HEXAWARE ASSIGNMENT 2

Ticket Booking System

You are tasked with creating a ticket booking system for a Event. The system should support booking tickets for different types of events, such as movies, concerts, and plays. Each event has its own pricing strategy, and the system should also track available seats and customer bookings.

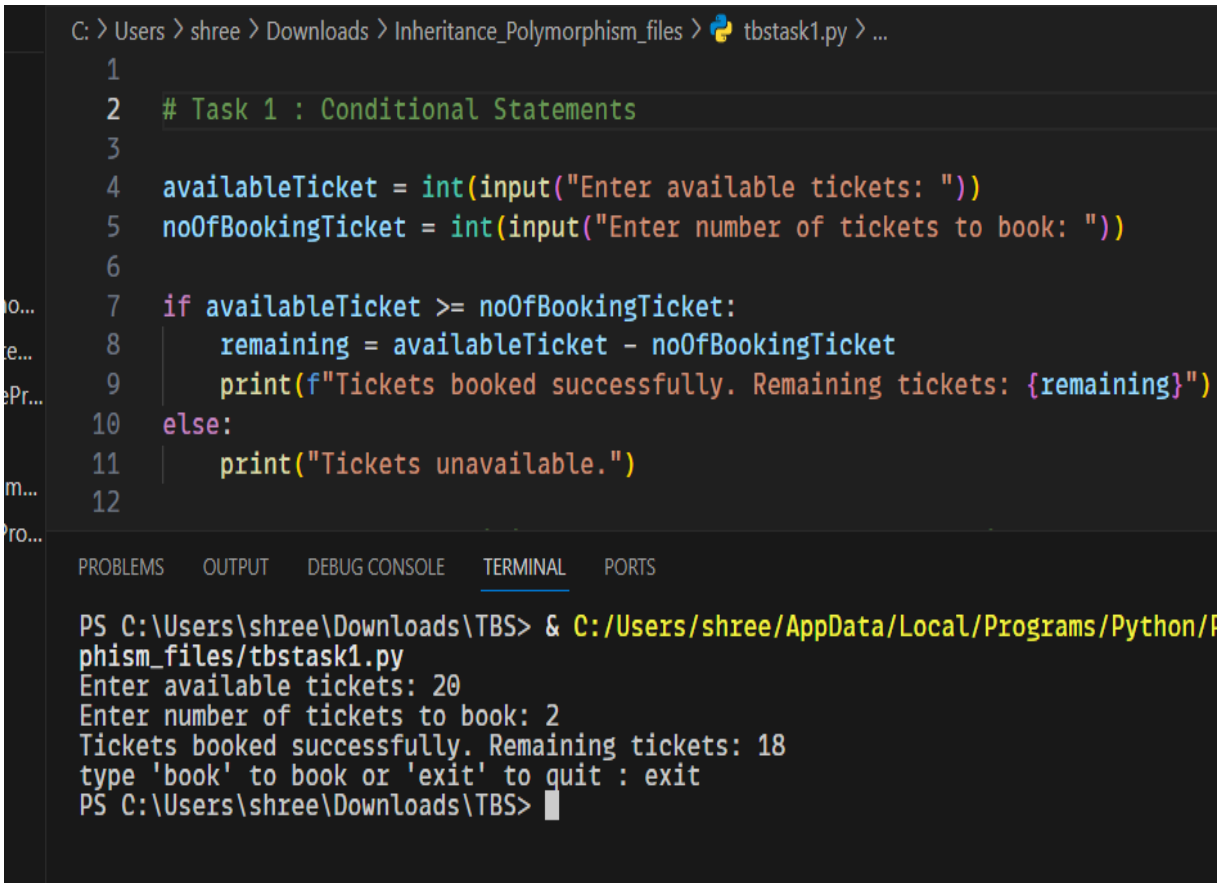
Control structure

Task 1:

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

Tasks:

1. Write a program that takes the availableTicket and noOfBookingTicket as input.
2. Use conditional statements (if-else) to determine if the ticket is available or not.
3. Display an appropriate message based on ticket availability.



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.

Task 3:

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

```
12
13  # Task 2 : Nested Conditional Statements AND Task 3: Looping
14
15  while True:
16      command = input("type 'book' to book or 'exit' to quit : ").lower()
17      if command == "exit":
18          break
19      print("choose Ticket Type : SILVER / GOLD / DIAMOND ")
20      ticket_type = input("enter ticket type : ").lower()
21      no_of_tickets = int(input("enter no. of tickets : "))
22
23      if ticket_type == "silver":
24          price = 100
25      elif ticket_type == "gold":
26          price = 200
27      elif ticket_type == "diamond":
28          price = 300
29      else:
30          print("invalid ticket type")
31          price = 0
32
33      if price > 0 :
34          total = price * no_of_tickets
35          print(f"Booking Successfull ! Total cost for {ticket_type.title()} tickets: rs{total}")
36
37
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
phism_files/tbstask1.py
type 'book' to book or 'exit' to quit : book
choose Ticket Type : SILVER / GOLD / DIAMOND
enter ticket type : gold
enter no. of tickets : 2
Booking Successfull ! Total cost for Gold tickets: rs400
type 'book' to book or 'exit' to quit : book
choose Ticket Type : SILVER / GOLD / DIAMOND
enter ticket type : silver
enter no. of tickets : 1
Booking Successfull ! Total cost for Silver tickets: rs100
type 'book' to book or 'exit' to quit :
```

Task 4: Class & Object

Create a Following classes with the following attributes and methods:

1. Event Class:
   - Attributes:
     - o event_name,
     - o event_date DATE,
     - o event_time TIME,
     - o venue_name,
     - o total_seats,
     - o available_seats,
     - o ticket_price DECIMAL,
     - o event_type ENUM('Movie', 'Sports', 'Concert')

   - Methods and Constuctors:
     - o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.
     - o calculate_total_revenue(): Calculate and return the total revenue based on the number of tickets sold.
     - o getBookedNoOfTickets(): return the total booked tickets
     - o 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.
     - o cancel_booking(num_tickets): Cancel the booking and update the available seats.
     - o display_event_details(): Display event details, including event name, date time seat availability.

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

3. Customer Class
   - Attributes:
     - o customer_name,
     - o email,
     - o phone_number,
   - Methods and Constuctors:

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

o display_customer_details(): Display customer details.

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:

o calculate_booking_cost(num_tickets): Calculate and set the total cost of the booking.

o book_tickets(num_tickets): Book a specified number of tickets for an event.

o cancel_booking(num_tickets): Cancel the booking and update the available seats.

o getAvailableNoOfTickets(): return the total available tickets

o getEventDetails(): return event details from the event class

CONSOLE OUTPUT USER INTERFACE

```
13
14  # Initialize service
15  system = BookingSystemServiceProviderImpl()
16
17  while True:
18      print("\n--- Ticket Booking System ---")
19      print("1. Create Event")
20      print("2. Book Tickets")
21      print("3. Cancel Booking")
22      print("4. View Booking")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice:
```

Create Event

- Prompts the user to enter event details (name, date, time, type, total seats, ticket price, and venue).
- Stores the event and venue in the database.

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 1
Enter Venue Name: sneha stage
Enter Venue Address: 12,Gandhi street
Enter Event Name: dance
Enter Event Date (YYYY-MM-DD): 2025-07-30
Enter Event Time (HH:MM): 16:00
Enter Total Seats: 3
Enter Ticket Price: 260
Enter Event Type (Movie/Concert/Sports): Concert
✅ Event created successfully!
```

Book Tickets

- Takes event name and number of tickets.
- Collects customer info for each ticket.
- Books the tickets and generates a unique booking ID.

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 2
Enter Event Name to Book: ai fusion
Enter Number of Tickets: 1
Enter Customer 1 Details:
Name: Pooja
Email: pooj@gmail
Phone: 1234567890

✅ Booking successful! Your Booking ID is: 14

✅ Booking Confirmed!
Booking ID: 14
Event Name: ai fusion
No. of Tickets: 1

Customer 1:
Name: Pooja
Email: pooj@gmail
Phone: 1234567890
```

Cancel Booking

- Asks for a booking ID and removes the booking.
- Restores the cancelled tickets to the event's available seats.

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 3
Enter Booking ID to Cancel: 14
✅ Booking 14 canceled and 1 tickets restored.
```

View Booking

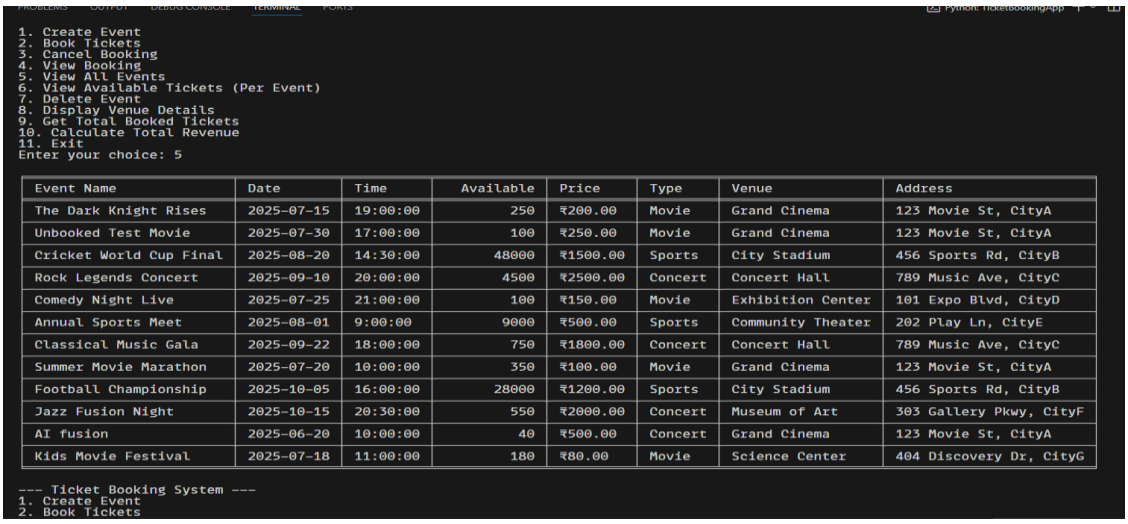
- Takes booking ID and displays all booking details (customers, event, number of tickets, etc.).

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 4
Enter Booking ID to View: 1

--- Booking Details ---
Booking ID      : 1
Customer Name   : Alice Smith
Email           : alice.s@example.com
Phone Number    : 9876543210
Event Name      : The Dark Knight Rises
Event Date      : 2025-07-15
Event Time      : 19:00:00
Event Type      : Movie
No. of Tickets  : 2
Total Cost      : ₹400.00
Booking Date    : 2025-06-24 10:27:51
```

View All Events

- Displays all upcoming events in a **tabular format**, including event name, date, time, available seats, type, venue, and address.



```
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 5
```

Event Name	Date	Time	Available	Price	Type	Venue	Address
The Dark Knight Rises	2025-07-15	19:00:00	250	₹200.00	Movie	Grand Cinema	123 Movie St, CityA
Unbooked Test Movie	2025-07-30	17:00:00	100	₹250.00	Movie	Grand Cinema	123 Movie St, CityA
Cricket World Cup Final	2025-08-20	14:30:00	48000	₹1500.00	Sports	City Stadium	456 Sports Rd, CityB
Rock Legends Concert	2025-09-10	20:00:00	4500	₹2500.00	Concert	Concert Hall	789 Music Ave, CityC
Comedy Night Live	2025-07-25	21:00:00	100	₹150.00	Movie	Exhibition Center	101 Expo Blvd, CityD
Annual Sports Meet	2025-08-01	9:00:00	9000	₹500.00	Sports	Community Theater	202 Play Ln, CityE
Classical Music Gala	2025-09-22	18:00:00	750	₹1800.00	Concert	Concert Hall	789 Music Ave, CityC
Summer Movie Marathon	2025-07-20	10:00:00	350	₹100.00	Movie	Grand Cinema	123 Movie St, CityA
Football Championship	2025-10-05	16:00:00	28000	₹1200.00	Sports	City Stadium	456 Sports Rd, CityB
Jazz Fusion Night	2025-10-15	20:30:00	550	₹2000.00	Concert	Museum of Art	303 Gallery Pkwy, CityF
AI fusion	2025-06-20	10:00:00	40	₹500.00	Concert	Grand Cinema	123 Movie St, CityA
Kids Movie Festival	2025-07-18	11:00:00	180	₹80.00	Movie	Science Center	404 Discovery Dr, CityG

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
```

View Available Tickets (Per Event)

- Lists the number of remaining tickets for each event in tabular format.

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 6
```

Event Name	Available Tickets
The Dark Knight Rises	250
Unbooked Test Movie	100
Cricket World Cup Final	48000
Rock Legends Concert	4500
Comedy Night Live	100
Annual Sports Meet	9000
Classical Music Gala	750
Summer Movie Marathon	350
Football Championship	28000
Jazz Fusion Night	550
AI fusion	40
Kids Movie Festival	180
dance	3

Delete Event

- Asks for event name, date, and time.
- Deletes that specific event and may also remove related bookings if cascade is enabled.

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 7
🗑️Delete Event
Enter Event Name: dance
Enter Event Date (YYYY-MM-DD): 2025-07-30
Enter Event Time (HH:MM:SS): 16:00:00
✅ Event deleted successfully.
```

Display Venue Details

- Shows all venue names and addresses

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 8
Venue: Grand Cinema | Address: 123 Movie St, CityA
Venue: City Stadium | Address: 456 Sports Rd, CityB
Venue: Concert Hall | Address: 789 Music Ave, CityC
Venue: Exhibition Center | Address: 101 Expo Blvd, CityD
Venue: Community Theater | Address: 202 Play Ln, CityE
Venue: Museum of Art | Address: 303 Gallery Pkwy, CityF
Venue: Science Center | Address: 404 Discovery Dr, CityG
Venue: Botanical Gardens | Address: 505 Green St, CityH
Venue: Waterfront Park | Address: 606 Oceanfront Rd, CityI
Venue: The Jazz Club | Address: 888 Melody Lane, CityK
Venue: Children's Museum Auditorium | Address: 999 Imagination Blvd, CityL
Venue: Innovation Hub | Address: 707 Tech Ave, CityJ
Venue: home | Address: 1   , home
Venue: sneha stage | Address: 12,Gandhi street
```

Get Total Booked Tickets

- Returns the total number of tickets sold (sum of all bookings).

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 9
Total Booked Tickets: 35
```

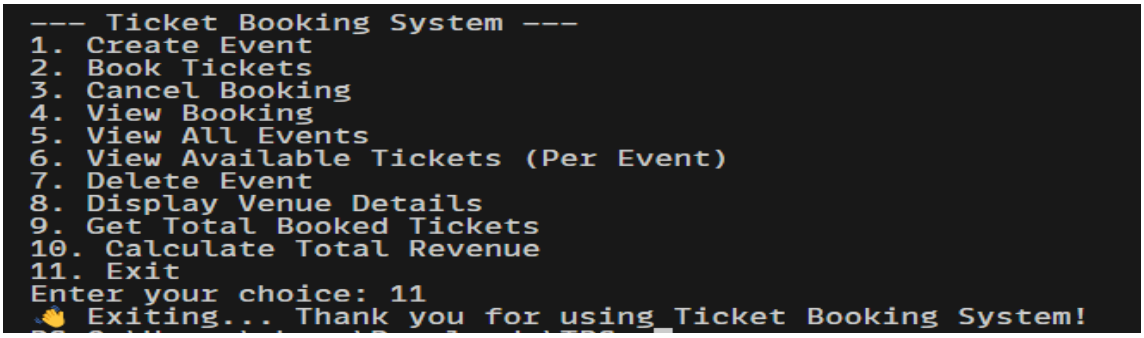
Calculate Total Revenue

- Calculates and displays the total revenue earned from all bookings by summing num_tickets × ticket_price.

```
--- Ticket Booking System ---
1. Create Event
2. Book Tickets
3. Cancel Booking
4. View Booking
5. View All Events
6. View Available Tickets (Per Event)
7. Delete Event
8. Display Venue Details
9. Get Total Booked Tickets
10. Calculate Total Revenue
11. Exit
Enter your choice: 10
Total Revenue: ₹23650.00
```

Exit

- Safely exits the program.



TICKET BOOKING SYSTEM APP (Main code.py)

```
import sys
import os
from tabulate import tabulate

# Add TBS root directory to sys.path
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from service.BookingSystemServiceProviderImpl import
BookingSystemServiceProviderImpl
from entity.venue import Venue
from entity.customer import Customer
from exception.EventNotFoundException import EventNotFoundException
from exception.InvalidBookingIDException import InvalidBookingIDException

# Initialize service
system = BookingSystemServiceProviderImpl()

while True:
    print("\n--- Ticket Booking System ---")
    print("1. Create Event")
    print("2. Book Tickets")
    print("3. Cancel Booking")
    print("4. View Booking")
    print("5. View All Events")
    print("6. View Available Tickets (Per Event)")
    print("7. Delete Event")
    print("8. Display Venue Details")
    print("9. Get Total Booked Tickets")
```

```
    print("10. Calculate Total Revenue")
    print("11. Exit")

    choice = input("Enter your choice: ")

    try:
        if choice == "1":
            venue = Venue(
                input("Enter Venue Name: "),
                input("Enter Venue Address: ")
            )
            system.create_event(
                input("Enter Event Name: "),
                input("Enter Event Date (YYYY-MM-DD): "),
                input("Enter Event Time (HH:MM): ") + ":00",
                int(input("Enter Total Seats: ")),
                float(input("Enter Ticket Price: ")),
                input("Enter Event Type (Movie/Concert/Sports): "),
                venue
            )
            print("✅ Event created successfully!")

        elif choice == "2":
            event_name = input("Enter Event Name to Book: ")
            num = int(input("Enter Number of Tickets: "))
            customers = []
            for i in range(num):
                print(f"Enter Customer {i+1} Details:")
                name = input("Name: ")
                email = input("Email: ")
                phone = input("Phone: ")
                customers.append(Customer(name, email, phone))

            booking = system.book_tickets(event_name, num, customers)
            print(f"\n✅ Booking successful! Your Booking ID is:
{booking.booking_id}")
            booking.display_booking_details()

        elif choice == "3":
            booking_id = int(input("Enter Booking ID to Cancel: "))
            system.cancel_booking(booking_id)

        elif choice == "4":
```

```
            booking_id = int(input("Enter Booking ID to View: "))
            system.get_booking_details(booking_id)

        elif choice == "5":
            events = system.get_event_details()
            if not events:
                print("No events found.")
            else:
                headers = ["Event Name", "Date", "Time", "Available", "Price", "Type",
"Venue", "Address"]
                table = [e.to_row() for e in events]
                print("\n" + tabulate(table, headers=headers, tablefmt="fancy_grid"))

        elif choice == "6":
            event_tickets = system.get_available_tickets_per_event()
            if not event_tickets:
                print("No events found.")
            else:
                headers = ["Event Name", "Available Tickets"]
                print("\n" + tabulate(event_tickets, headers=headers,
tablefmt="fancy_grid"))

        elif choice == "7":
            print("🗑️ Delete Event")
            name = input("Enter Event Name: ")
            date = input("Enter Event Date (YYYY-MM-DD): ")
            time = input("Enter Event Time (HH:MM:SS): ")
            system.delete_event(name, date, time)

        elif choice == "8":
            venues = system.get_venue_details()
            if not venues:
                print("❌ No venues found.")
            else:
                headers = ["Venue Name", "Address"]
                print("\n📍 Venue Details:\n")
                print(tabulate(venues, headers=headers, tablefmt="fancy_grid"))

        elif choice == "9":
            total_booked = system.get_booked_no_of_tickets()
            print(f"Total Booked Tickets: {total_booked}")

        elif choice == "10":
```

```
            revenue = system.calculate_total_revenue()
            print(f"Total Revenue: ₹{revenue:.2f}")

        elif choice == "11":
            print("👋 Exiting... Thank you for using Ticket Booking System!")
            break

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

    except EventNotFoundException as e:
        print("[Event Error]", e)
    except InvalidBookingIDException as e:
        print("[Booking Error]", e)
    except Exception as e:
        print("[Unexpected Error]", e)
```

DATABASE FOLDER :

```
import mysql.connector

def run_setup():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Poojashree",
            database= "TicketBookingSystem"

        )
        cursor = conn.cursor()
        cursor.execute("CREATE DATABASE IF NOT EXISTS TicketBookingSystem")
        cursor.execute("USE TicketBookingSystem")

        # ---------------------- CREATE TABLES ----------------------
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS Venue (
                venue_id INT AUTO_INCREMENT PRIMARY KEY,
                venue_name VARCHAR(255),
                address VARCHAR(255)
            );
        """)
```

```
cursor.execute("""
   CREATE TABLE IF NOT EXISTS Event (
      event_id INT AUTO_INCREMENT PRIMARY KEY,
      event_name VARCHAR(200) NOT NULL,
      event_date DATE NOT NULL,
      event_time TIME NOT NULL,
      total_seats INT NOT NULL,
      available_seats INT NOT NULL,
      venue_id INT NOT NULL,
      ticket_price DECIMAL(10,2) NOT NULL,
      event_type ENUM('Movie', 'Sports', 'Concert'),
      FOREIGN KEY (venue_id) REFERENCES Venue(venue_id)
   );
""")

cursor.execute("""
   CREATE TABLE IF NOT EXISTS Customer (
      customer_id INT AUTO_INCREMENT PRIMARY KEY,
      customer_name VARCHAR(200) NOT NULL,
      email VARCHAR(200) UNIQUE NOT NULL,
      phone_number VARCHAR(20) NOT NULL
   );
""")

cursor.execute("""
   CREATE TABLE IF NOT EXISTS Booking (
      booking_id INT AUTO_INCREMENT PRIMARY KEY,
      customer_id INT NOT NULL,
      event_id INT NOT NULL,
      num_tickets INT NOT NULL,
      total_cost DECIMAL(10,2) NOT NULL,
      booking_date DATETIME DEFAULT CURRENT_TIMESTAMP,
      FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
      FOREIGN KEY (event_id) REFERENCES Event(event_id)
   );
""")

# ---------- CLEAN OLD DATA ----------
cursor.execute("DELETE FROM Booking")
cursor.execute("DELETE FROM Event")
cursor.execute("DELETE FROM Customer")
cursor.execute("DELETE FROM Venue")

cursor.execute("ALTER TABLE Venue AUTO_INCREMENT = 1")
```

```
        cursor.execute("ALTER TABLE Event AUTO_INCREMENT = 1")
        cursor.execute("ALTER TABLE Customer AUTO_INCREMENT = 1")
        cursor.execute("ALTER TABLE Booking AUTO_INCREMENT = 1")

        # ---------- INSERT VENUES ----------
        cursor.executemany("""
            INSERT INTO Venue (venue_name, address) VALUES (%s, %s)
        """, [
            ('Grand Cinema', '123 Movie St, CityA'),
            ('City Stadium', '456 Sports Rd, CityB'),
            ('Concert Hall', '789 Music Ave, CityC'),
            ('Exhibition Center', '101 Expo Blvd, CityD'),
            ('Community Theater', '202 Play Ln, CityE'),
            ('Museum of Art', '303 Gallery Pkwy, CityF'),
            ('Science Center', '404 Discovery Dr, CityG'),
            ('Botanical Gardens', '505 Green St, CityH'),
            ('Waterfront Park', '606 Oceanfront Rd, CityI'),
            ('The Jazz Club', '888 Melody Lane, CityK'),
            ("Children's Museum Auditorium", '999 Imagination Blvd, CityL'),
            ('Innovation Hub', '707 Tech Ave, CityJ')
        ])
        conn.commit()  # ✅ Commit venue insert

        # ---------- INSERT EVENTS ----------
        cursor.executemany("""
            INSERT INTO Event (event_name, event_date, event_time, venue_id,
                               total_seats, available_seats, ticket_price, event_type)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """, [
            ('The Dark Knight Rises', '2025-07-15', '19:00:00', 1, 300, 250, 200.00,
'Movie'),
            ('Unbooked Test Movie', '2025-07-30', '17:00:00', 1, 100, 100, 250.00,
'Movie'),
            ('Cricket World Cup Final', '2025-08-20', '14:30:00', 2, 50000, 48000, 1500.00,
'Sports'),
            ('Rock Legends Concert', '2025-09-10', '20:00:00', 3, 5000, 4500, 2500.00,
'Concert'),
            ('Comedy Night Live', '2025-07-25', '21:00:00', 4, 150, 100, 150.00, 'Movie'),
            ('Annual Sports Meet', '2025-08-01', '09:00:00', 5, 10000, 9000, 500.00,
'Sports'),
            ('Classical Music Gala', '2025-09-22', '18:00:00', 3, 800, 750, 1800.00,
'Concert'),
```

```
        ('Summer Movie Marathon', '2025-07-20', '10:00:00', 1, 400, 350, 100.00,
'Movie'),
        ('Football Championship', '2025-10-05', '16:00:00', 2, 30000, 28000, 1200.00,
'Sports'),
        ('Jazz Fusion Night', '2025-10-15', '20:30:00', 6, 600, 550, 2000.00, 'Concert'),
        ('AI fusion', '2025-06-20', '10:00:00', 1, 100, 40, 500.00, 'Concert'),
        ('Kids Movie Festival', '2025-07-18', '11:00:00', 7, 200, 180, 80.00, 'Movie')
    ])
    conn.commit()

    # ---------- INSERT CUSTOMERS ----------
    cursor.executemany("""
        INSERT INTO Customer (customer_name, email, phone_number)
        VALUES (%s, %s, %s)
    """, [
        ('Alice Smith', 'alice.s@example.com', '9876543210'),
        ('John NoBook', 'john.nobook@example.com', '9999988888'),
        ('Bob Johnson', 'bob.j@example.com', '9988776655'),
        ('Charlie Brown', 'charlie.b@example.com', '9000000000'),
        ('Diana Prince', 'diana.p@example.com', '9111111111'),
        ('Eve Adams', 'eve.a@example.com', '9222222222'),
        ('Frank Green', 'frank.g@example.com', '9333333333'),
        ('Grace Hopper', 'grace.h@example.com', '9444444444'),
        ('Harry Potter', 'harry.p@example.com', '9555555555'),
        ('Ivy Rose', 'ivy.r@example.com', '9666666666'),
        ('Jack Sparrow', 'jack.s@example.com', '9777777777')
    ])
    conn.commit()

    # ---------- INSERT BOOKINGS ----------
    cursor.executemany("""
        INSERT INTO Booking (customer_id, event_id, num_tickets, total_cost)
        VALUES (%s, %s, %s, %s)
    """, [
        (1, 1, 2, 400.00),
        (2, 2, 5, 7500.00),
        (3, 3, 1, 2500.00),
        (4, 1, 3, 600.00),
        (5, 4, 1, 150.00),
        (6, 5, 6, 3000.00),
        (7, 6, 2, 3600.00),
        (8, 7, 4, 400.00),
        (9, 8, 5, 6000.00),
        (10, 9, 3, 6000.00),
```

```
            (1, 10, 2, 160.00),
            (2, 1, 1, 200.00)
        ])
        conn.commit()

        print("✅ Database setup completed successfully.")

    except mysql.connector.Error as err:
        print("[❌ MySQL Error]", err)

    finally:
        try:
            if cursor:
                cursor.close()
        except:
            pass
        try:
            if conn:
                conn.close()
        except:
            pass

# Run setup
run_setup()
```

ENTITY FOLDER

```
class Booking:
    def __init__(self, booking_id, event_name, num_tickets, customers):
        self.booking_id = booking_id
        self.event_name = event_name
        self.num_tickets = num_tickets
        self.customers = customers  # List of Customer objects

    def display_booking_details(self):
        print(f"\n✅ Booking Confirmed!")
        print(f"Booking ID: {self.booking_id}")
        print(f"Event Name: {self.event_name}")
        print(f"No. of Tickets: {self.num_tickets}")
        for i, customer in enumerate(self.customers, start=1):
            print(f"\nCustomer {i}:")
            print(f"Name: {customer.name}")
```

```
            print(f"Email: {customer.email}")
            print(f"Phone: {customer.phone}")

class Customer:
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone

class Event:
    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 = event_date
        self.event_time = event_time
        self.venue = venue  # Venue object
        self.total_seats = total_seats
        self.available_seats = available_seats
        self.ticket_price = ticket_price
        self.event_type = event_type

    def book_tickets(self, num_tickets):
        if self.available_seats >= num_tickets:
            self.available_seats -= num_tickets
            return True
        return False

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

    def display_event_details(self):
        print(f"Event: {self.event_name}, Date: {self.event_date}, Time:
{self.event_time}, "
              f"Available: {self.available_seats}, Price: ₹{self.ticket_price:.2f}, Type:
{self.event_type}")
        self.venue.display_venue_details()

    def to_row(self):
        return [
            self.event_name,
            str(self.event_date),
            str(self.event_time),
            str(self.available_seats),
            f"₹{self.ticket_price:.2f}",
```

```
            self.event_type,
            self.venue.venue_name,
            self.venue.address
        ]

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

    def to_row(self):
        return [self.venue_name, self.address]
```

EXCEPTION HANDLING FOLDER

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

class InvalidBookingIDException(Exception):
    def __init__(self, message="Invalid Booking ID"):
        super().__init__(message)

class InvalidInputException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

SERVICE FOLDER

#BookingSystemServiceProviderImpl.py

```
import mysql.connector
from tabulate import tabulate
from entity.event import Event
from entity.venue import Venue
from entity.customer import Customer
from entity.booking import Booking
from exception.InvalidBookingIDException import InvalidBookingIDException
from exception.EventNotFoundException import EventNotFoundException
```

```
class BookingSystemServiceProviderImpl:

    def __init__(self):
        self.conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Poojashree",
            database="TicketBookingSystem"
        )
        self.cursor = self.conn.cursor(dictionary=True)

    def create_event(self, event_name, event_date, event_time, total_seats,
ticket_price, event_type, venue_obj):
        try:
            self.cursor.execute("SELECT venue_id FROM Venue WHERE venue_name =
%s AND address = %s",
                                (venue_obj.venue_name, venue_obj.address))
            result = self.cursor.fetchone()

            if result:
                venue_id = result['venue_id']
            else:
                self.cursor.execute("INSERT INTO Venue (venue_name, address) VALUES
(%s, %s)",
                                    (venue_obj.venue_name, venue_obj.address))
                venue_id = self.cursor.lastrowid

            self.cursor.execute("""
                INSERT INTO Event (event_name, event_date, event_time, total_seats,
                                   available_seats, venue_id, ticket_price, event_type)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """, (event_name, event_date, event_time, total_seats, total_seats,
                  venue_id, ticket_price, event_type))
            self.conn.commit()
        except Exception as e:
            print("[Error creating event]", e)
            self.conn.rollback()

    def get_event_details(self):
        try:
            self.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_name, v.address
        FROM Event e
        JOIN Venue v ON e.venue_id = v.venue_id
    """)
    rows = self.cursor.fetchall()
    events = []

    for row in rows:
        venue = Venue(row['venue_name'], row['address'])
        event = Event(
            event_name=row['event_name'],
            event_date=row['event_date'],
            event_time=row['event_time'],
            total_seats=row['total_seats'],
            available_seats=row['available_seats'],
            ticket_price=row['ticket_price'],
            event_type=row['event_type'],
            venue=venue
        )
        events.append(event)
    return events
except Exception as e:
    print("[Error fetching event details]", e)
    return []

def get_booking_details(self, booking_id):
    try:
        self.cursor.execute("""
            SELECT b.booking_id, b.num_tickets, b.total_cost, b.booking_date,
                c.customer_name, c.email, c.phone_number,
                e.event_name, e.event_date, e.event_time, e.event_type
            FROM Booking b
            JOIN Customer c ON b.customer_id = c.customer_id
            JOIN Event e ON b.event_id = e.event_id
            WHERE b.booking_id = %s
        """, (booking_id,))
        row = self.cursor.fetchone()

        if not row:
            raise InvalidBookingIDException(f"Booking ID {booking_id} not found")

        print("\n--- Booking Details ---")
        print(f"Booking ID     : {row['booking_id']}")
        print(f"Customer Name  : {row['customer_name']}")
```

```
            print(f"Email         : {row['email']}")
            print(f"Phone Number  : {row['phone_number']}")
            print(f"Event Name    : {row['event_name']}")
            print(f"Event Date    : {row['event_date']}")
            print(f"Event Time    : {row['event_time']}")
            print(f"Event Type    : {row['event_type']}")
            print(f"No. of Tickets : {row['num_tickets']}")
            print(f"Total Cost    : ₹{row['total_cost']}")
            print(f"Booking Date  : {row['booking_date']}")

        except InvalidBookingIDException as ibe:
            raise ibe
        except Exception as e:
            print("[Error fetching booking details]", e)

    def get_available_no_of_tickets(self):
        try:
            self.cursor.execute("SELECT SUM(available_seats) AS total FROM Event")
            result = self.cursor.fetchone()
            return result['total'] if result['total'] else 0
        except Exception as e:
            print("[Error getting available tickets]", e)
            return 0

    def get_available_tickets_per_event(self):
        try:
            self.cursor.execute("""
                SELECT event_name, available_seats
                FROM Event
            """)
            return [(row['event_name'], row['available_seats']) for row in
self.cursor.fetchall()]
        except Exception as e:
            print("[Error fetching per-event tickets]", e)
            return []

    def cancel_booking(self, booking_id):
        try:
            self.cursor.execute("SELECT * FROM Booking WHERE booking_id = %s",
(booking_id,))
            booking = self.cursor.fetchone()

            if not booking:
                raise InvalidBookingIDException(f"Booking ID {booking_id} not found.")
```

```
            event_id = booking['event_id']
            num_tickets = booking['num_tickets']

            self.cursor.execute("DELETE FROM Booking WHERE booking_id = %s",
(booking_id,))
            self.cursor.execute("""
                UPDATE Event
                SET available_seats = available_seats + %s
                WHERE event_id = %s
            """, (num_tickets, event_id))

            self.conn.commit()
            print(f"✅ Booking {booking_id} canceled and {num_tickets} tickets
restored.")

        except InvalidBookingIDException as ibe:
            raise ibe
        except Exception as e:
            self.conn.rollback()
            print("[Error cancelling booking]", e)

    def book_tickets(self, event_name, num_tickets, customers):
        try:
            self.cursor.execute("SELECT * FROM Event WHERE event_name = %s",
(event_name,))
            event = self.cursor.fetchone()

            if not event:
                raise EventNotFoundException(f"Event '{event_name}' not found.")
            if event['available_seats'] < num_tickets:
                raise Exception("Not enough tickets available.")

            event_id = event['event_id']
            ticket_price = event['ticket_price']
            total_cost = ticket_price * num_tickets

            customer = customers[0]
            self.cursor.execute("""
                INSERT INTO Customer (customer_name, email, phone_number)
                VALUES (%s, %s, %s)
            """, (customer.name, customer.email, customer.phone))
            customer_id = self.cursor.lastrowid
```

```
            self.cursor.execute("""
                INSERT INTO Booking (customer_id, event_id, num_tickets, total_cost)
                VALUES (%s, %s, %s, %s)
            """, (customer_id, event_id, num_tickets, total_cost))
            booking_id = self.cursor.lastrowid

            self.cursor.execute("""
                UPDATE Event
                SET available_seats = available_seats - %s
                WHERE event_id = %s
            """, (num_tickets, event_id))

            self.conn.commit()

            return Booking(booking_id, event_name, num_tickets, [customer])

        except EventNotFoundException as enfe:
            raise enfe
        except Exception as e:
            self.conn.rollback()
            print("[Error booking tickets]", e)

    def delete_event(self, event_name, event_date, event_time):
        try:
            self.cursor.execute("""
                DELETE FROM Event
                WHERE event_name = %s AND event_date = %s AND event_time = %s
            """, (event_name, event_date, event_time))
            self.conn.commit()
            print("✅ Event deleted successfully.")
        except Exception as e:
            self.conn.rollback()
            print("[Error deleting event]", e)

    def calculate_total_revenue(self):
        try:
            self.cursor.execute("SELECT SUM(total_cost) AS total FROM Booking")
            result = self.cursor.fetchone()
            return result['total'] if result['total'] else 0.0
        except Exception as e:
            print("[Error calculating revenue]", e)
            return 0.0
```

```
    def get_booked_no_of_tickets(self):
        try:
            self.cursor.execute("SELECT SUM(num_tickets) AS total FROM Booking")
            result = self.cursor.fetchone()
            return result['total'] if result['total'] else 0
        except Exception as e:
            print("[Error counting booked tickets]", e)
            return 0

    def get_venue_details(self):
        try:
            self.cursor.execute("SELECT venue_name, address FROM Venue ORDER
BY venue_name ASC")
            venues = self.cursor.fetchall()
            return [Venue(v['venue_name'], v['address']).to_row() for v in venues]
        except Exception as e:
            print("[Error fetching venue details]", e)
            return []


    def calculate_booking_cost(self, num_tickets, ticket_price):
        try:
            return num_tickets * ticket_price
        except Exception as e:
            print("[Error calculating cost]", e)
            return 0.0
```

UTIL FOLDER

```
import mysql.connector

def fetch_table_data(cursor, table_name):
    print(f"\n--- {table_name.upper()} TABLE ---")
    try:
        cursor.execute(f"SELECT * FROM {table_name}")
        rows = cursor.fetchall()
        if rows:
            for row in rows:
                print(row)
        else:
            print("No records found.")
    except Exception as e:
```

```
        print(f"[Error retrieving data from {table_name}]:", e)

def display_all_tables():
    try:
        # Step 1: Connect to MySQL
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Poojashree",
            database="TicketBookingSystem"
        )

        cursor = conn.cursor()

        # Step 2: List all tables you want to fetch
        tables = ['events', 'venue', 'bookings']

        # Step 3: Fetch data for each table
        for table in tables:
            fetch_table_data(cursor, table)

        # Step 4: Close connections
        cursor.close()
        conn.close()

    except mysql.connector.Error as err:
        print("[MySQL Error]", err)
    except Exception as e:
        print("[Error]", e)

#  Step 5: Call display_all_tables() when script runs
if __name__ == "__main__":
    display_all_tables()
```

PYTEST FILES

#test_db.py

```
from util.DBConnUtil import DBConnUtil

conn = DBConnUtil.get_connection()
if conn:
    print(" ✅ Connection successful!")
```

```
    conn.close()
else:
    print("❌ Connection failed.")
```

 # test_booking.py

```
import pytest
from entity.customer import Customer
from service.BookingSystemServiceProviderImpl import
BookingSystemServiceProviderImpl
from exception.EventNotFoundException import EventNotFoundException
import uuid

system = BookingSystemServiceProviderImpl()

def test_successful_booking():
    unique_email = f"test_{uuid.uuid4().hex[:6]}@example.com"
    customer = Customer("Test User", unique_email, "9876543210")
    booking = system.book_tickets("The Dark Knight Rises", 1, [customer])

    assert booking is not None
    assert booking.booking_id is not None
    assert booking.event_name == "The Dark Knight Rises"

def test_booking_invalid_event():
    with pytest.raises(EventNotFoundException):
        system.book_tickets("Nonexistent Event", 1, [Customer("X",
"x@example.com", "123")])
```

 # test_event_creation.py

```
import pytest # type: ignore
from entity.venue import Venue
from service.BookingSystemServiceProviderImpl import
BookingSystemServiceProviderImpl

system = BookingSystemServiceProviderImpl()

def test_create_event():
    venue = Venue("Test Venue", "123 Test St")
    result = system.create_event(
        event_name="Test Event",
```

```
        event_date="2025-12-25",
        event_time="18:00:00",
        total_seats=100,
        ticket_price=300.0,
        event_type="Movie",
        venue_obj=venue
    )
    assert result is None  # create_event does not return; use DB to verify in integration
tests

# test_exceptions.py
import pytest # type: ignore
from service.BookingSystemServiceProviderImpl import
BookingSystemServiceProviderImpl
from exception.InvalidBookingIDException import InvalidBookingIDException

system = BookingSystemServiceProviderImpl()

def test_invalid_booking_id():
    with pytest.raises(InvalidBookingIDException):
        system.get_booking_details(-999)
```