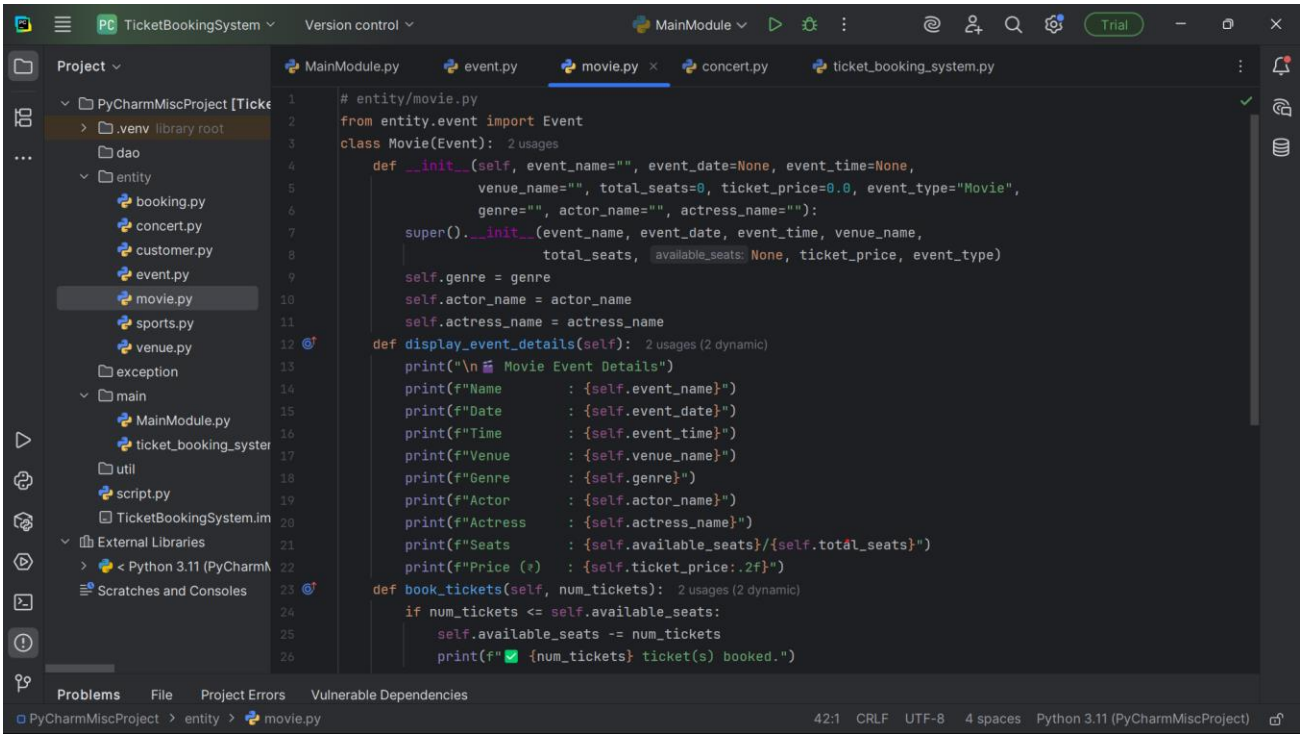
- Create three concrete classes that inherit from **Event** abstract class and override abstract methods in concrete class should declare the variables as mentioned in above *Task 2*:
- Movie.

Code :

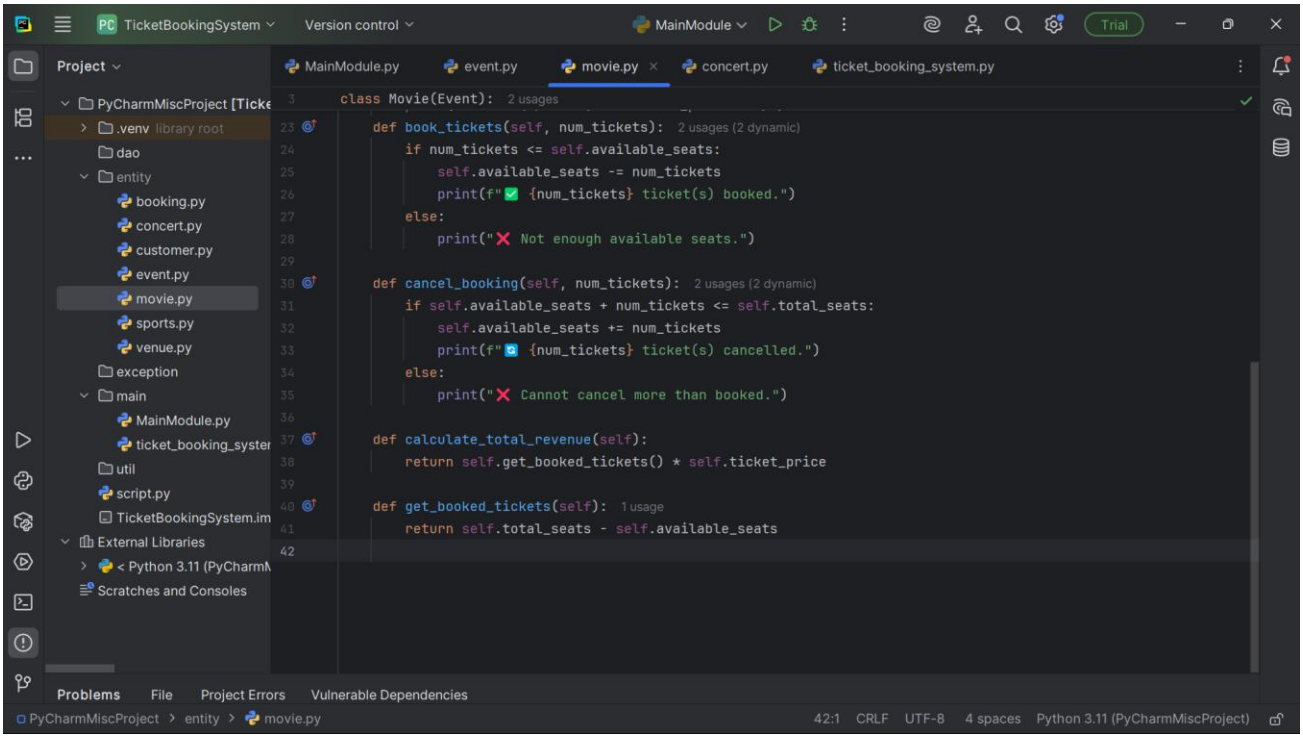
```
from entity.event import Event

class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time, venue_name,
                         total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def display_event_details(self):
        print("\n🎬 Movie Event Details")
        print(f"Name       : {self.event_name}")
        print(f"Date       : {self.event_date}")
        print(f"Time       : {self.event_time}")
        print(f"Venue      : {self.venue_name}")
        print(f"Genre      : {self.genre}")
```

Done by
Dhamini machireddy

```
        print(f"Actor      : {self.actor_name}")
        print(f"Actress    : {self.actress_name}")
        print(f"Seats      : {self.available_seats}/{self.total_seats}")
        print(f"Price (₹)  : {self.ticket_price:.2f}")

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"☑ {num_tickets} ticket(s) booked.")
        else:
            print("❌ Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"🔄 {num_tickets} ticket(s) cancelled.")
        else:
            print("❌ Cannot cancel more than booked.")

    def calculate_total_revenue(self):
        return self.get_booked_tickets() * self.ticket_price

    def get_booked_tickets(self):
        return self.total_seats - self.available_seats
```



Done by
Dhamini machireddy



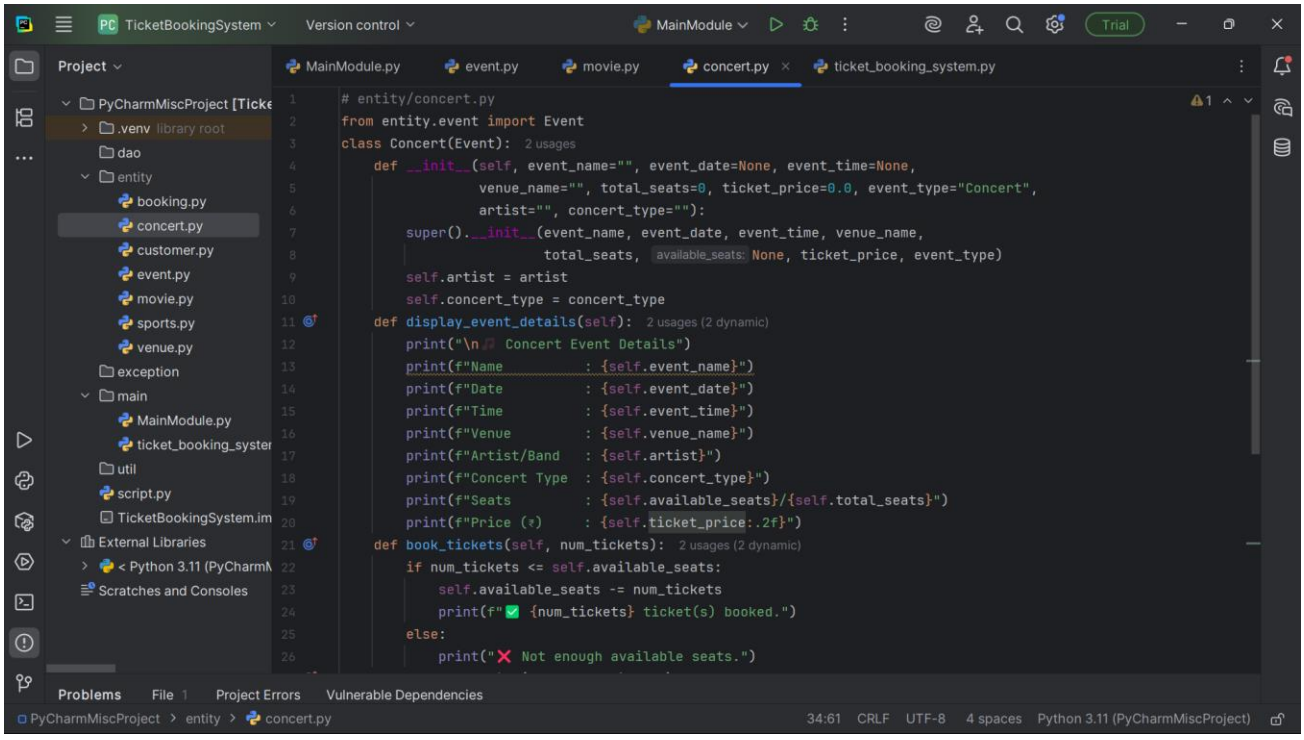
- Concert.

Code :

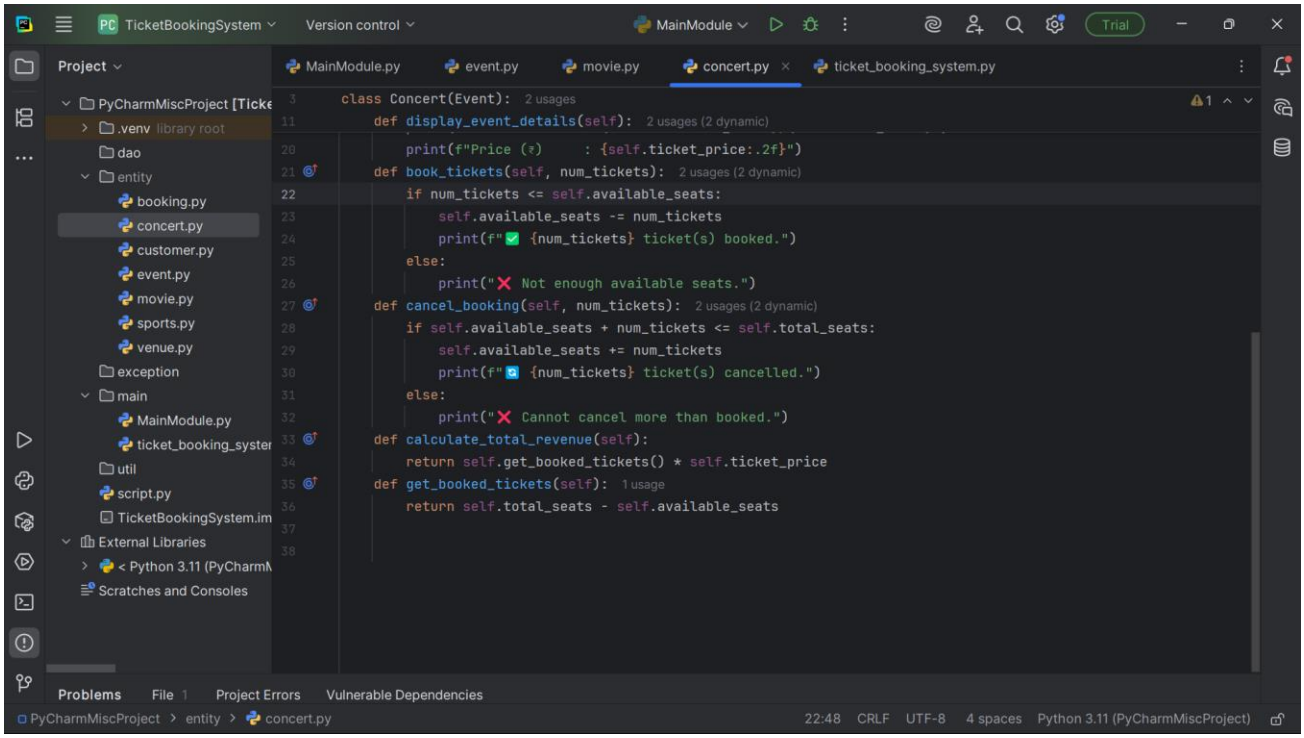
```
from entity.event import Event

class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time, venue_name,
                         total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        print("\n🎵 Concert Event Details")
        print(f"Name         : {self.event_name}")
        print(f"Date         : {self.event_date}")
        print(f"Time         : {self.event_time}")
        print(f"Venue        : {self.venue_name}")
        print(f"Artist/Band  : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
        print(f"Seats        : {self.available_seats}/{self.total_seats}")
        print(f"Price (₹)    : {self.ticket_price:.2f}")
```

Done by
Dhamini machireddy

```
def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"☑ {num_tickets} ticket(s) booked.")
    else:
        print("❌ Not enough available seats.")

def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"🔄 {num_tickets} ticket(s) cancelled.")
    else:
        print("❌ Cannot cancel more than booked.")

def calculate_total_revenue(self):
    return self.get_booked_tickets() * self.ticket_price

def get_booked_tickets(self):
    return self.total_seats - self.available_seats
```



Done by
Dhamini machireddy



- Sport.

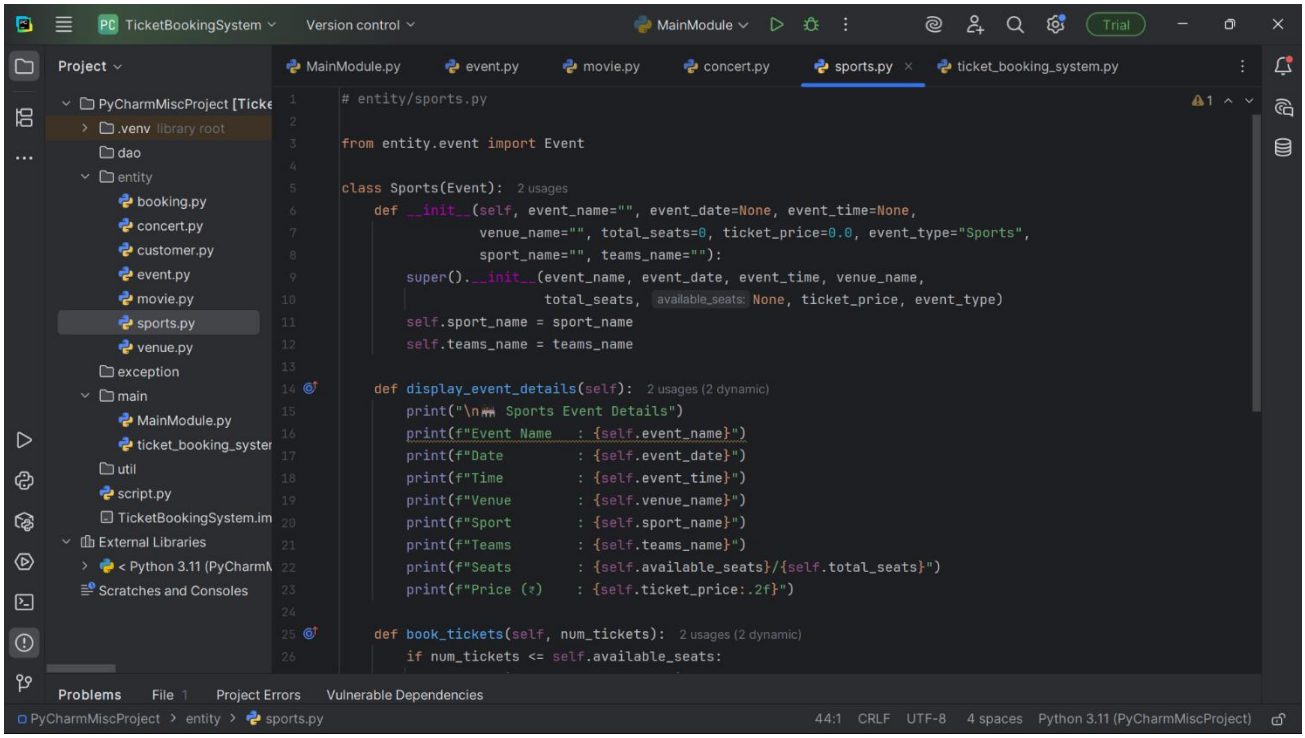
Code :

```
# entity/sports.py

from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time, venue_name,
                         total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        print("\n🏟 Sports Event Details")
        print(f"Event Name   : {self.event_name}")
        print(f"Date         : {self.event_date}")
        print(f"Time         : {self.event_time}")
        print(f"Venue        : {self.venue_name}")
        print(f"Sport        : {self.sport_name}")
        print(f"Teams        : {self.teams_name}")
```

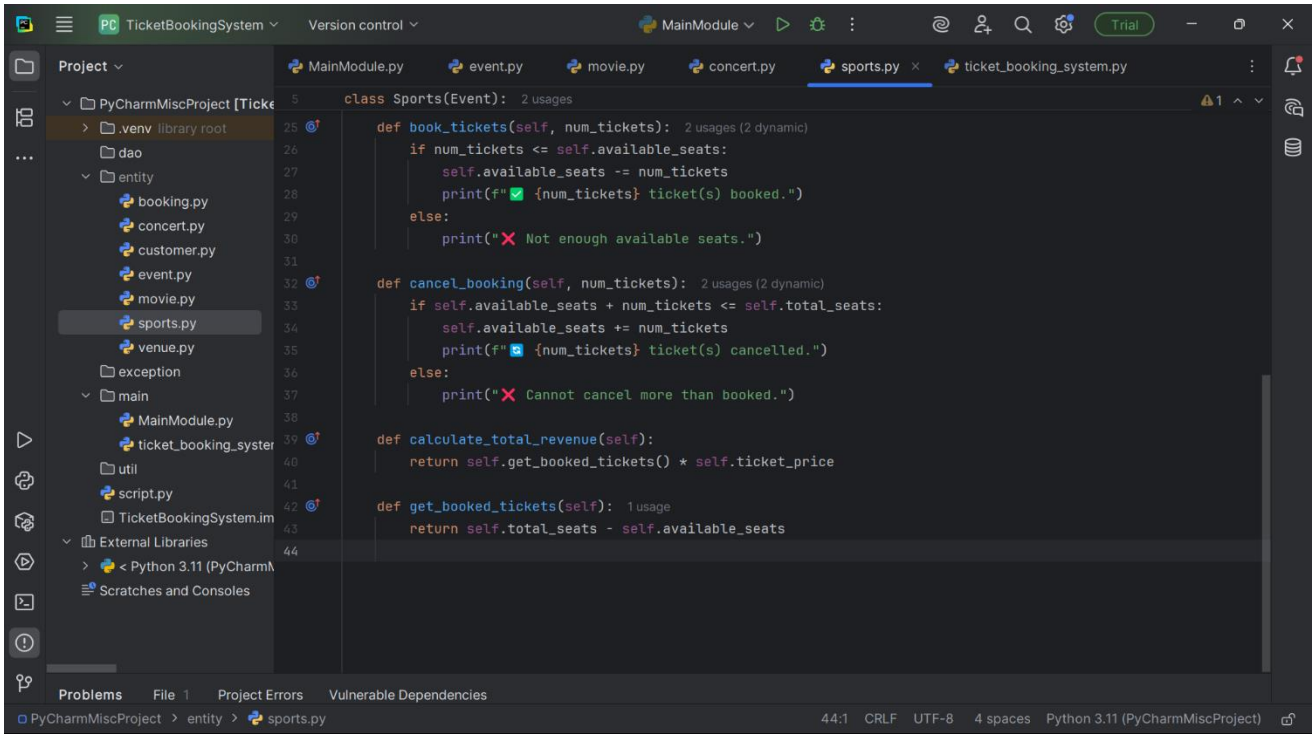
Done by
Dhamini machireddy

```
    print(f"Seats       : {self.available_seats}/{self.total_seats}")
    print(f"Price (₹)   : {self.ticket_price:.2f}")

def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"☑ {num_tickets} ticket(s) booked.")
    else:
        print("❌ Not enough available seats.")

def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"🔄 {num_tickets} ticket(s) cancelled.")
    else:
        print("❌ Cannot cancel more than booked.")

def calculate_total_revenue(self):
    return self.get_booked_tickets() * self.ticket_price

def get_booked_tickets(self):
    return self.total_seats - self.available_seats
```

Done by
Dhamini machireddy



```
# entity/sports.py

from entity.event import Event

class Sports(Event):  2 usages
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue_name="", total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time, venue_name,
                         total_seats, available_seats: None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):  2 usages (2 dynamic)
        print("\n🏏 Sports Event Details")
        print(f"Event Name   : {self.event_name}")
        print(f"Date         : {self.event_date}")
        print(f"Time         : {self.event_time}")
        print(f"Venue        : {self.venue_name}")
        print(f"Sport        : {self.sport_name}")
        print(f"Teams        : {self.teams_name}")
        print(f"Seats        : {self.available_seats}/{self.total_seats}")
        print(f"Price (₹)    : {self.ticket_price:.2f}")

    def book_tickets(self, num_tickets):  2 usages (2 dynamic)
        if num_tickets <= self.available_seats:
```



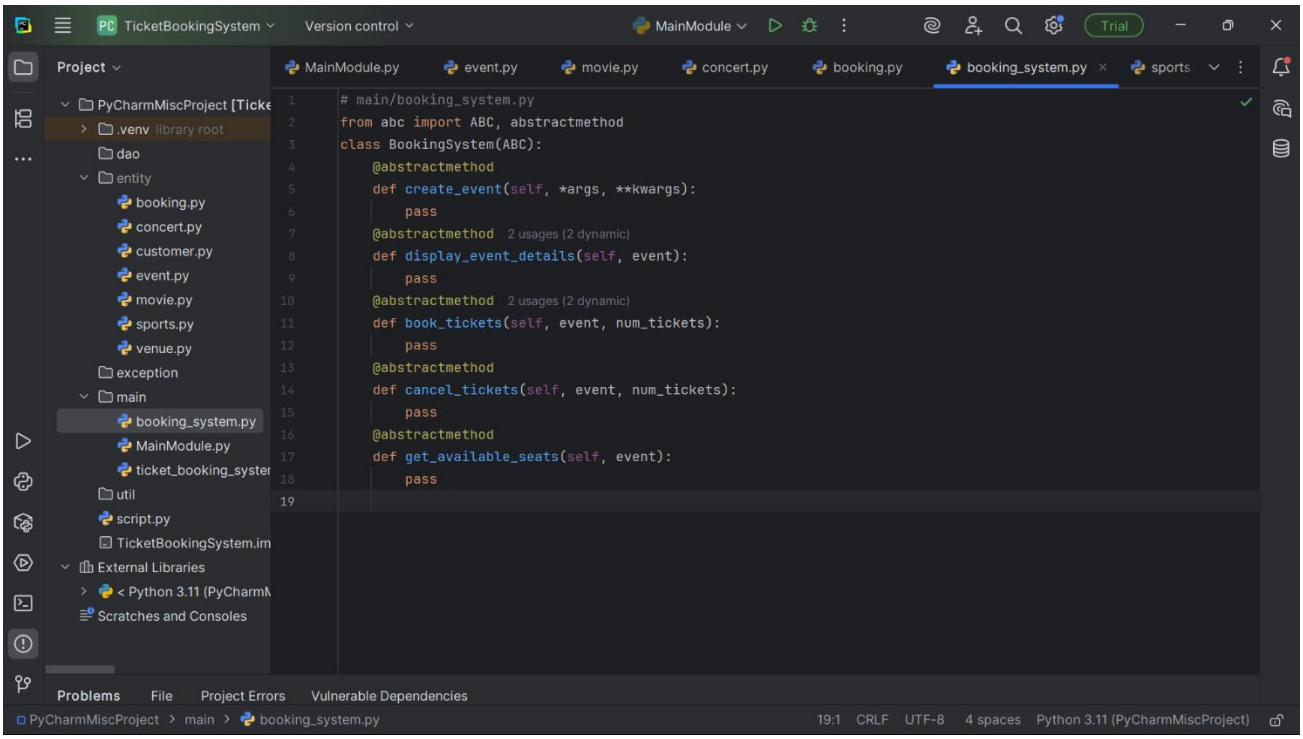
```
class Sports(Event):  2 usages
    def book_tickets(self, num_tickets):  2 usages (2 dynamic)
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"✅ {num_tickets} ticket(s) booked.")
        else:
            print("❌ Not enough available seats.")

    def cancel_booking(self, num_tickets):  2 usages (2 dynamic)
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"🔄 {num_tickets} ticket(s) cancelled.")
        else:
            print("❌ Cannot cancel more than booked.")

    def calculate_total_revenue(self):
        return self.get_booked_tickets() * self.ticket_price

    def get_booked_tickets(self):  1 usage
        return self.total_seats - self.available_seats
```

Done by
Dhamini machireddy

3. **BookingSystem** Abstraction:
   - Create an abstract class **BookingSystem** that represents the ticket booking system. It should include the methods of TASK 2 **TicketBookingSystem**:

Code :

```
# main/booking_system.py
from abc import ABC, abstractmethod
class BookingSystem(ABC):
    @abstractmethod
    def create_event(self, *args, **kwargs):
        pass
    @abstractmethod
    def display_event_details(self, event):
        pass
    @abstractmethod
    def book_tickets(self, event, num_tickets):
        pass
    @abstractmethod
    def cancel_tickets(self, event, num_tickets):
        pass
    @abstractmethod
    def get_available_seats(self, event):
        pass
```



Done by
Dhamini machireddy

4. Concrete **TicketBookingSystem** Class:
   - Create a concrete class **TicketBookingSystem** that inherits from BookingSystem:
   - **TicketBookingSystem**: Implement the abstract methods to create events, book tickets, and retrieve available seats. Maintain an array of events in this class.
   - Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats," and "exit."

   Code :

```
# main/ticket_booking_system.py

from main.booking_system import BookingSystem
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime


class TicketBookingSystem(BookingSystem):

    def __init__(self):
        self.events = []  # Store all created events

    def create_event(self, event_name, date_str, time_str, venue_name, total_seats,
                     ticket_price, event_type):
        event_date = datetime.strptime(date_str, "%Y-%m-%d").date()
        event_time = datetime.strptime(time_str, "%H:%M").time()

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue_name,
                          total_seats, ticket_price, "Movie", genre, actor, actress)

        elif event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type (Rock, Classical, etc.): ")
            event = Concert(event_name, event_date, event_time, venue_name,
                            total_seats, ticket_price, "Concert", artist, concert_type)

        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
```

Done by
Dhamini machireddy

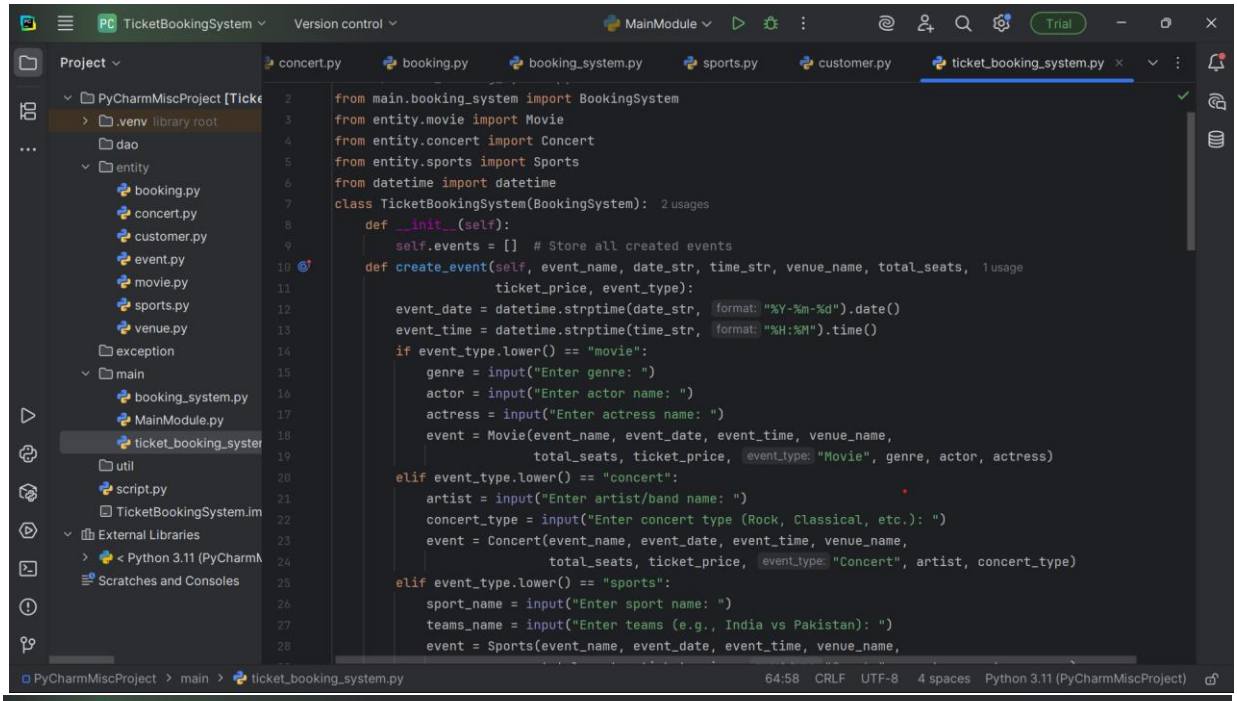
```
            teams_name = input("Enter teams (e.g., India vs Pakistan): ")
            event = Sports(event_name, event_date, event_time, venue_name,
                           total_seats, ticket_price, "Sports", sport_name, teams_name)

        else:
            print("❌ Invalid event type!")
            return None

        self.events.append(event)
        print("☑ Event created successfully!")
        return event

    def display_event_details(self, event):
        event.display_event_details()

    def book_tickets(self, event, num_tickets):
        event.book_tickets(num_tickets)

    def cancel_tickets(self, event, num_tickets):
        event.cancel_booking(num_tickets)

    def get_available_seats(self, event):
        return event.available_seats

    def run(self):
        while True:
            print("\n📌 Ticket Booking Menu")
            print("Commands: create_event, list_events, book_tickets, cancel_tickets,
get_available_seats, exit")
            command = input("Enter command: ").strip().lower()

            if command == "create_event":
                name = input("Enter event name: ")
                date_str = input("Enter event date (YYYY-MM-DD): ")
                time_str = input("Enter event time (HH:MM): ")
                venue = input("Enter venue name: ")
                seats = int(input("Enter total seats: "))
                price = float(input("Enter ticket price: "))
                e_type = input("Enter event type (Movie/Concert/Sports): ")

                self.create_event(name, date_str, time_str, venue, seats, price, e_type)
```

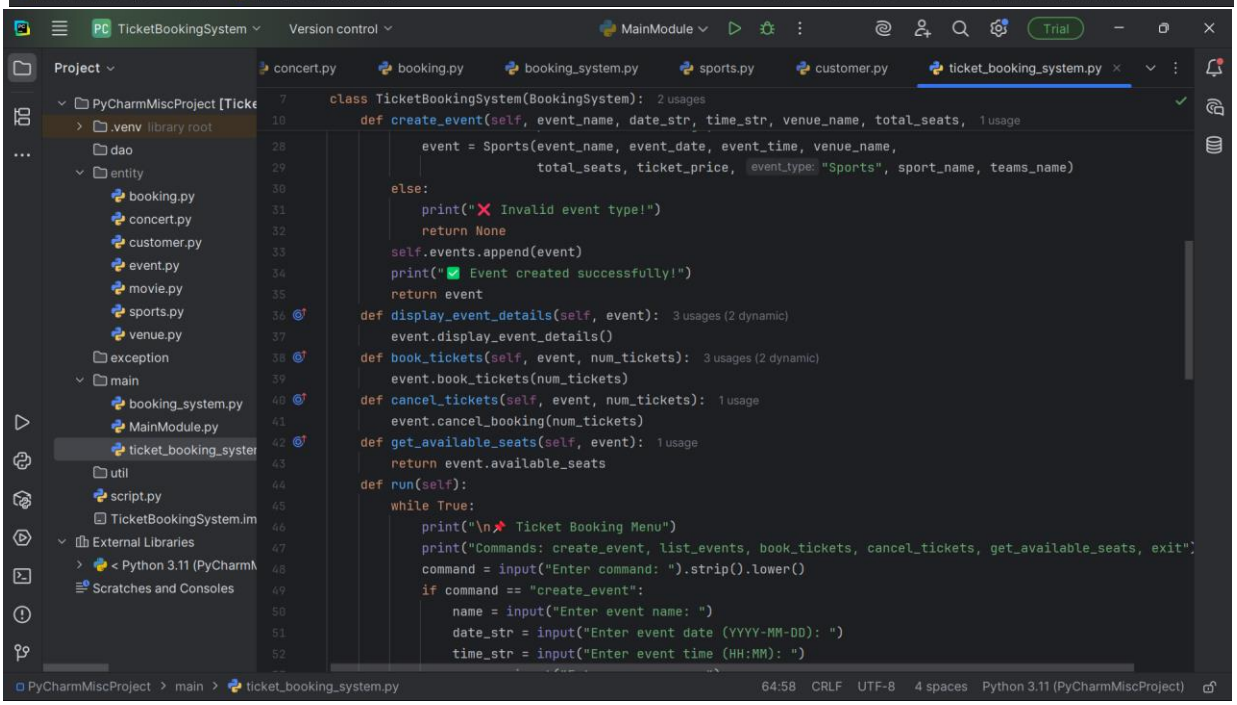
Done by
Dhamini machireddy

```
elif command == "list_events":
    if not self.events:
        print("⚠ No events available.")
    else:
        for idx, event in enumerate(self.events):
            print(f"\nEvent #{idx + 1}")
            self.display_event_details(event)

elif command == "book_tickets":
    if not self.events:
        print("⚠ No events to book.")
        continue
    event_id = int(input("Enter event number to book tickets: ")) - 1
    if 0 <= event_id < len(self.events):
        count = int(input("Enter number of tickets: "))
        self.book_tickets(self.events[event_id], count)
    else:
        print("❌ Invalid event number.")

elif command == "cancel_tickets":
    if not self.events:
        print("⚠ No events to cancel.")
        continue
    event_id = int(input("Enter event number to cancel tickets: ")) - 1
    if 0 <= event_id < len(self.events):
        count = int(input("Enter number of tickets to cancel: "))
        self.cancel_tickets(self.events[event_id], count)
    else:
        print("❌ Invalid event number.")

elif command == "get_available_seats":
    if not self.events:
        print("⚠ No events.")
        continue
    event_id = int(input("Enter event number: ")) - 1
    if 0 <= event_id < len(self.events):
        seats = self.get_available_seats(self.events[event_id])
        print(f"🎟 Available Seats: {seats}")
    else:
        print("❌ Invalid event number.")
```

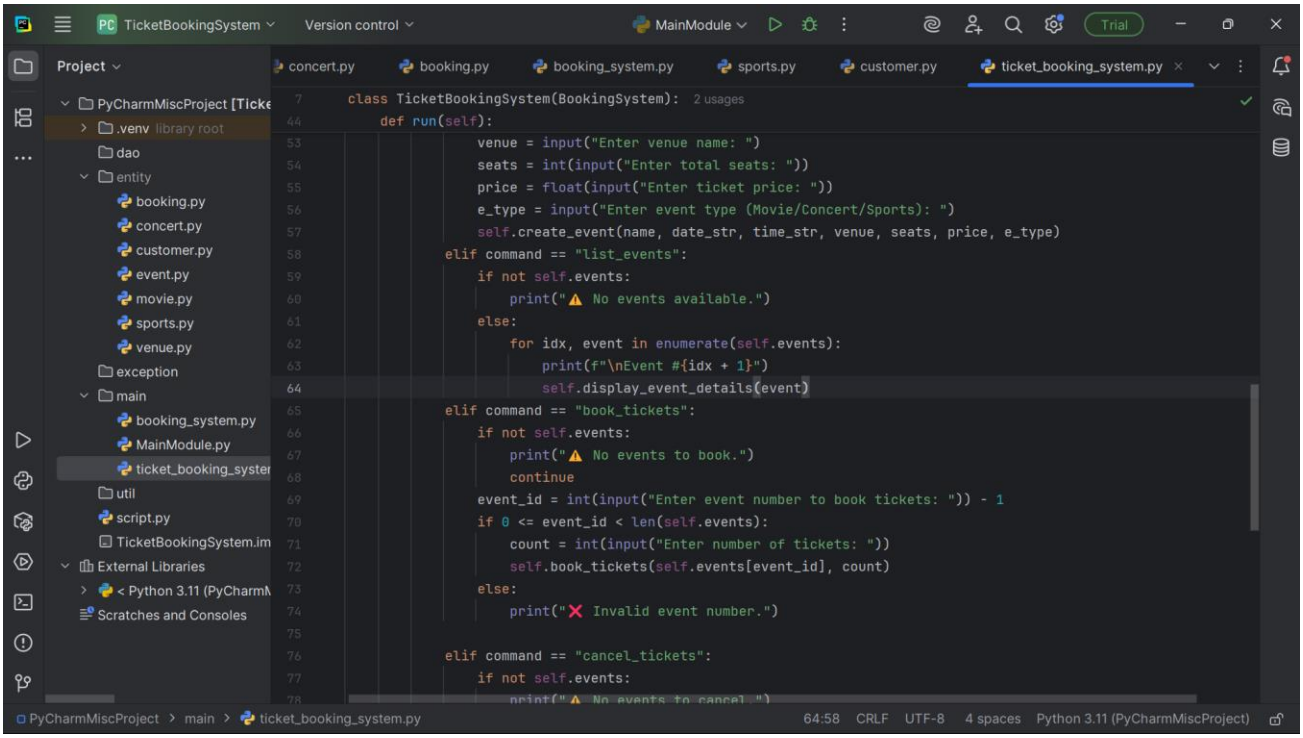
Done by
Dhamini machireddy

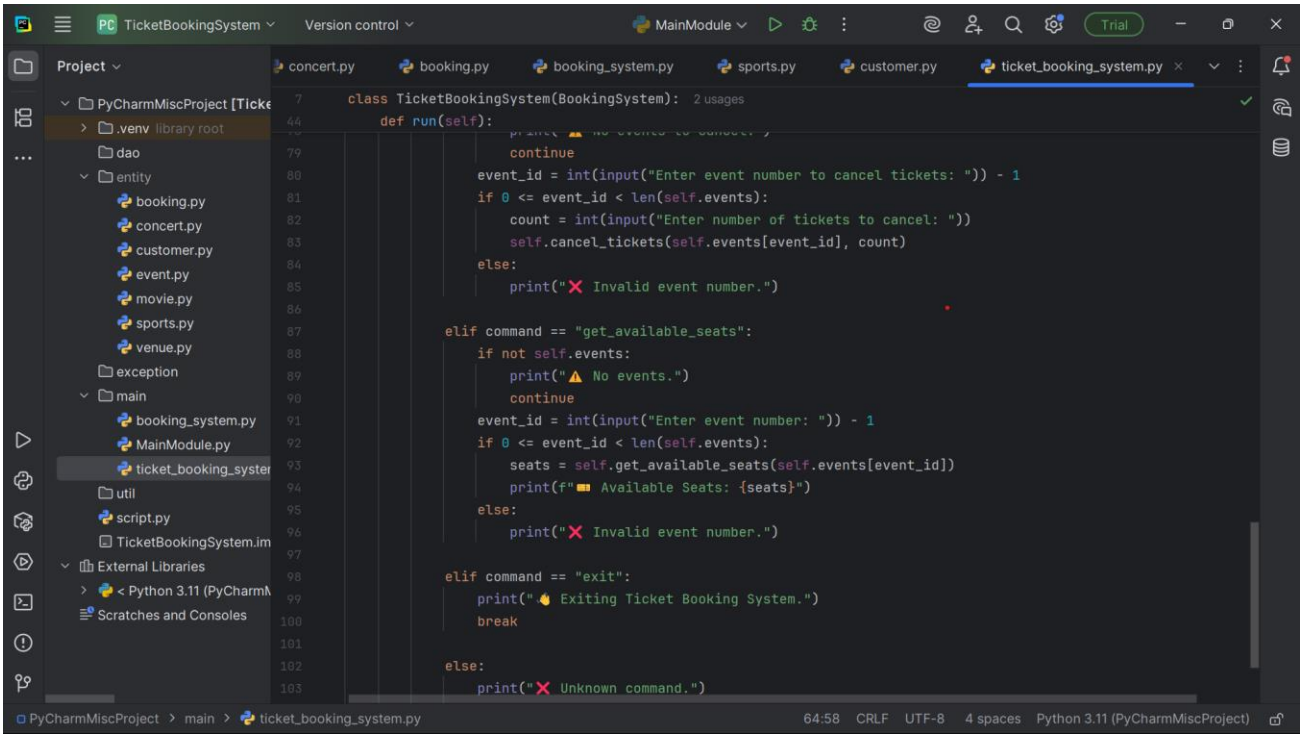
```
elif command == "exit":
    print("👋 Exiting Ticket Booking System.")
    break

else:
    print("❌ Unknown command.")
```





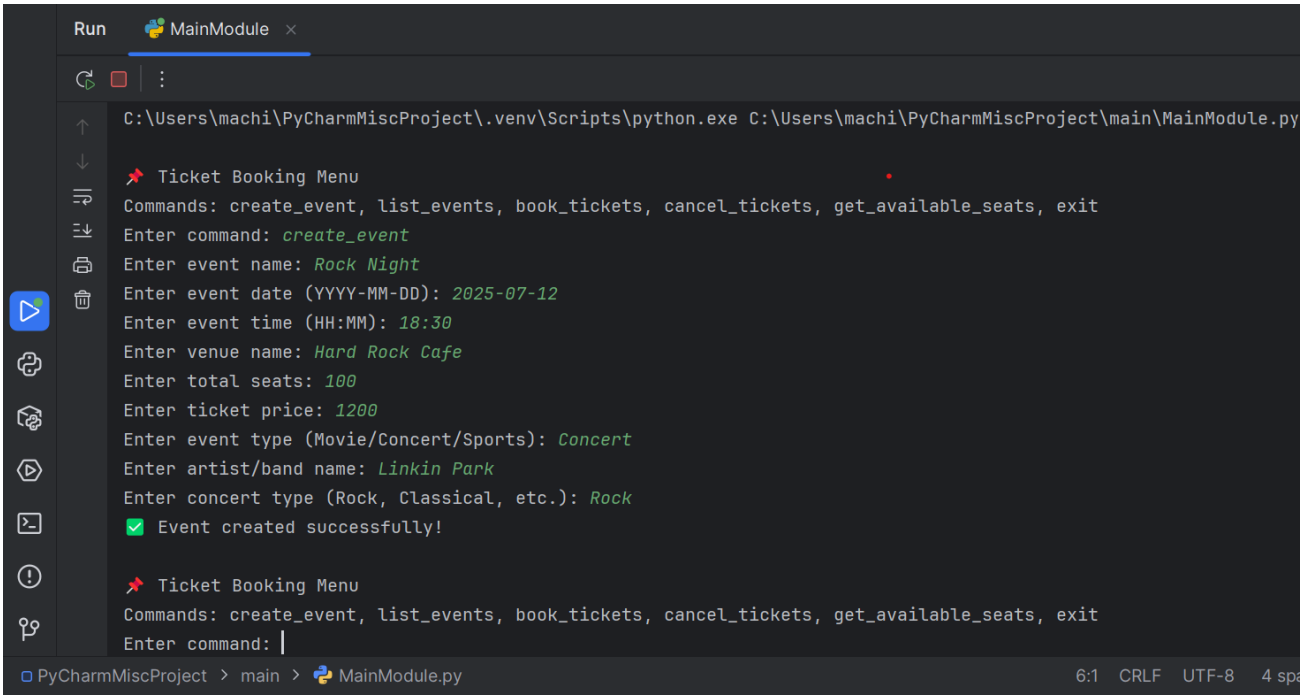
Done by
Dhamini machireddy





Output:

Done by
Dhamini machireddy



Task 7: Has A Relation / Association

Create a Following classes with the following attributes and methods:

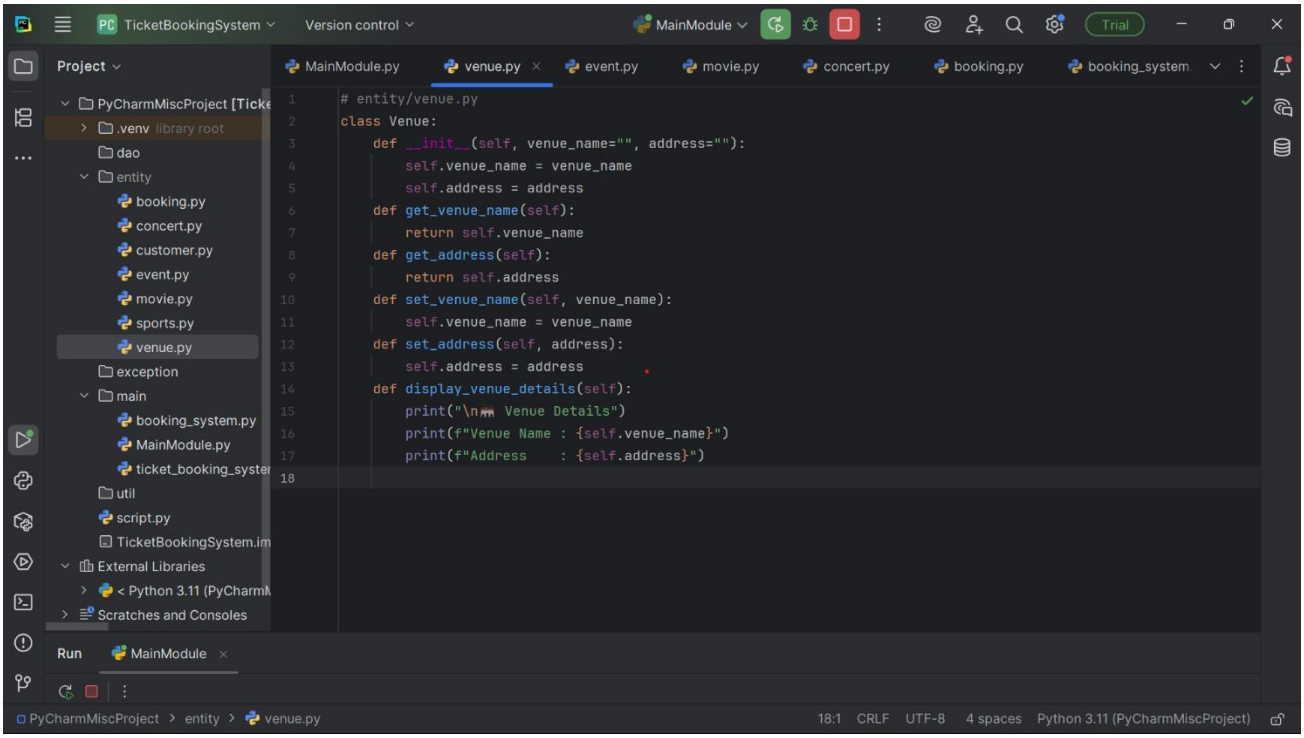
1. **Venue** Class
   - **Attributes**:
   - venue_name,
   - address
   - **Methods and Constuctors:**
   - **display_venue_details():** Display venue details.
   - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.

Code :

```
# entity/venue.py
class Venue:
    def __init__(self, venue_name="", address=""):
        self.venue_name = venue_name
        self.address = address
    def get_venue_name(self):
        return self.venue_name
    def get_address(self):
        return self.address
    def set_venue_name(self, venue_name):
        self.venue_name = venue_name
    def set_address(self, address):
        self.address = address
    def display_venue_details(self):
```

Done by
Dhamini machireddy

```
print("\n🏟 Venue Details")
print(f"Venue Name : {self.venue_name}")
print(f"Address    : {self.address}")
```



2. **Event** Class**:**
    - **Attributes:**

- event_name, ○ event_date DATE, ○ event_time TIME,
- venue (reference of class **Venu**), ○ total_seats, ○ available_seats, ○ ticket_price DECIMAL,
    - event_type ENUM('Movie', 'Sports', 'Concert')
    - **Methods and Constuctors:**
- Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes. ○ **calculate_total_revenue()**: Calculate and return the total revenue based on the number of tickets sold.
- **getBookedNoOfTickets()**: return the total booked tickets
- **book_tickets(num_tickets)**: Book a specified number of tickets for an event. Initially available seats are equal to total seats when tickets are booked available seats number should be reduced.
- **cancel_booking(num_tickets)**: Cancel the booking and update the available seats. ○ **display_event_details():** Display event details, including event name, date time seat availability.

Code :

```
# entity/event.py

from abc import ABC, abstractmethod
```

Done by
Dhamini machireddy

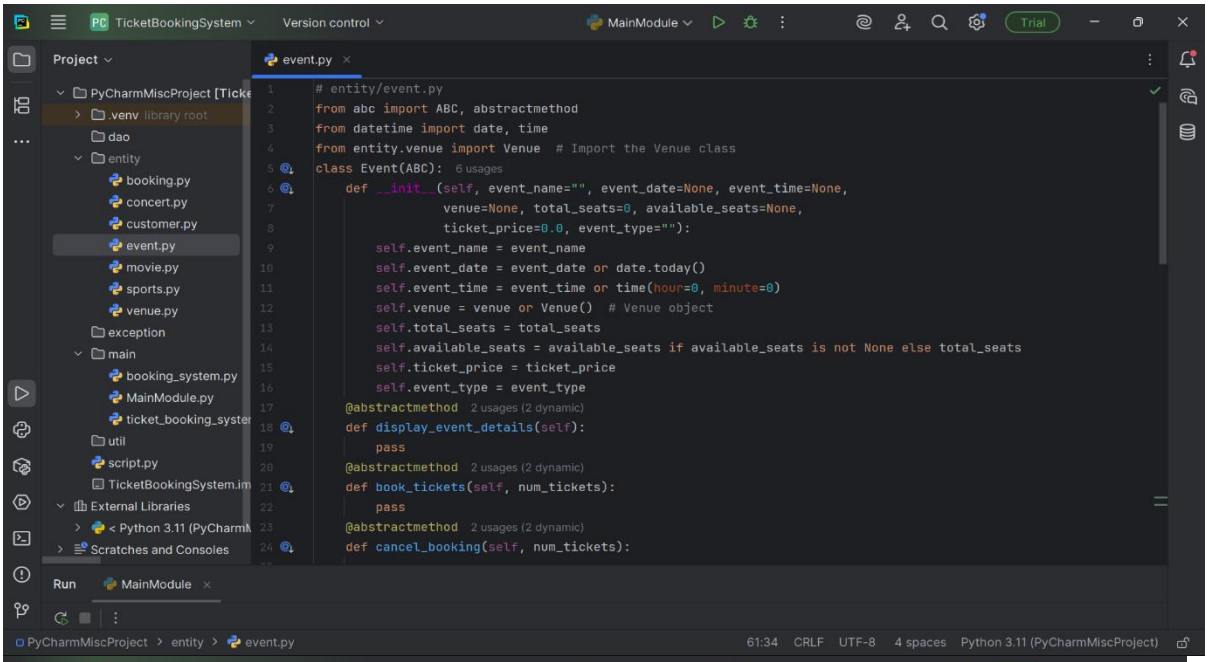
```
from datetime import date, time
from entity.venue import Venue  # Import the Venue class

class Event(ABC):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type=""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue = venue or Venue()  # Venue object
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # -------- Abstract Methods --------
    @abstractmethod
    def display_event_details(self):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, num_tickets):
        pass

    @abstractmethod
    def calculate_total_revenue(self):
        pass

    @abstractmethod
    def get_booked_tickets(self):
        pass

    # -------- Getters --------
    def get_event_name(self):
        return self.event_name
```

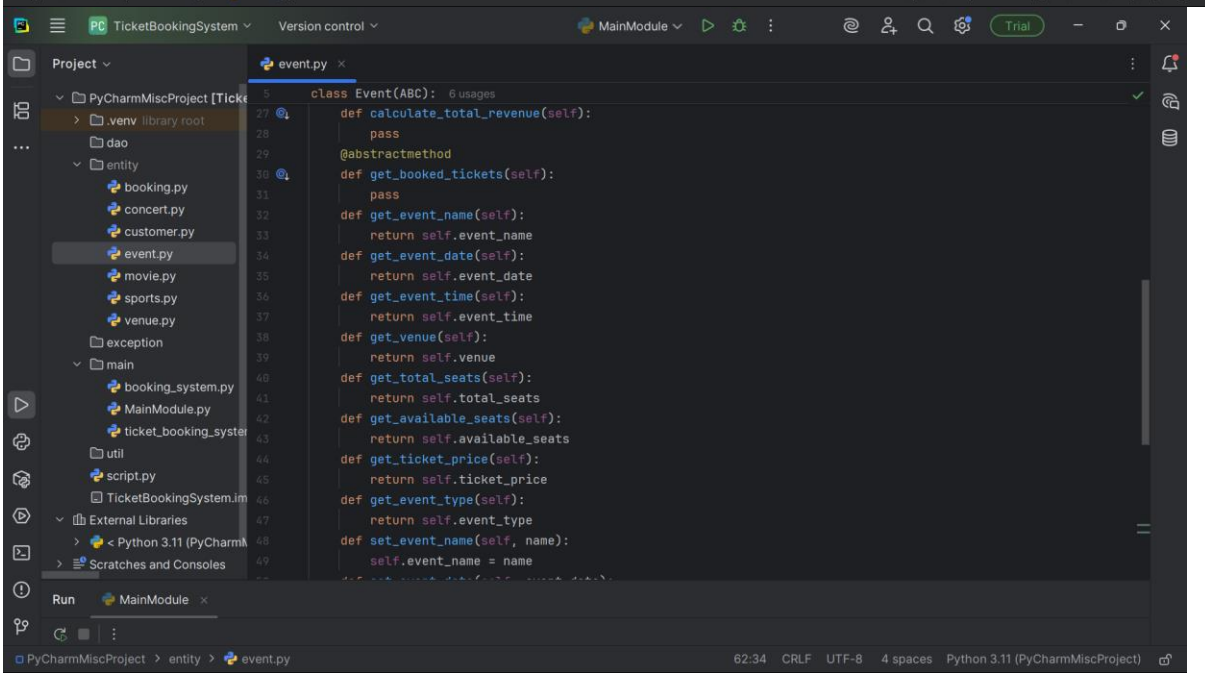
Done by
Dhamini machireddy

```
    def get_event_date(self):
        return self.event_date

    def get_event_time(self):
        return self.event_time

    def get_venue(self):
        return self.venue

    def get_total_seats(self):
        return self.total_seats

    def get_available_seats(self):
        return self.available_seats

    def get_ticket_price(self):
        return self.ticket_price

    def get_event_type(self):
        return self.event_type

    # -------- Setters --------
    def set_event_name(self, name):
        self.event_name = name

    def set_event_date(self, event_date):
        self.event_date = event_date

    def set_event_time(self, event_time):
        self.event_time = event_time

    def set_venue(self, venue):
        self.venue = venue

    def set_total_seats(self, seats):
        self.total_seats = seats
    def set_available_seats(self, seats):
        self.available_seats = seats
    def set_ticket_price(self, price):
```

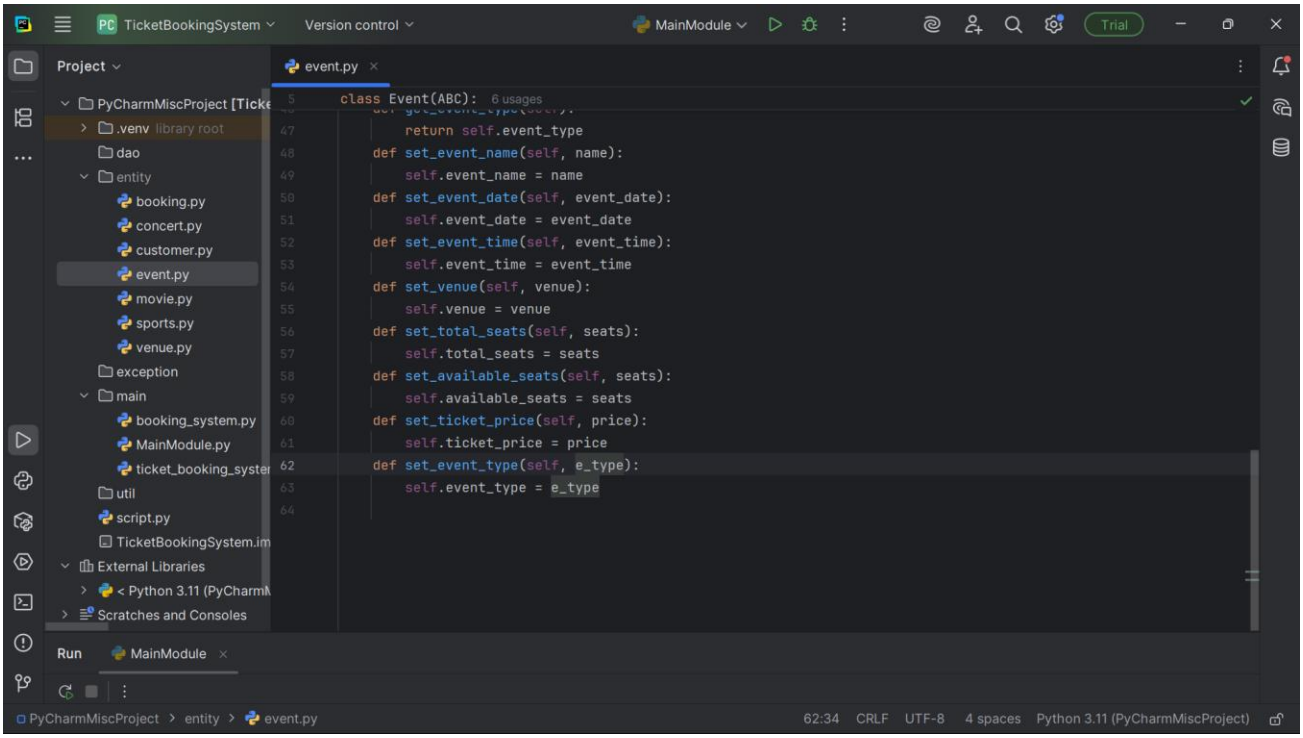
Done by
Dhamini machireddy

```
    self.ticket_price = price
def set_event_type(self, e_type):
    self.event_type = e_type
```





Done by
Dhamini machireddy



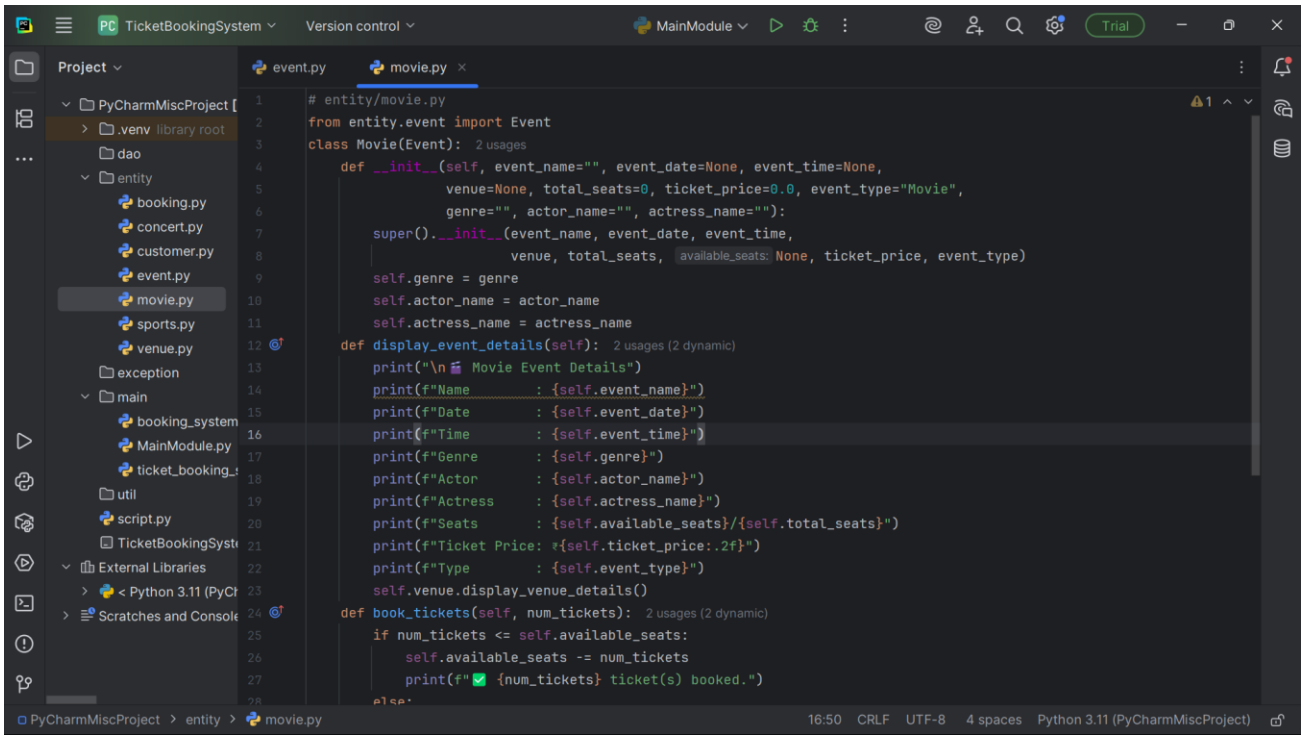
3. **Event** sub classes**:**
   - Create three sub classes that inherit from **Event** abstract class and override abstract methods in concrete class should declare the variables as mentioned in above
   - *Task 2*:
     - Movie.

     Code :

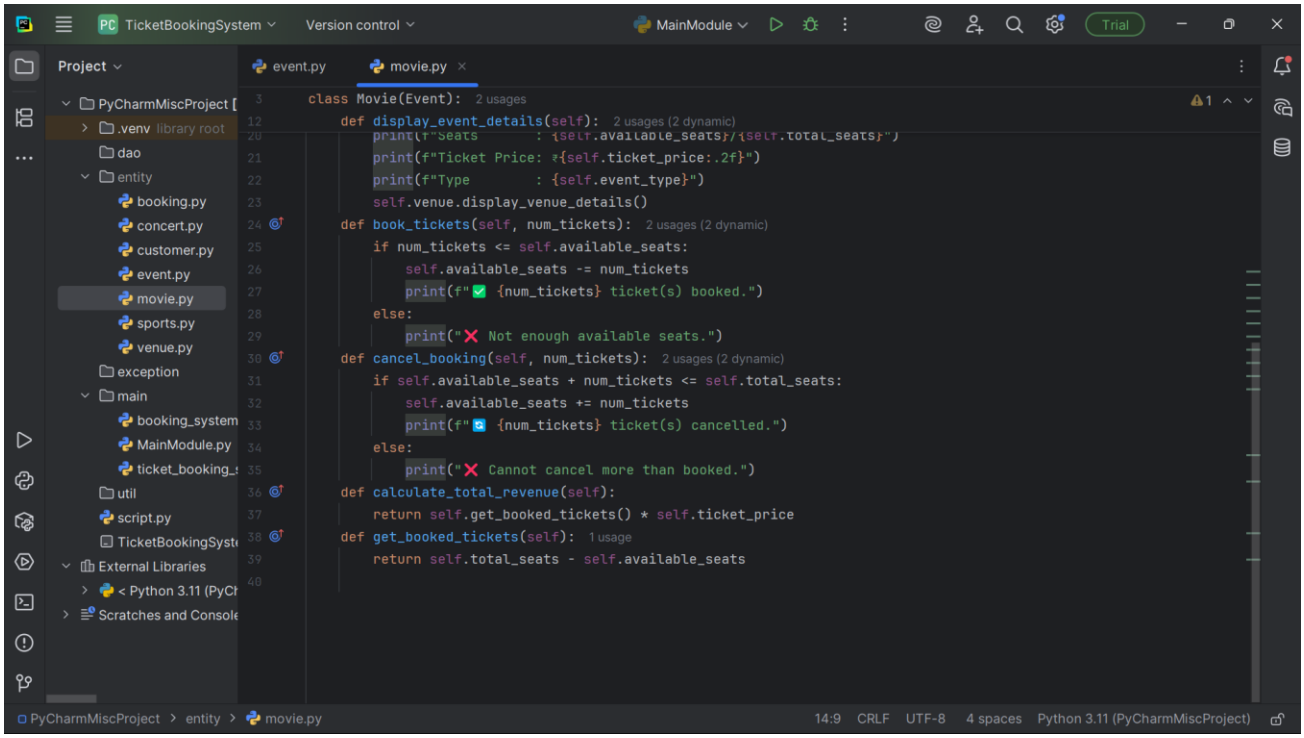
```
# entity/movie.py
from entity.event import Event
class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name
    def display_event_details(self):
        print("\n🎬 Movie Event Details")
        print(f"Name       : {self.event_name}")
        print(f"Date       : {self.event_date}")
        print(f"Time       : {self.event_time}")
        print(f"Genre      : {self.genre}")
        print(f"Actor      : {self.actor_name}")
```

Done by
Dhamini machireddy

```
        print(f"Actress    : {self.actress_name}")
        print(f"Seats      : {self.available_seats}/{self.total_seats}")
        print(f"Ticket Price: ₹{self.ticket_price:.2f}")
        print(f"Type       : {self.event_type}")
        self.venue.display_venue_details()
    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"☑ {num_tickets} ticket(s) booked.")
        else:
            print("❌ Not enough available seats.")
    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"🔄 {num_tickets} ticket(s) cancelled.")
        else:
            print("❌ Cannot cancel more than booked.")
    def calculate_total_revenue(self):
        return self.get_booked_tickets() * self.ticket_price
    def get_booked_tickets(self):
        return self.total_seats - self.available_seats
```



Done by
Dhamini machireddy



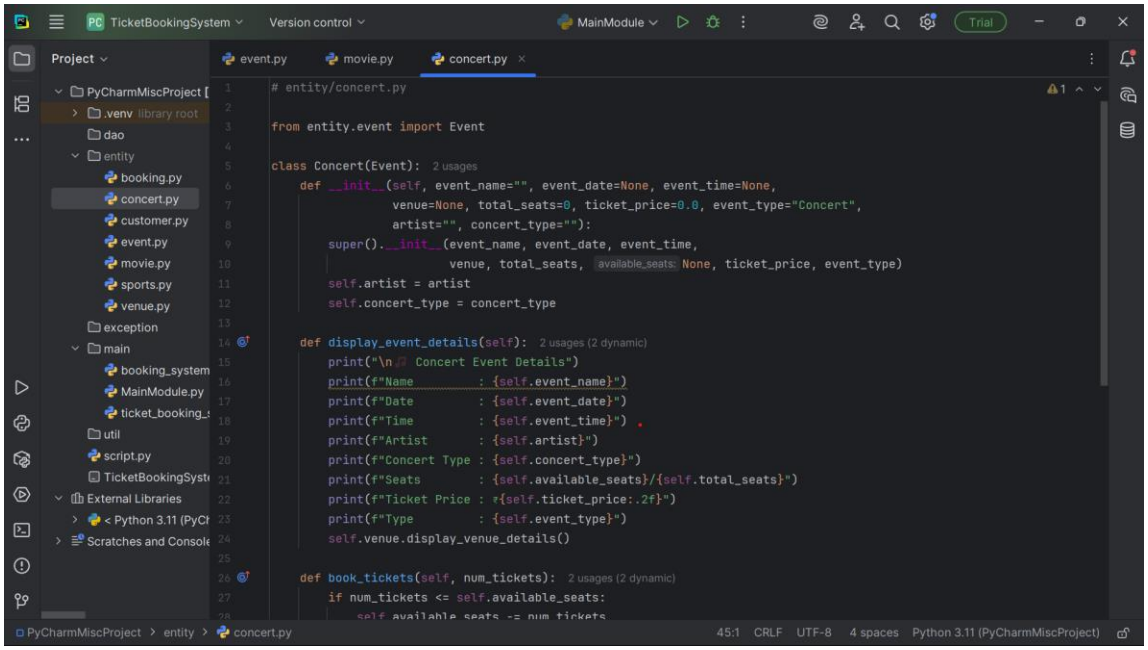
- Concert.

Code:

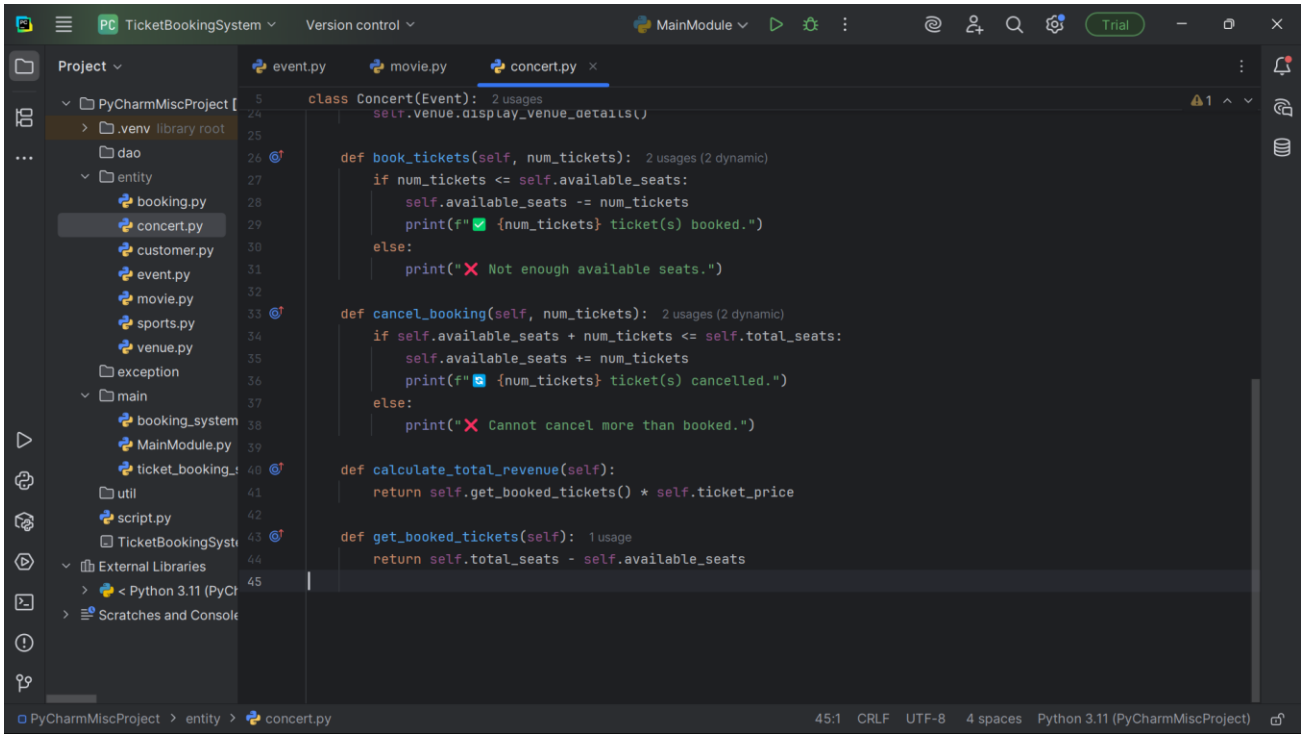
```
# entity/concert.py

from entity.event import Event

class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        print("\n🎵 Concert Event Details")
        print(f"Name        : {self.event_name}")
        print(f"Date        : {self.event_date}")
        print(f"Time        : {self.event_time}")
        print(f"Artist      : {self.artist}")
        print(f"Concert Type : {self.concert_type}")
```

Done by
Dhamini machireddy

```
    print(f"Seats       : {self.available_seats}/{self.total_seats}")
    print(f"Ticket Price : ₹{self.ticket_price:.2f}")
    print(f"Type        : {self.event_type}")
    self.venue.display_venue_details()

def book_tickets(self, num_tickets):
    if num_tickets <= self.available_seats:
        self.available_seats -= num_tickets
        print(f"☑ {num_tickets} ticket(s) booked.")
    else:
        print("❌ Not enough available seats.")

def cancel_booking(self, num_tickets):
    if self.available_seats + num_tickets <= self.total_seats:
        self.available_seats += num_tickets
        print(f"🔄 {num_tickets} ticket(s) cancelled.")
    else:
        print("❌ Cannot cancel more than booked.")

def calculate_total_revenue(self):
    return self.get_booked_tickets() * self.ticket_price

def get_booked_tickets(self):
    return self.total_seats - self.available_seats
```



Done by
Dhamini machireddy



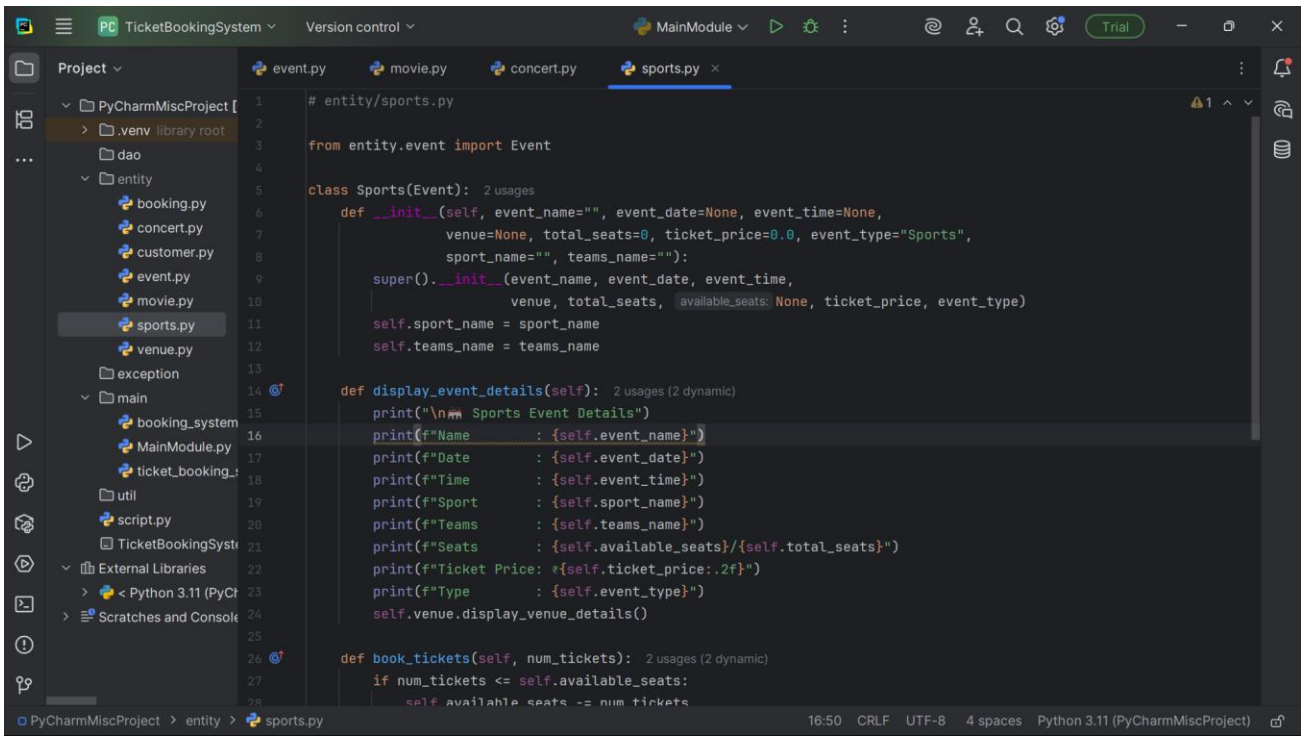
- Sport.

Code :

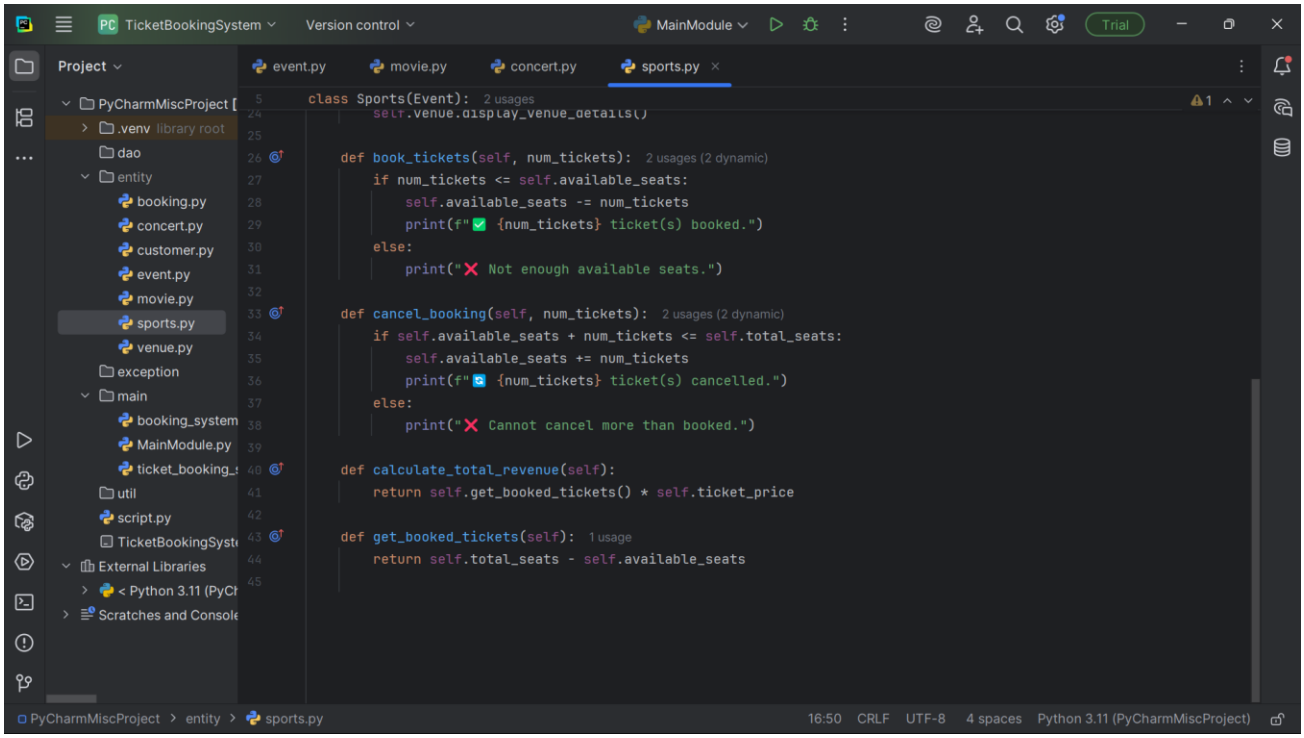
```
# entity/sports.py

from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        print("\n Sports Event Details")
        print(f"Name       : {self.event_name}")
        print(f"Date       : {self.event_date}")
        print(f"Time       : {self.event_time}")
        print(f"Sport      : {self.sport_name}")
        print(f"Teams      : {self.teams_name}")
        print(f"Seats      : {self.available_seats}/{self.total_seats}")
        print(f"Ticket Price: ₹{self.ticket_price:.2f}")
```

Done by
Dhamini machireddy

```
        print(f"Type       : {self.event_type}")
        self.venue.display_venue_details()

    def book_tickets(self, num_tickets):
        if num_tickets <= self.available_seats:
            self.available_seats -= num_tickets
            print(f"☑ {num_tickets} ticket(s) booked.")
        else:
            print("❌ Not enough available seats.")

    def cancel_booking(self, num_tickets):
        if self.available_seats + num_tickets <= self.total_seats:
            self.available_seats += num_tickets
            print(f"🔄 {num_tickets} ticket(s) cancelled.")
        else:
            print("❌ Cannot cancel more than booked.")

    def calculate_total_revenue(self):
        return self.get_booked_tickets() * self.ticket_price

    def get_booked_tickets(self):
        return self.total_seats - self.available_seats
```



Done by
Dhamini machireddy



4. **Customer** Class
    - **Attributes:**

- customer_name,
- email,
- phone_number,
    - **Methods and Constuctors:**

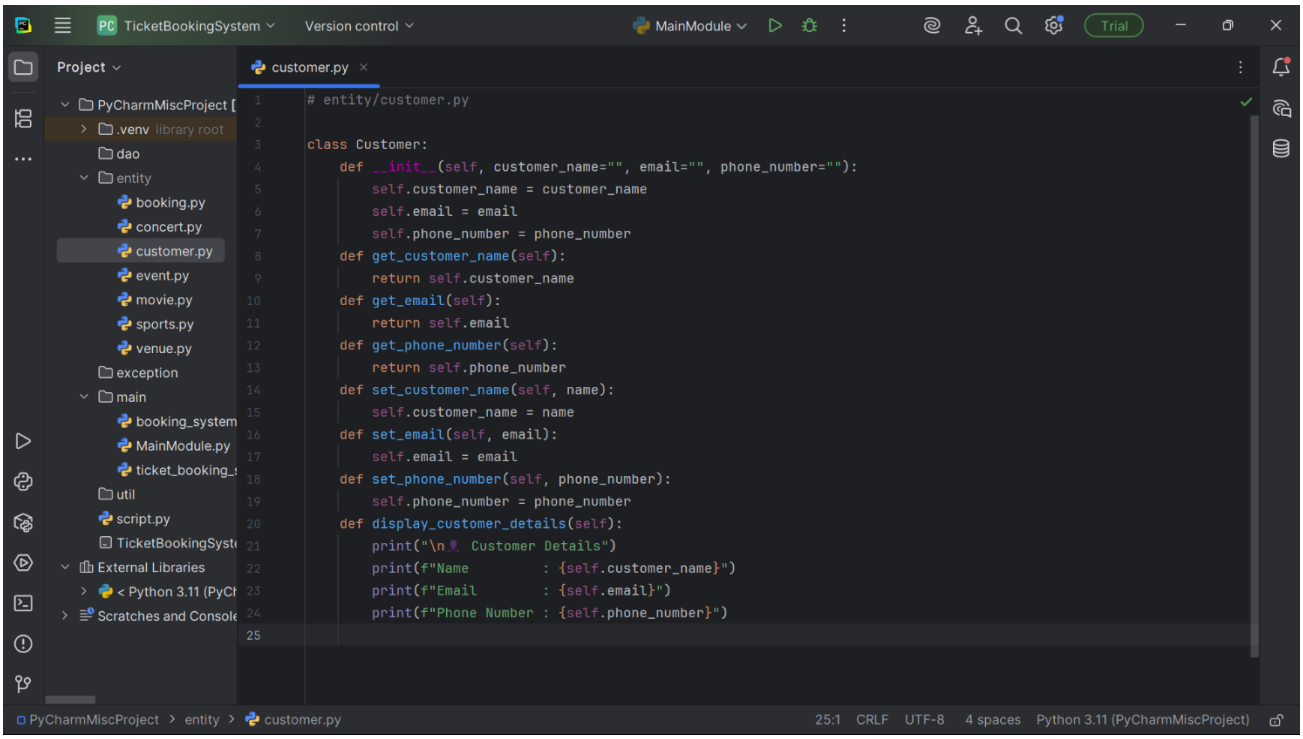
- Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
- **display_customer_details()**: Display customer details.

Code :

```
class Customer:
    def __init__(self, customer_name="", email="", phone_number=""):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number
    def get_customer_name(self):
        return self.customer_name
    def get_email(self):
        return self.email
    def get_phone_number(self):
        return self.phone_number
    def set_customer_name(self, name):
        self.customer_name = name
    def set_email(self, email):
```

Done by
Dhamini machireddy

```
        self.email = email
    def set_phone_number(self, phone_number):
        self.phone_number = phone_number
    def display_customer_details(self):
        print("\n👤 Customer Details")
        print(f"Name         : {self.customer_name}")
        print(f"Email        : {self.email}")
        print(f"Phone Number : {self.phone_number}")
```



5. Create a class **Booking** with the following attributes:
    - bookingId (should be incremented for each booking)
    - array of customer (reference to the customer who made the booking)
    - event (reference to the event booked)
    - num_tickets(no of tickets and array of customer must equal)
    - total_cost
    - booking_date (timestamp of when the booking was made)
    - **Methods and Constuctors:**

- Implement default constructors and overload the constructor with Customer attributes, generate getter and setter methods.
- **display_booking_details()**: Display customer details.

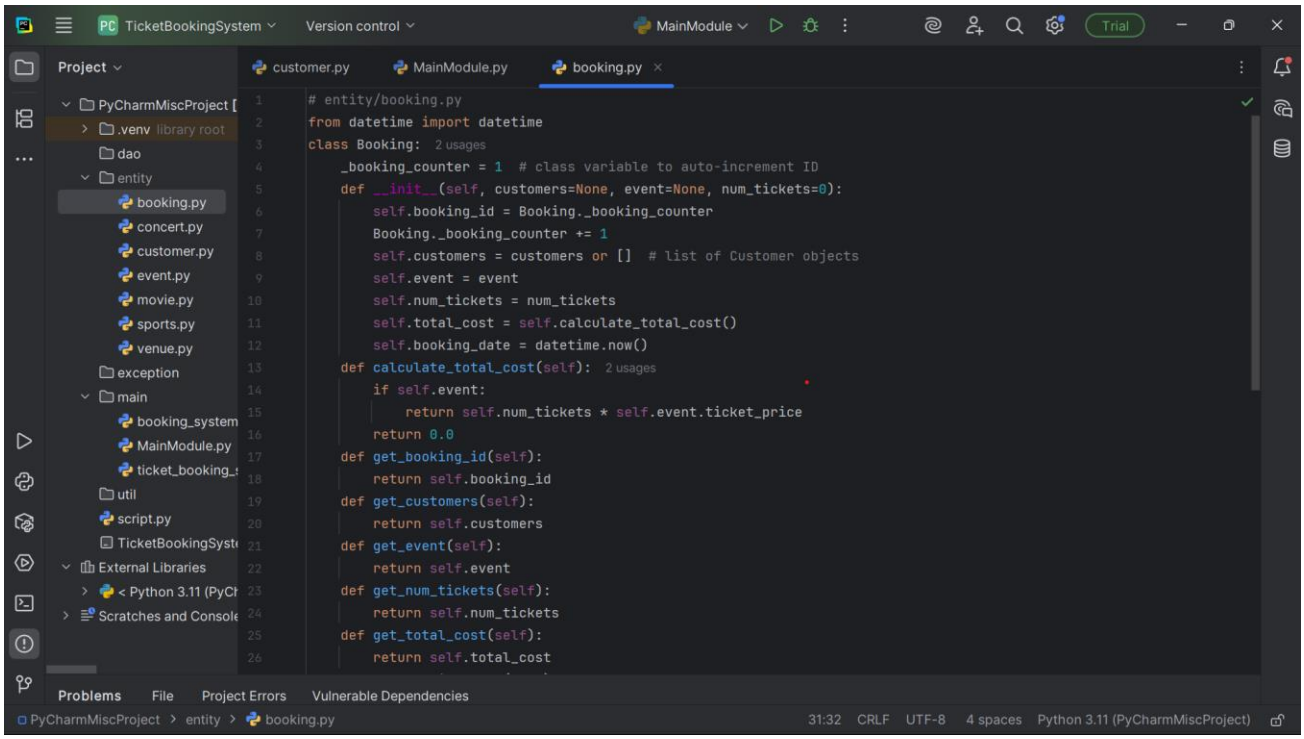
Code :

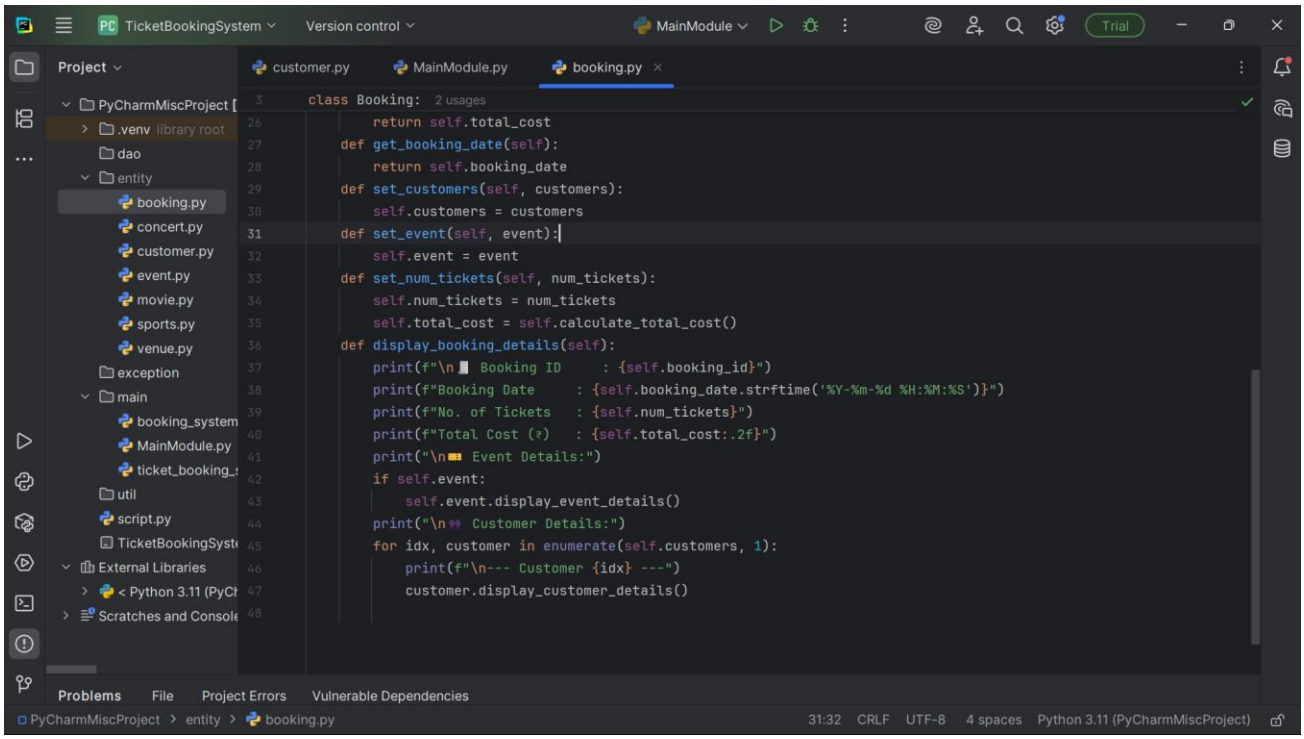
Done by
Dhamini machireddy

```
from datetime import datetime
class Booking:
    _booking_counter = 1  # class variable to auto-increment ID
    def __init__(self, customers=None, event=None, num_tickets=0):
        self.booking_id = Booking._booking_counter
        Booking._booking_counter += 1
        self.customers = customers or []  # list of Customer objects
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()
    def calculate_total_cost(self):
        if self.event:
            return self.num_tickets * self.event.ticket_price
        return 0.0
    def get_booking_id(self):
        return self.booking_id
    def get_customers(self):
        return self.customers
    def get_event(self):
        return self.event
    def get_num_tickets(self):
        return self.num_tickets
    def get_total_cost(self):
        return self.total_cost
    def get_booking_date(self):
        return self.booking_date
    def set_customers(self, customers):
        self.customers = customers
    def set_event(self, event):
        self.event = event
    def set_num_tickets(self, num_tickets):
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
    def display_booking_details(self):
        print(f"\n📄 Booking ID     : {self.booking_id}")
        print(f"Booking Date     : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"No. of Tickets   : {self.num_tickets}")
        print(f"Total Cost (₹)   : {self.total_cost:.2f}")
        print("\n🎟️ Event Details:")
        if self.event:
            self.event.display_event_details()
```

Done by
Dhamini machireddy

```
print("\n👥 Customer Details:")
for idx, customer in enumerate(self.customers, 1):
    print(f"\n--- Customer {idx} ---")
    customer.display_customer_details()
```





Done by
Dhamini machireddy

6. **BookingSystem** Class to represent the Ticket booking system. Perform the following operation in main method. Note: - Use Event class object for the following operation.
   - **Attributes** ○ array of events
   - **Methods and Constuctors:**

○ **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu:Venu):**
Create a new event with the specified details and event type (movie, sport or concert) and return event object.

- ○ **calculate_booking_cost(num_tickets)**: Calculate and set the total cost of the booking.
- ○ **book_tickets(eventname:str, num_tickets, arrayOfCustomer)**: Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of **Booking** class.
- ○ **cancel_booking(booking_id)**: Cancel the booking and update the available seats.
- ○ **getAvailableNoOfTickets()**: return the total available tickets ○ **getEventDetails()**: return event details from the event class
- ○ Create a simple user interface in a **main method** that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."

Code :

```
# main/ticket_booking_system.py
from entity.venue import Venue
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from entity.customer import Customer
from entity.booking import Booking
from datetime import datetime

class TicketBookingSystem:
    def __init__(self):
        self.events = []      # list of all events
        self.bookings = []    # list of all bookings

    def create_event(self, event_name, date_str, time_str, total_seats, ticket_price, event_type, venue_obj):
        event_date = datetime.strptime(date_str, "%Y-%m-%d").date()
        event_time = datetime.strptime(time_str, "%H:%M").time()

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue_obj,
```

Done by
Dhamini machireddy

```
                total_seats, ticket_price, "Movie", genre, actor, actress)

        elif event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue_obj,
                            total_seats, ticket_price, "Concert", artist, concert_type)

        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams name (e.g., India vs Pakistan): ")
            event = Sports(event_name, event_date, event_time, venue_obj,
                           total_seats, ticket_price, "Sports", sport_name, teams_name)

        else:
            print("❌ Invalid event type.")
            return None

        self.events.append(event)
        print("☑ Event created successfully!")
        return event

    def calculate_booking_cost(self, event, num_tickets):
        return event.ticket_price * num_tickets

    def book_tickets(self, event_name, num_tickets):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
            print("❌ Event not found.")
            return

        if event.available_seats < num_tickets:
            print("❌ Not enough tickets available.")
            return

        customers = []
        for i in range(num_tickets):
            print(f"\nEnter details for Customer {i + 1}:")
            name = input("Name: ")
            email = input("Email: ")
            phone = input("Phone: ")
            customers.append(Customer(name, email, phone))

        event.book_tickets(num_tickets)
```

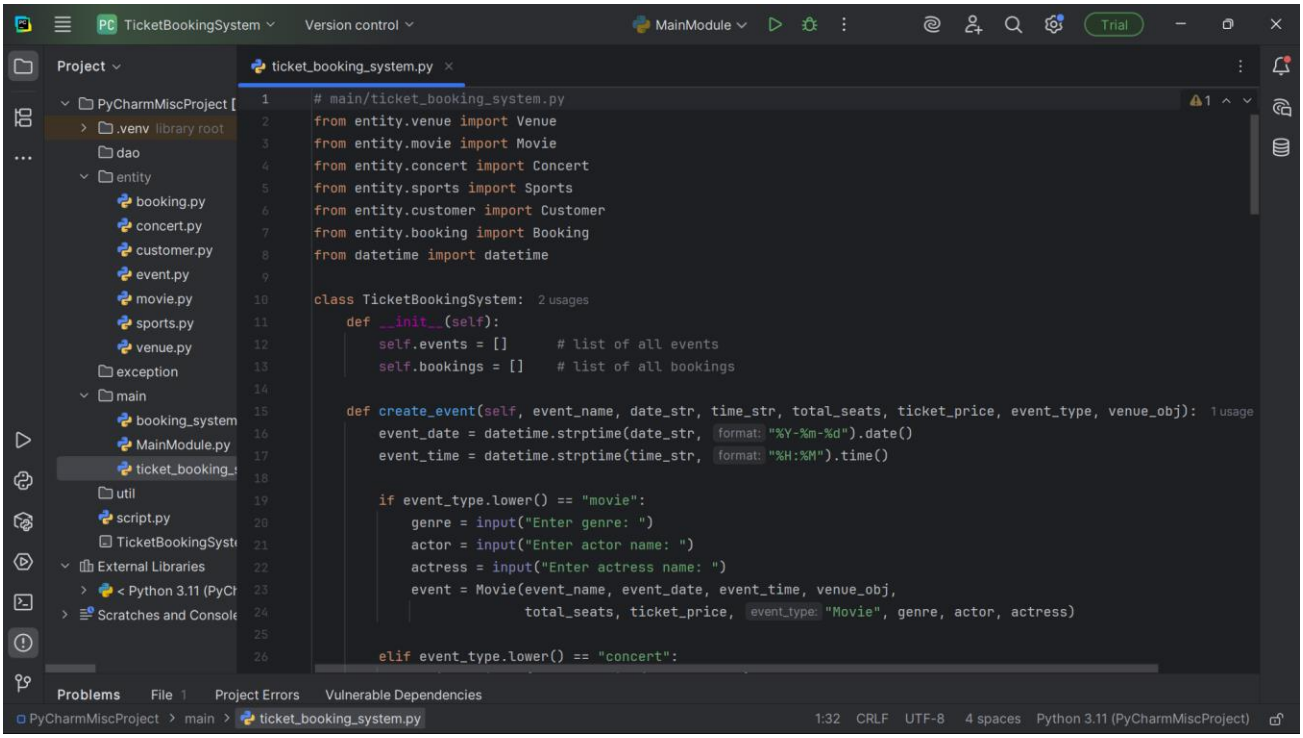
Done by
Dhamini machireddy

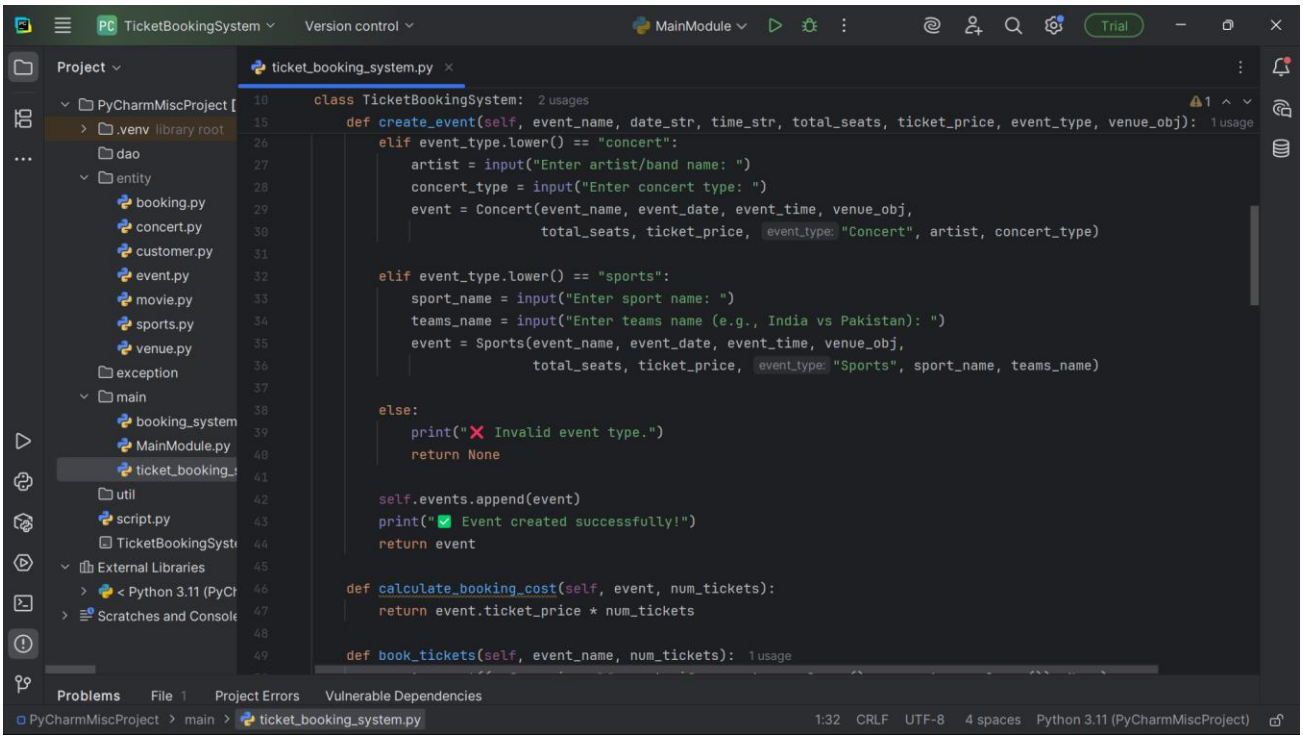
```
        booking = Booking(customers, event, num_tickets)
        self.bookings.append(booking)
        print("☑ Booking completed!")
        booking.display_booking_details()

    def cancel_booking(self, booking_id):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("❌ Booking ID not found.")
            return
        booking.event.cancel_booking(booking.num_tickets)
        self.bookings.remove(booking)
        print("☑ Booking cancelled successfully.")

    def get_available_no_of_tickets(self, event_name):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if event:
            print(f"🎟 Available Seats: {event.available_seats}")
        else:
            print("❌ Event not found.")

    def get_event_details(self, event_name):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if event:
            event.display_event_details()
        else:
            print("❌ Event not found.")

    def run(self):
        while True:
            print("\n📌 Ticket Booking Menu")
            print("Commands: create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details,
exit")
            command = input("Enter command: ").strip().lower()

            if command == "create_event":
                name = input("Enter event name: ")
                date_str = input("Enter date (YYYY-MM-DD): ")
                time_str = input("Enter time (HH:MM): ")
                venue_name = input("Enter venue name: ")
                address = input("Enter venue address: ")
                venue = Venue(venue_name, address)
                seats = int(input("Enter total seats: "))
```

Done by
Dhamini machireddy

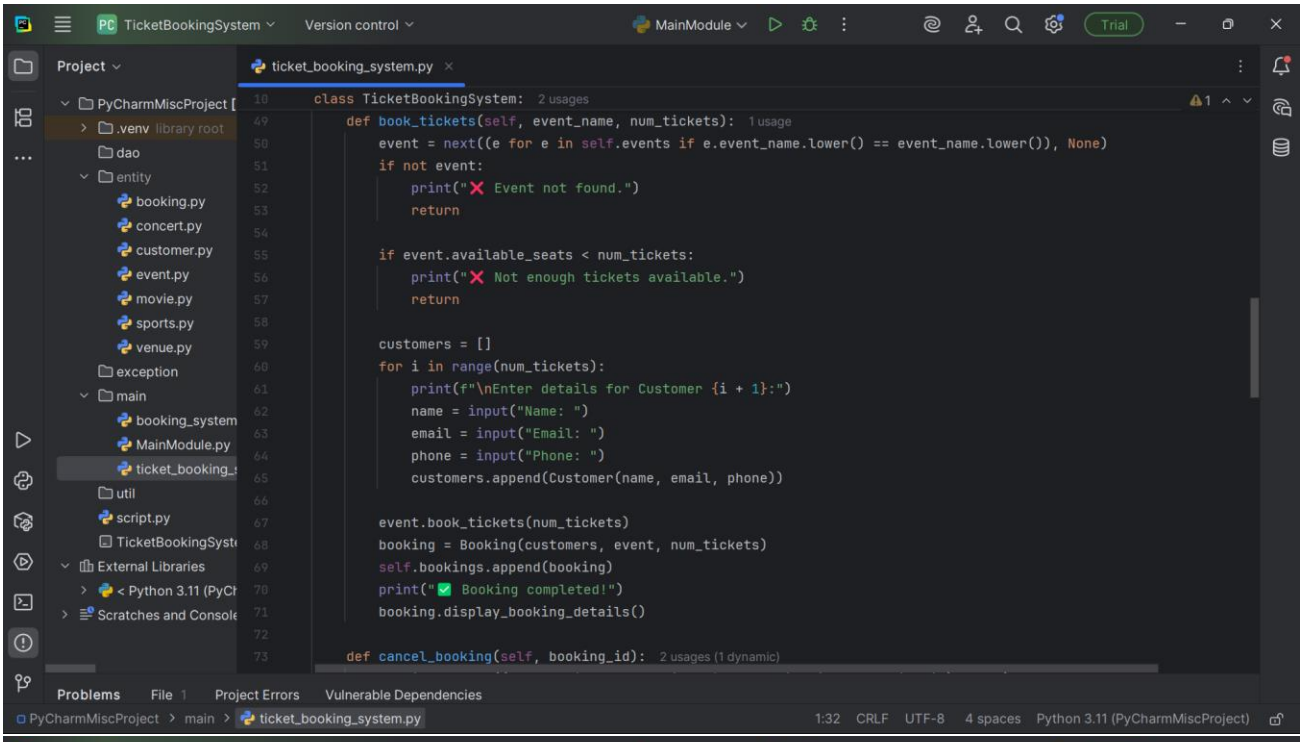
```
        price = float(input("Enter ticket price: "))
        event_type = input("Enter event type (Movie/Concert/Sports): ")

        self.create_event(name, date_str, time_str, seats, price, event_type, venue)

    elif command == "book_tickets":
        name = input("Enter event name: ")
        count = int(input("Enter number of tickets: "))
        self.book_tickets(name, count)

    elif command == "cancel_tickets":
        bid = int(input("Enter booking ID to cancel: "))
        self.cancel_booking(bid)

    elif command == "get_available_seats":
        name = input("Enter event name: ")
        self.get_available_no_of_tickets(name)

    elif command == "get_event_details":
        name = input("Enter event name: ")
        self.get_event_details(name)

    elif command == "exit":
        print("👋 Exiting system. Goodbye!")
        break

    else:
        print("❌ Invalid command.")
```

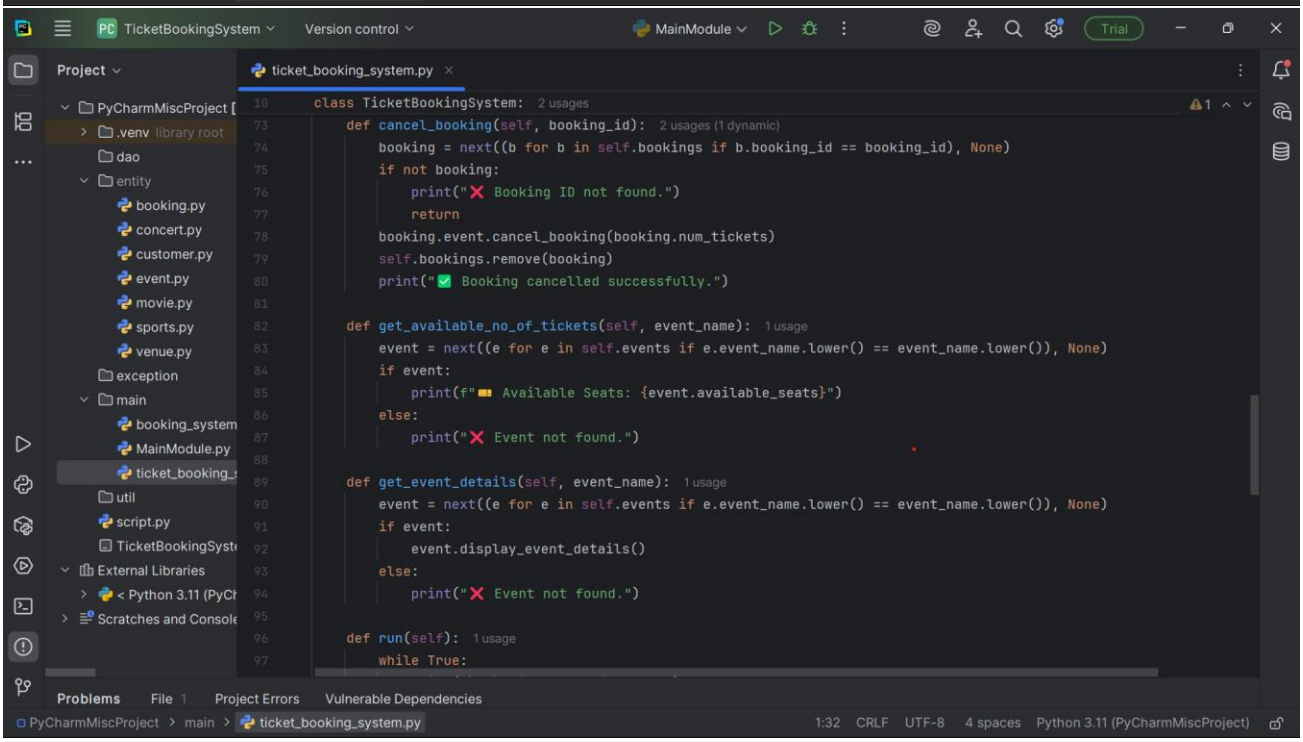
Done by
Dhamini machireddy





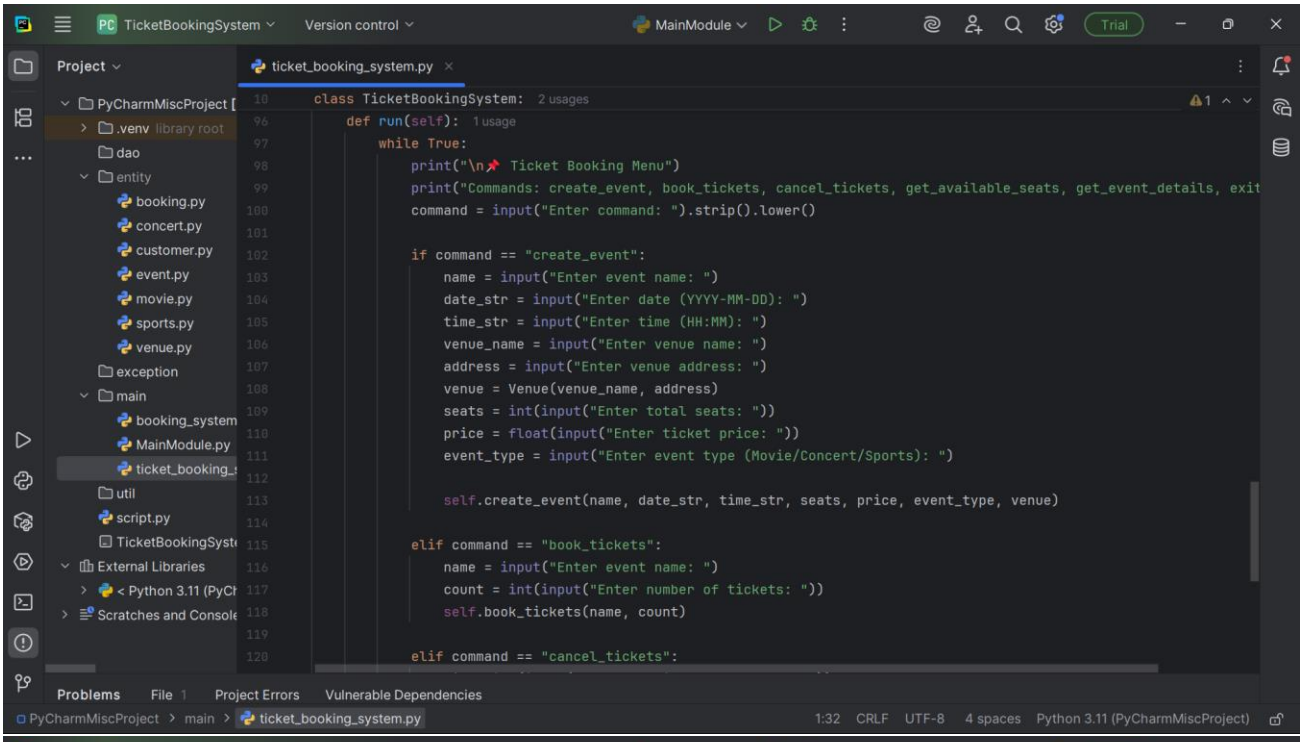
Done by
Dhamini machireddy



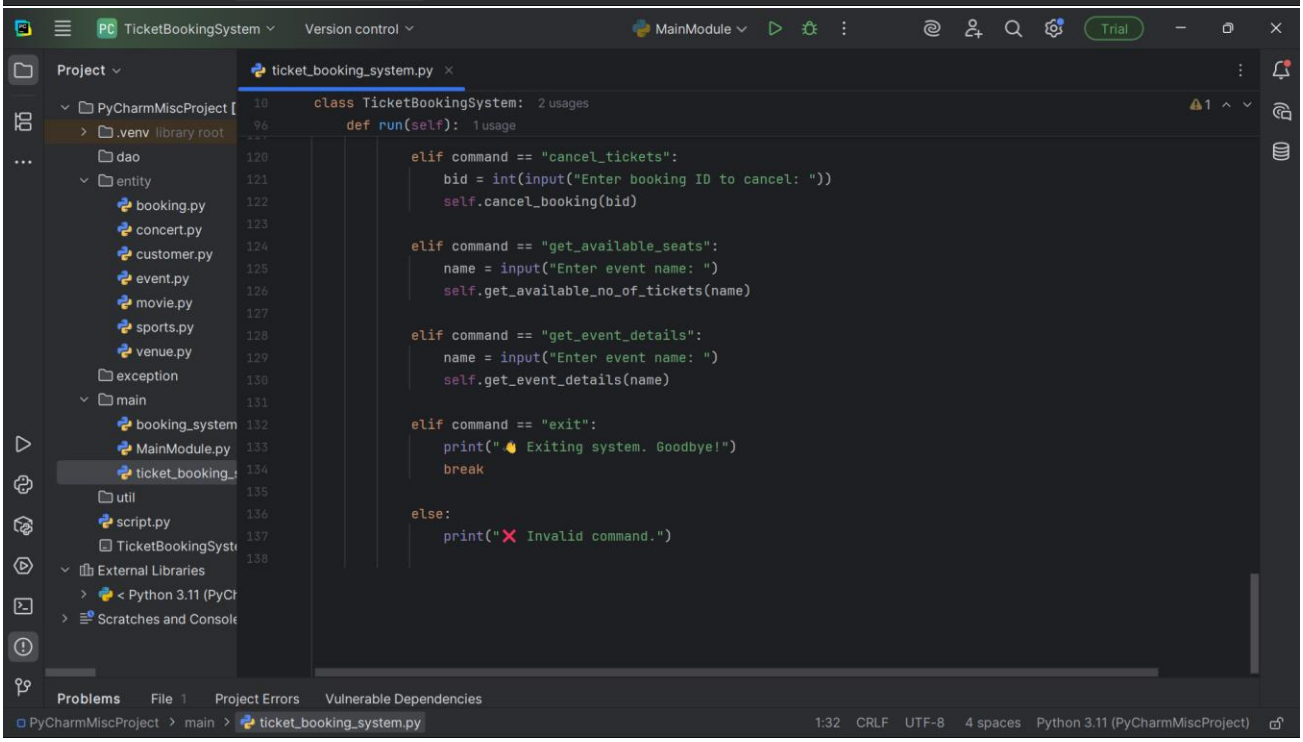


Done by

Dhamini machireddy



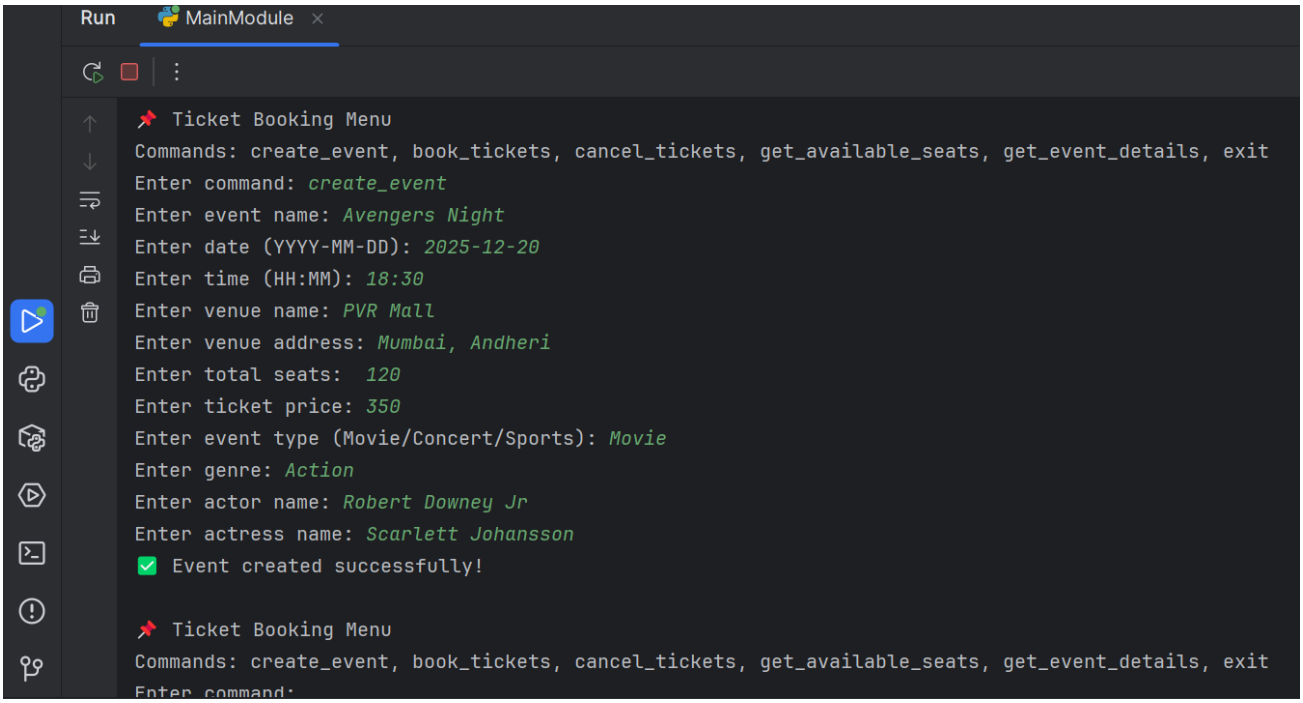
```
class TicketBookingSystem:  2 usages
    def run(self):  1 usage
        while True:
            print("\n📌 Ticket Booking Menu")
            print("Commands: create_event, book_tickets, cancel_tickets, get_available_seats, get_event_details, exit
            command = input("Enter command: ").strip().lower()

            if command == "create_event":
                name = input("Enter event name: ")
                date_str = input("Enter date (YYYY-MM-DD): ")
                time_str = input("Enter time (HH:MM): ")
                venue_name = input("Enter venue name: ")
                address = input("Enter venue address: ")
                venue = Venue(venue_name, address)
                seats = int(input("Enter total seats: "))
                price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (Movie/Concert/Sports): ")

                self.create_event(name, date_str, time_str, seats, price, event_type, venue)

            elif command == "book_tickets":
                name = input("Enter event name: ")
                count = int(input("Enter number of tickets: "))
                self.book_tickets(name, count)

            elif command == "cancel_tickets":
```



```
class TicketBookingSystem:  2 usages
    def run(self):  1 usage
            elif command == "cancel_tickets":
                bid = int(input("Enter booking ID to cancel: "))
                self.cancel_booking(bid)

            elif command == "get_available_seats":
                name = input("Enter event name: ")
                self.get_available_no_of_tickets(name)

            elif command == "get_event_details":
                name = input("Enter event name: ")
                self.get_event_details(name)

            elif command == "exit":
                print("👋 Exiting system. Goodbye!")
                break

            else:
                print("❌ Invalid command.")
```

Output :

Done by
Dhamini machireddy



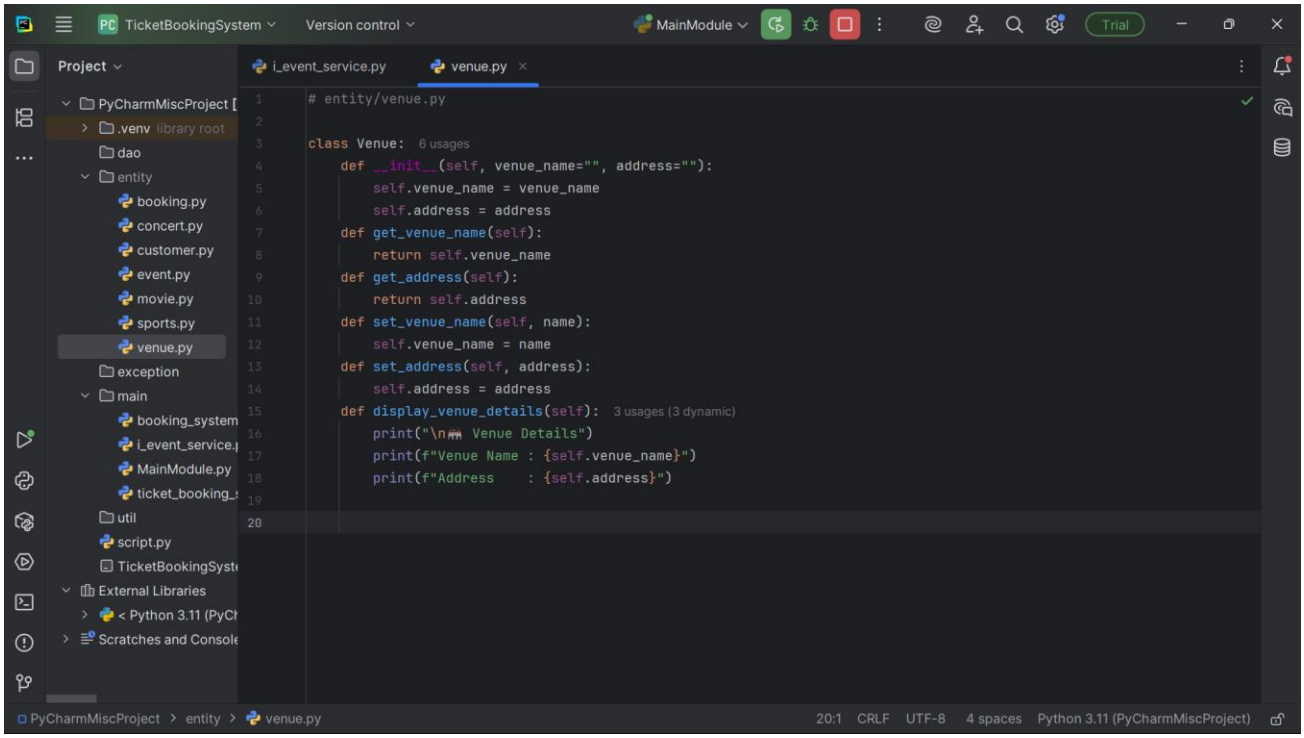
Task 8: Interface/abstract class, and Single Inheritance, static variable

1. Create **Venue,** class as mentioned above Task 4.

   Code :

```
# entity/venue.py

class Venue:
    def __init__(self, venue_name="", address=""):
        self.venue_name = venue_name
        self.address = address
    def get_venue_name(self):
        return self.venue_name
    def get_address(self):
        return self.address
    def set_venue_name(self, name):
        self.venue_name = name
    def set_address(self, address):
        self.address = address
    def display_venue_details(self):
        print("\n🏟 Venue Details")
        print(f"Venue Name : {self.venue_name}")
        print(f"Address    : {self.address}")
```

Done by
Dhamini machireddy



2. **Event** Class**:**
   - **Attributes:**
     - event_name, ○ event_date DATE, ○ event_time TIME,
     - venue (reference of class Venu), ○ total_seats, ○ available_seats, ○ ticket_price DECIMAL,
     - event_type ENUM('Movie', 'Sports', 'Concert')
   - **Methods and Constuctors:**
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

Code :

```
# entity/event.py

from datetime import date, time
from entity.venue import Venue

class Event:
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type=""):
        self.event_name = event_name
        self.event_date = event_date or date.today()
```

Done by
Dhamini machireddy

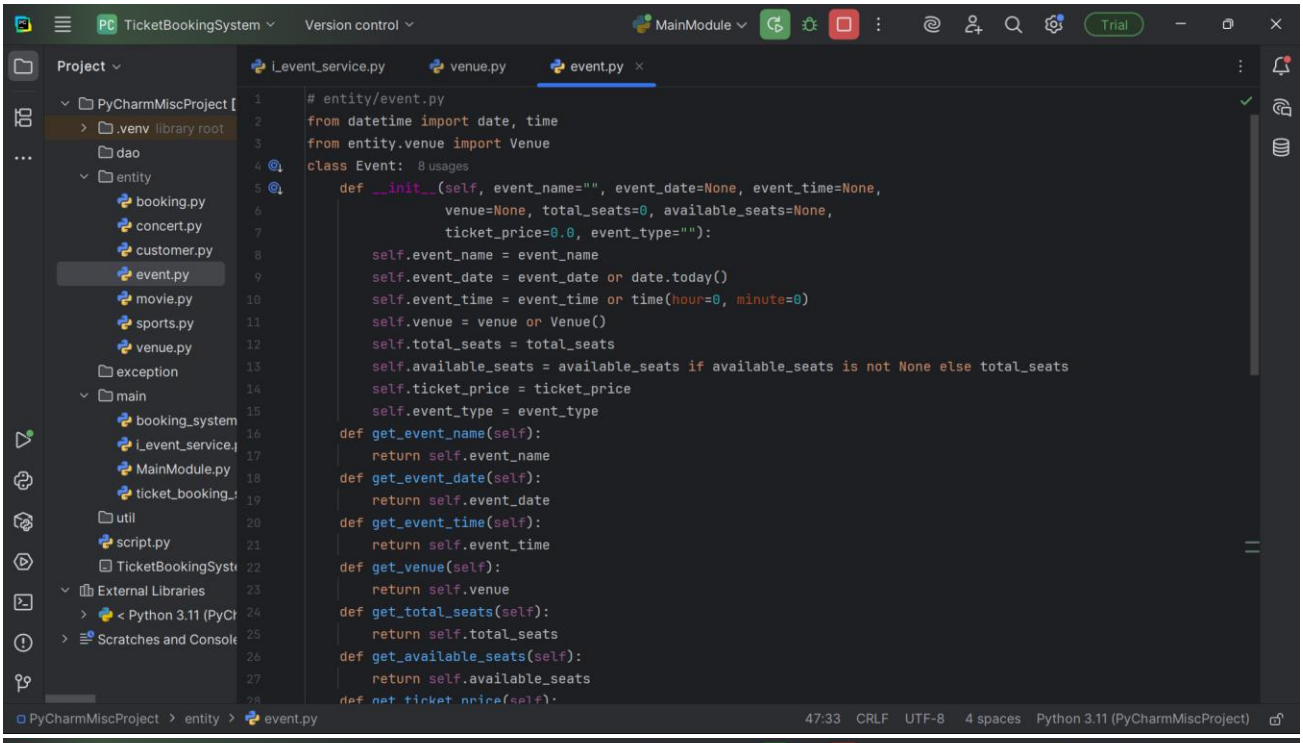
```
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue = venue or Venue()
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else
total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # ---------- Getters ----------
    def get_event_name(self):
        return self.event_name

    def get_event_date(self):
        return self.event_date

    def get_event_time(self):
        return self.event_time

    def get_venue(self):
        return self.venue

    def get_total_seats(self):
        return self.total_seats

    def get_available_seats(self):
        return self.available_seats

    def get_ticket_price(self):
        return self.ticket_price

    def get_event_type(self):
        return self.event_type

    # ---------- Setters ----------
    def set_event_name(self, name):
        self.event_name = name

    def set_event_date(self, event_date):
        self.event_date = event_date

    def set_event_time(self, event_time):
        self.event_time = event_time
```

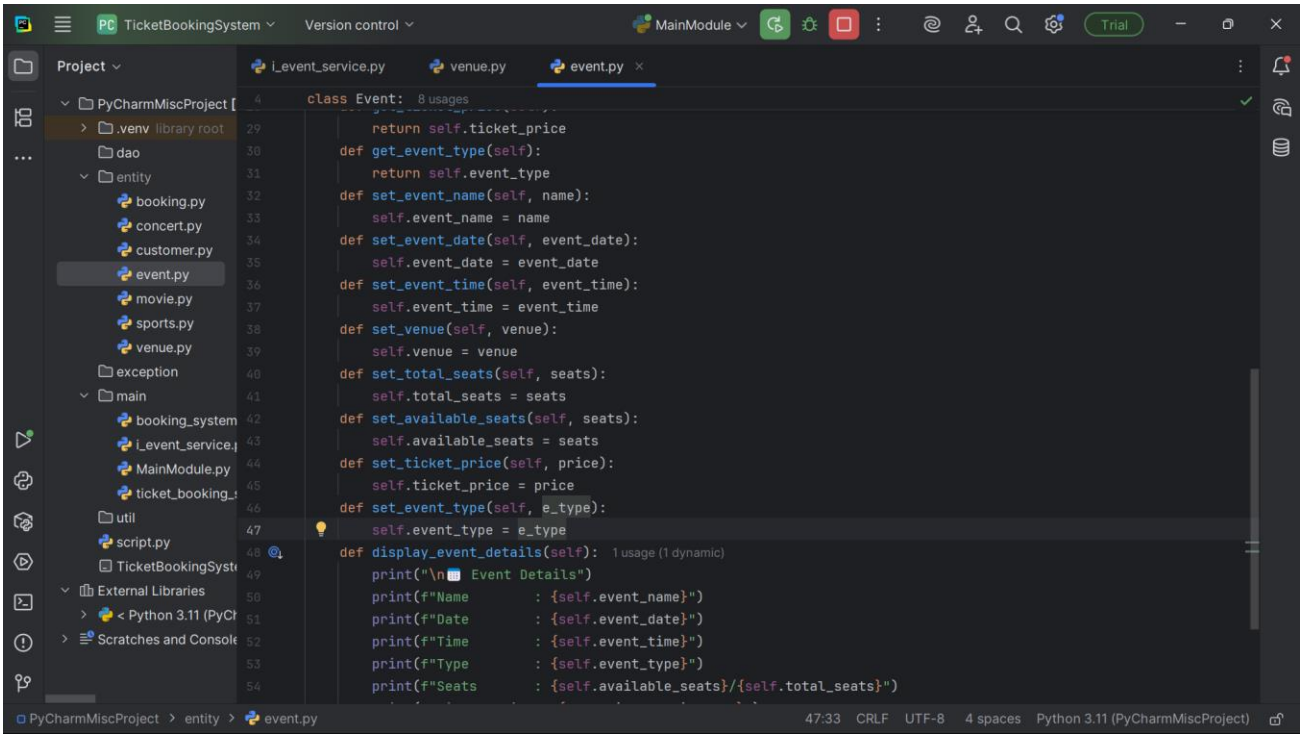
Done by
Dhamini machireddy

```
def set_venue(self, venue):
    self.venue = venue

def set_total_seats(self, seats):
    self.total_seats = seats

def set_available_seats(self, seats):
    self.available_seats = seats

def set_ticket_price(self, price):
    self.ticket_price = price

def set_event_type(self, e_type):
    self.event_type = e_type

# ---------- Display ----------
def display_event_details(self):
    print("\n🧮 Event Details")
    print(f"Name       : {self.event_name}")
    print(f"Date       : {self.event_date}")
    print(f"Time       : {self.event_time}")
    print(f"Type       : {self.event_type}")
    print(f"Seats      : {self.available_seats}/{self.total_seats}")
    print(f"Ticket Price: ₹{self.ticket_price:.2f}")
    self.venue.display_venue_details()
```

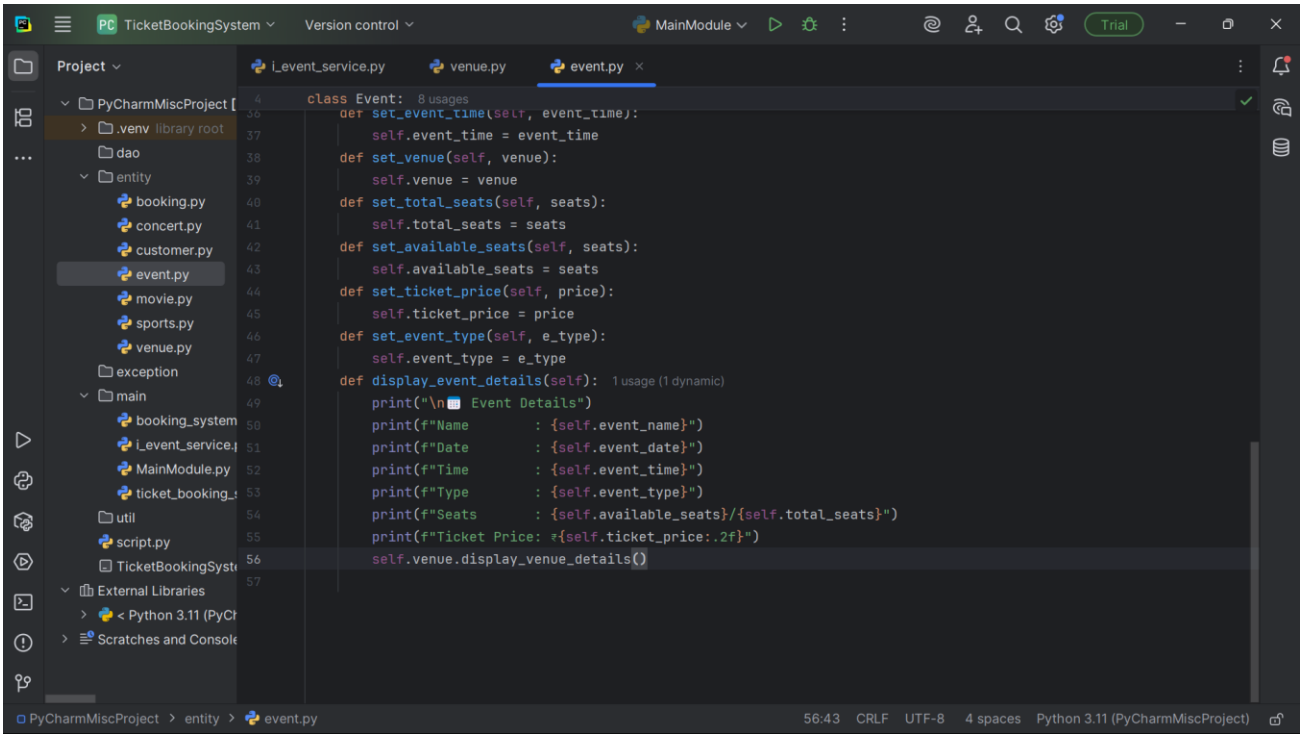
Done by

Dhamini machireddy





Done by
Dhamini machireddy



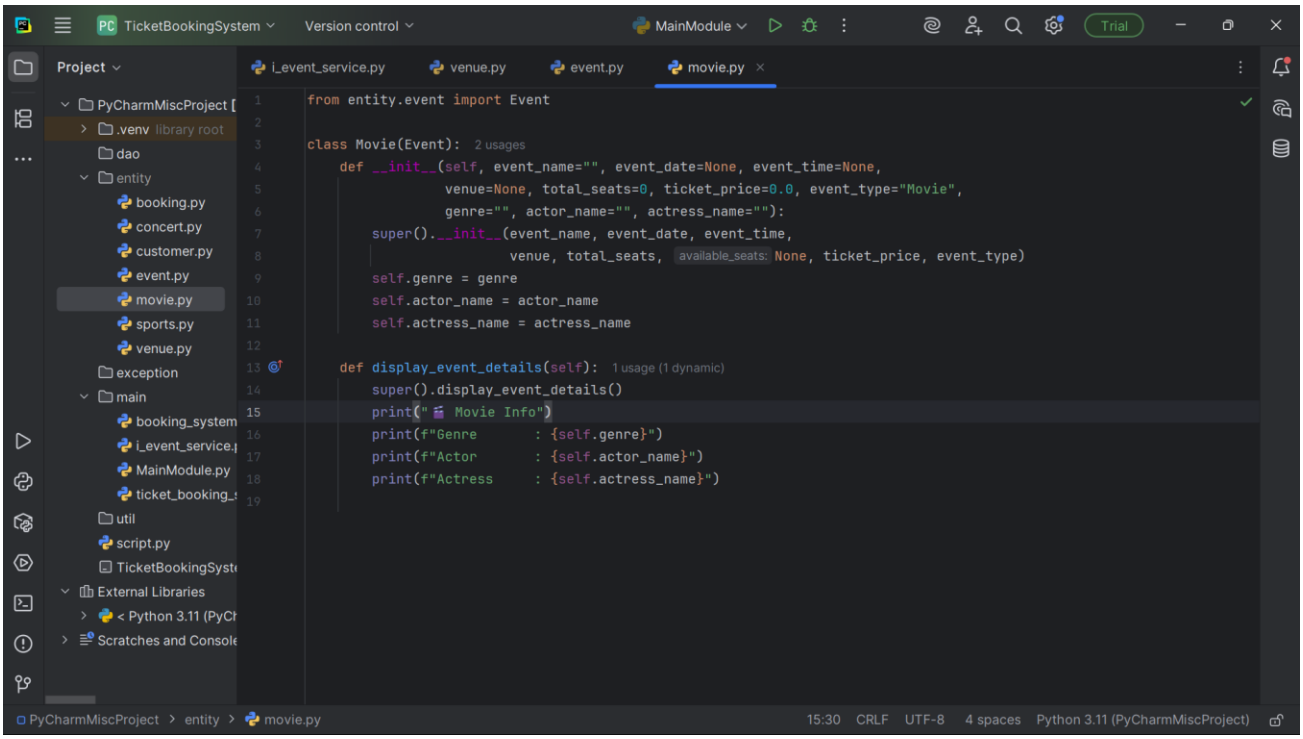
3. Create **Event** sub classes as mentioned in above Task 4**.**

Movie :

Code :

```
from entity.event import Event

class Movie(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Movie",
                 genre="", actor_name="", actress_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

    def display_event_details(self):
        super().display_event_details()
        print(" 🎬 Movie Info")
        print(f"Genre      : {self.genre}")
        print(f"Actor      : {self.actor_name}")
        print(f"Actress    : {self.actress_name}")
```

Done by
Dhamini machireddy

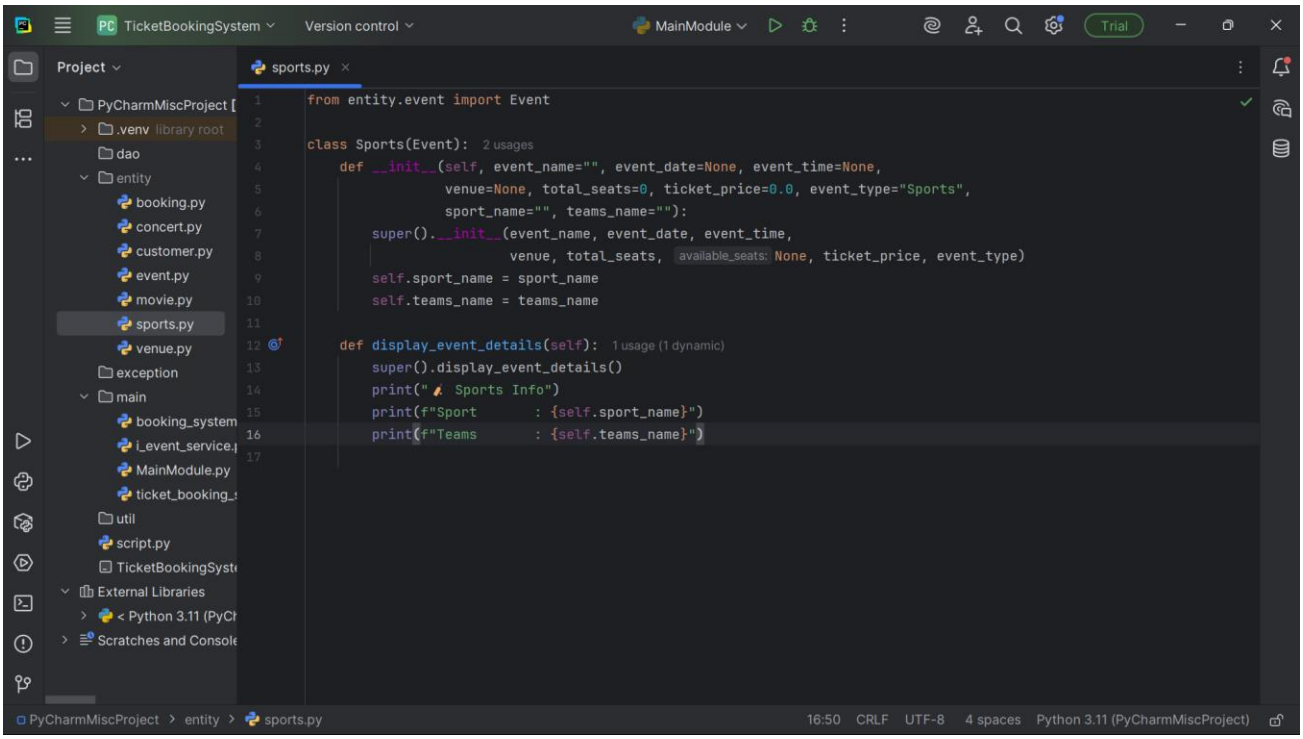


Sports :

Code :

```
from entity.event import Event

class Sports(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Sports",
                 sport_name="", teams_name=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.sport_name = sport_name
        self.teams_name = teams_name

    def display_event_details(self):
        super().display_event_details()
        print(" Sports Info")
        print(f"Sport      : {self.sport_name}")
        print(f"Teams      : {self.teams_name}")
```

Done by
Dhamini machireddy

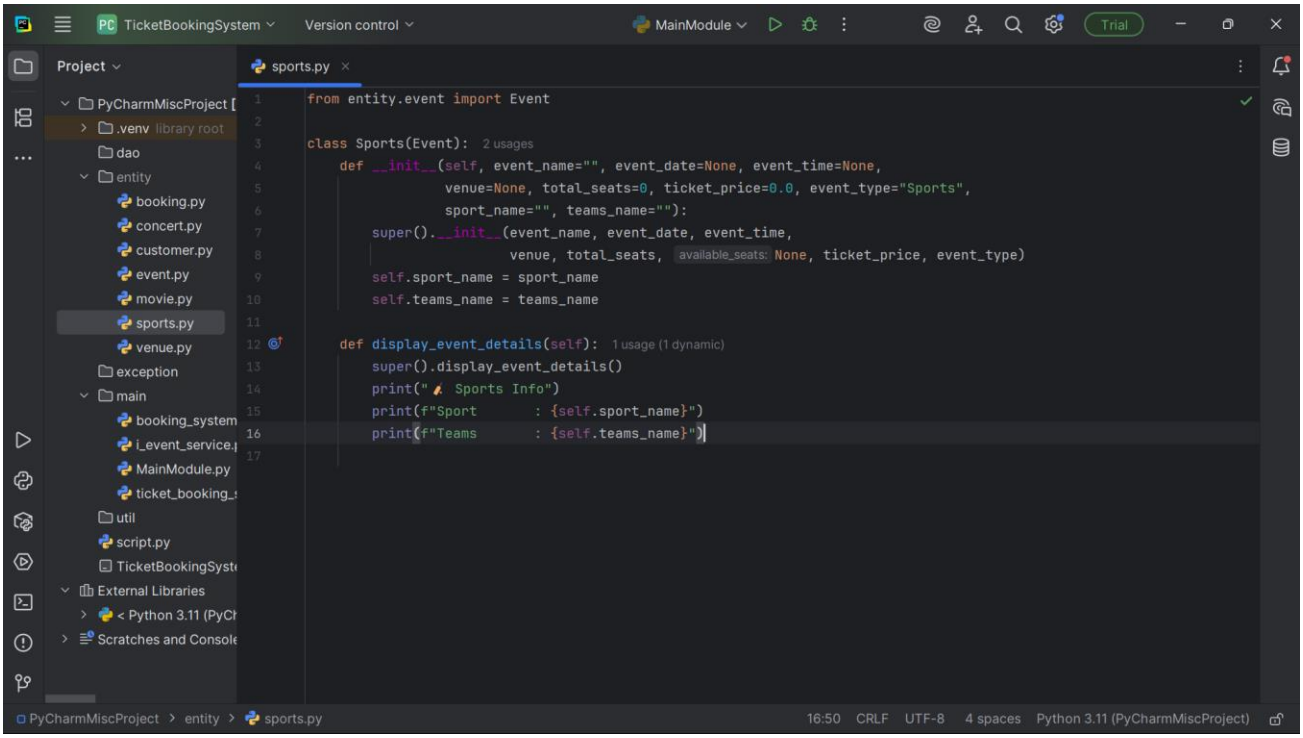


Concert :

Code :

```
from entity.event import Event

class Concert(Event):
    def __init__(self, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, ticket_price=0.0, event_type="Concert",
                 artist="", concert_type=""):
        super().__init__(event_name, event_date, event_time,
                         venue, total_seats, None, ticket_price, event_type)
        self.artist = artist
        self.concert_type = concert_type

    def display_event_details(self):
        super().display_event_details()
        print(" Concert Info")
        print(f"Artist       : {self.artist}")
        print(f"Concert Type  : {self.concert_type}")
```

Done by
Dhamini machireddy



4. Create a class **Customer** and **Booking** as mentioned in above Task 4**.**

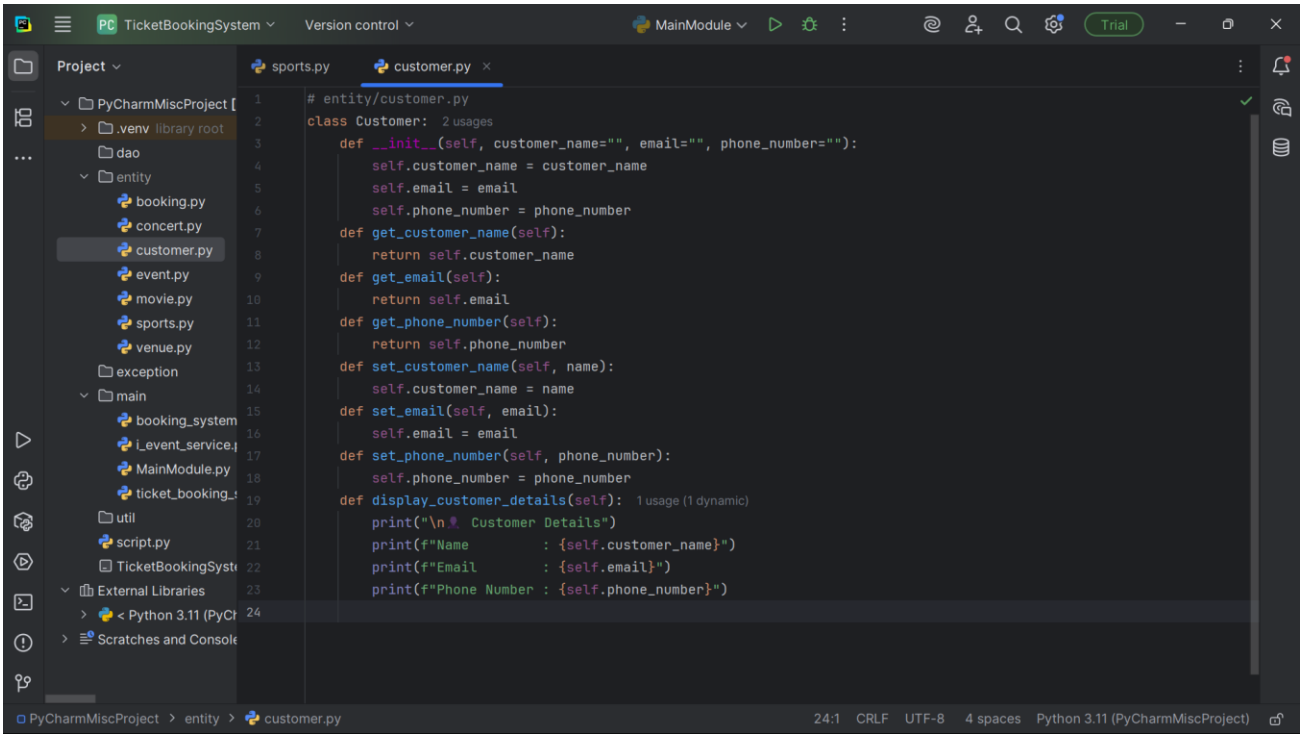
Customer :

Code :

```
# entity/customer.py
class Customer:
    def __init__(self, customer_name="", email="", phone_number=""):
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number
    def get_customer_name(self):
        return self.customer_name
    def get_email(self):
        return self.email
    def get_phone_number(self):
        return self.phone_number
    def set_customer_name(self, name):
        self.customer_name = name
    def set_email(self, email):
        self.email = email
    def set_phone_number(self, phone_number):
        self.phone_number = phone_number
    def display_customer_details(self):
        print("\n Customer Details")
        print(f"Name        : {self.customer_name}")
```

Done by
Dhamini machireddy

```
        print(f"Email       : {self.email}")
        print(f"Phone Number : {self.phone_number}")
```



Booking :

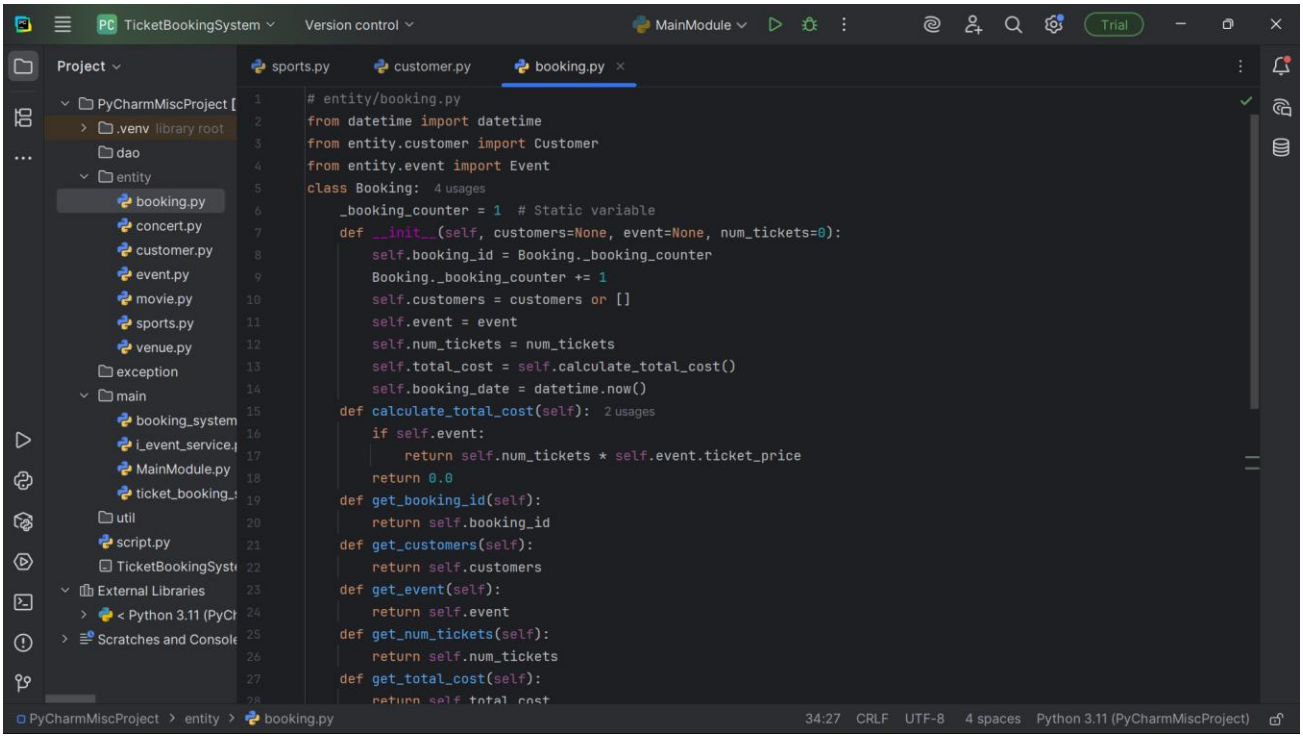
Code :

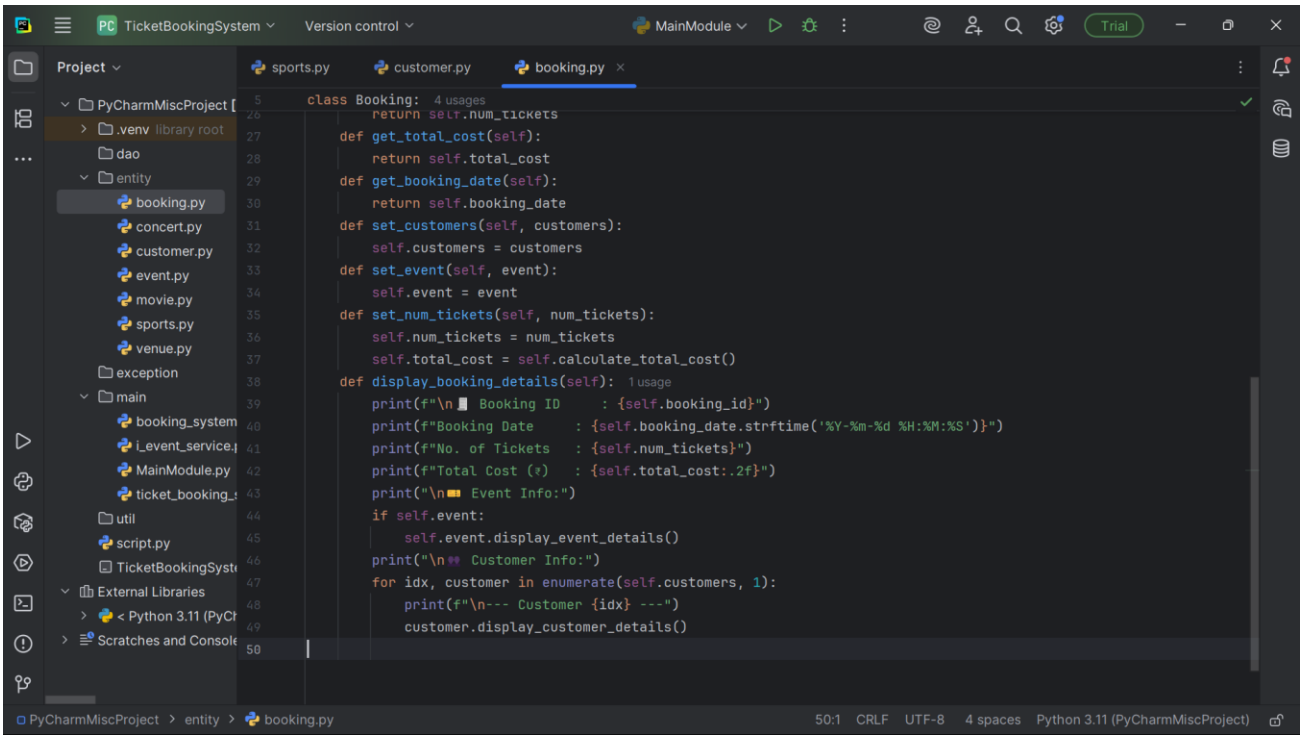
```
# entity/booking.py
from datetime import datetime
from entity.customer import Customer
from entity.event import Event
class Booking:
    _booking_counter = 1  # Static variable
    def __init__(self, customers=None, event=None, num_tickets=0):
        self.booking_id = Booking._booking_counter
        Booking._booking_counter += 1
        self.customers = customers or []
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()
    def calculate_total_cost(self):
        if self.event:
            return self.num_tickets * self.event.ticket_price
        return 0.0
    def get_booking_id(self):
        return self.booking_id
```

Done by
Dhamini machireddy

```
def get_customers(self):
    return self.customers
def get_event(self):
    return self.event
def get_num_tickets(self):
    return self.num_tickets
def get_total_cost(self):
    return self.total_cost
def get_booking_date(self):
    return self.booking_date
def set_customers(self, customers):
    self.customers = customers
def set_event(self, event):
    self.event = event
def set_num_tickets(self, num_tickets):
    self.num_tickets = num_tickets
    self.total_cost = self.calculate_total_cost()
def display_booking_details(self):
    print(f"\n📝 Booking ID     : {self.booking_id}")
    print(f"Booking Date     : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"No. of Tickets   : {self.num_tickets}")
    print(f"Total Cost (₹)   : {self.total_cost:.2f}")
    print("\n🎟️ Event Info:")
    if self.event:
        self.event.display_event_details()
    print("\n👥 Customer Info:")
    for idx, customer in enumerate(self.customers, 1):
        print(f"\n--- Customer {idx} ---")
        customer.display_customer_details()
```

Done by
Dhamini machireddy





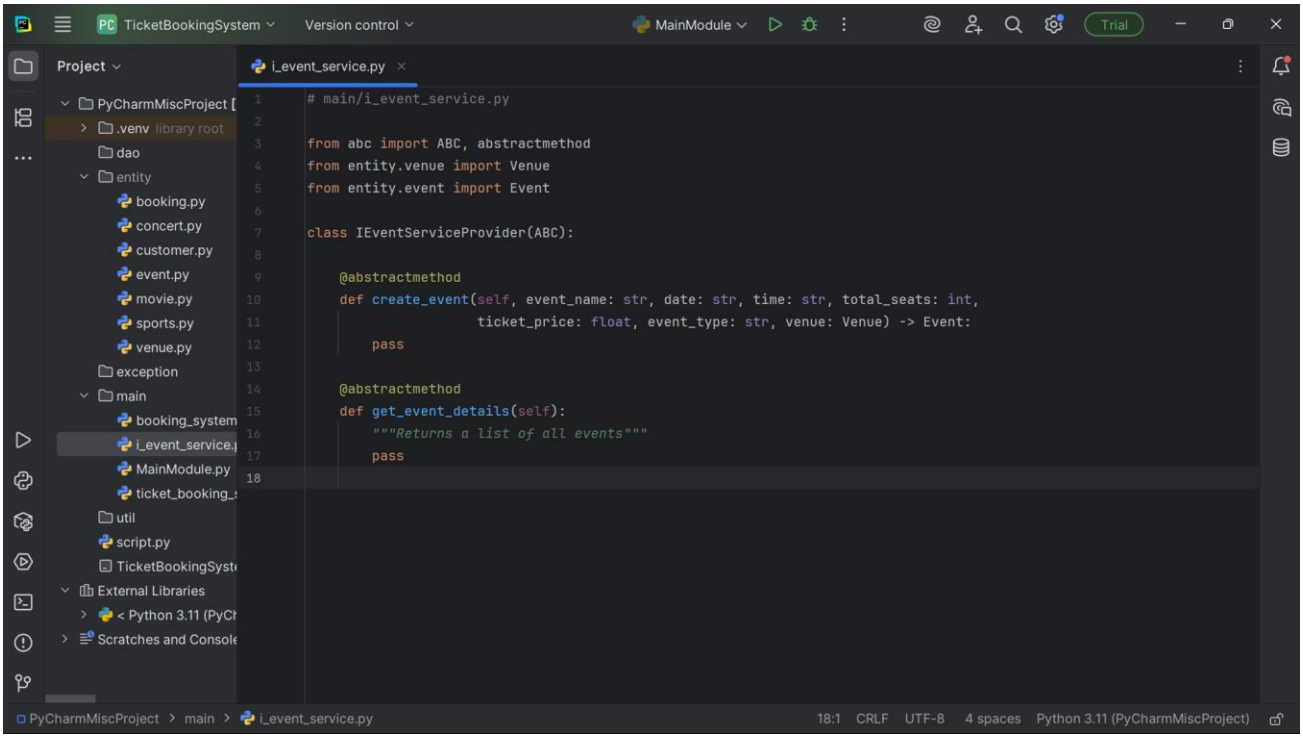
5. Create interface/abstract class **IEventServiceProvider** with following methods:
   - **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu: Venu):** Create a new event with the specified details and event type (movie, sport or concert) and return event object.
   - **getEventDetails():** return array of event details from the event class.

Done by
Dhamini machireddy

- **getAvailableNoOfTickets()**: return the total available tickets.

Code :

```
# main/i_event_service.py

from abc import ABC, abstractmethod
from entity.venue import Venue
from entity.event import Event

class IEventServiceProvider(ABC):

    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str,
                     total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue) -> Event:
        pass

    @abstractmethod
    def getEventDetails(self) -> list:
        pass

    @abstractmethod
    def getAvailableNoOfTickets(self) -> int:
        pass
```

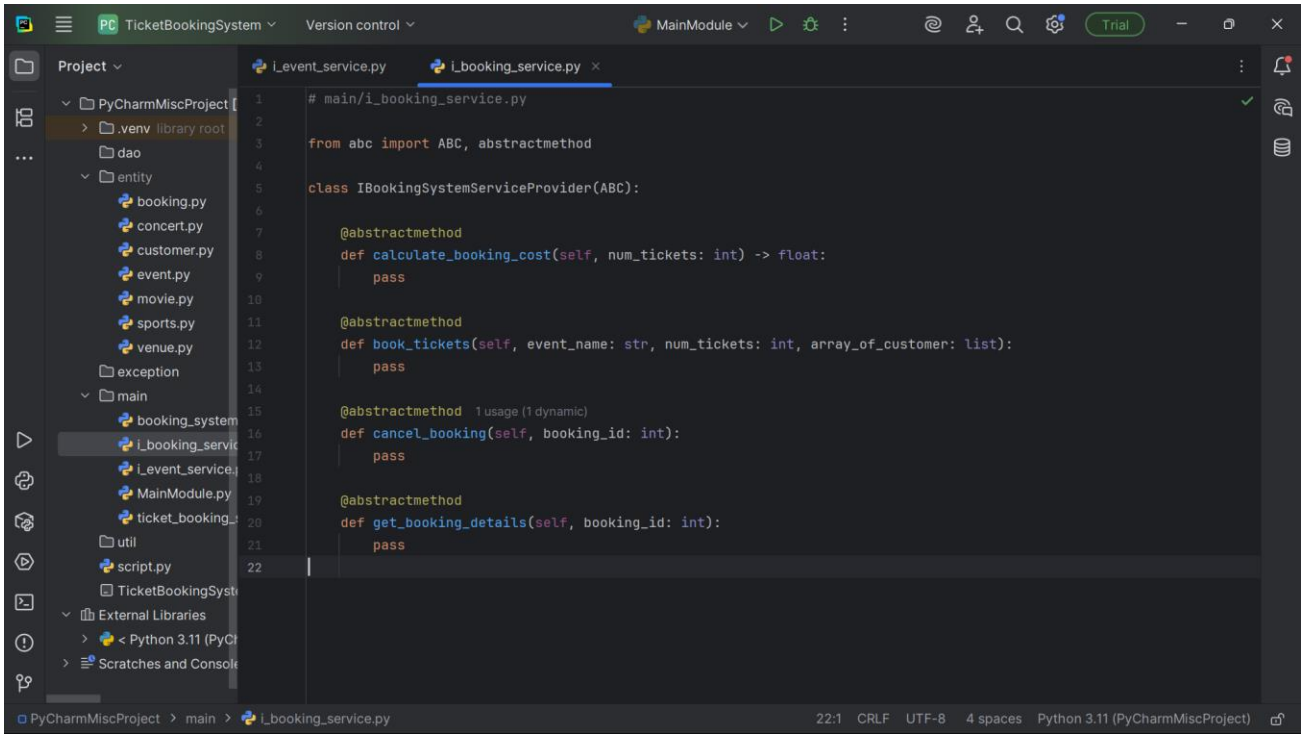


Done by
Dhamini machireddy

6. Create interface/abstract class **IBookingSystemServiceProvider** with following methods:
    - **calculate_booking_cost(num_tickets):** Calculate and set the total cost of the booking.
    - **book_tickets(eventname:str, num_tickets, arrayOfCustomer):** Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class.
    - **cancel_booking(booking_id):** Cancel the booking and update the available seats.
    - **get_booking_details(booking_id):**get the booking details.

Code :

```
# main/i_booking_service.py

from abc import ABC, abstractmethod
class IBookingSystemServiceProvider(ABC):
    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int) -> float:
        pass
    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, array_of_customer: list):
        pass
    @abstractmethod
    def cancel_booking(self, booking_id: int):
        pass
    @abstractmethod
    def get_booking_details(self, booking_id: int):
        pass
```



Done by
Dhamini machireddy

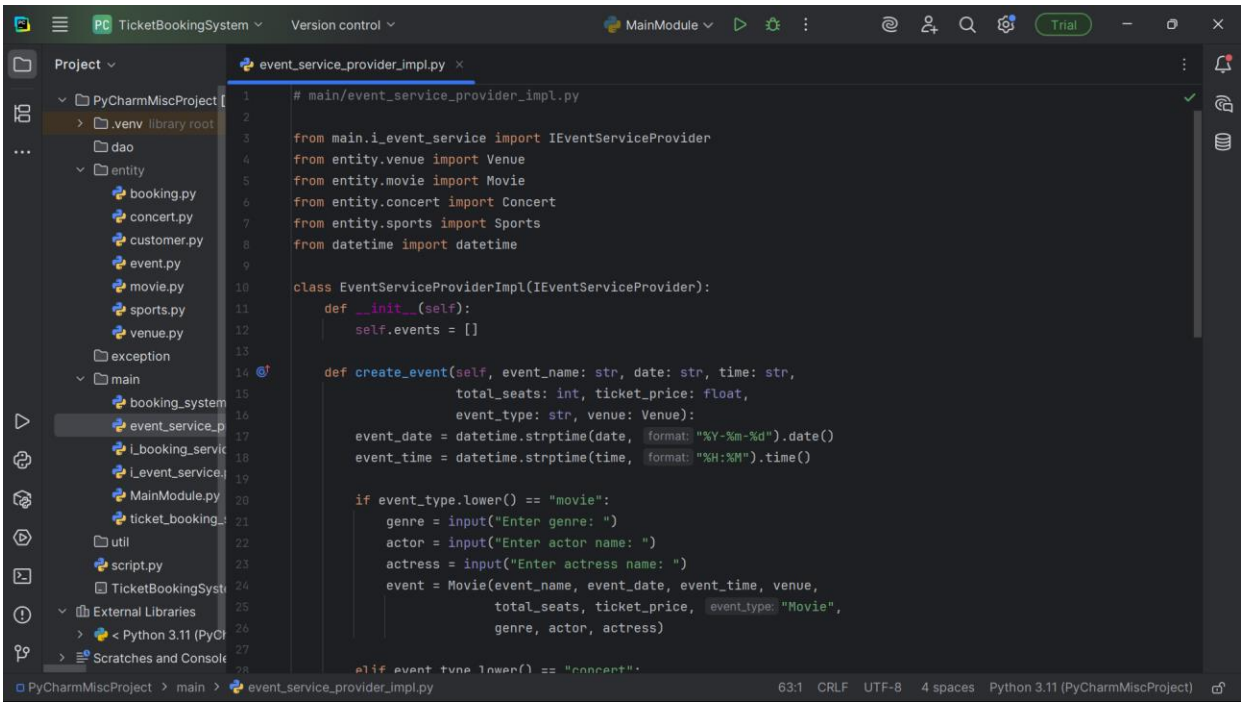
7. Create **EventServiceProviderImpl** class which implements **IEventServiceProvider** provide all implementation methods.

Code :

```
# main/event_service_provider_impl.py

from main.i_event_service import IEventServiceProvider
from entity.venue import Venue
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime

class EventServiceProviderImpl(IEventServiceProvider):
    def __init__(self):
        self.events = []

    def create_event(self, event_name: str, date: str, time: str,
                     total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue):
        event_date = datetime.strptime(date, "%Y-%m-%d").date()
        event_time = datetime.strptime(time, "%H:%M").time()

        if event_type.lower() == "movie":
            genre = input("Enter genre: ")
            actor = input("Enter actor name: ")
            actress = input("Enter actress name: ")
            event = Movie(event_name, event_date, event_time, venue,
                          total_seats, ticket_price, "Movie",
                          genre, actor, actress)

        elif event_type.lower() == "concert":
            artist = input("Enter artist/band name: ")
            concert_type = input("Enter concert type: ")
            event = Concert(event_name, event_date, event_time, venue,
                            total_seats, ticket_price, "Concert",
                            artist, concert_type)

        elif event_type.lower() == "sports":
            sport_name = input("Enter sport name: ")
            teams_name = input("Enter teams name: ")
            event = Sports(event_name, event_date, event_time, venue,
```

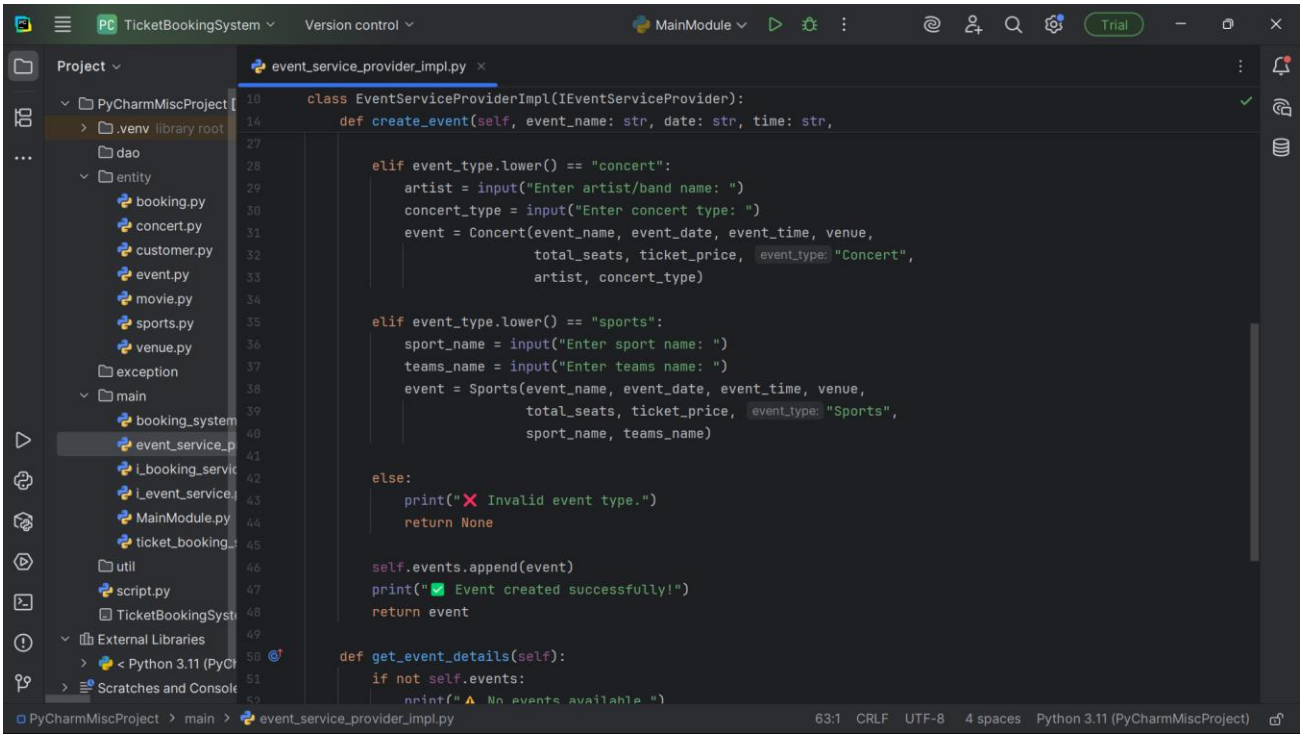
Done by
Dhamini machireddy

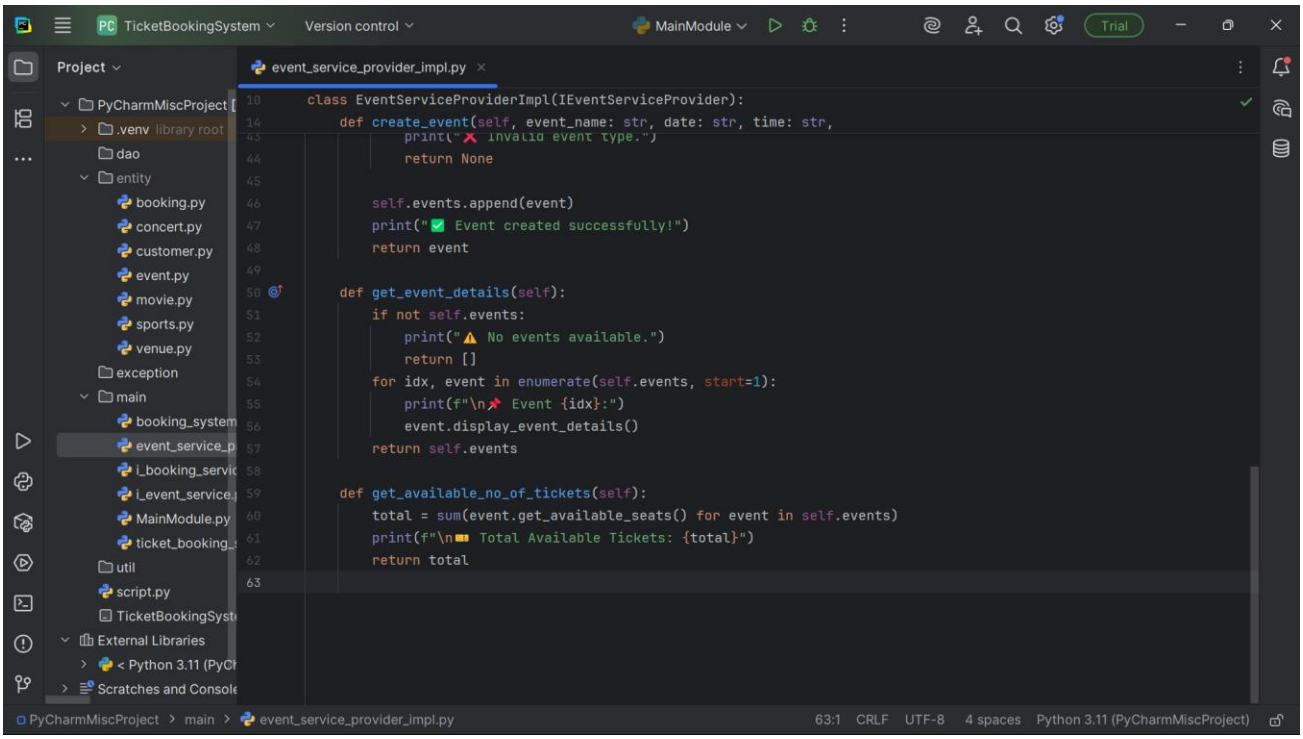
```
                total_seats, ticket_price, "Sports",
                sport_name, teams_name)

        else:
            print("❌ Invalid event type.")
            return None

        self.events.append(event)
        print("☑ Event created successfully!")
        return event

    def get_event_details(self):
        if not self.events:
            print("⚠ No events available.")
            return []
        for idx, event in enumerate(self.events, start=1):
            print(f"\n📌 Event {idx}:")
            event.display_event_details()
        return self.events

    def get_available_no_of_tickets(self):
        total = sum(event.get_available_seats() for event in self.events)
        print(f"\n🎟 Total Available Tickets: {total}")
        return total
```



```
```

Done by
Dhamini machireddy





8. Create **BookingSystemServiceProviderImpl** class which implements **IBookingSystemServiceProvider** provide all implementation methods and inherits **EventServiceProviderImpl** class with following attributes.
    - **Attributes** ○ array of events

Code :

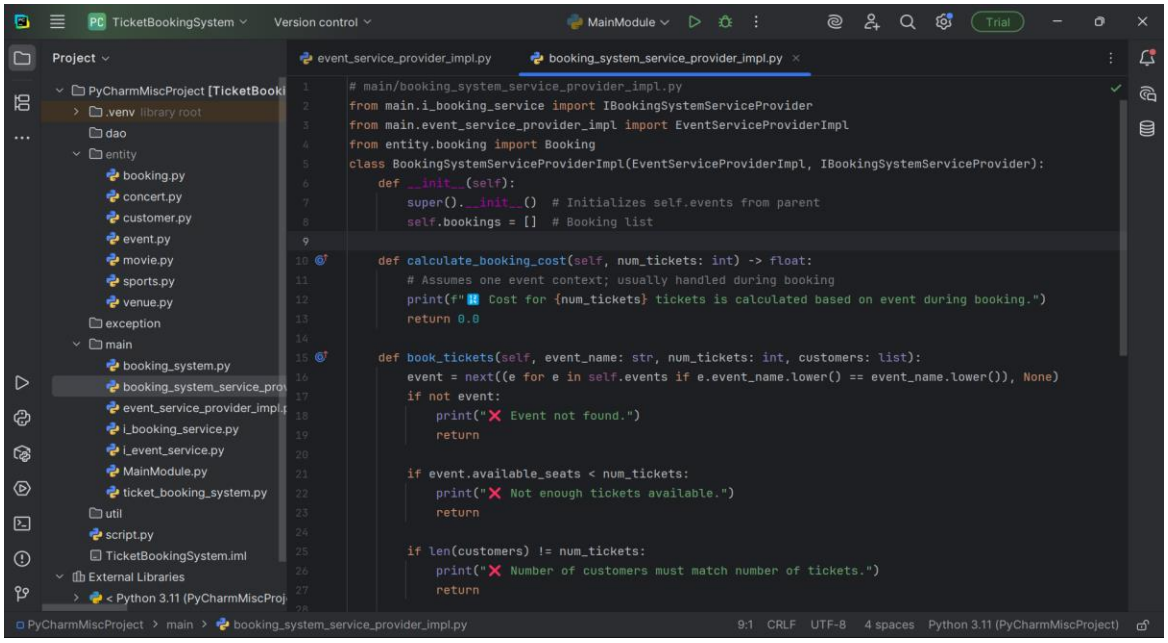
main/booking_system_service_provider_impl.py

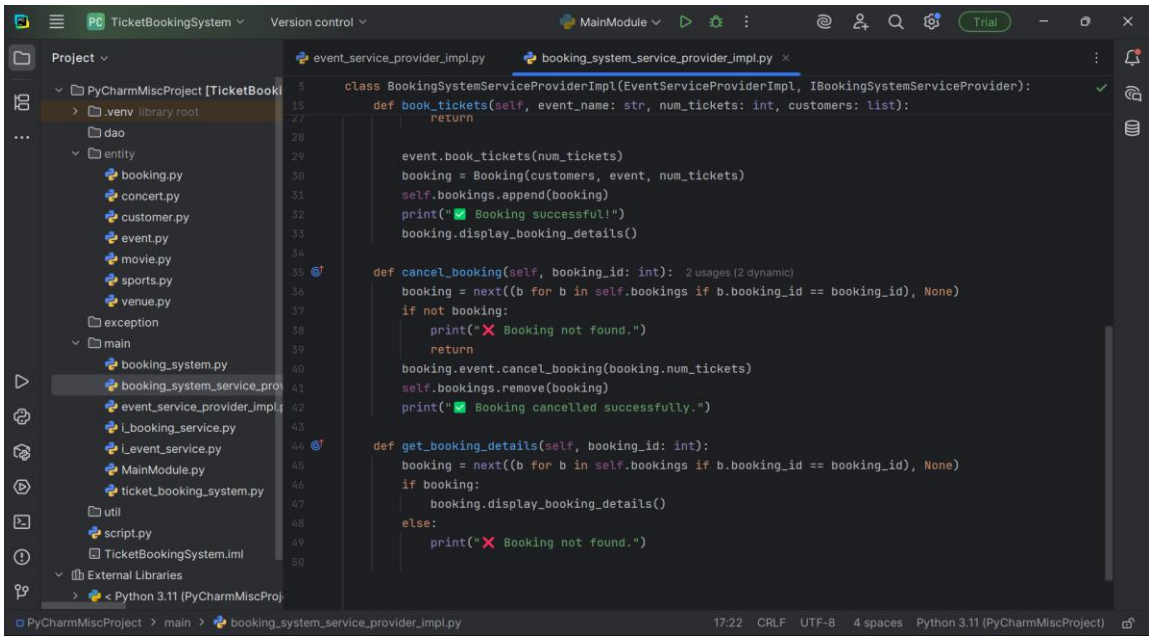
Done by
Dhamini machireddy

```
from main.i_booking_service import IBookingSystemServiceProvider
from main.event_service_provider_impl import EventServiceProviderImpl
from entity.booking import Booking
from entity.customer import Customer

class BookingSystemServiceProviderImpl(EventServiceProviderImpl, IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()  # Initializes self.events from parent
        self.bookings = []  # Booking list

    def calculate_booking_cost(self, num_tickets: int) -> float:
        # Assumes one event context; usually handled during booking
        print(f"🔢 Cost for {num_tickets} tickets is calculated based on event during booking.")
        return 0.0

    def book_tickets(self, event_name: str, num_tickets: int, customers: list):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
            print("❌ Event not found.")
            return

        if event.available_seats < num_tickets:
            print("❌ Not enough tickets available.")
            return

        if len(customers) != num_tickets:
            print("❌ Number of customers must match number of tickets.")
            return

        event.book_tickets(num_tickets)
        booking = Booking(customers, event, num_tickets)
        self.bookings.append(booking)
        print("☑️ Booking successful!")
        booking.display_booking_details()

    def cancel_booking(self, booking_id: int):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("❌ Booking not found.")
            return
```

Done by
Dhamini machireddy

```
        booking.event.cancel_booking(booking.num_tickets)
        self.bookings.remove(booking)
        print("☑ Booking cancelled successfully.")

    def get_booking_details(self, booking_id: int):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if booking:
            booking.display_booking_details()
        else:
            print("❌ Booking not found.")
```





Done by
Dhamini machireddy

9. Create **TicketBookingSystem** class and perform following operations:
    - Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."

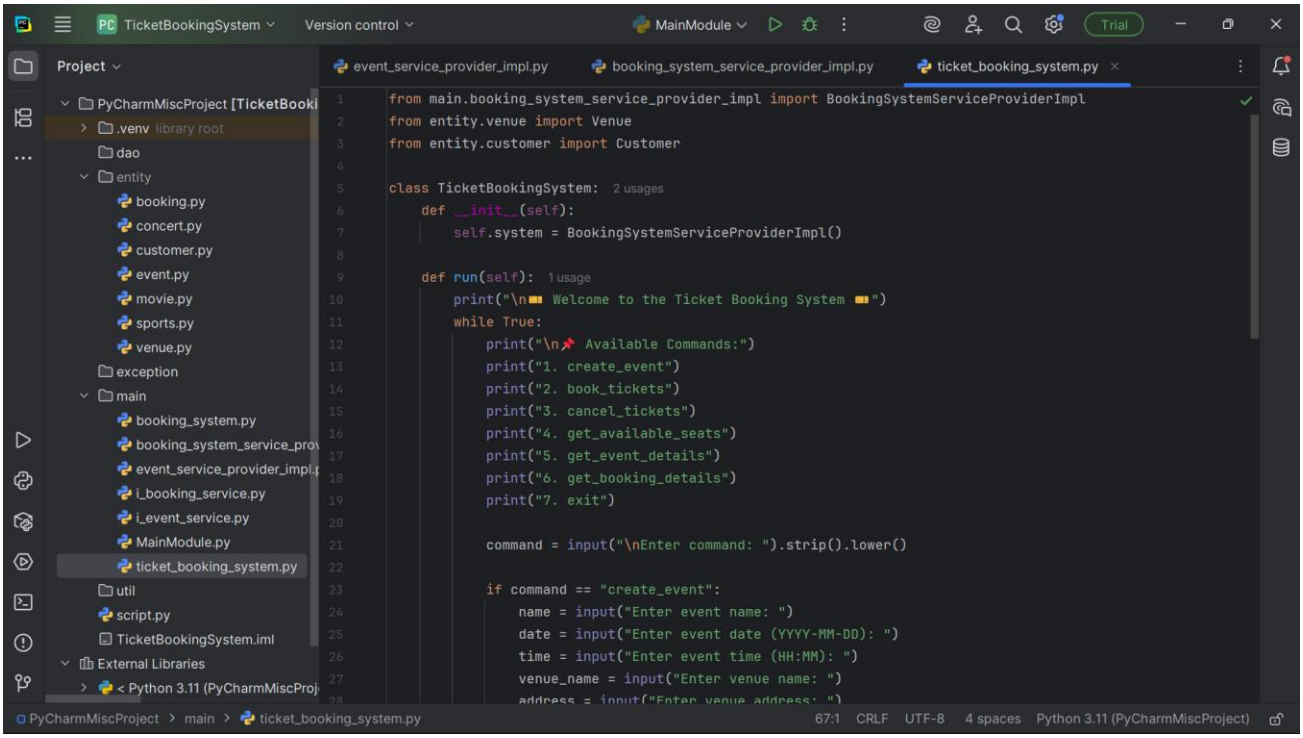
Code :

```
from main.booking_system_service_provider_impl import BookingSystemServiceProviderImpl
from entity.venue import Venue
from entity.customer import Customer

class TicketBookingSystem:
    def __init__(self):
        self.system = BookingSystemServiceProviderImpl()

    def run(self):
        print("\n🎟️ Welcome to the Ticket Booking System 🎟️")
        while True:
            print("\n📌 Available Commands:")
            print("1. create_event")
            print("2. book_tickets")
            print("3. cancel_tickets")
            print("4. get_available_seats")
            print("5. get_event_details")
            print("6. get_booking_details")
            print("7. exit")

            command = input("\nEnter command: ").strip().lower()

            if command == "create_event":
                name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time = input("Enter event time (HH:MM): ")
                venue_name = input("Enter venue name: ")
                address = input("Enter venue address: ")
                venue = Venue(venue_name, address)
                total_seats = int(input("Enter total seats: "))
                price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (Movie/Concert/Sports): ")
                self.system.create_event(name, date, time, total_seats, price, event_type, venue)
```

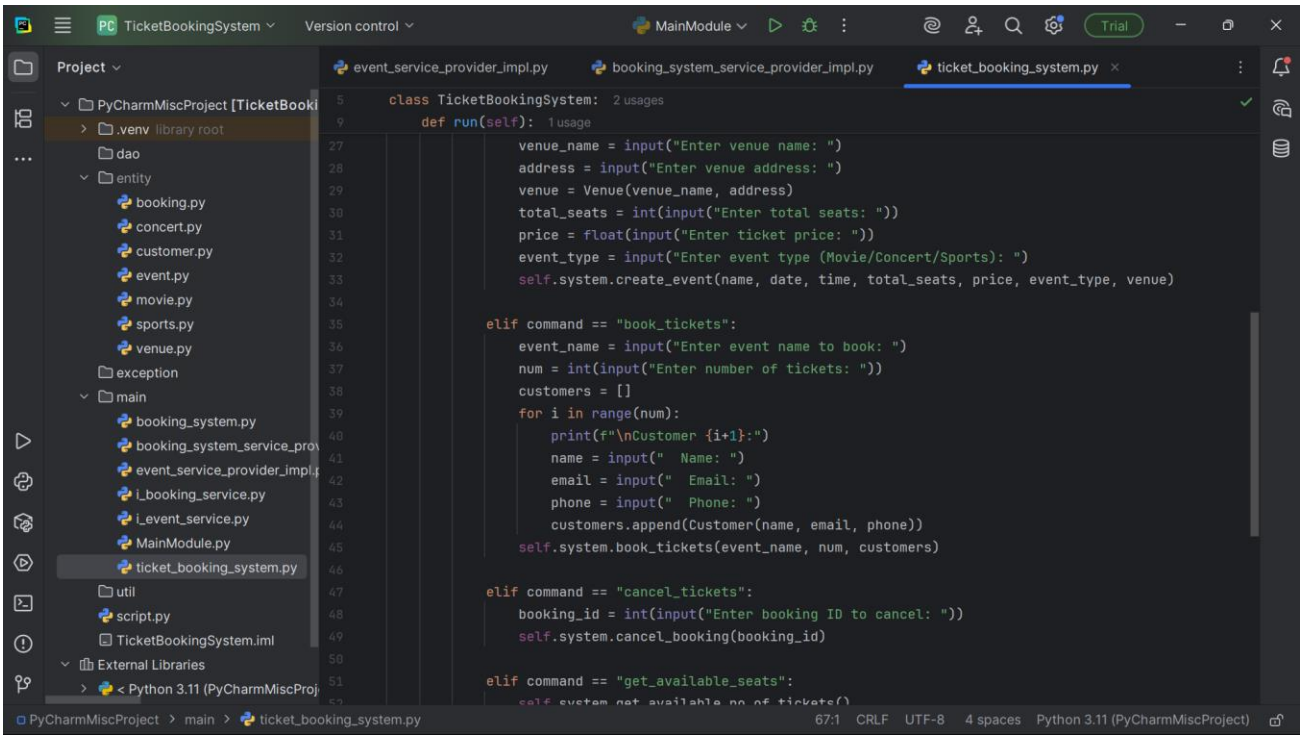
Done by
Dhamini machireddy

```
elif command == "book_tickets":
    event_name = input("Enter event name to book: ")
    num = int(input("Enter number of tickets: "))
    customers = []
    for i in range(num):
        print(f"\nCustomer {i+1}:")
        name = input("  Name: ")
        email = input("  Email: ")
        phone = input("  Phone: ")
        customers.append(Customer(name, email, phone))
    self.system.book_tickets(event_name, num, customers)

elif command == "cancel_tickets":
    booking_id = int(input("Enter booking ID to cancel: "))
    self.system.cancel_booking(booking_id)

elif command == "get_available_seats":
    self.system.get_available_no_of_tickets()

elif command == "get_event_details":
    self.system.get_event_details()

elif command == "get_booking_details":
    booking_id = int(input("Enter booking ID: "))
    self.system.get_booking_details(booking_id)

elif command == "exit":
    print(" 👋 Exiting Ticket Booking System. Goodbye!")
    break

else:
    print(" ❌ Invalid command. Try again.")
```

Done by
Dhamini machireddy

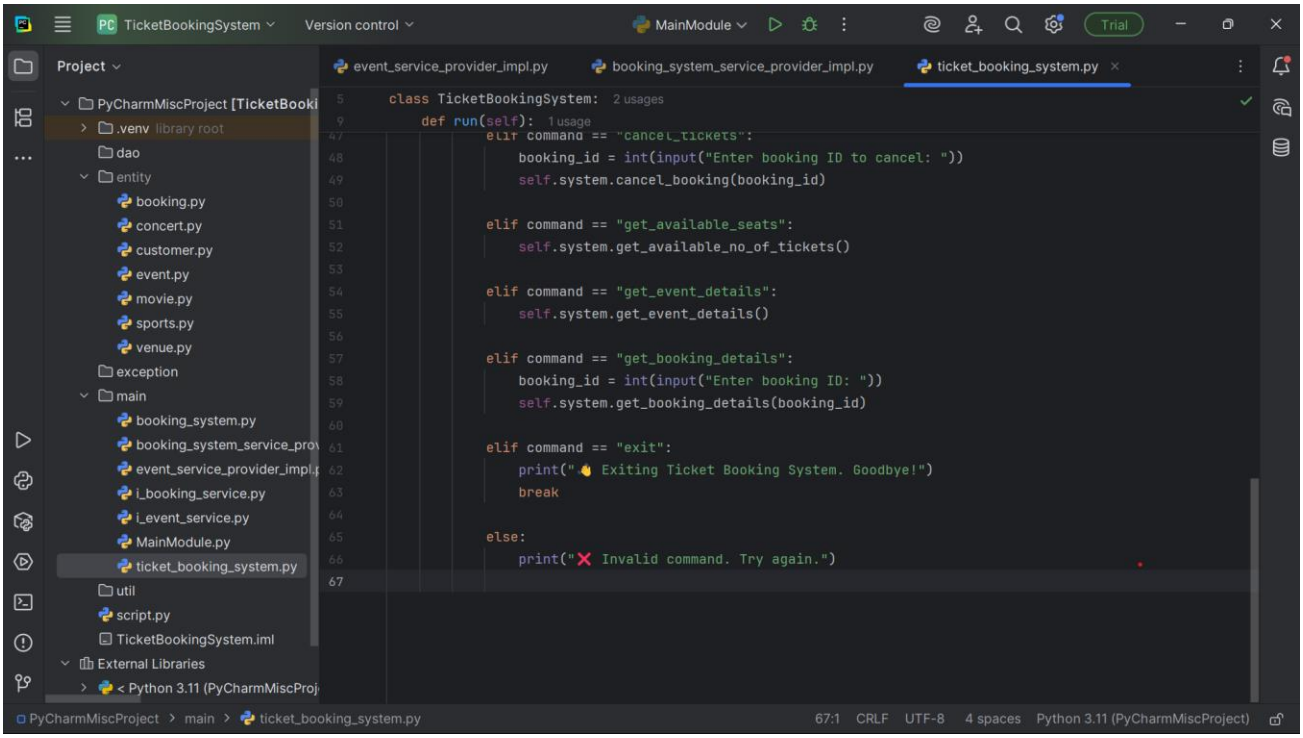


```
from main.booking_system_service_provider_impl import BookingSystemServiceProviderImpl
from entity.venue import Venue
from entity.customer import Customer

class TicketBookingSystem:  2 usages
    def __init__(self):
        self.system = BookingSystemServiceProviderImpl()

    def run(self):  1 usage
        print("\n Welcome to the Ticket Booking System ")
        while True:
            print("\n Available Commands:")
            print("1. create_event")
            print("2. book_tickets")
            print("3. cancel_tickets")
            print("4. get_available_seats")
            print("5. get_event_details")
            print("6. get_booking_details")
            print("7. exit")

            command = input("\nEnter command: ").strip().lower()

            if command == "create_event":
                name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time = input("Enter event time (HH:MM): ")
                venue_name = input("Enter venue name: ")
                address = input("Enter venue address: ")
```

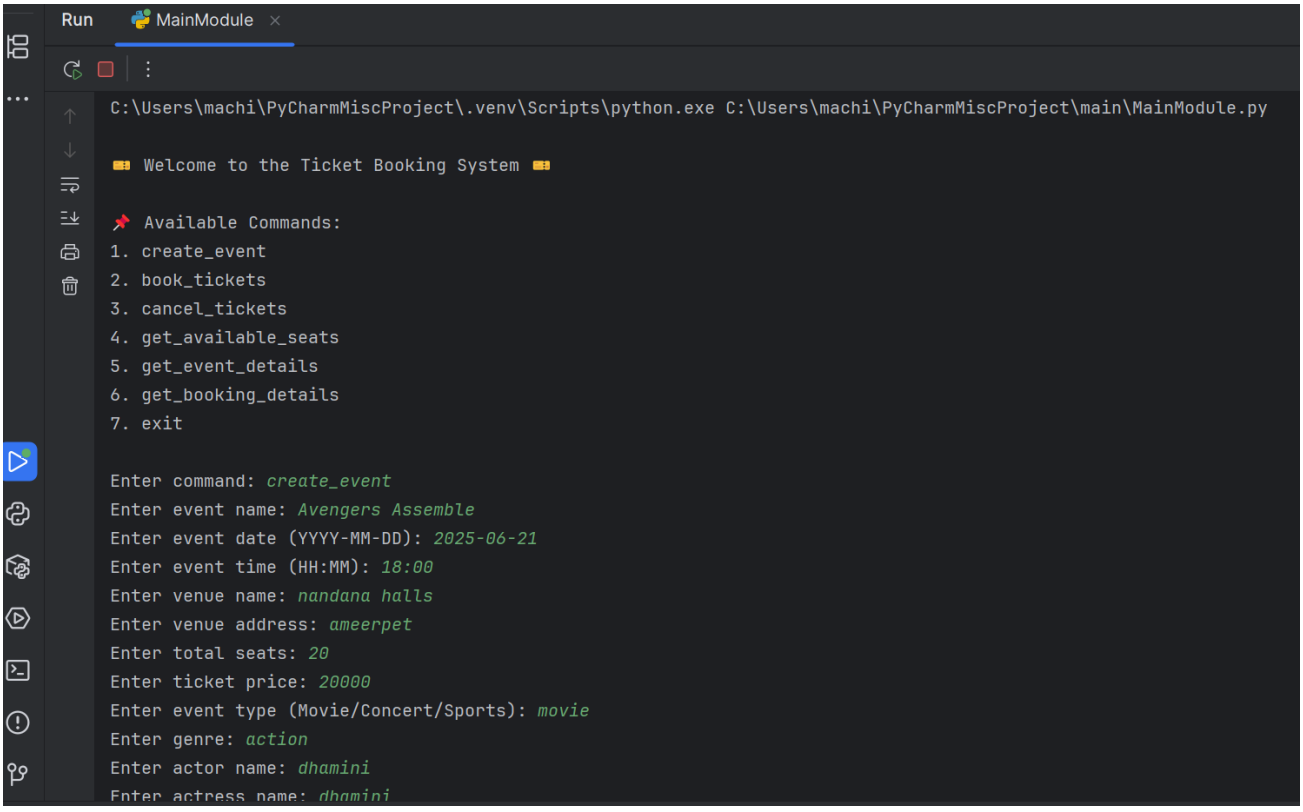


```
class TicketBookingSystem:  2 usages
    def run(self):  1 usage
                venue_name = input("Enter venue name: ")
                address = input("Enter venue address: ")
                venue = Venue(venue_name, address)
                total_seats = int(input("Enter total seats: "))
                price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (Movie/Concert/Sports): ")
                self.system.create_event(name, date, time, total_seats, price, event_type, venue)

            elif command == "book_tickets":
                event_name = input("Enter event name to book: ")
                num = int(input("Enter number of tickets: "))
                customers = []
                for i in range(num):
                    print(f"\nCustomer {i+1}:")
                    name = input("  Name: ")
                    email = input("  Email: ")
                    phone = input("  Phone: ")
                    customers.append(Customer(name, email, phone))
                self.system.book_tickets(event_name, num, customers)

            elif command == "cancel_tickets":
                booking_id = int(input("Enter booking ID to cancel: "))
                self.system.cancel_booking(booking_id)

            elif command == "get_available_seats":
```

Done by
Dhamini machireddy

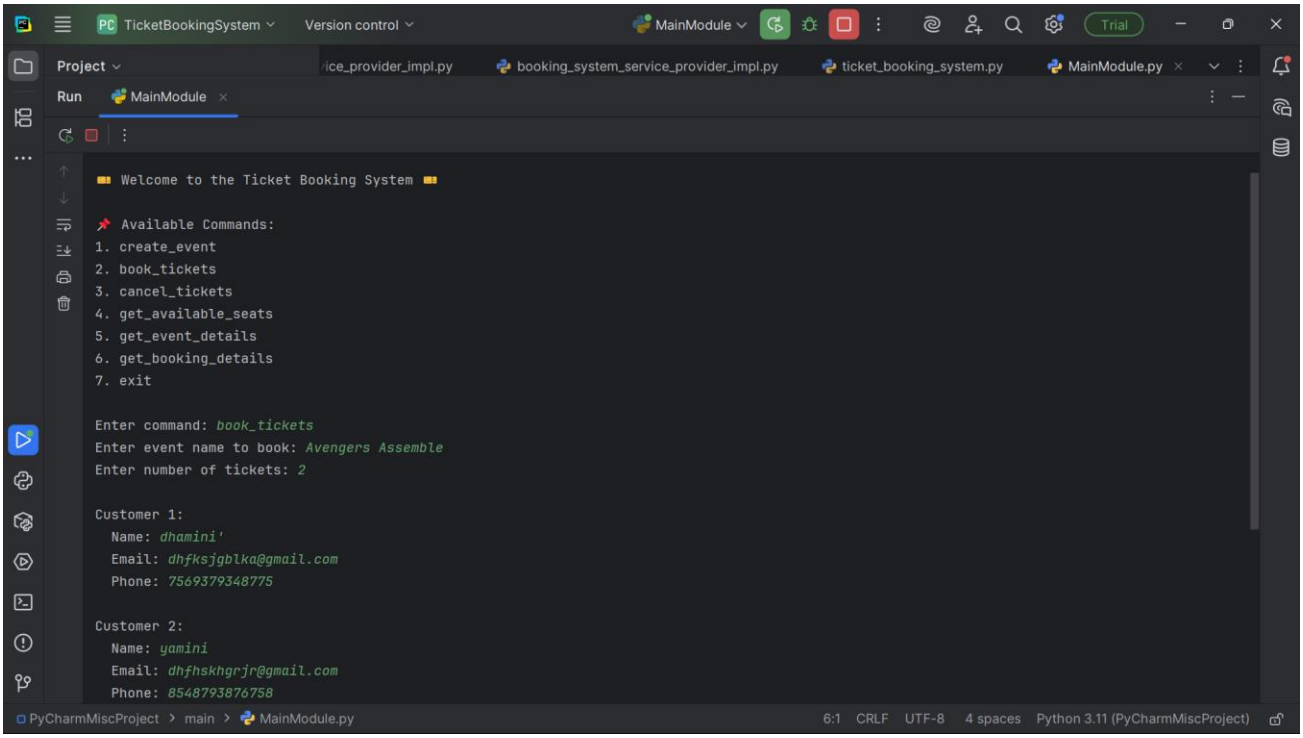


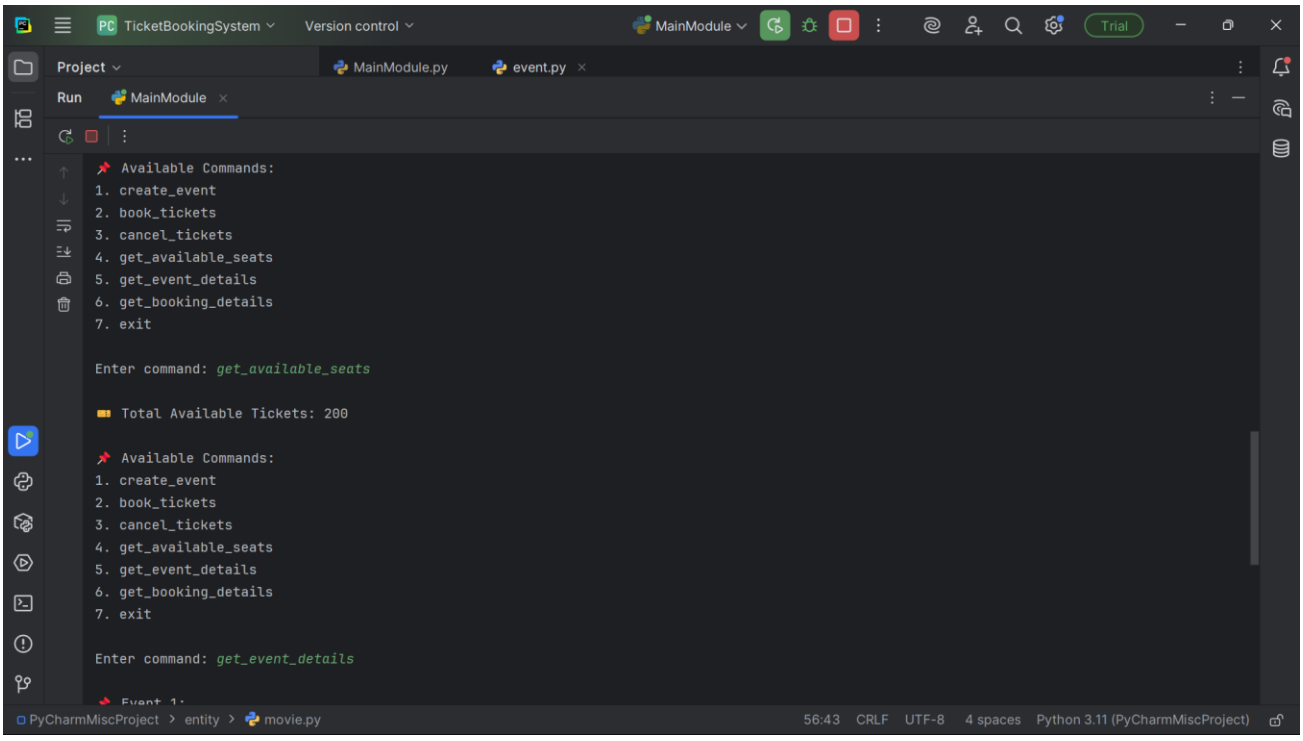
Output :



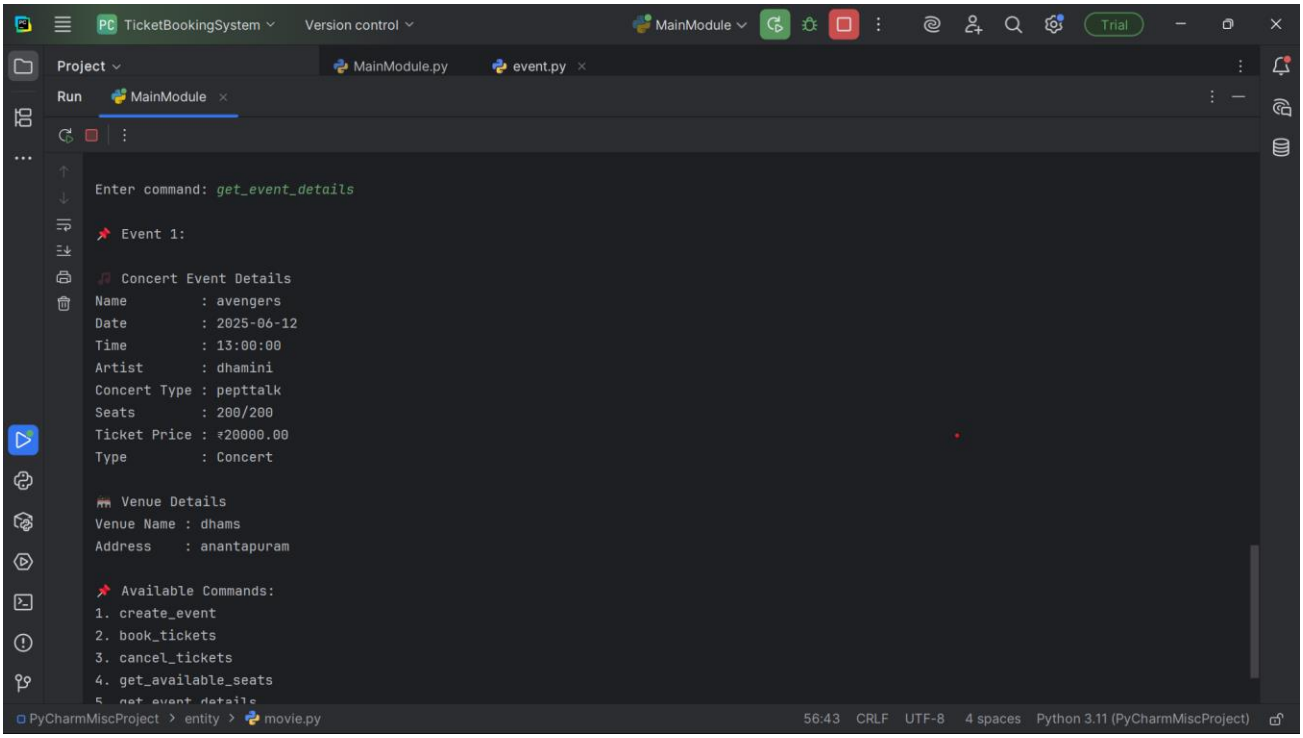
Done by

Dhamini machireddy

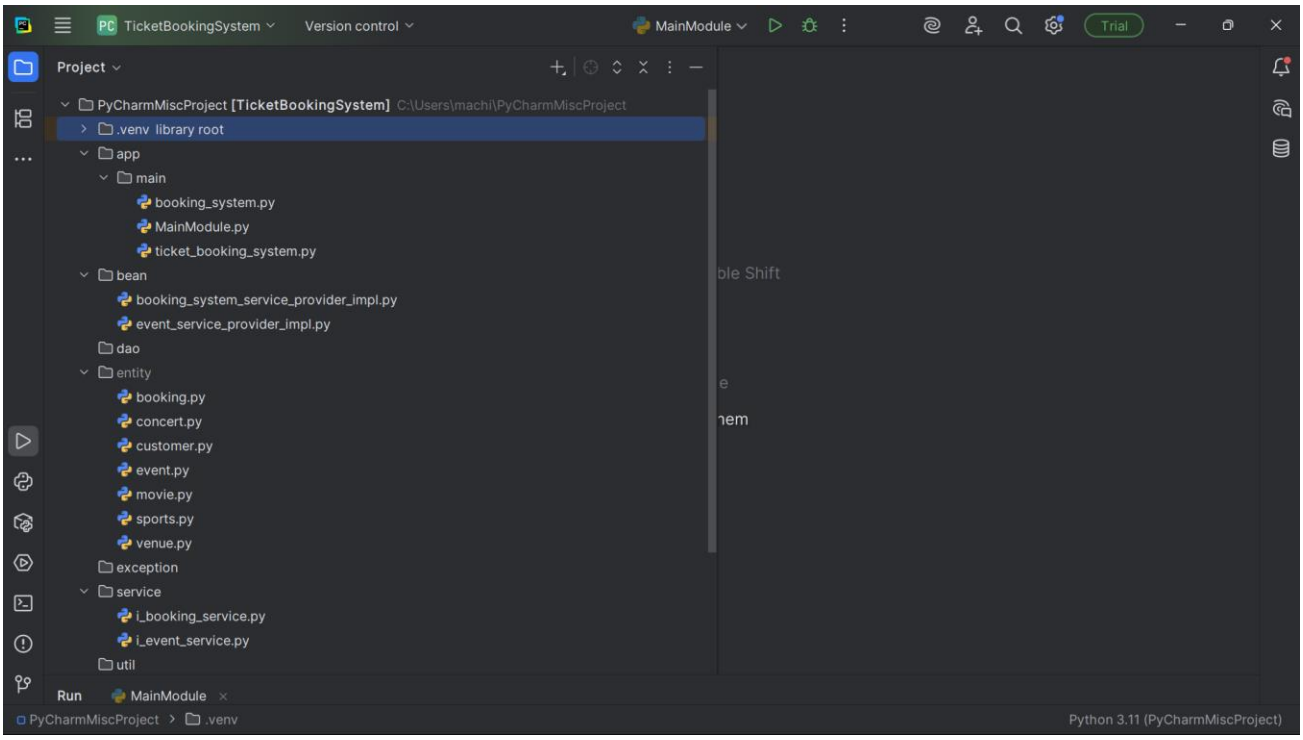




Done by
Dhamini machireddy

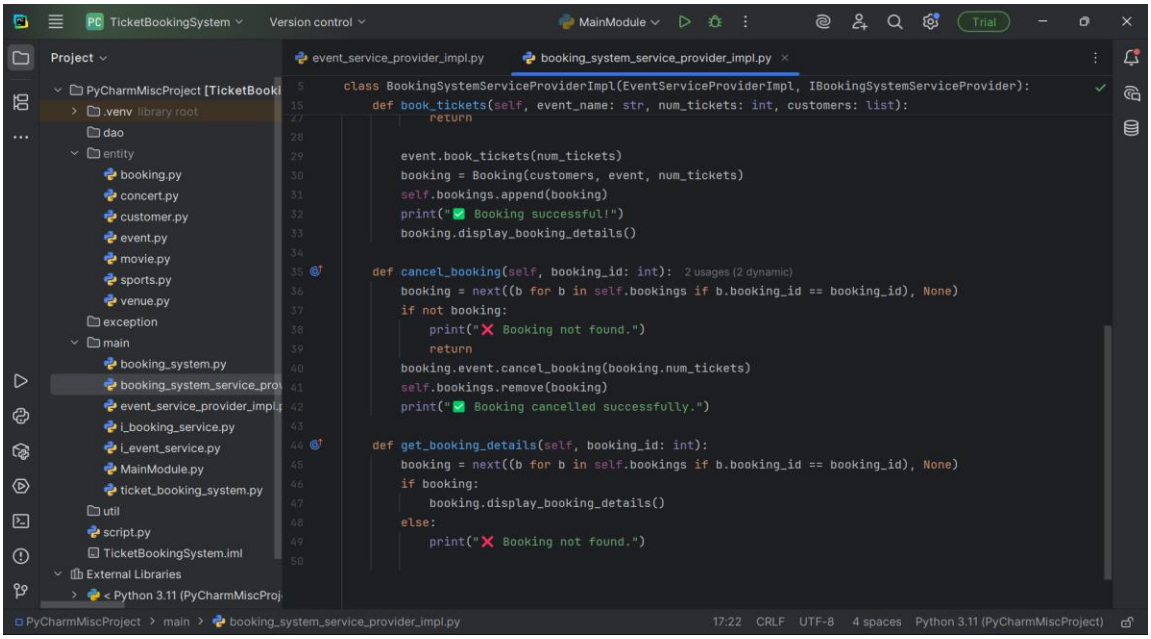


10. Place the interface/abstract class in service package and interface/abstract class implementation class, all concrete class in bean package and **TicketBookingSystem** class in app package.



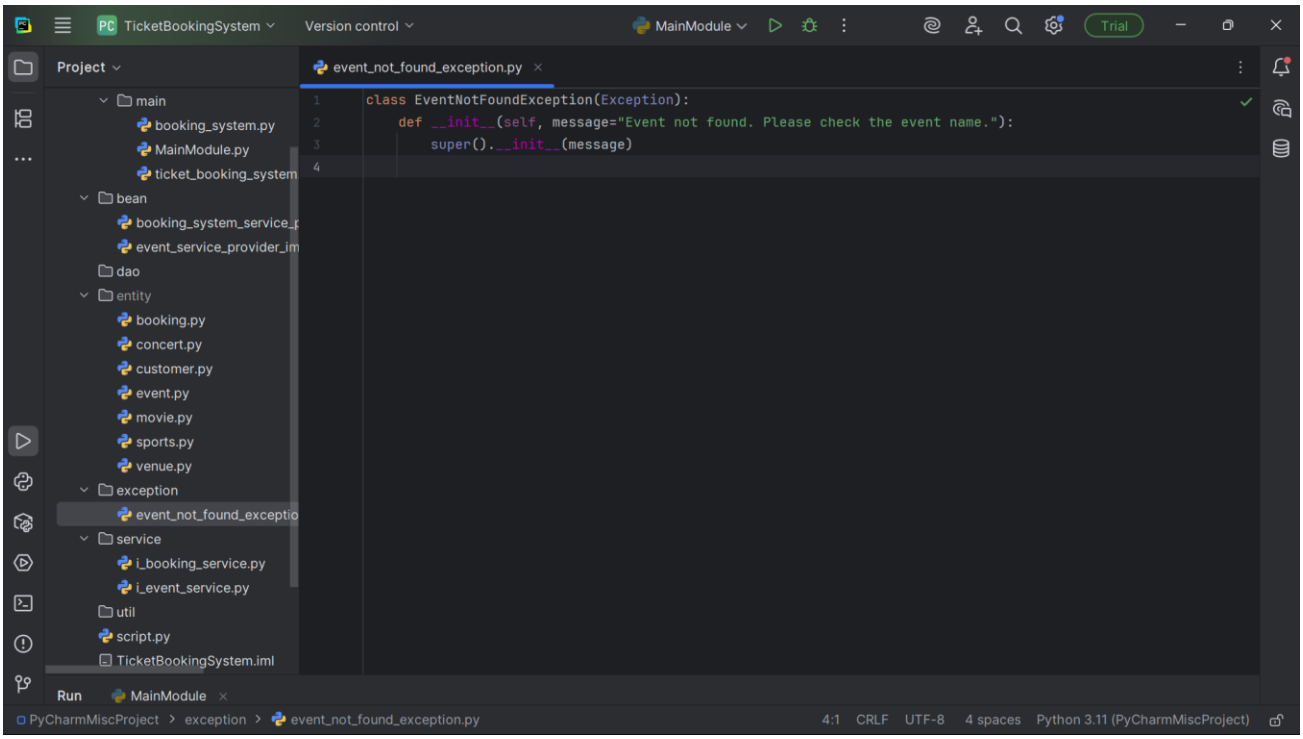
Done by
Dhamini machireddy

11. Should display appropriate message when the event or booking id is not found or any other wrong information provided.



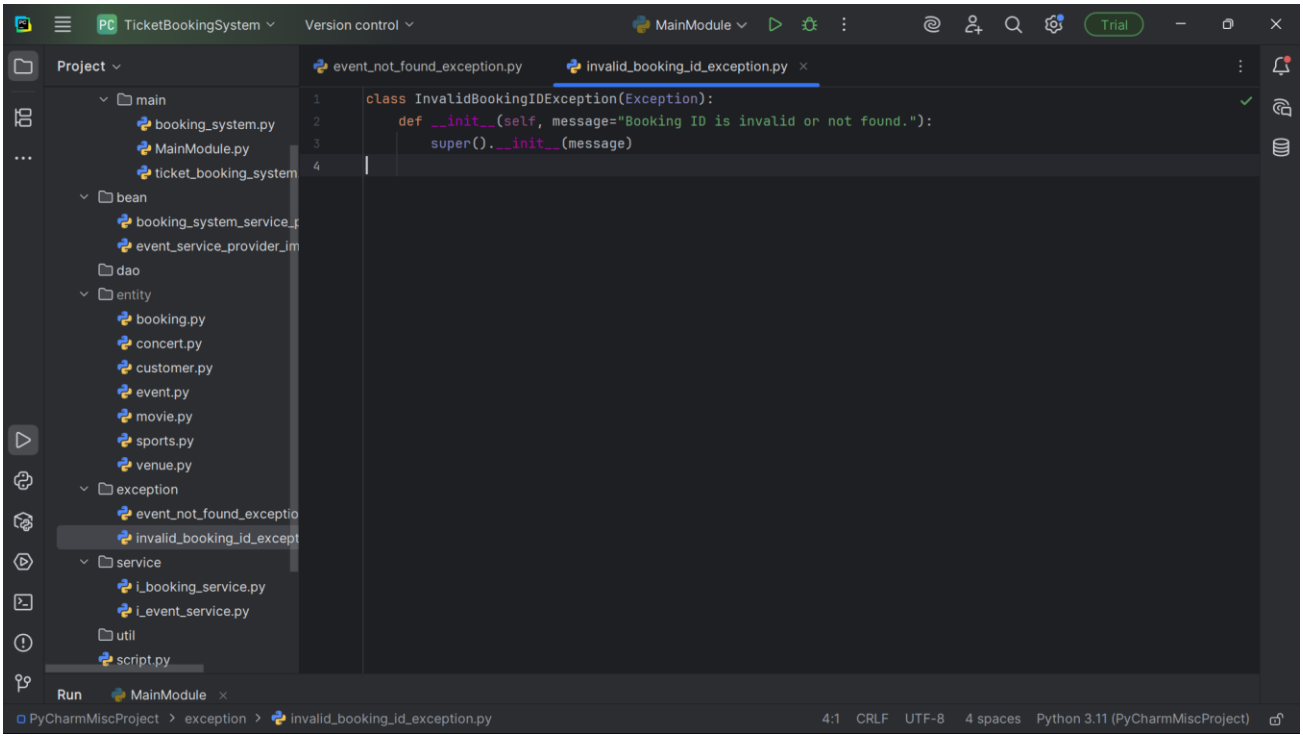
Task 9: Exception Handling throw the exception whenever needed and Handle in main method,

1. **EventNotFoundException** throw this exception when user try to book the tickets for Event not listed in the menu.



Done by
Dhamini machireddy

2. **InvalidBookingIDException** throw this exception when user entered the invalid bookingId when he tries to view the booking or cancel the booking.



3. **NullPointerException** handle in main method**.**

Throw these exceptions from the methods in **TicketBookingSystem** class. Make necessary changes to accommodate exception in the source code. Handle all these exceptions from the main program.

Code :

```
from bean.booking_system_service_provider_impl import BookingSystemServiceProviderImpl
from entity.venue import Venue
from entity.customer import Customer

# Custom exceptions
from exception.event_not_found_exception import EventNotFoundException
from exception.invalid_booking_id_exception import InvalidBookingIDException


class TicketBookingSystem:
    def __init__(self):
        self.system = BookingSystemServiceProviderImpl()

    def book_tickets(self):
        event_name = input("Enter event name to book: ")
        event = next((e for e in self.system.events if e.event_name.lower() == event_name.lower()), None)
        if not event:
```

Done by
Dhamini machireddy

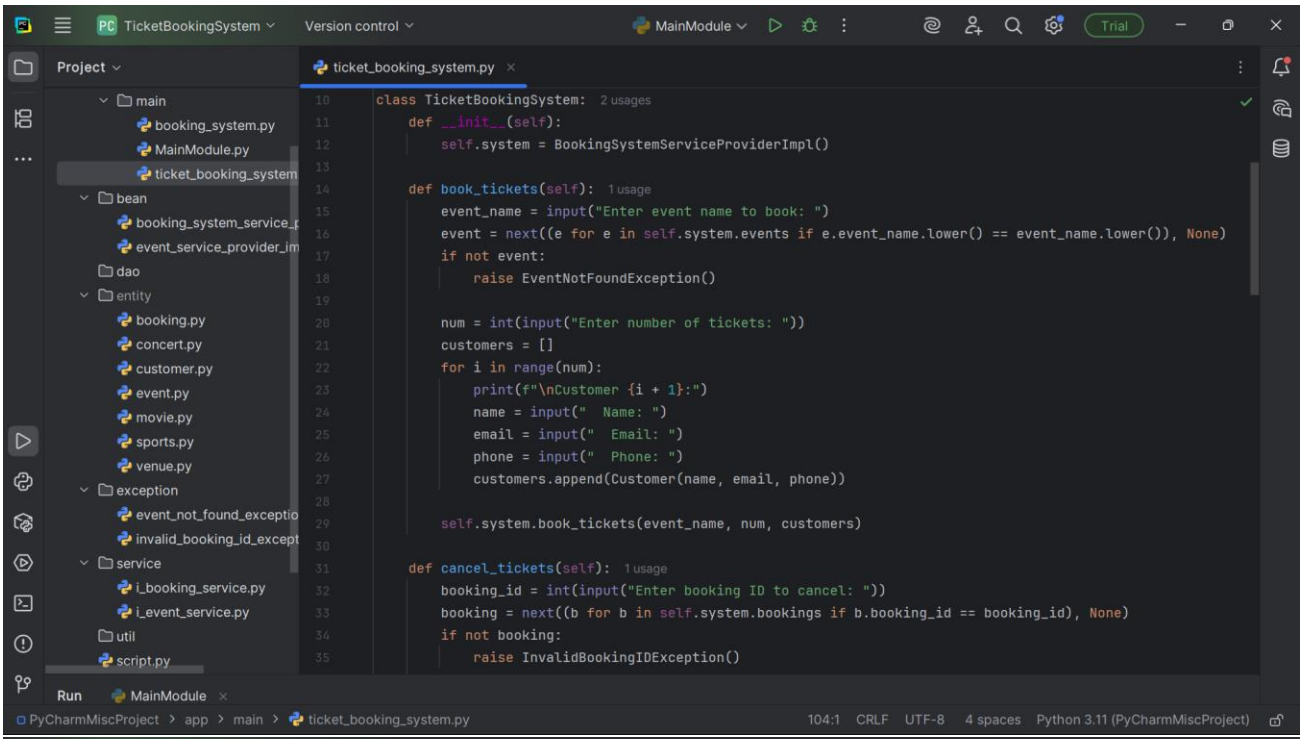
```
            raise EventNotFoundException()

        num = int(input("Enter number of tickets: "))
        customers = []
        for i in range(num):
            print(f"\nCustomer {i + 1}:")
            name = input("  Name: ")
            email = input("  Email: ")
            phone = input("  Phone: ")
            customers.append(Customer(name, email, phone))

        self.system.book_tickets(event_name, num, customers)

    def cancel_tickets(self):
        booking_id = int(input("Enter booking ID to cancel: "))
        booking = next((b for b in self.system.bookings if b.booking_id == booking_id), None)
        if not booking:
            raise InvalidBookingIDException()

        self.system.cancel_booking(booking_id)

    def get_booking_details(self):
        booking_id = int(input("Enter booking ID to view: "))
        booking = next((b for b in self.system.bookings if b.booking_id == booking_id), None)
        if not booking:
            raise InvalidBookingIDException()

        self.system.get_booking_details(booking_id)

    def run(self):
        print("\n🎟️ Welcome to the Ticket Booking System 🎟️")
        while True:
            try:
                print("\n📌 Available Commands:")
                print("1. create_event")
                print("2. book_tickets")
                print("3. cancel_tickets")
                print("4. get_available_seats")
                print("5. get_event_details")
                print("6. get_booking_details")
                print("7. exit")
```

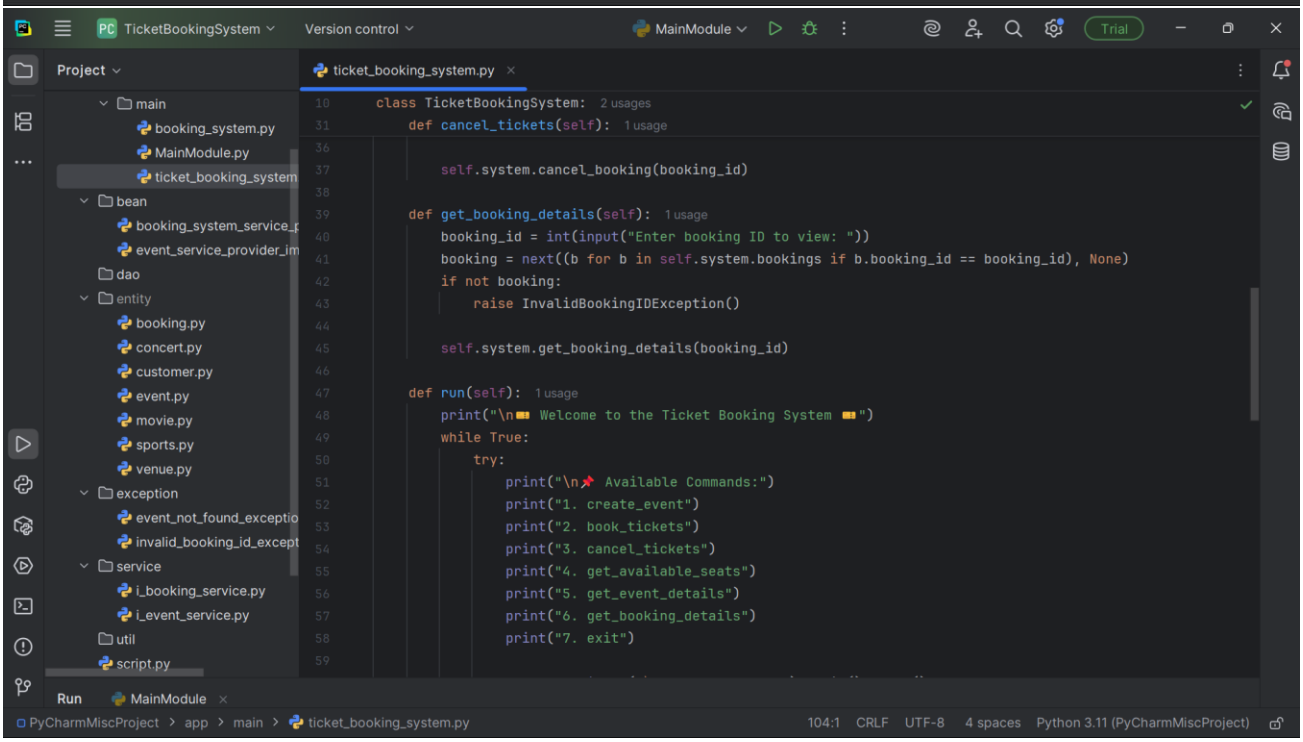
Done by
Dhamini machireddy

```
            command = input("\nEnter command: ").strip().lower()

            if command == "create_event":
                name = input("Enter event name: ")
                date = input("Enter event date (YYYY-MM-DD): ")
                time = input("Enter event time (HH:MM): ")
                venue_name = input("Enter venue name: ")
                address = input("Enter venue address: ")
                venue = Venue(venue_name, address)
                total_seats = int(input("Enter total seats: "))
                price = float(input("Enter ticket price: "))
                event_type = input("Enter event type (Movie/Concert/Sports): ")
                self.system.create_event(name, date, time, total_seats, price, event_type, venue)

            elif command == "book_tickets":
                self.book_tickets()

            elif command == "cancel_tickets":
                self.cancel_tickets()

            elif command == "get_available_seats":
                self.system.get_available_no_of_tickets()

            elif command == "get_event_details":
                self.system.get_event_details()

            elif command == "get_booking_details":
                self.get_booking_details()

            elif command == "exit":
                print("👋 Exiting Ticket Booking System. Goodbye!")
                break

            else:
                print("❌ Invalid command. Please try again.")

        except EventNotFoundException as e:
            print(f"❌ {e}")
        except InvalidBookingIDException as e:
            print(f"❌ {e}")
```

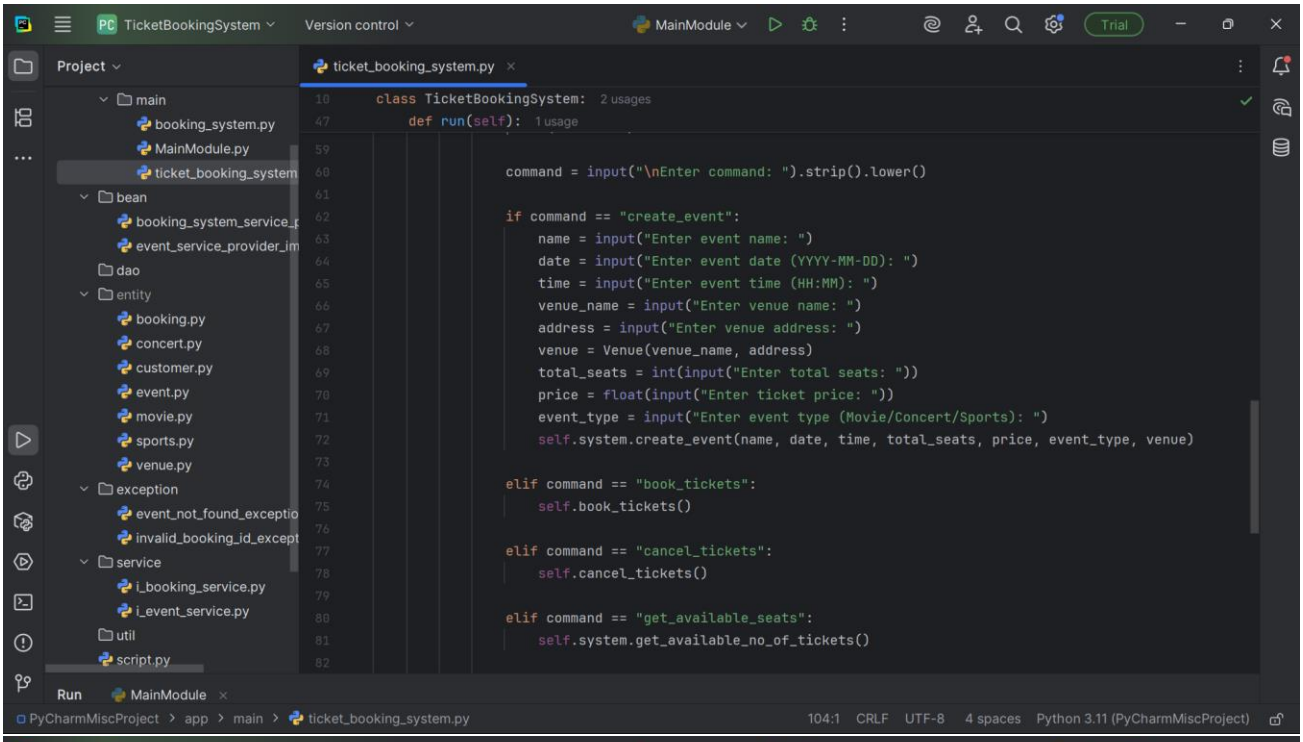
Done by
Dhamini machireddy

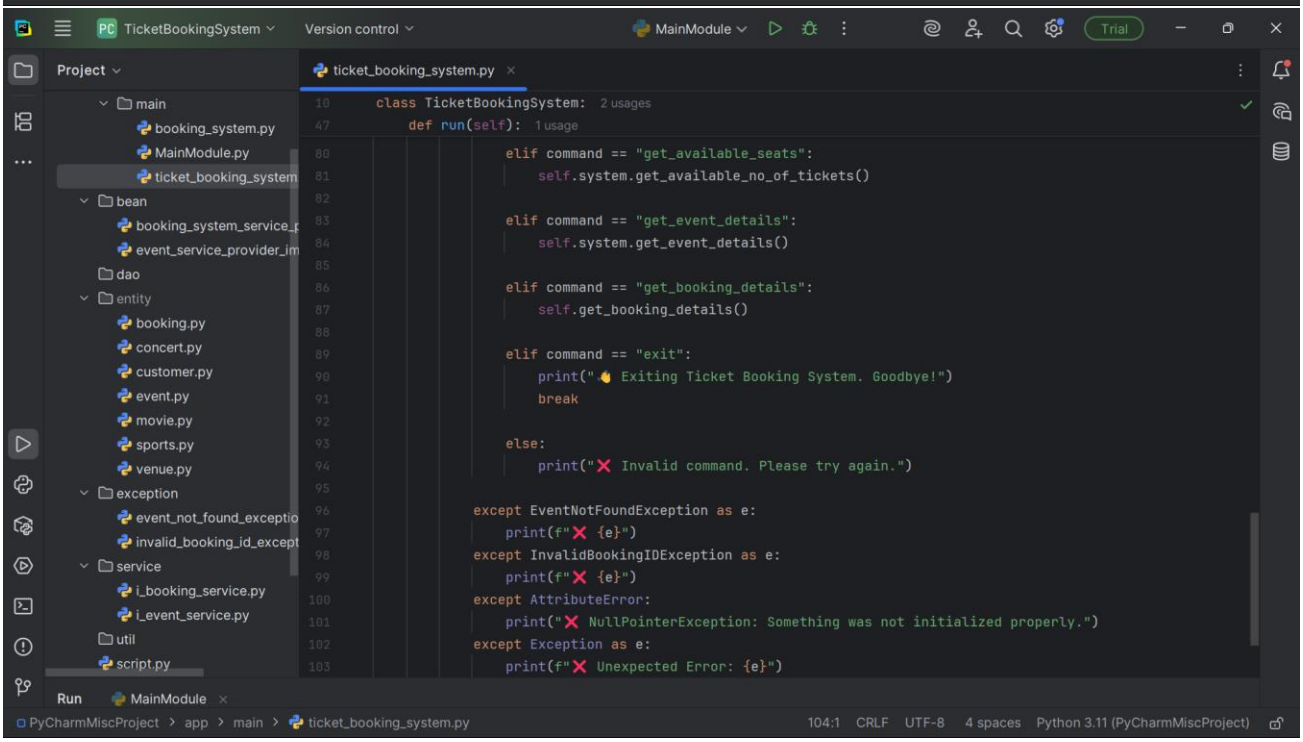
```
except AttributeError:
    print("❌ NullPointerException: Something was not initialized properly.")
except Exception as e:
    print(f"❌ Unexpected Error: {e}")
```





Done by
Dhamini machireddy





Task 10: Collection

1. From the previous task change the **Booking** class attribute customers to List of customers and **BookingSystem** class attribute events to List of events and perform the same operation.
   a. **Booking** class attribute customers to List of customers

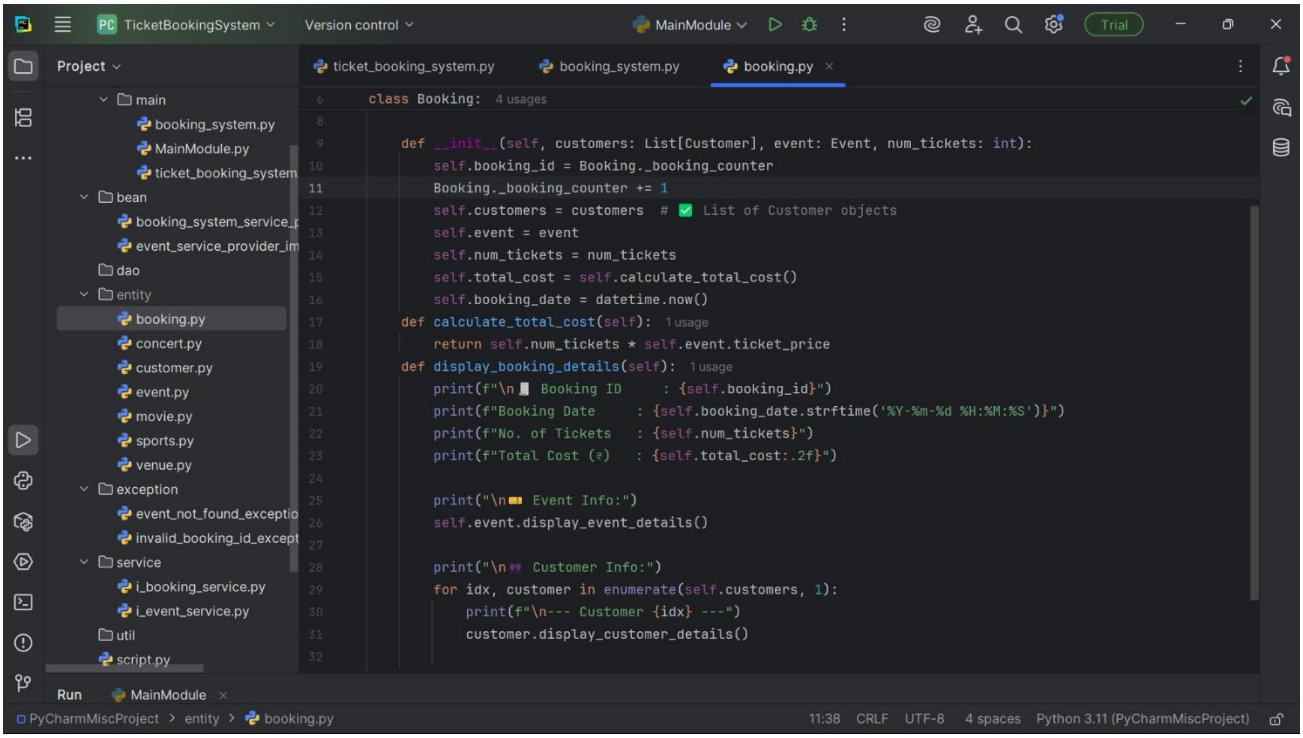
   Code :

   from datetime import datetime

Done by
Dhamini machireddy

```
from typing import List
from entity.customer import Customer
from entity.event import Event

class Booking:
    _booking_counter = 1  # static booking ID

    def __init__(self, customers: List[Customer], event: Event, num_tickets: int):
        self.booking_id = Booking._booking_counter
        Booking._booking_counter += 1

        self.customers = customers  # ☑ List of Customer objects
        self.event = event
        self.num_tickets = num_tickets
        self.total_cost = self.calculate_total_cost()
        self.booking_date = datetime.now()

    def calculate_total_cost(self):
        return self.num_tickets * self.event.ticket_price

    def display_booking_details(self):
        print(f"\n📄 Booking ID     : {self.booking_id}")
        print(f"Booking Date     : {self.booking_date.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"No. of Tickets   : {self.num_tickets}")
        print(f"Total Cost (₹)   : {self.total_cost:.2f}")

        print("\n🎟️ Event Info:")
        self.event.display_event_details()

        print("\n👥 Customer Info:")
        for idx, customer in enumerate(self.customers, 1):
            print(f"\n--- Customer {idx} ---")
            customer.display_customer_details()
```

Done by
Dhamini machireddy



b. **BookingSystem** class attribute events to List of events and perform the same operation.

Code

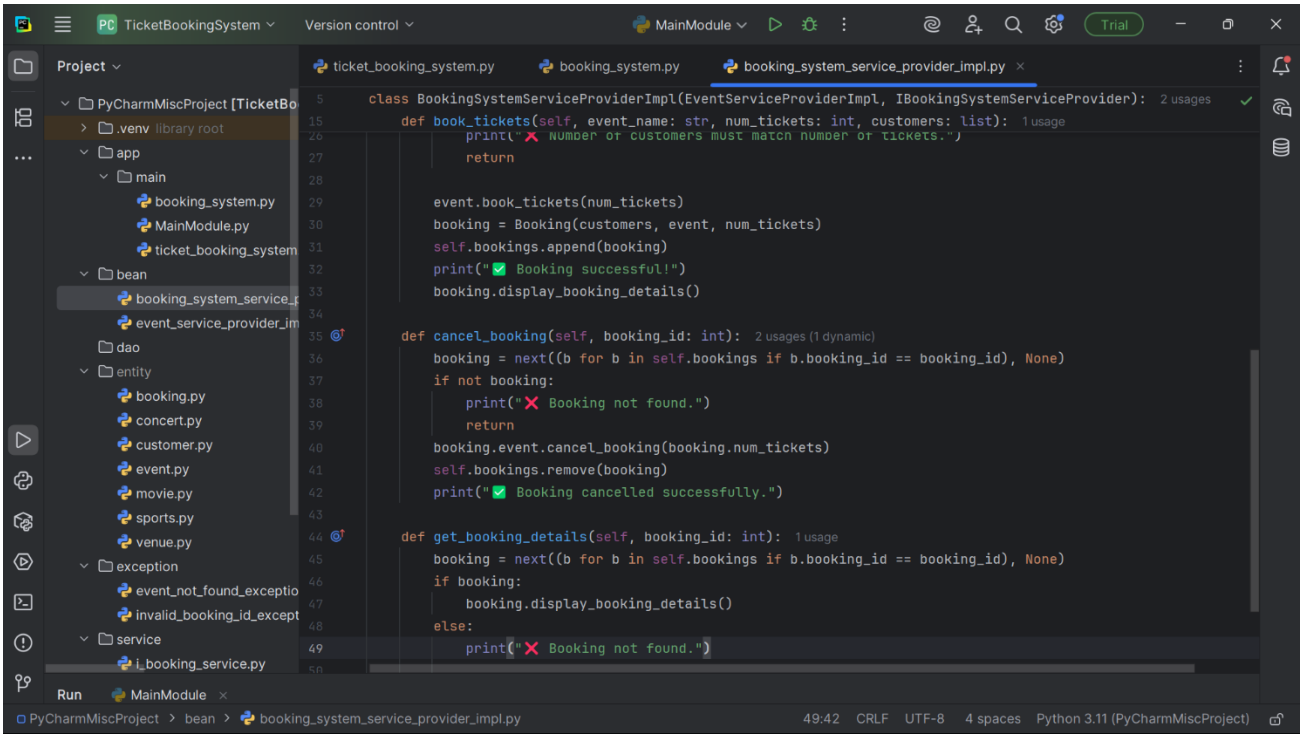
main/booking_system_service_provider_impl.py

```
from service.i_booking_service import IBookingSystemServiceProvider
from bean.event_service_provider_impl import EventServiceProviderImpl
from entity.booking import Booking
class BookingSystemServiceProviderImpl(EventServiceProviderImpl,
IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()  # Initializes self.events from parent
        self.bookings = []  # Booking list

    def calculate_booking_cost(self, num_tickets: int) -> float:
        # Assumes one event context; usually handled during booking
        print(f"🔢 Cost for {num_tickets} tickets is calculated based on event during booking.")
        return 0.0

    def book_tickets(self, event_name: str, num_tickets: int, customers: list):
        event = next((e for e in self.events if e.event_name.lower() == event_name.lower()),
None)
        if not event:
            print("❌ Event not found.")
```

Done by
Dhamini machireddy

```
            return

        if event.available_seats < num_tickets:
            print("❌ Not enough tickets available.")
            return

        if len(customers) != num_tickets:
            print("❌ Number of customers must match number of tickets.")
            return

        event.book_tickets(num_tickets)
        booking = Booking(customers, event, num_tickets)
        self.bookings.append(booking)
        print("☑ Booking successful!")
        booking.display_booking_details()

    def cancel_booking(self, booking_id: int):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if not booking:
            print("❌ Booking not found.")
            return
        booking.event.cancel_booking(booking.num_tickets)
        self.bookings.remove(booking)
        print("☑ Booking cancelled successfully.")

    def get_booking_details(self, booking_id: int):
        booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
        if booking:
            booking.display_booking_details()
        else:
            print("❌ Booking not found.")
```

Done by
Dhamini machireddy



2. From the previous task change all list type of attribute to type Set in **Booking** and **BookingSystem** class and perform the same operation.
    - Avoid adding duplicate Account object to the set.
    - Create Comparator<Event> object to sort the event based on event name and location in alphabetical order.

    Code:

```
from typing import Set

class Booking:
    def __init__(self, customers: Set[Customer], event: Event, num_tickets: int):
        self.customers = customers  # ☑ Now a set
self.bookings.append(Booking(set(customers), event, num_tickets))
        from typing import Set

class BookingSystemServiceProviderImpl:
    def __init__(self):
        self.events: Set[Event] = set()    # ☑ Set of events
        self.bookings = []
def __eq__(self, other):
    return isinstance(other, Customer) and self.email == other.email

def __hash__(self):
```

Done by
Dhamini machireddy

```
        return hash(self.email)
    def __eq__(self, other):
        return isinstance(other, Event) and self.event_name.lower() == other.event_name.lower() and
    self.event_date == other.event_date

    def __hash__(self):
        return hash((self.event_name.lower(), self.event_date))
    sorted_events = sorted(
        self.events,
        key=lambda e: (e.event_name.lower(), e.venue.venue_name.lower())
    )

    for event in sorted_events:
        event.display_event_details()
```

3. From the previous task change all list type of attribute to type Map object in **Booking** and **BookingSystem** class and perform the same operation.

Code :

```
from typing import Dict
from entity.customer import Customer

class Booking:
    def __init__(self, customers: Dict[int, Customer], event, num_tickets: int):
        self.customers = customers  # ☑ Map: ticket_no → Customer

# Convert list to dict {1: customer1, 2: customer2, ...}
customer_dict = {i+1: customers[i] for i in range(len(customers))}
booking = Booking(customer_dict, event, num_tickets)

for ticket_no, customer in self.customers.items():
    print(f"\n--- Ticket {ticket_no} ---")
    customer.display_customer_details()

self.events: Dict[str, Event] = {}  # ☑ event_name.lower() → Event

event = self.events.get(event_name.lower())

 self.events[event.event_name.lower()] = event
```

Done by
Dhamini machireddy

```
sorted_events = sorted(self.events.values(), key=lambda e: (e.event_name.lower(),
e.venue.venue_name.lower()))
for idx, event in enumerate(sorted_events, 1):
    event.display_event_details()
```

made all the changes

Task 11: Database Connectivity.

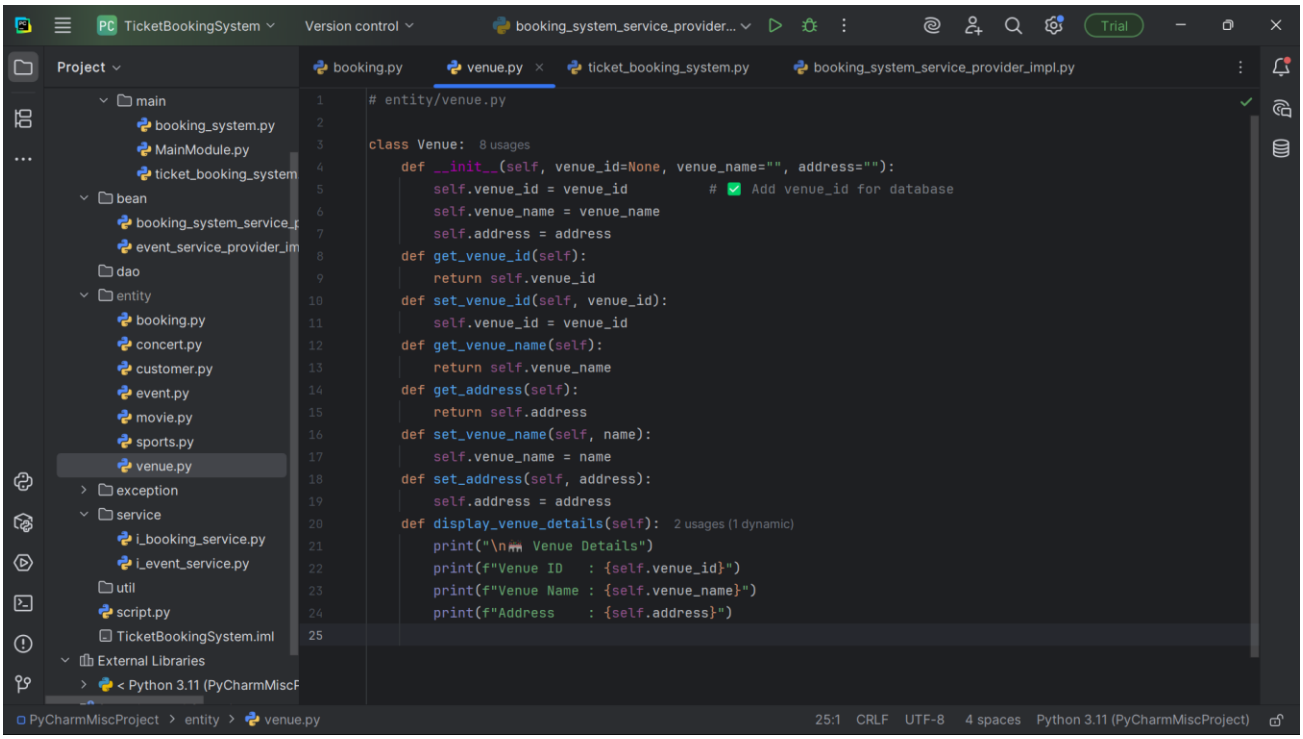
1. Create **Venue, Event, Customer** and **Booking** class as mentioned above Task 5.

Venue

Code :

```
# entity/venue.py

class Venue:
    def __init__(self, venue_id=None, venue_name="", address=""):
        self.venue_id = venue_id        # ☑ Add venue_id for database
        self.venue_name = venue_name
        self.address = address

    def get_venue_id(self):
        return self.venue_id

    def set_venue_id(self, venue_id):
        self.venue_id = venue_id

    def get_venue_name(self):
        return self.venue_name

    def get_address(self):
        return self.address

    def set_venue_name(self, name):
        self.venue_name = name

    def set_address(self, address):
        self.address = address

    def display_venue_details(self):
        print("\n🏟 Venue Details")
        print(f"Venue ID   : {self.venue_id}")
        print(f"Venue Name : {self.venue_name}")
        print(f"Address    : {self.address}")
```

Done by
Dhamini machireddy



Event :

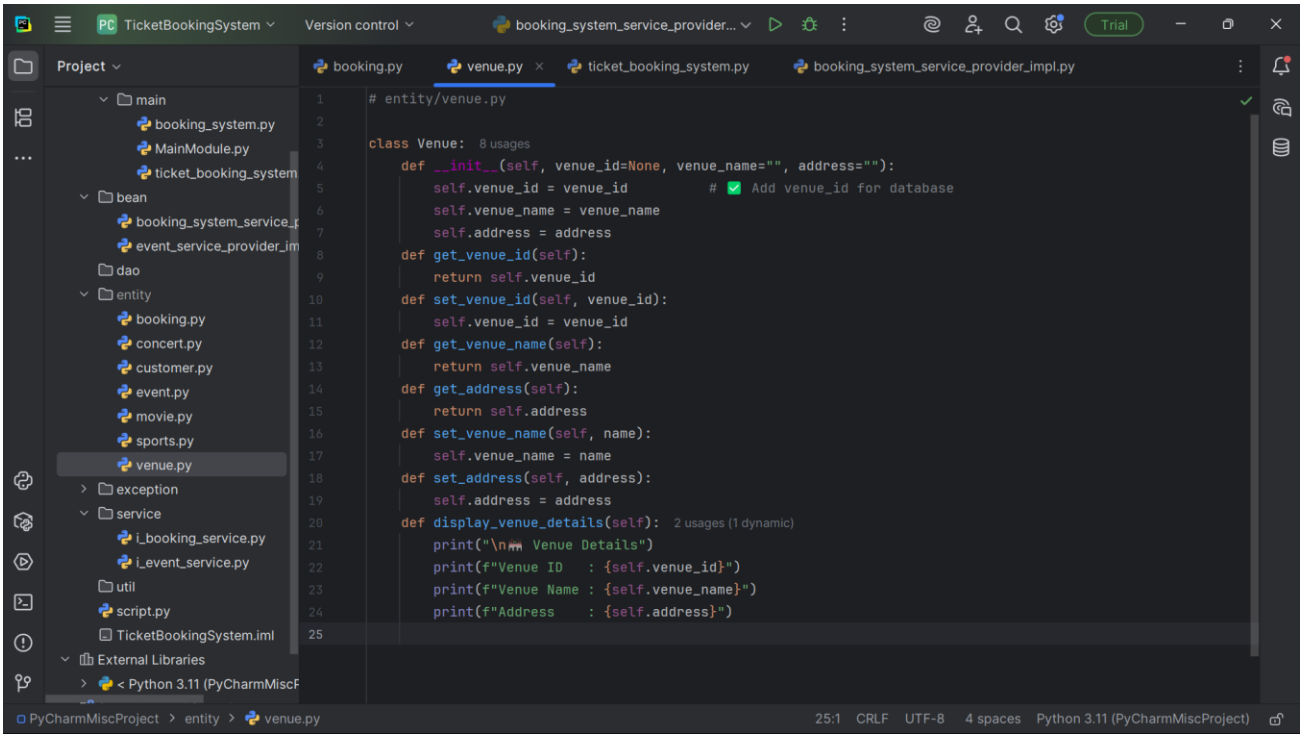
```
# entity/event.py
from datetime import date, time
from entity.venue import Venue

class Event:
    def __init__(self, event_id=None, event_name="", event_date=None, event_time=None,
                 venue=None, total_seats=0, available_seats=None,
                 ticket_price=0.0, event_type=""):
        self.event_id = event_id                    # ☑ Add event_id
        self.event_name = event_name
        self.event_date = event_date or date.today()
        self.event_time = event_time or time(hour=0, minute=0)
        self.venue = venue or Venue()
        self.total_seats = total_seats
        self.available_seats = available_seats if available_seats is not None else total_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    # ---------- Getters ----------
    def get_event_id(self):
        return self.event_id
```

Done by
Dhamini machireddy

```
def get_event_name(self):
    return self.event_name

def get_event_date(self):
    return self.event_date

def get_event_time(self):
    return self.event_time

def get_venue(self):
    return self.venue

def get_total_seats(self):
    return self.total_seats

def get_available_seats(self):
    return self.available_seats

def get_ticket_price(self):
    return self.ticket_price

def get_event_type(self):
    return self.event_type

# ---------- Setters ----------
def set_event_id(self, eid):
    self.event_id = eid

def set_event_name(self, name):
    self.event_name = name

def set_event_date(self, event_date):
    self.event_date = event_date

def set_event_time(self, event_time):
    self.event_time = event_time

def set_venue(self, venue):
    self.venue = venue

def set_total_seats(self, seats):
    self.total_seats = seats
```

Done by
Dhamini machireddy

```
def set_available_seats(self, seats):
    self.available_seats = seats

def set_ticket_price(self, price):
    self.ticket_price = price

def set_event_type(self, e_type):
    self.event_type = e_type

# ---------- Display ----------
def display_event_details(self):
    print("\n Event Details")
    print(f"Event ID    : {self.event_id}")
    print(f"Name        : {self.event_name}")
    print(f"Date        : {self.event_date}")
    print(f"Time        : {self.event_time}")
    print(f"Type        : {self.event_type}")
    print(f"Seats       : {self.available_seats}/{self.total_seats}")
    print(f"Ticket Price: ₹{self.ticket_price:.2f}")
    self.venue.display_venue_details()
```



Customer :

Code :

entity/customer.py

Done by
Dhamini machireddy

```
class Customer:
    def __init__(self, customer_id=None, customer_name="", email="", phone_number=""):
        self.customer_id = customer_id       # ☑ Add this
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number

    # ---------- Getters ----------
    def get_customer_id(self):
        return self.customer_id

    def get_customer_name(self):
        return self.customer_name

    def get_email(self):
        return self.email

    def get_phone_number(self):
        return self.phone_number

    # ---------- Setters ----------
    def set_customer_id(self, cid):
        self.customer_id = cid

    def set_customer_name(self, name):
        self.customer_name = name

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

    def set_phone_number(self, phone_number):
        self.phone_number = phone_number

    # ---------- Display ----------
    def display_customer_details(self):
        print("\n👤 Customer Details")
        print(f"Customer ID  : {self.customer_id}")
        print(f"Name         : {self.customer_name}")
        print(f"Email        : {self.email}")
        print(f"Phone Number : {self.phone_number}")
```

Done by
Dhamini machireddy

```
class Customer:  4 usages
    def __init__(self, customer_id=None, customer_name="", email="", phone_number=""):
        self.customer_id = customer_id        # ✅ Add this
        self.customer_name = customer_name
        self.email = email
        self.phone_number = phone_number
```

Booking :

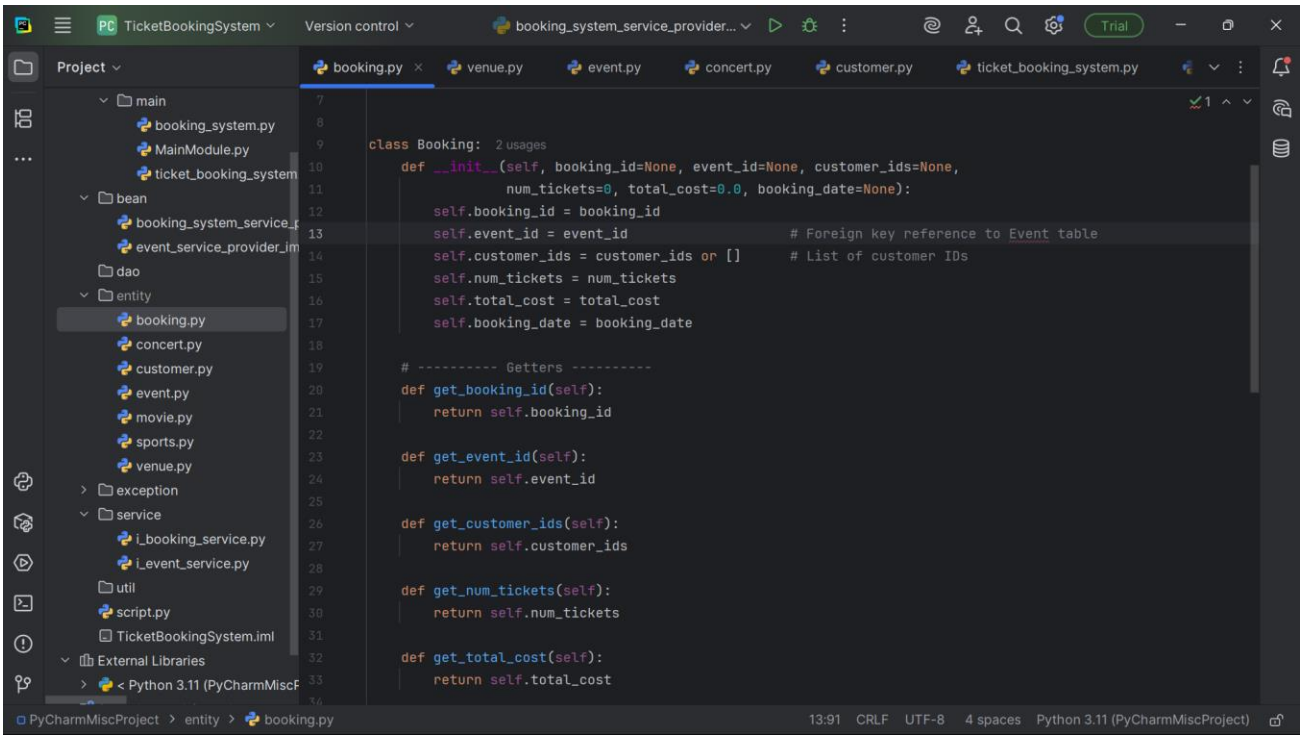
Code :

```
# entity/booking.py

class Booking:
    def __init__(self, booking_id=None, event_id=None, customer_ids=None,
                 num_tickets=0, total_cost=0.0, booking_date=None):
        self.booking_id = booking_id
        self.event_id = event_id                # Foreign key reference to Event table
        self.customer_ids = customer_ids or []      # List of customer IDs
        self.num_tickets = num_tickets
        self.total_cost = total_cost
        self.booking_date = booking_date

    # ---------- Getters ----------
    def get_booking_id(self):
        return self.booking_id

    def get_event_id(self):
        return self.event_id

    def get_customer_ids(self):
        return self.customer_ids

    def get_num_tickets(self):
        return self.num_tickets

    def get_total_cost(self):
        return self.total_cost

    def get_booking_date(self):
        return self.booking_date

    # ---------- Setters ----------
    def set_booking_id(self, bid):
        self.booking_id = bid
```

Done by
Dhamini machireddy

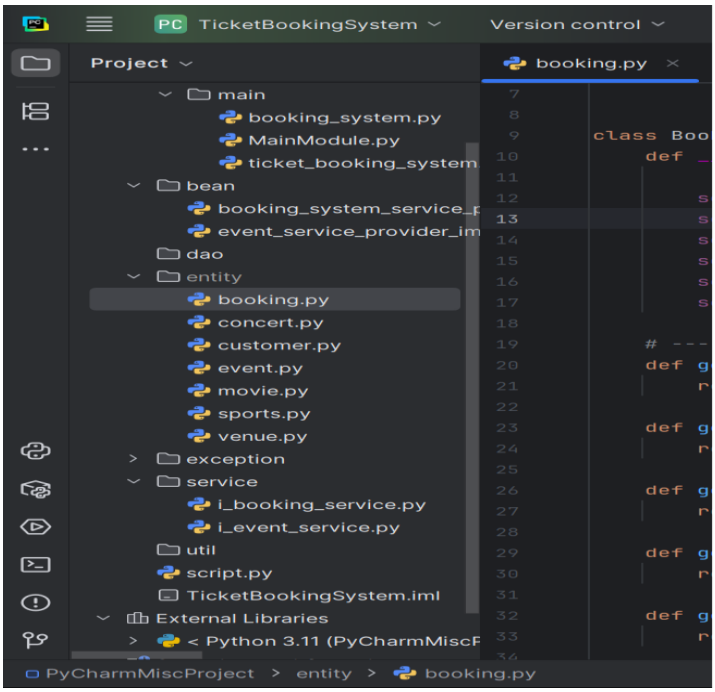
```
def set_event_id(self, eid):
    self.event_id = eid

def set_customer_ids(self, ids):
    self.customer_ids = ids

def set_num_tickets(self, tickets):
    self.num_tickets = tickets

def set_total_cost(self, cost):
    self.total_cost = cost

def set_booking_date(self, b_date):
    self.booking_date = b_date

# ---------- Display ----------
def display_booking_details(self):
    print("\n📦 Booking Details")
    print(f"Booking ID    : {self.booking_id}")
    print(f"Event ID      : {self.event_id}")
    print(f"Customer IDs  : {', '.join(map(str, self.customer_ids))}")
    print(f"No. of Tickets: {self.num_tickets}")
    print(f"Total Cost    : ₹{self.total_cost:.2f}")
    print(f"Booking Date  : {self.booking_date}")
```

Done by
Dhamini machireddy



2. Create **Event** sub classes as mentioned in above Task 4**.**

   Performed in the previous steps



3. Create interface/abstract class **IEventServiceProvider, IBookingSystemServiceProvider** and its implementation classes as mentioned in above Task 5**.**

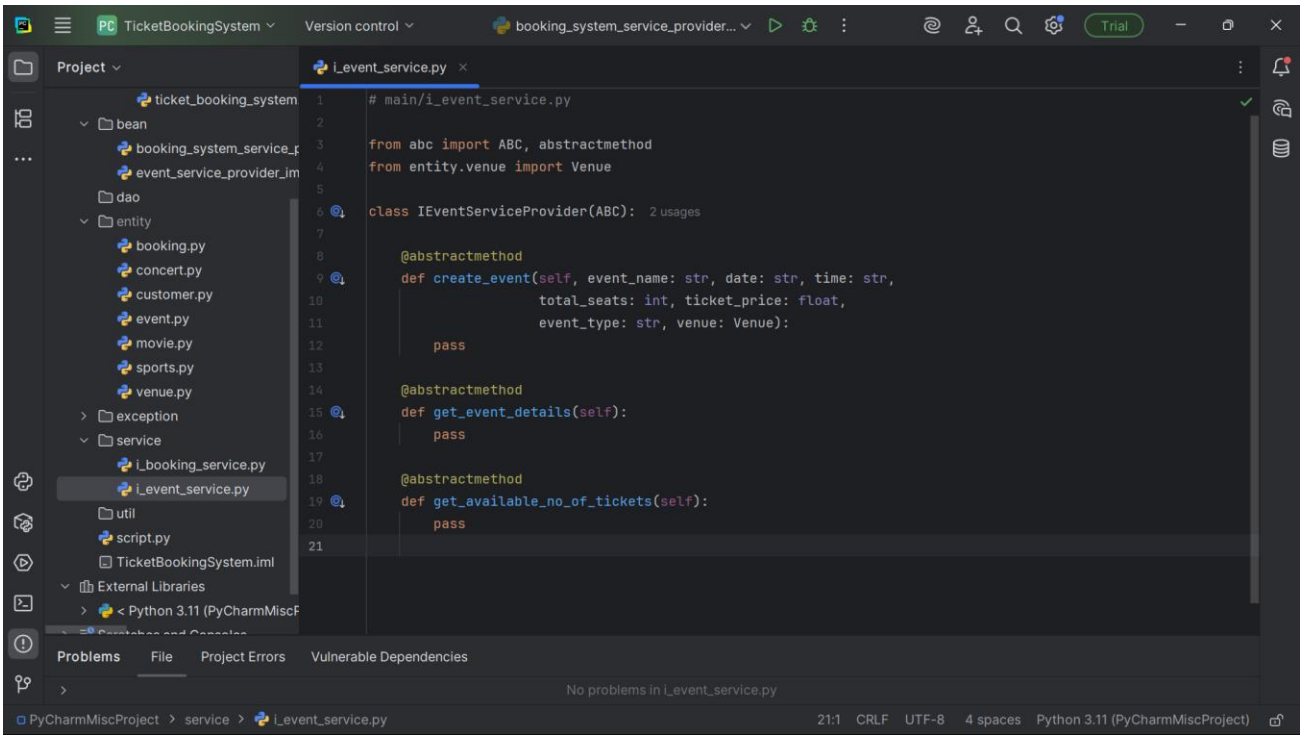
   Code :

   from abc import ABC, abstractmethod

   from entity.venue import Venue

Done by
Dhamini machireddy

```
class IEventServiceProvider(ABC):

    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str,
                     total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

    @abstractmethod
    def get_available_no_of_tickets(self):
        pass
```

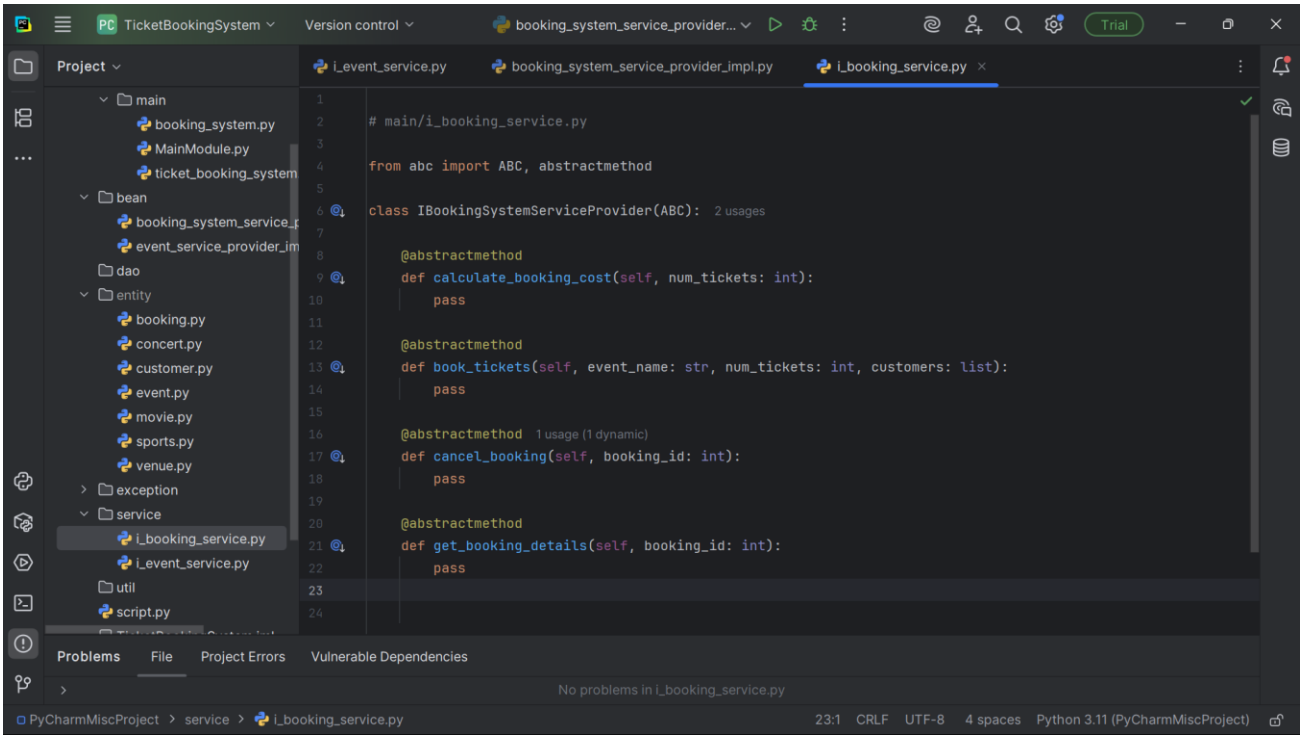


Code :

```
from abc import ABC, abstractmethod

class IBookingSystemServiceProvider(ABC):

    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int):
```

Done by
Dhamini machireddy

```
    pass

@abstractmethod
def book_tickets(self, event_name: str, num_tickets: int, customers: list):
    pass

@abstractmethod
def cancel_booking(self, booking_id: int):
    pass

@abstractmethod
def get_booking_details(self, booking_id: int):
    pass
```



Code:

main/event_service_provider_impl.py

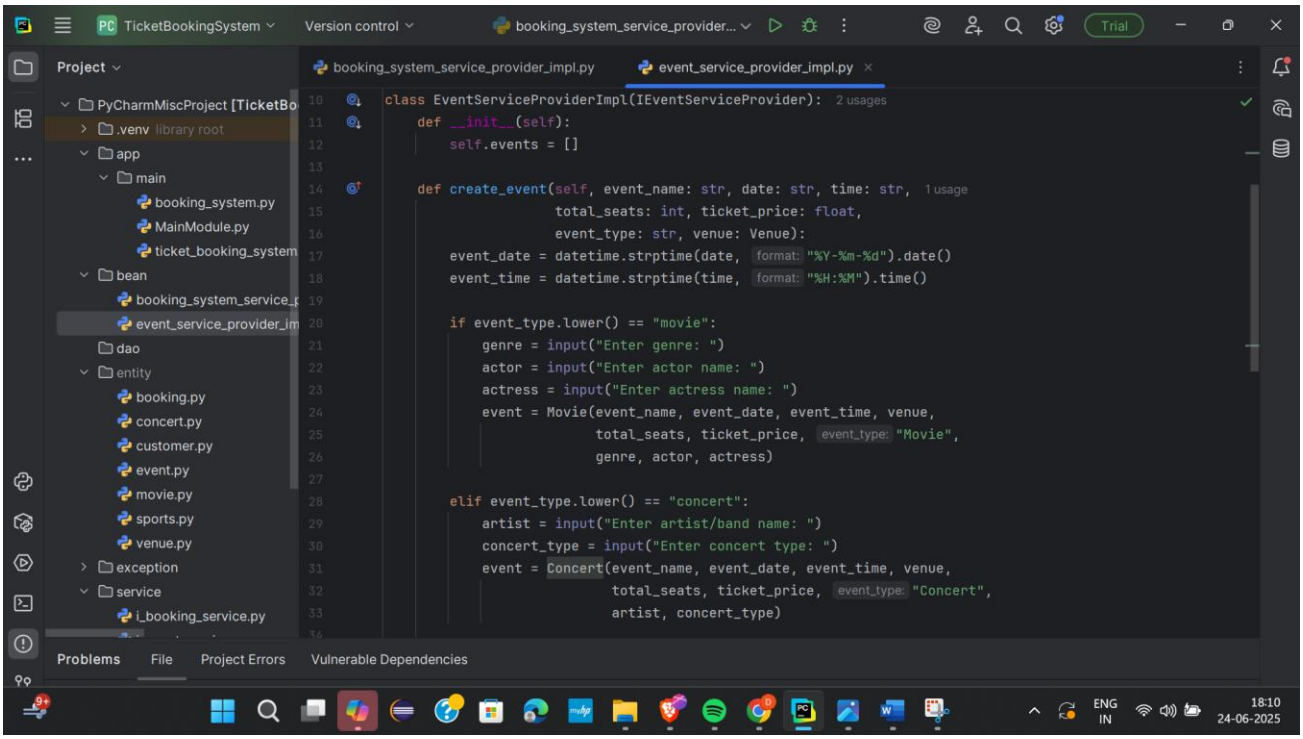
```
from service.i_event_service import IEventServiceProvider
from entity.venue import Venue
from entity.movie import Movie
from entity.concert import Concert
from entity.sports import Sports
from datetime import datetime

class EventServiceProviderImpl(IEventServiceProvider):
```

Done by
Dhamini machireddy

```
def __init__(self):
    self.events = []

def create_event(self, event_name: str, date: str, time: str,
                 total_seats: int, ticket_price: float,
                 event_type: str, venue: Venue):
    event_date = datetime.strptime(date, "%Y-%m-%d").date()
    event_time = datetime.strptime(time, "%H:%M").time()

    if event_type.lower() == "movie":
        genre = input("Enter genre: ")
        actor = input("Enter actor name: ")
        actress = input("Enter actress name: ")
        event = Movie(event_name, event_date, event_time, venue,
                      total_seats, ticket_price, "Movie",
                      genre, actor, actress)

    elif event_type.lower() == "concert":
        artist = input("Enter artist/band name: ")
        concert_type = input("Enter concert type: ")
        event = Concert(event_name, event_date, event_time, venue,
                        total_seats, ticket_price, "Concert",
                        artist, concert_type)

    elif event_type.lower() == "sports":
        sport_name = input("Enter sport name: ")
        teams_name = input("Enter teams name: ")
        event = Sports(event_name, event_date, event_time, venue,
                       total_seats, ticket_price, "Sports",
                       sport_name, teams_name)

    else:
        print("❌ Invalid event type.")
        return None

    self.events.append(event)
    print("☑ Event created successfully!")
    return event

def get_event_details(self):
    if not self.events:
        print("⚠ No events available.")
        return []
```

Done by
Dhamini machireddy

```
        for idx, event in enumerate(self.events, start=1):
            print(f"\n📌 Event {idx}:")
            event.display_event_details()
        return self.events

    def get_available_no_of_tickets(self):
        total = sum(event.get_available_seats() for event in self.events)
        print(f"\n🎟️ Total Available Tickets: {total}")
        return total
```



Code :

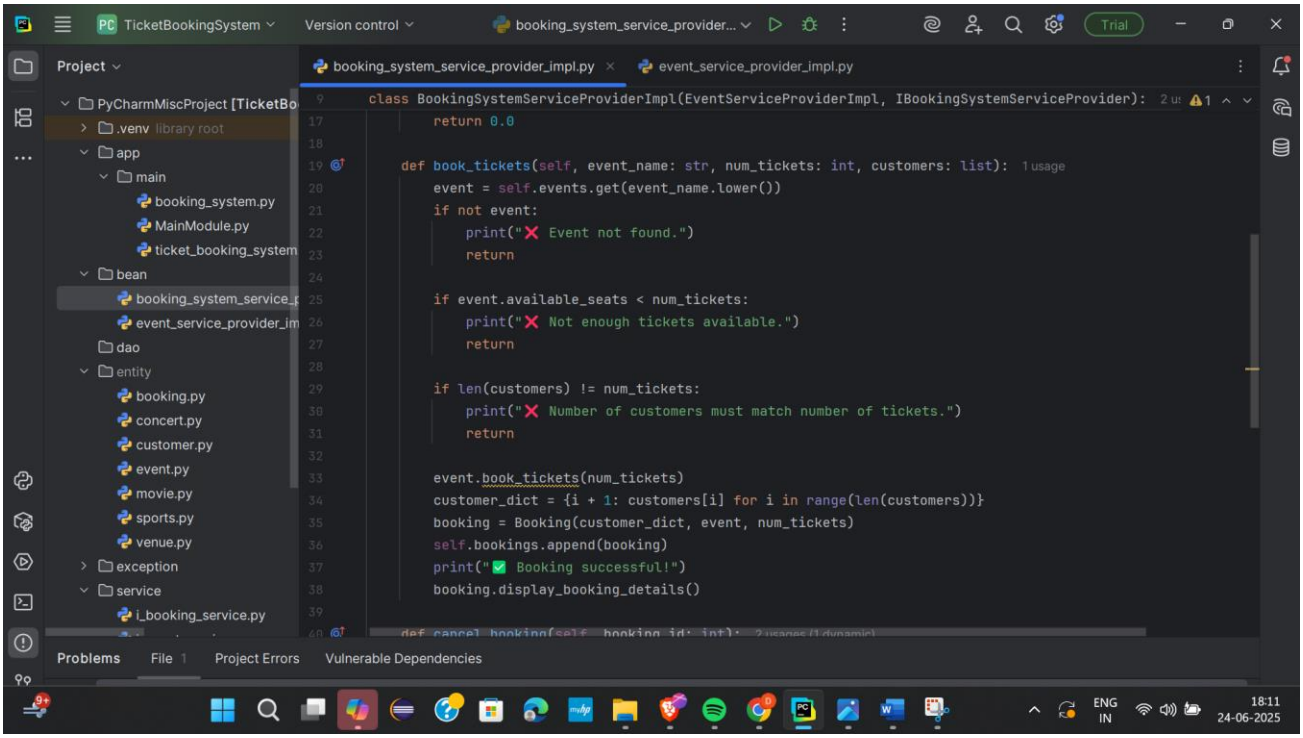
```
# Updated BookingSystemServiceProviderImpl class using Dict for Task 10.3
from service.i_booking_service import IBookingSystemServiceProvider
from bean.event_service_provider_impl import EventServiceProviderImpl
from entity.booking import Booking

from entity.event import Event
from typing import Dict

class BookingSystemServiceProviderImpl(EventServiceProviderImpl,
IBookingSystemServiceProvider):
    def __init__(self):
        super().__init__()
        self.events: Dict[str, Event] = {}
        self.bookings = []
```

Done by
Dhamini machireddy

```
def calculate_booking_cost(self, num_tickets: int) -> float:
    print(f"🔢 Cost for {num_tickets} tickets is calculated based on event during booking.")
    return 0.0

def book_tickets(self, event_name: str, num_tickets: int, customers: list):
    event = self.events.get(event_name.lower())
    if not event:
        print("❌ Event not found.")
        return

    if event.available_seats < num_tickets:
        print("❌ Not enough tickets available.")
        return

    if len(customers) != num_tickets:
        print("❌ Number of customers must match number of tickets.")
        return

    event.book_tickets(num_tickets)
    customer_dict = {i + 1: customers[i] for i in range(len(customers))}
    booking = Booking(customer_dict, event, num_tickets)
    self.bookings.append(booking)
    print("☑️ Booking successful!")
    booking.display_booking_details()

def cancel_booking(self, booking_id: int):
    booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
    if not booking:
        print("❌ Booking not found.")
        return
    booking.event.cancel_booking(booking.num_tickets)
    self.bookings.remove(booking)
    print("☑️ Booking cancelled successfully.")

def get_booking_details(self, booking_id: int):
    booking = next((b for b in self.bookings if b.booking_id == booking_id), None)
    if booking:
        booking.display_booking_details()
    else:
        print("❌ Booking not found.")
```

Done by
Dhamini machireddy

```
def get_event_details(self):
    sorted_events = sorted(self.events.values(), key=lambda e: (e.event_name.lower(),
e.venue.venue_name.lower()))
    for idx, event in enumerate(sorted_events, 1):
        print(f"\n📌 Event {idx}:")
        event.display_event_details()
```



4. Create I**BookingSystemRepository** interface/abstract class which include following methods to interact with database.
    - **create_event(event_name: str, date:str, time:str, total_seats: int, ticket_price: float, event_type: str, venu: Venu):** Create a new event with the specified details and event type (movie, sport or concert) and return event object and should store in database.
    - **getEventDetails():** return array of event details from the database.
    - **getAvailableNoOfTickets()**: return the total available tickets from the database.
    - **calculate_booking_cost(num_tickets):** Calculate and set the total cost of the booking.
    - **book_tickets(eventname:str, num_tickets, listOfCustomer):** Book a specified number of tickets for an event. for each tickets customer object should be created and stored in array also should update the attributes of Booking class and stored in database.
    - **cancel_booking(booking_id):** Cancel the booking and update the available seats and stored in database.
    - **get_booking_details(booking_id):** get the booking details from database.

      Code :

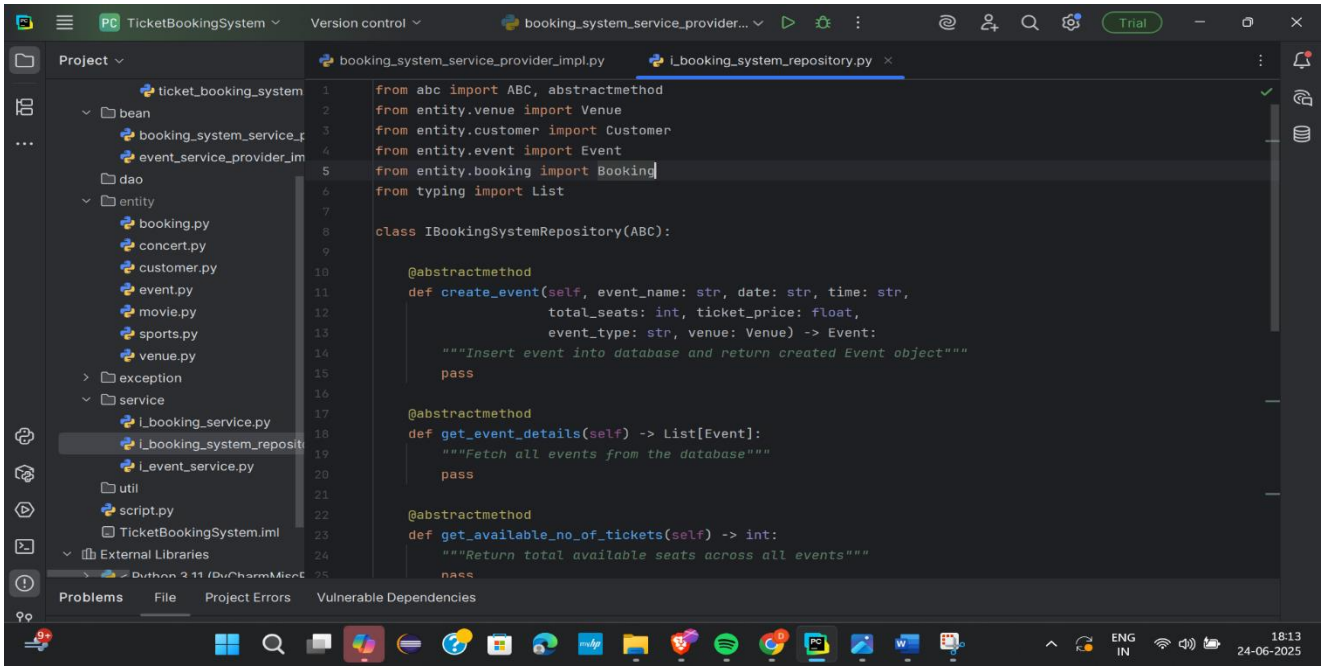
      ```
      from abc import ABC, abstractmethod
      from entity.venue import Venue
      ```

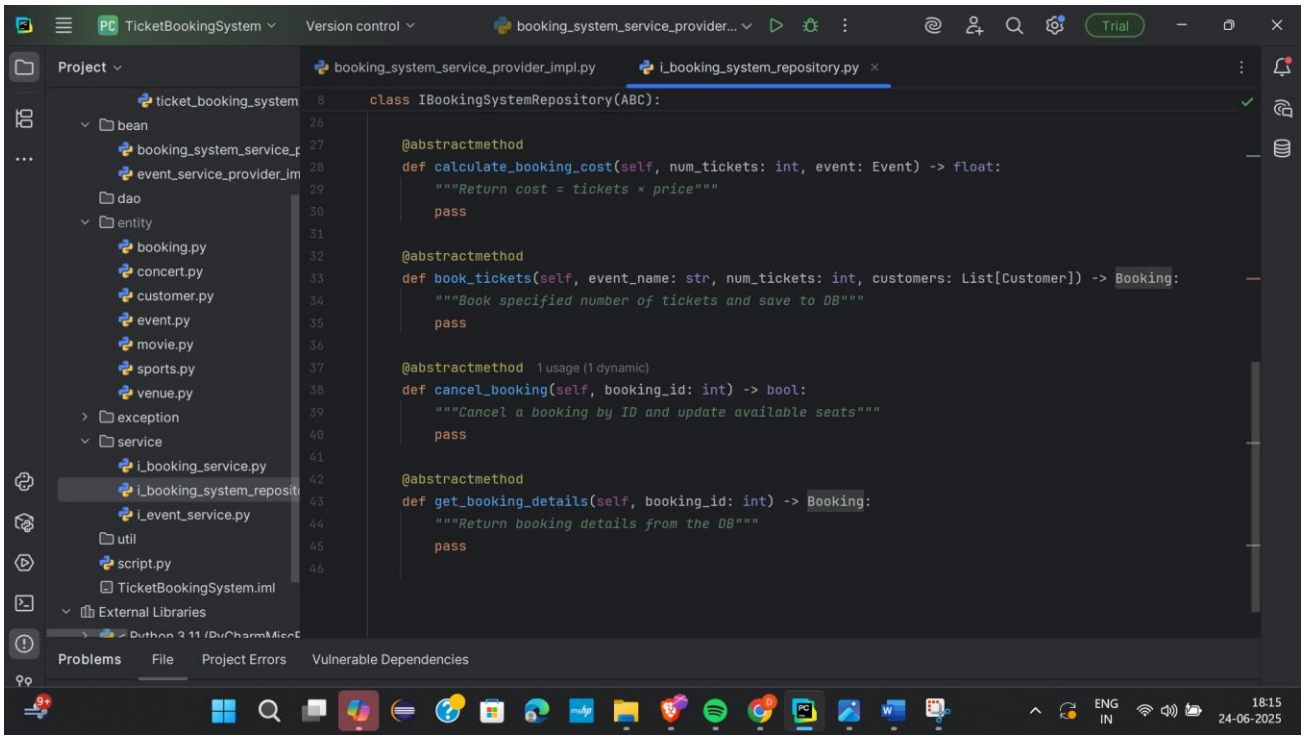
Done by
Dhamini machireddy

```
from entity.customer import Customer
from entity.event import Event
from entity.booking import Booking
from typing import List

class IBookingSystemRepository(ABC):

    @abstractmethod
    def create_event(self, event_name: str, date: str, time: str,
                     total_seats: int, ticket_price: float,
                     event_type: str, venue: Venue) -> Event:
        """Insert event into database and return created Event object"""
        pass

    @abstractmethod
    def get_event_details(self) -> List[Event]:
        """Fetch all events from the database"""
        pass

    @abstractmethod
    def get_available_no_of_tickets(self) -> int:
        """Return total available seats across all events"""
        pass

    @abstractmethod
    def calculate_booking_cost(self, num_tickets: int, event: Event) -> float:
        """Return cost = tickets × price"""
        pass

    @abstractmethod
    def book_tickets(self, event_name: str, num_tickets: int, customers: List[Customer]) ->
Booking:
        """Book specified number of tickets and save to DB"""
        pass

    @abstractmethod
    def cancel_booking(self, booking_id: int) -> bool:
        """Cancel a booking by ID and update available seats"""
        pass

    @abstractmethod
    def get_booking_details(self, booking_id: int) -> Booking:
```

Done by
Dhamini machireddy

"""Return booking details from the DB"""
pass





5. Create **BookingSystemRepositoryImpl** interface/abstract class which implements I**BookingSystemRepository** interface/abstract class and provide implementation of all methods and perform the database operations.

   Code :

Done by
Dhamini machireddy

```
import mysql.connector
from service.i_booking_system_repository import IBookingSystemRepository
from util.db_util import DBUtil
from entity.venue import Venue
from entity.customer import Customer
from entity.event import Event
from entity.booking import Booking
from typing import List


class BookingSystemRepositoryImpl(IBookingSystemRepository):

    def create_event(self, event_name, date, time, total_seats, ticket_price, event_type, venue:
Venue) -> Event:
        conn = DBUtil.get_db_conn()
        try:
            cursor = conn.cursor()
            # Insert venue first (if not exists)
            cursor.execute("INSERT INTO venue (venue_name, address) VALUES (%s, %s)",
                           (venue.venue_name, venue.address))
            venue_id = cursor.lastrowid

            # Insert event
            cursor.execute("""
                           INSERT INTO event (event_name, event_date, event_time, total_seats,
available_seats,
                                              ticket_price, event_type, venue_id)
                           VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                           """, (event_name, date, time, total_seats, total_seats, ticket_price, event_type,
venue_id))

            conn.commit()
            print("☑ Event saved to database.")
            return Event(cursor.lastrowid, event_name, date, time, venue, total_seats, total_seats,
ticket_price,
                         event_type)
        except mysql.connector.Error as e:
            print("❌ DB Error:", e)
        finally:
            conn.close()

    def get_event_details(self) -> List[Event]:
        events = []
```

Done by
Dhamini machireddy

```
    conn = DBUtil.get_db_conn()
    try:
        cursor = conn.cursor()
        cursor.execute("""
                    SELECT e.event_id,
                        e.event_name,
                        e.event_date,
                        e.event_time,
                        e.total_seats,
                        e.available_seats,
                        e.ticket_price,
                        e.event_type,
                        v.venue_id,
                        v.venue_name,
                        v.address
                    FROM event e
                        JOIN venue v ON e.venue_id = v.venue_id
                    """)
        for row in cursor.fetchall():
            venue = Venue(row[8], row[9], row[10])
            event = Event(*row[:8], venue)
            events.append(event)
    except mysql.connector.Error as e:
        print("❌ DB Error:", e)
    finally:
        conn.close()
    return events

def get_available_no_of_tickets(self) -> int:
    conn = DBUtil.get_db_conn()
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT SUM(available_seats) FROM event")
        result = cursor.fetchone()
        return result[0] if result and result[0] is not None else 0
    except mysql.connector.Error as e:
        print("❌ DB Error:", e)
    finally:
        conn.close()

def calculate_booking_cost(self, num_tickets: int, event: Event) -> float:
    return num_tickets * event.ticket_price
```

Done by
Dhamini machireddy

```
    def book_tickets(self, event_name: str, num_tickets: int, customers: List[Customer]) -> Booking:
        conn = DBUtil.get_db_conn()
        try:
            cursor = conn.cursor()
            # Get event
            cursor.execute("SELECT event_id, available_seats, ticket_price FROM event WHERE
LOWER(event_name) = %s",
                           (event_name.lower(),))
            row = cursor.fetchone()
            if not row:
                print("❌ Event not found.")
                return None

            event_id, available_seats, price = row
            if available_seats < num_tickets:
                print("❌ Not enough tickets available.")
                return None

            # Update seats
            cursor.execute("UPDATE event SET available_seats = available_seats - %s WHERE event_id =
%s",
                           (num_tickets, event_id))

            # Insert customers and collect their IDs
            customer_ids = []
            for customer in customers:
                cursor.execute("INSERT INTO customer (customer_name, email, phone_number) VALUES
(%s, %s, %s)",
                               (customer.customer_name, customer.email, customer.phone_number))
                customer_ids.append(cursor.lastrowid)

            total_cost = num_tickets * price

            # Insert booking
            cursor.execute(
                "INSERT INTO booking (event_id, num_tickets, total_cost, booking_date) VALUES (%s, %s,
%s, NOW())",
                (event_id, num_tickets, total_cost))
            booking_id = cursor.lastrowid

            # Link customers to booking (if needed in your schema)
            # Optional join table logic here
```

Done by
Dhamini machireddy

```
            conn.commit()
            print("☑ Booking completed.")
            return Booking(booking_id, event_id, customer_ids, num_tickets, total_cost, None)
        except mysql.connector.Error as e:
            print("❌ DB Error:", e)
        finally:
            conn.close()

    def cancel_booking(self, booking_id: int) -> bool:
        conn = DBUtil.get_db_conn()
        try:
            cursor = conn.cursor()
            cursor.execute("SELECT event_id, num_tickets FROM booking WHERE booking_id = %s",
(booking_id,))
            booking = cursor.fetchone()
            if not booking:
                print("❌ Booking not found.")
                return False

            event_id, tickets = booking
            cursor.execute("UPDATE event SET available_seats = available_seats + %s WHERE event_id =
%s",
                           (tickets, event_id))
            cursor.execute("DELETE FROM booking WHERE booking_id = %s", (booking_id,))
            conn.commit()
            print("☑ Booking cancelled.")
            return True
        except mysql.connector.Error as e:
            print("❌ DB Error:", e)
            return False
        finally:
            conn.close()

    def get_booking_details(self, booking_id: int) -> Booking:
        conn = DBUtil.get_db_conn()
        try:
            cursor = conn.cursor()
            cursor.execute("SELECT event_id, num_tickets, total_cost, booking_date FROM booking
WHERE booking_id = %s",
                           (booking_id,))
            result = cursor.fetchone()
            if not result:
```

Done by
Dhamini machireddy

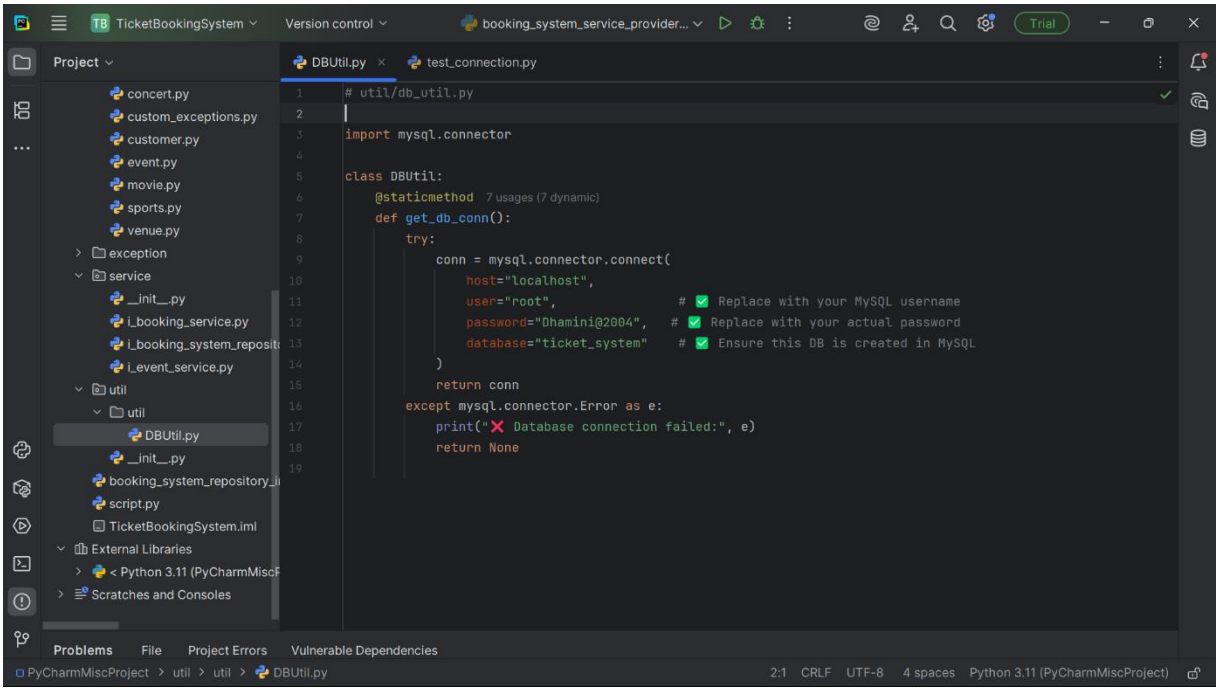
```
            print("❌ Booking not found.")
            return None

        event_id, num_tickets, total_cost, booking_date = result
        return Booking(booking_id, event_id, [], num_tickets, total_cost, booking_date)
    except mysql.connector.Error as e:
        print("❌ DB Error:", e)
        return None
    finally:
        conn.close()
```

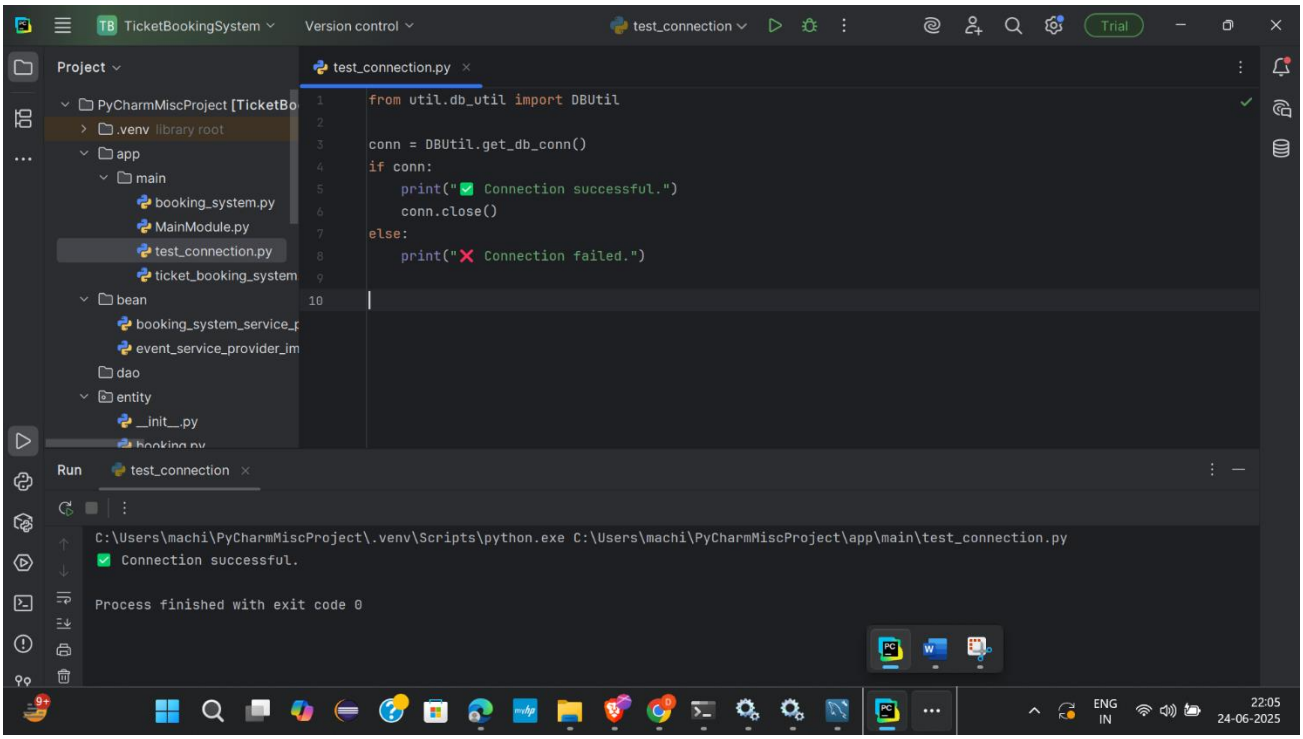
6. Create **DBUtil** class and add the following method.
   - **static getDBConn():**
   - **Connection** Establish a connection to the database and return Connection reference

     Code :

```
import mysql.connector

class DBUtil:
    @staticmethod
    def get_db_conn():
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",            # ☑ Replace with your MySQL username
                password="password",   # ☑ Replace with your actual password
                database="ticket_system"    # ☑ Ensure this DB is created in MySQL
            )
            return conn
        except mysql.connector.Error as e:
            print("❌ Database connection failed:", e)
            return None
```

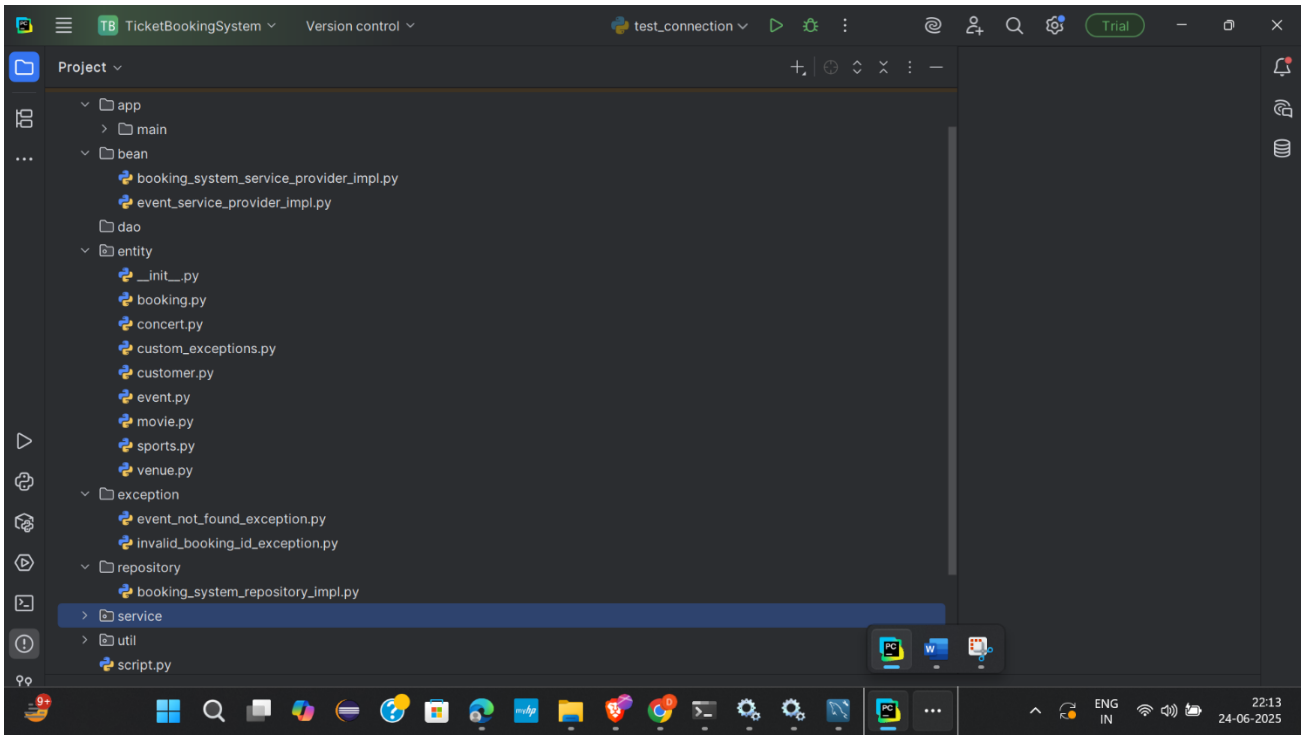
Done by
Dhamini machireddy





7. Place the interface/abstract class in service package and interface implementation class, concrete class in bean package and **TicketBookingSystemRepository** class in app package.

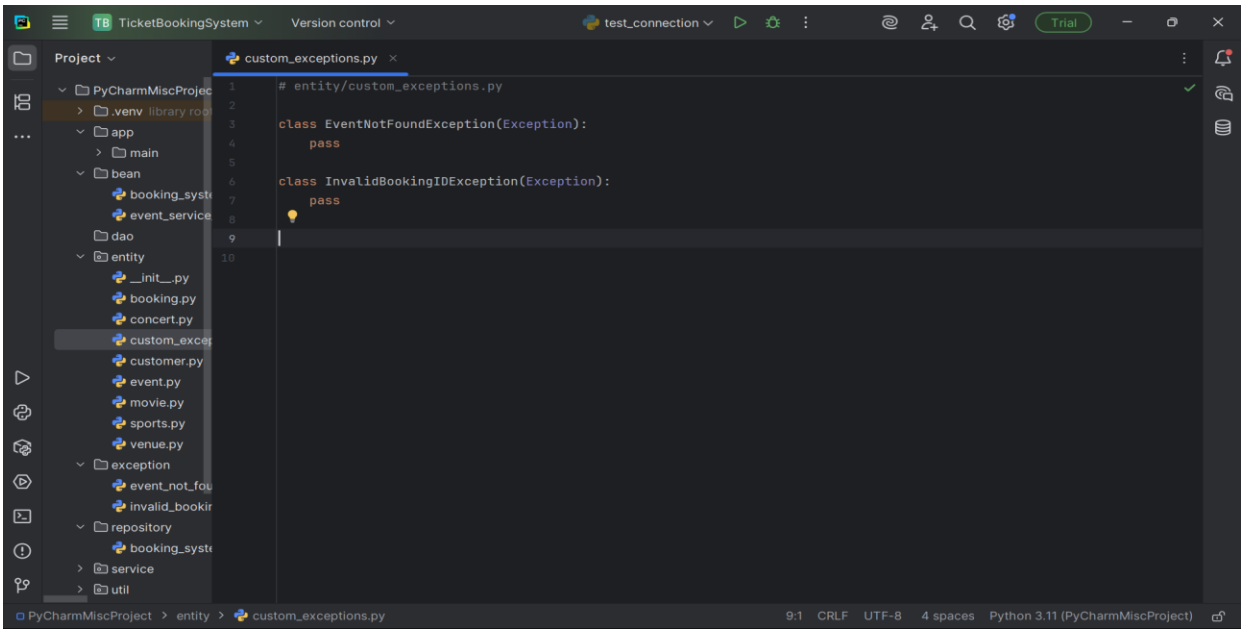
Done by
Dhamini machireddy



8. Should throw appropriate exception as mentioned in above task along with handle **SQLException**.

Code :

```
class EventNotFoundException(Exception):
pass

 class InvalidBookingIDException(Exception):
pass
```



Done by
Dhamini machireddy

9. Create **TicketBookingSystem** class and perform following operations:
    - Create a simple user interface in a main method that allows users to interact with the ticket booking system by entering commands such as "create_event", "book_tickets", "cancel_tickets", "get_available_seats,", "get_event_details," and "exit."

Code :

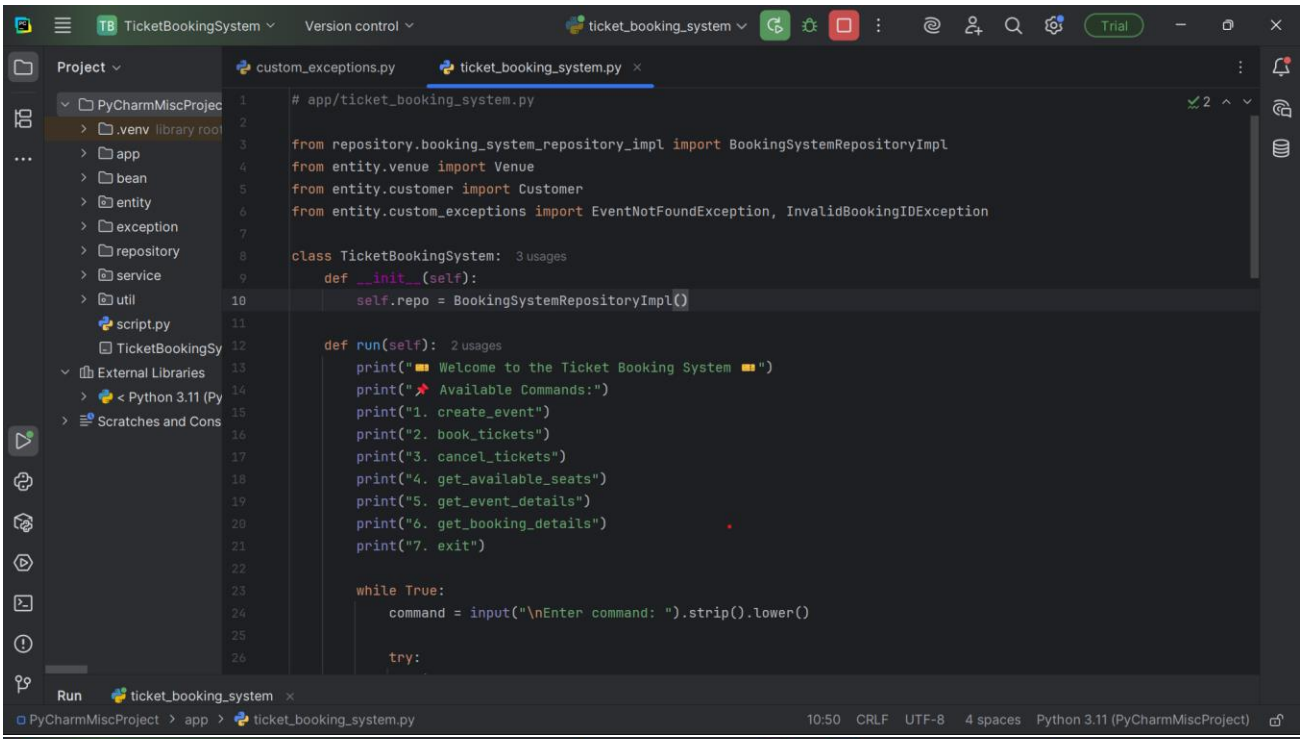
```
# app/ticket_booking_system.py

from repository.booking_system_repository_impl import BookingSystemRepositoryImpl
from entity.venue import Venue
from entity.customer import Customer
from entity.custom_exceptions import EventNotFoundException, InvalidBookingIDException

class TicketBookingSystem:
    def __init__(self):
        self.repo = BookingSystemRepositoryImpl()

    def run(self):
        print("🎟️ Welcome to the Ticket Booking System 🎟️")
        print("📌 Available Commands:")
        print("1. create_event")
        print("2. book_tickets")
        print("3. cancel_tickets")
        print("4. get_available_seats")
        print("5. get_event_details")
        print("6. get_booking_details")
        print("7. exit")

        while True:
            command = input("\nEnter command: ").strip().lower()

            try:
                if command == "create_event":
                    name = input("Event name: ")
                    date = input("Date (YYYY-MM-DD): ")
                    time = input("Time (HH:MM): ")
                    seats = int(input("Total seats: "))
                    price = float(input("Ticket price: "))
                    type_ = input("Event type (Movie, Concert, Sports): ")
                    vname = input("Venue name: ")
                    vaddr = input("Venue address: ")
                    venue = Venue(vname, vaddr)
```

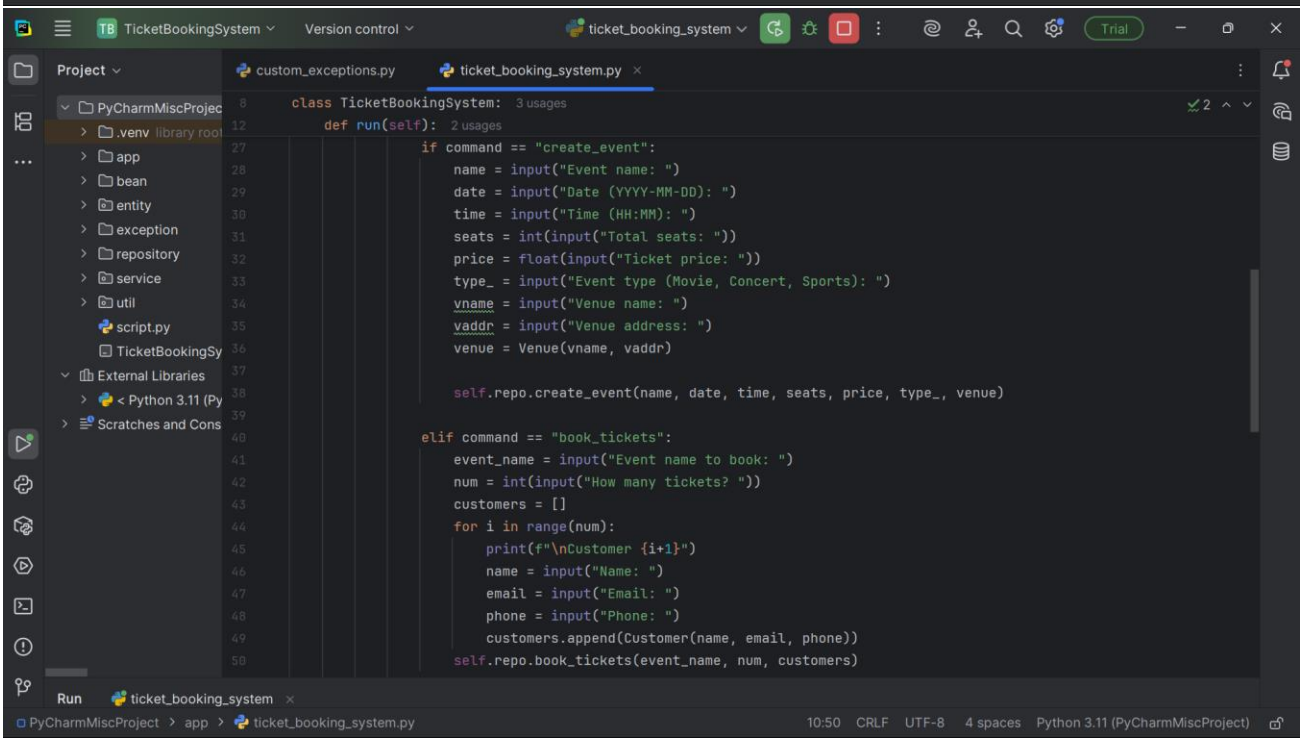
Done by
Dhamini machireddy

```
                self.repo.create_event(name, date, time, seats, price, type_, venue)

            elif command == "book_tickets":
                event_name = input("Event name to book: ")
                num = int(input("How many tickets? "))
                customers = []
                for i in range(num):
                    print(f"\nCustomer {i+1}")
                    name = input("Name: ")
                    email = input("Email: ")
                    phone = input("Phone: ")
                    customers.append(Customer(name, email, phone))
                self.repo.book_tickets(event_name, num, customers)

            elif command == "cancel_tickets":
                bid = int(input("Booking ID to cancel: "))
                self.repo.cancel_booking(bid)

            elif command == "get_available_seats":
                self.repo.get_available_no_of_tickets()

            elif command == "get_event_details":
                self.repo.get_event_details()

            elif command == "get_booking_details":
                bid = int(input("Booking ID to view: "))
                self.repo.get_booking_details(bid)

            elif command == "exit":
                print("👋 Goodbye!")
                break

            else:
                print("❌ Invalid command.")

        except EventNotFoundException as e:
            print("❗", e)
        except InvalidBookingIDException as e:
            print("❗", e)
        except Exception as e:
            print("❌ Error:", e)

# Run the system
if __name__ == "__main__":
```

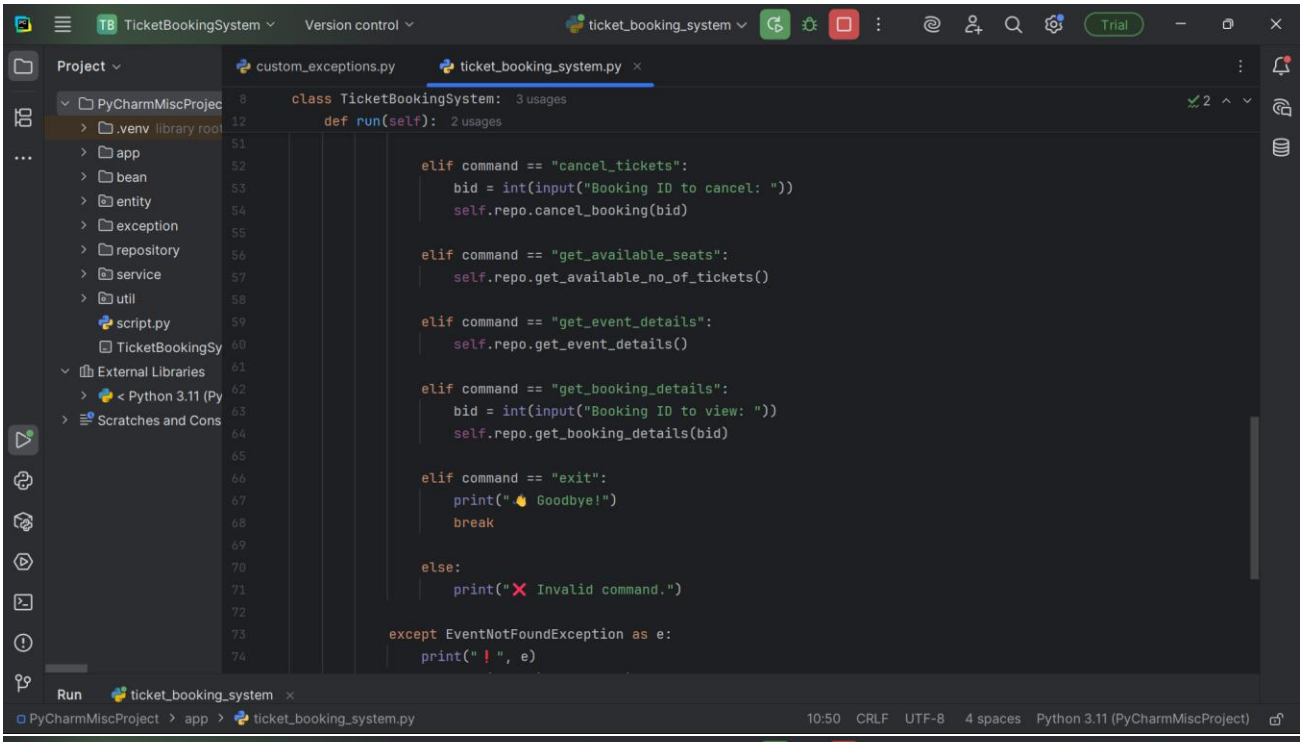
Done by
Dhamini machireddy

```
app = TicketBookingSystem()
app.run()
```

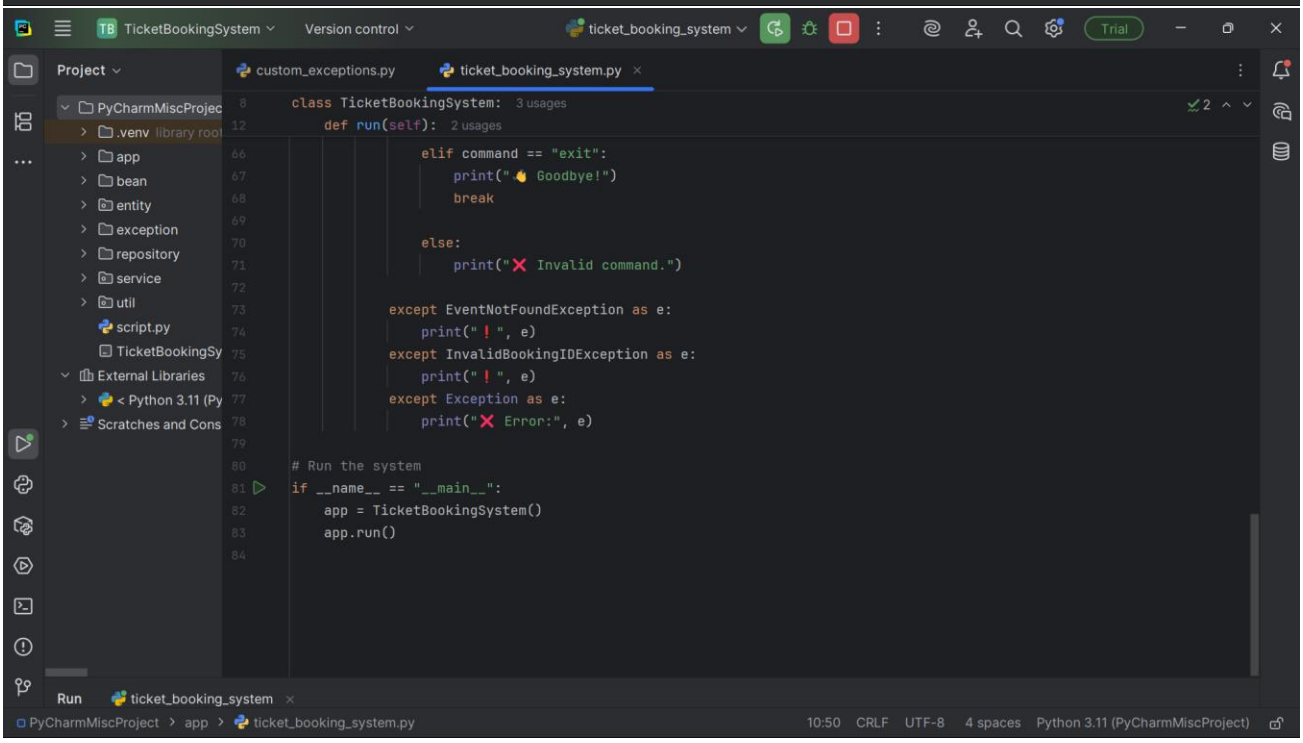




Done by
Dhamini machireddy



```
class TicketBookingSystem:  3 usages
    def run(self):  2 usages
                elif command == "cancel_tickets":
                    bid = int(input("Booking ID to cancel: "))
                    self.repo.cancel_booking(bid)

                elif command == "get_available_seats":
                    self.repo.get_available_no_of_tickets()

                elif command == "get_event_details":
                    self.repo.get_event_details()

                elif command == "get_booking_details":
                    bid = int(input("Booking ID to view: "))
                    self.repo.get_booking_details(bid)

                elif command == "exit":
                    print("👋 Goodbye!")
                    break

                else:
                    print("❌ Invalid command.")

            except EventNotFoundException as e:
                print("❗", e)
```



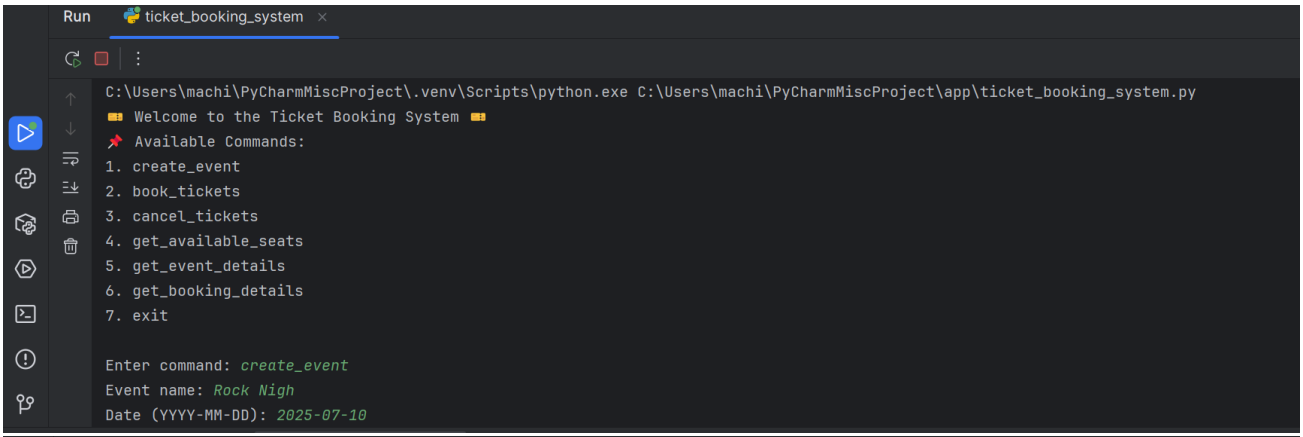
```
class TicketBookingSystem:  3 usages
    def run(self):  2 usages
                elif command == "exit":
                    print("👋 Goodbye!")
                    break

                else:
                    print("❌ Invalid command.")

            except EventNotFoundException as e:
                print("❗", e)
            except InvalidBookingIDException as e:
                print("❗", e)
            except Exception as e:
                print("❌ Error:", e)

# Run the system
if __name__ == "__main__":
    app = TicketBookingSystem()
    app.run()
```

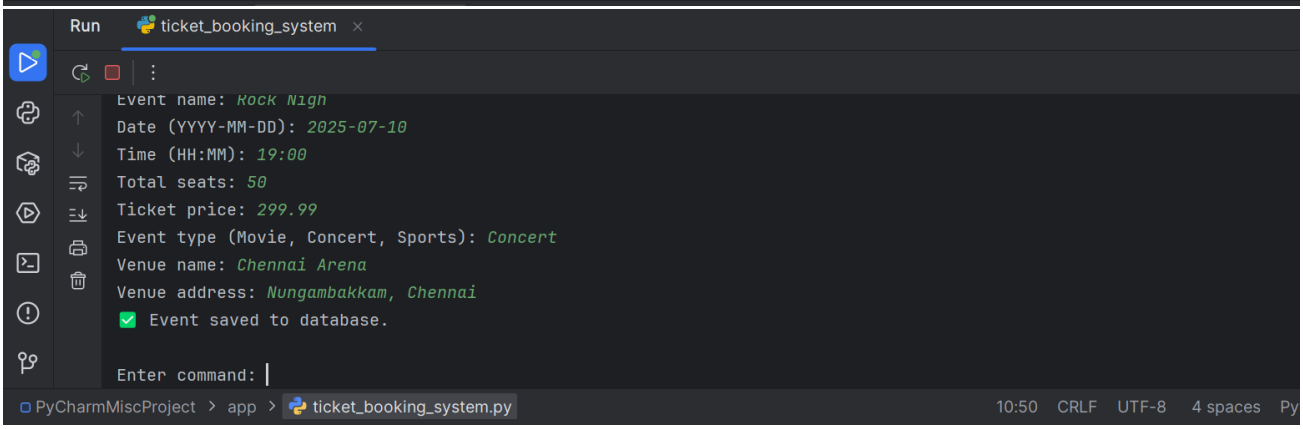
Output :

From ticket booking system

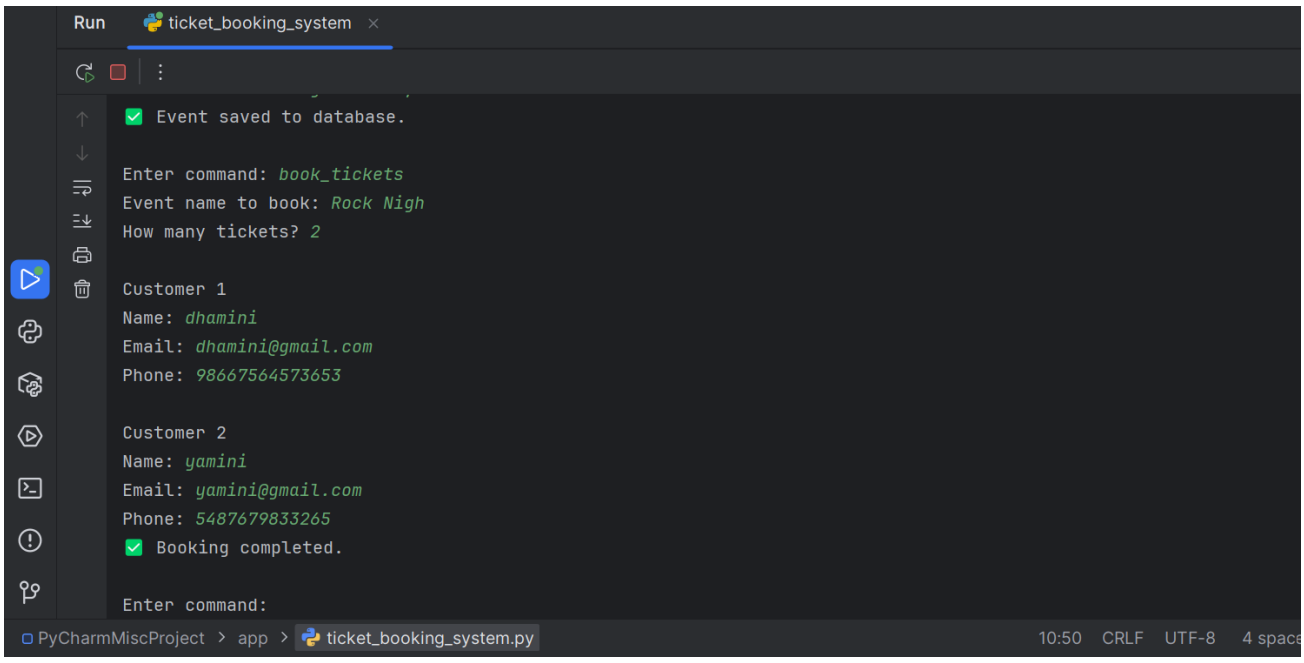
1. Create event details

Done by
Dhamini machireddy

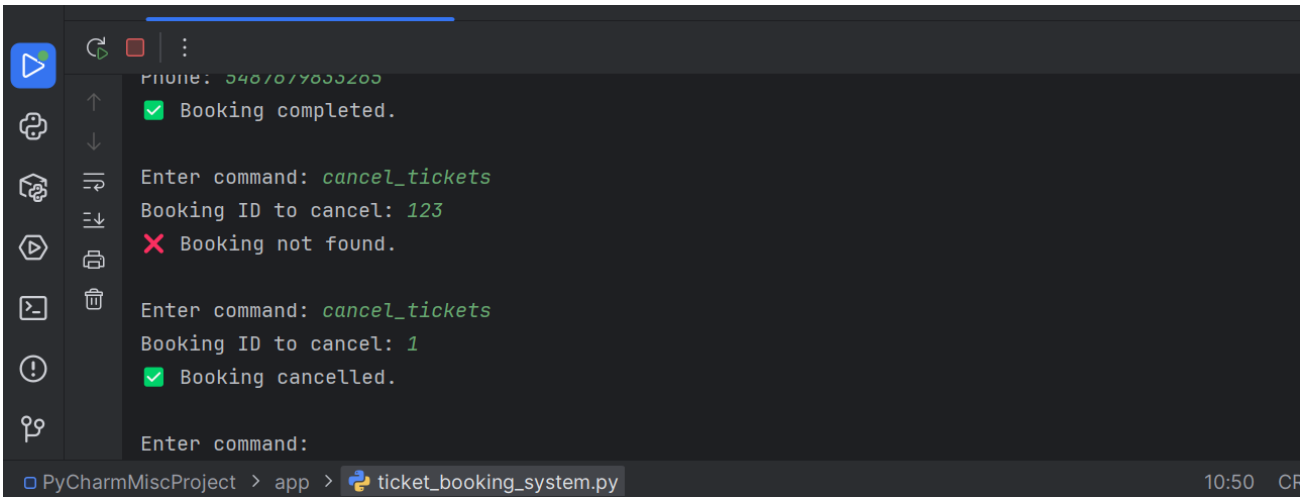




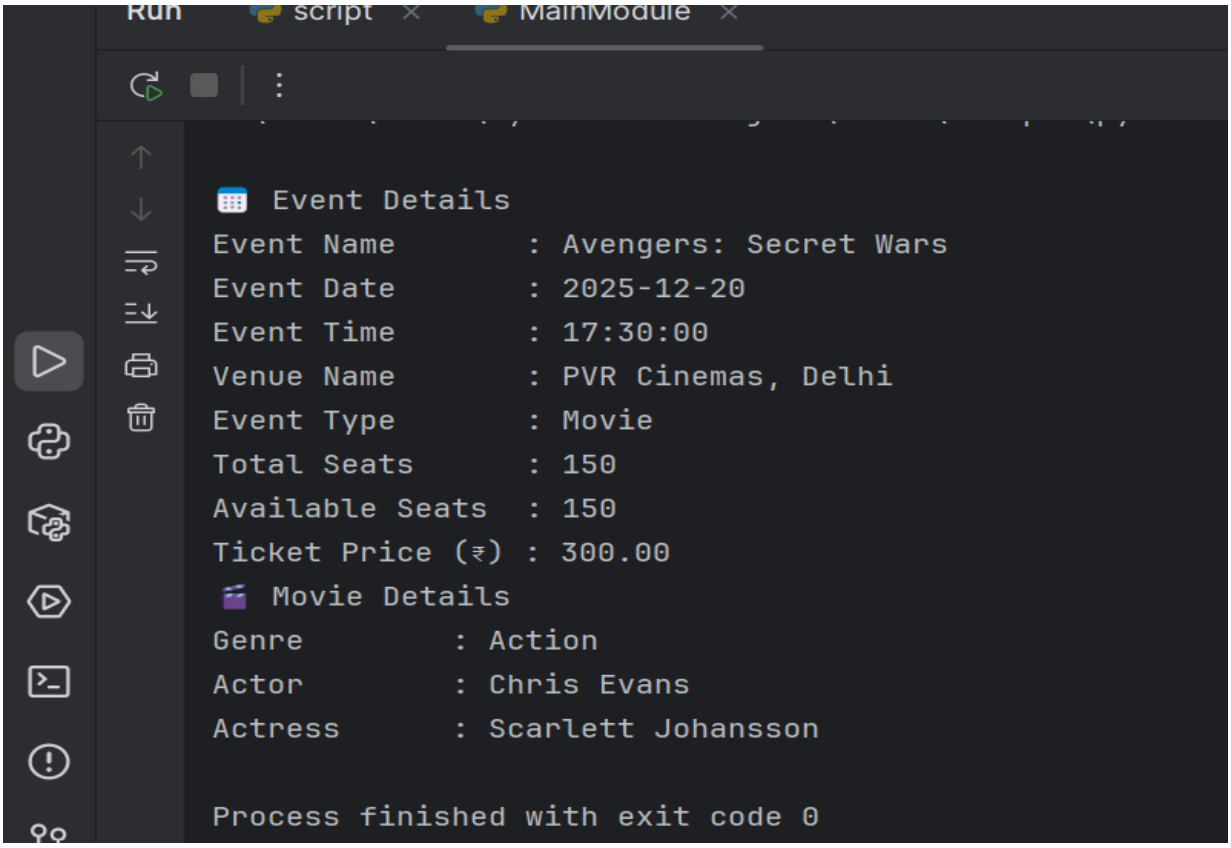
2. Booking tickets for events



Done by
Dhamini machireddy



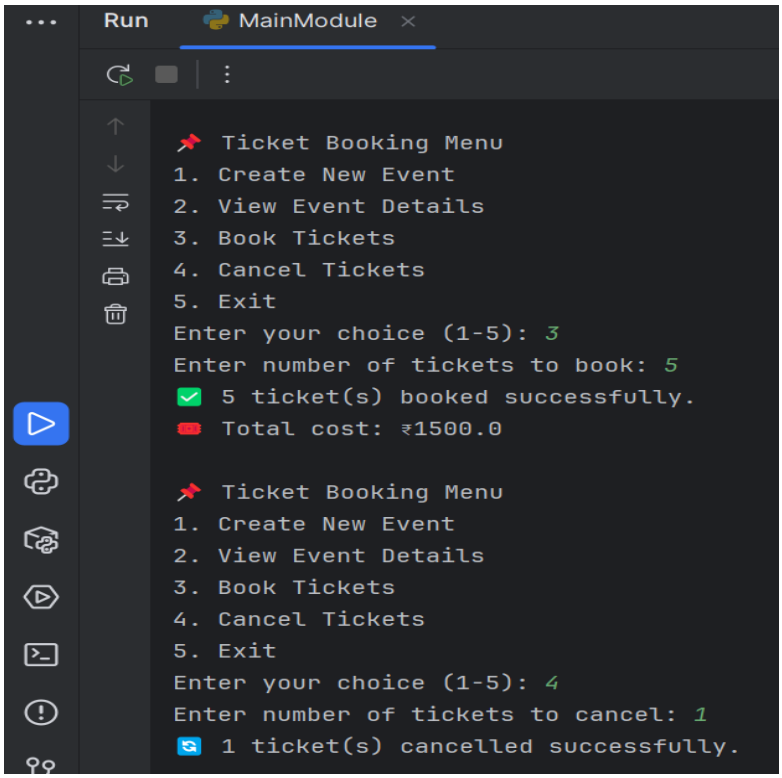
3. Checking bookid details to cancel



4. Getting event details

Done by
Dhamini machireddy

5. Ticket booked successfully and 1 ticket canceled



6. Exiting the ticket system