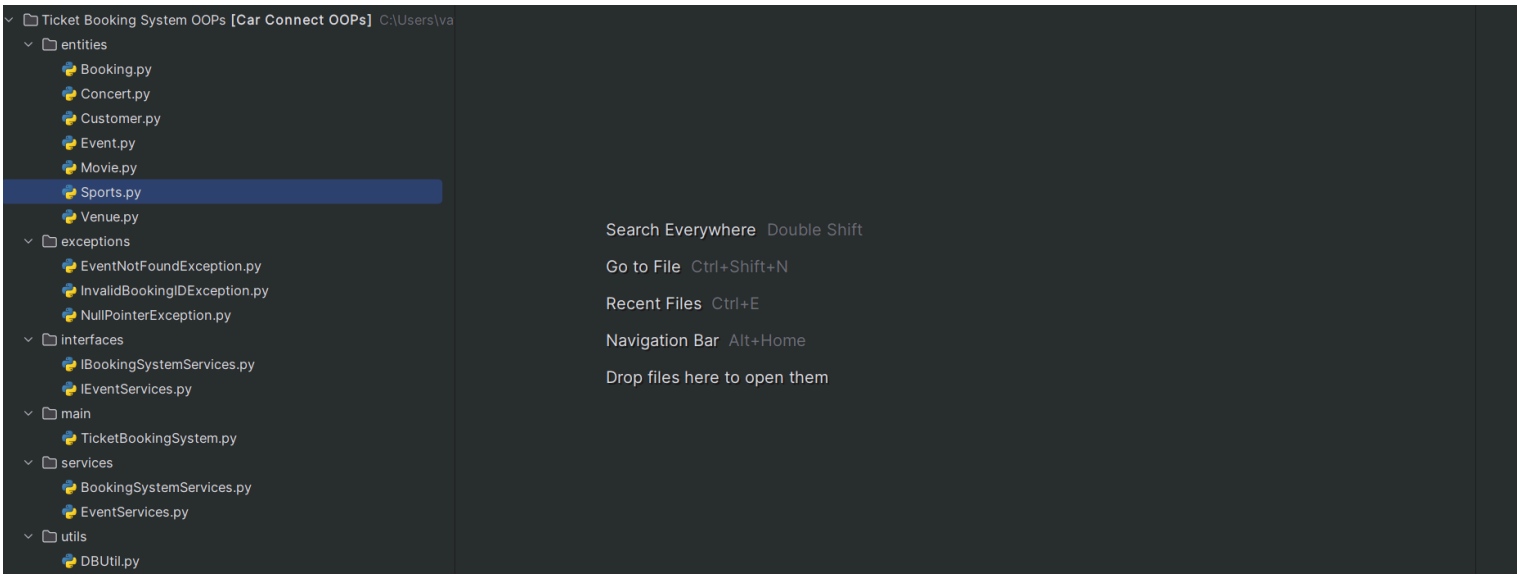
Ticket Booking System

Folder Structure



Task 1: Conditional Statements

```
        # Select Event and Book Event
        name_of_event = input("\nEnter event name: ")
        cursor.execute("select event_id,ticket_price from events where event_name = %s", (name_of_event,))
        rows = cursor.fetchone()
        event_id = -1
        price = 0
        available_seats = 0
        if rows:
            event_id, price, available_seats = rows

        if available_seats < num_tickets:
            raise Exception('Not enough tickets available. Tickets available: ', available_seats)
```

Task 2: Nested Conditional Statements

```
56
57          ticket_type = input("What type of ticket you want? - 1.Silver(x1) 2.Gold(x2) 3.Dimond(x3) ?")
58          total_cost = price * num_tickets * ticket_type
59          today = date.today()
60          query = "insert into bookings (customer_id, event_id, num_tickets, total_cost, booking_date) values (
61          cursor.execute(query, (customer_id, event_id, num_tickets, total_cost, today))
```

Task 3: Looping

```
1 usage
def main_menu(self):
    while True:
        print("\nSelect one options from the options given below : ")
        print("1. Create a new event.")
        print("2. Book tickets.")
        print("3. Cancel Tickets.")
        print("4. Know how many seats are Available.")
        print("5. See every event and it's details.")
        print("6. Exit.")
        choice = input("Enter your choice here : ")

        try:
            match choice:
                case "1":
                    self.create_event()
                    print()
                case "2":
                    num_tickets = int(input("\nPlease enter the number of tickets you want to book : "))
                    self.book_tickets(num_tickets)
                    print()
                case "3":
                    booking_id = int(input("\nPlease enter your booking id here : "))
                    self.cancel_booking(booking_id)
                    print()
                case "4":
```

Task 4: Class & Object

Booking

```
class Booking(Event):
    def __init__(self, event, customer):
        self.booking_id = random.randint( a: 10000,  b: 99999)
        self.customer = customer
        self.event = event
        self.num_tickets = len(customer)
        self.total_cost = 0
        self.booking_date = date.today()

    def calculate_booking_cost(self, num_tickets):
        pass

    def book_tickets(self, num_tickets):
        super().book_ticket(num_tickets)

    def cancel_booking(self, num_tickets):
        super().cancel_booking(num_tickets)

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

    def get_event_details(self):
        pass
```

Customer

```python
class Customer:
    def __init__(self, customer_name, email, phone):
        self.customer_name = customer_name
        self.email = email
        self.phone = phone

    def display_customer_details(self):
        print(f"Customer Name : {self.customer_name}")
        print(f"Email : {self.email}")
        print(f"Phone Number : {self.phone}")
```

Event

```python
class Event(Venue):
    def __init__(self, event_name, event_date, event_time, venue, total_seats, available_seats, ticket_price, event_
        self.event_name = event_name
        self.event_date = datetime.strptime(event_date, __format: "%Y-%m-%d").date()
        self.event_time = datetime.strptime(event_time, __format: "%H:%M").time()
        self.venue_name = venue.venue_name
        self.total_seats = total_seats
        self.available_seats = available_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    def calculate_total_revenue(self):
        return self.ticket_price * (self.total_seats - self.available_seats)

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

    1 usage
    def book_ticket(self, num_tickets):
        self.available_seats = self.available_seats - num_tickets

    1 usage
    def cancel_booking(self, num_tickets):
        self.available_seats = self.available_seats + num_tickets

    def display_event_details(self):
        print(f"Event name = {self.event_name}")
        print(f"Date of event = {self.event_date}")
        print(f"Time of event = {self.event_time}")
        print(f"Venue name = {self.venue_name}")
```

Venu

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

    def display_venue_details(self):
        print(f"Venue Name = {self.venue_name}")
        print(f"Address of venue = {self.address}")
```

Task 5: Inheritance and polymorphism

Movie

```python
from entities.Event import Event


class Movie(Event):
    def __init__(self, event_name, event_date, genre, actor_name, actress_name, customer=None):
        super().__init__(event_name, event_date, customer)
        self.genre = genre
        self.actor_name = actor_name
        self.actress_name = actress_name

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

Concert

```python
from entities.Event import Event


class Concert(Event):
    def __init__(self, event_name, event_date, artist, concert_type, customer=None):
        super().__init__(event_name, event_date, customer)
        self.artist = artist
        self.concert_type = concert_type

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

Sports

```python
from entities.Event import Event


class Sports(Event):
    def __init__(self, event_name, event_date, sport_name, teams_name, customer=None):
        super().__init__(event_name, event_date, customer)
        self.sport_name = sport_name
        self.teams_name = teams_name

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

TicketBookingSystem

```python
class TicketBookingSystem(EventServices, BookingSystemServices):
    def __init__(self, new_dbutil):
        super().__init__(new_dbutil)

    1 usage
    def main_menu(self):
        while True:
            print("\nSelect one options from the options given below : ")
            print("1. Create a new event.")
            print("2. Book tickets.")
            print("3. Cancel Tickets.")
            print("4. Know how many seats are Available.")
            print("5. See every event and it's details.")
            print("6. Exit.")
            choice = input("Enter your choice here : ")

            try:
                match choice:
                    case "1":
                        self.create_event()
                        print()
                    case "2":
                        num_tickets = int(input("\nPlease enter the number of tickets you want to book : "))
                        self.book_tickets(num_tickets)
                        print()
                    case "3":
                        booking_id = int(input("\nPlease enter your booking id here : "))
                        self.cancel_booking(booking_id)
                        print()
                    case "4":
                        self.get_available_tickets_count()
```

```python
                        num_tickets = int(input("\nPlease enter the number of tickets you want to book : "))
                        self.book_tickets(num_tickets)
                        print()
                    case "3":
                        booking_id = int(input("\nPlease enter your booking id here : "))
                        self.cancel_booking(booking_id)
                        print()
                    case "4":
                        self.get_available_tickets_count()
                        print()
                    case "5":
                        self.get_event_details()
                        print()
                    case "6":
                        break
                    case _:
                        print("Invalid input! Please Try Again.")
            except EventNotFoundException as e1:
                print("EventNotFoundException Exception occurred: ", e1)
            except InvalidBookingIDException as e2:
                print("InvalidBookingIDException Exception occurred: ", e2)
            except NullPointerException as e3:
                print("NullPointerException Exception occurred: ", e3)
            except Exception as ex:
                print("Exception occurred: ", ex)
```

Task 6: Abstraction

IBookingSystemServices

```python
from abc import *


2 usages
class IBookingSystemServices:

    @abstractmethod
    def calculate_booking_cost(self, num_tickets):
        pass

    @abstractmethod
    def book_tickets(self, num_tickets):
        pass

    @abstractmethod
    def cancel_booking(self, booking_id):
        pass

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

IEventServices

```python
from abc import *


2 usages
class IEventServices(ABC):
    @abstractmethod
    def create_event(self):
        pass

    @abstractmethod
    def get_event_details(self):
        pass

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

Task 7: Has A Relation / Association

Booking

```python
class Booking(Event):
    def __init__(self, event, customer):
        self._booking_id = random.randint( a: 10000,  b: 99999)
        self._customer = customer
        self._event = event
        self._num_tickets = len(customer)
        self._total_cost = 0
        self._booking_date = date.today()

    @property
    def booking_id(self):
        return self._booking_id

    1 usage
    @property
    def customer(self):
        return self._customer

    @customer.setter
    def customer(self, value):
        self._customer = value

    1 usage
    @property
    def event(self):
        return self._event
```

```python
    @property
    def num_tickets(self):
        return self._num_tickets

    @num_tickets.setter
    def num_tickets(self, value):
        self._num_tickets = value

    1 usage
    @property
    def total_cost(self):
        return self._total_cost

    @total_cost.setter
    def total_cost(self, value):
        self._total_cost = value

    1 usage
    @property
    def booking_date(self):
        return self._booking_date

    @booking_date.setter
    def booking_date(self, value):
        self._booking_date = value
```

Customer

```python
class Customer:
    def __init__(self, customer_name, email, phone):
        self._customer_name = customer_name
        self._email = email
        self._phone = phone

    1 usage
    @property
    def customer_name(self):
        return self._customer_name

    @customer_name.setter
    def customer_name(self, value):
        self._customer_name = value

    1 usage
    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, value):
        self._email = value

    1 usage
    @property
    def phone(self):
        return self._phone

    @phone.setter
    def phone(self, value):
```

```python
    1 usage
    @property
    def phone(self):
        return self._phone

    @phone.setter
    def phone(self, value):
        self._phone = value

    def display_customer_details(self):
        print(f"Customer Name: {self._customer_name}")
        print(f"Email: {self._email}")
        print(f"Phone Number: {self._phone}")

```

Event

```python
class Event(Venue):
    def __init__(self, event_name, event_date, event_time, venue, total_seats, available_seats, ticket_price, event_type):
        self._event_name = event_name
        self._event_date = datetime.strptime(event_date, __format: "%Y-%m-%d").date()
        self._event_time = datetime.strptime(event_time, __format: "%H:%M").time()
        self._venue_name = venue.venue_name
        self._total_seats = total_seats
        self._available_seats = available_seats
        self._ticket_price = ticket_price
        self._event_type = event_type

    1 usage
    @property
    def event_name(self):
        return self._event_name

    @event_name.setter
    def event_name(self, value):
        self._event_name = value

    1 usage
    @property
    def event_date(self):
        return self._event_date

    @event_date.setter
    def event_date(self, value):
        self._event_date = value
```

```python
    1 usage
    @property
    def event_time(self):
        return self._event_time

    @event_time.setter
    def event_time(self, value):
        self._event_time = value

    2 usages (1 dynamic)
    @property
    def venue_name(self):
        return self._venue_name

    1 usage (1 dynamic)
    @venue_name.setter
    def venue_name(self, value):
        self._venue_name = value

    1 usage
    @property
    def total_seats(self):
        return self._total_seats

    @total_seats.setter
    def total_seats(self, value):
        self._total_seats = value
```

Venue

```python
class Venue:
    def __init__(self, venue_name, address):
        self._venue_name = venue_name
        self._address = address

    2 usages (1 dynamic)
    @property
    def venue_name(self):
        return self._venue_name

    1 usage (1 dynamic)
    @venue_name.setter
    def venue_name(self, value):
        self._venue_name = value

    1 usage
    @property
    def address(self):
        return self._address

    @address.setter
    def address(self, value):
        self._address = value

    def display_venue_details(self):
        print(f"Venue Name = {self._venue_name}")
        print(f"Address of venue = {self._address}")
```

Task 8: Interface/abstract class, and Single Inheritance, static variable
Task 10: Collection

BookingSystemServices

```python
class BookingSystemServices(IBookingSystemServices):
    def __init__(self, dbutil):
        self.dbutil = dbutil

    def calculate_booking_cost(self, num_tickets):
        pass

    1 usage
    def book_tickets(self, num_tickets):
        # Take User Input
        print("\nPlease enter customer details: ")
        customer_name = input("Enter your name: ")
        customer_email = input("Enter you email: ")
        customer_phone = input("Enter your phone number: ")

        # Create Customer
        cursor = self.dbutil.get_cursor()
        cursor.execute("insert into customers(customer_name,email,phone_number) values (%s,%s,%s)",
                       (customer_name, customer_email, customer_phone,))
        cursor.fetchall()
        self.dbutil.con.commit()

        # Set Customer ID
        cursor.execute("select customer_id from customers where customer_name=%s", (customer_name,))
        cursor.fetchall()
        customer_row = cursor.fetchone()
        customer_id = -1
        if customer_row:
            customer_id = customer_row[0]
```

```python
        ticket_type = input("What type of ticket you want? - 1.Silver(x1) 2.Gold(x2) 3.Dimond(x3) ?")
        total_cost = price * num_tickets * ticket_type
        today = date.today()
        query = "insert into bookings (customer_id, event_id, num_tickets, total_cost, booking_date) values (%s,%s,%
        cursor.execute(query, (customer_id, event_id, num_tickets, total_cost, today))

        result = cursor.fetchall()
        if result is None:
            raise EventNotFoundException()

        self.dbutil.con.commit()

        # Get Booking
        cursor.execute("select booking_id from bookings where customer_id = %s", (customer_id,))
        booking_id = cursor.fetchone()
        if booking_id:
            b_id = booking_id[0]
        print("\nCongratulations! Your booking is confirmed. Your booking id is ", b_id)

    1 usage
    def cancel_booking(self, booking_id):
        cursor = self.dbutil.get_cursor()
        query = "delete from bookings where booking_id = %s"
        cursor.execute(query, (booking_id,))

        result = cursor.fetchall()
        if result is None:
            raise InvalidBookingIDException()

        self.dbutil.con.commit()
        print("\nYour booking is cancelled successfully.")
```

EventServices

```python
class EventServices(IEventServices):
    def __init__(self, dbutil):
        self.dbutil = dbutil

    1 usage
    def create_event(self):
        # Take User Input
        event_name = input("\nEnter event name: ")
        date = input("Enter event Date(Y-m-d): ")
        event_date = datetime.datetime.strptime(date, __format: "%Y-%m-%d").date()
        time = input("Enter event time in format HH:MM:SS: ")
        event_time = datetime.datetime.strptime(time, __format: "%H:%M:%S").time()
        venue = input("Enter venue name: ")
        venue_address = input("Enter venue address: ")
        total_seats = int(input("Enter total seats: "))
        available_seats = int(input("Enter available seats: "))
        ticket_price = float(input("Enter ticket price: "))
        event_type = input("Enter event type ['Movie','Sports','Concert']: ")

        # Create Venue
        cursor = self.dbutil.get_cursor()
        query = "insert into venues (venue_name, address) values (%s, %s)"
        cursor.execute(query, (venue, venue_address))
        self.dbutil.con.commit()

        # Get Venue ID for Event Creation
        cursor.execute("select venue_id from venues where venue_name=%s", (venue, ))
        venue_id = cursor.fetchone()

        # Create Event
```

```python
    def get_event_details(self):
        cursor = self.dbutil.get_cursor()
        cursor.execute("select * from events")
        events = cursor.fetchall()

        # Print all events
        for event in events:
            print(event)

    1 usage
    def get_available_tickets_count(self):
        cursor = self.dbutil.get_cursor()
        query = "select event_name from events"
        cursor.execute(query)
        event_names = cursor.fetchall()

        # Print all events
        print("\nPlease select one events from below: ")
        for event in event_names:
            print(event)

        # Get Tickets Count
        selected_event = input("\nPlease type your event name: ")
        query = "select available_seats from events where event_name=%s"
        cursor.execute(query, (selected_event, ))
        seats = cursor.fetchall()
        print("\nAvailable seats: ", seats)
```

Task 9: Exception Handling

EventNotFoundException.py

```
4 usages
class EventNotFoundException(Exception):
    def __init__(self, message="Event not found"):
        self.message = message
        super().__init__(self.message)
```

EventNotFoundException.py | InvalidBookingIDException.py

```
4 usages
class InvalidBookingIDException(Exception):
    def __init__(self, message="Invalid Booking Id"):
        self.message = message
        super().__init__(self.message)
```

EventNotFoundException.py | InvalidBookingIDException.py | NullPointerException.py

```
2 usages
class NullPointerException(Exception):
    def __init__(self, message="Null pointer exception"):
        self.message = message
        super().__init__(self.message)
```

Task 11: Database Connectivity

All Database operations are mentioned in services.

DBUtil.py

```
from mysql import connector
import mysql


2 usages
class DBUtil:
    def __init__(self):
        self.con = mysql.connector.connect(
            host="localhost",
            port="3306",
            user="root",
            password="root",
            database="ticketbookingsystem"
        )

    5 usages (5 dynamic)
    def get_cursor(self):
        return self.con.cursor()
```

Conclusion

Overall it is a full-fledged backend and database connection implementation. I recommend you check the project file by file to see all the features and miscellaneous things implemented.

Thank You!