PROJECT NAME: EVENT MANAGEMENT SYSTEM

NAME: PRIYADHARSHINI. P

DATE SUBMITTED: 09/09/2024

EVENT MANAGEMENT SYSTEM

- An Event Management System (EMS) is a software platform that helps plan, organize, and execute events efficiently.

- It streamlines processes like registration, ticketing, and attendee management. EMS also facilitates communication among organizers, attendees, speakers, and sponsors. Additionally, it provides analytics and insights to enhance the event experience.

- Overall, an EMS reduces manual efforts, saves time, and improves event outcomes. It's a one-stop solution for all event management needs.

1. AIM OF THE PROJECT:

- The aim of this project is to design and develop a comprehensive Event Management System that streamlines the process of organizing and attending events.

- The system seeks to improve communication and collaboration among event stakeholders, enhance the overall event experience, and provide real-time insights and analytics to inform future events.

- By automating and simplifying event registration, ticketing, and attendee management, the system aims to reduce costs and manual efforts, increase attendee engagement and satisfaction, and ensure data security and integrity.

- Ultimately, the project aims to create a user-friendly, scalable, and flexible platform that meets the diverse needs of event organizers and attendees.

2. BUSINESS PROBLEM AND PROBLEM STATEMENT :

Manual User and Role Management:

- Difficulty in securely and efficiently managing user registrations, including different roles (e.g., organizers and attendees).
- Risks associated with handling user credentials manually.

Venue Capacity Constraints:

- Challenges in accurately tracking venue capacities and ensuring that events do not exceed these limits.
- Inefficiencies in managing and updating venue information in real-time.

Event and Attendee Coordination:

- Complexity in managing attendee lists, including adding and removing attendees as needed.
- Issues with ensuring that only registered users can attend and managing cancellations effectively.

Lack of Real-Time Updates:

- Without real-time data management, event organizers struggle to provide up-to-date information to attendees, which can lead to confusion and dissatisfaction.

Complex Booking and Cancellation Processes:

- Handling ticket bookings, seat assignments, and cancellations manually can lead to discrepancies, wasted resources, and poor customer experience.

Automated User Management:

- Facilitate easy user registration, authentication, and role management (organizers vs. attendees) with minimal manual intervention.

Effective Capacity Control:

- Implement a system to automatically monitor and enforce venue capacity limits, preventing overbooking and ensuring optimal utilization.

Booking and Cancellation:

- Develop a user-friendly interface for booking tickets, assigning seats, and processing cancellations to ensure accuracy and efficiency.

Real-Time Data Management:

- Provide real-time updates and information about events, bookings, and attendees to both organizers and users to enhance transparency and communication.

3. PROJECT DESCRIPTION :

The project is a comprehensive event management and booking system designed to streamline the process of organizing and attending events. The system comprises several key classes that encapsulate various aspects of event management, including Venue, Event, User, Ticket, Booking, and Authentication.

Entity Class:

- This is the base class for entities with a name, used by both Venue and Event to manage common attributes.

Venue Class:

- Inherits from Entity and extends it by including details specific to a venue, such as its address and capacity. This class ensures that venue information is encapsulated and can be easily retrieved.

User Class:

- Inherits from Person, representing an individual user of the system. It includes additional attributes for password and role, providing methods for user authentication and role management.

Event Class:

- Inherits from Entity and represents a specific event. It manages the event's date, venue, organizer, and attendees. Methods in this class handle adding and removing attendees, ensuring the event does not exceed its capacity.

Ticket Class:

- Represents a ticket for an event, storing details about the event, the user who booked the ticket, and the seat number.

Booking Class:

- Handles the booking process, including ticket creation and cancellation. It ensures that tickets are only booked if there is available capacity and maintains a record of all bookings.

Authentication Class:

- Manages user registration and login processes. It maintains a list of registered users and provides methods to authenticate users based on email and password.

The main function serves as the user interface for the system. It allows users to register as organizers or attendees, manage events, and handle ticket bookings and cancellations. The system ensures a smooth interaction flow by guiding users through the necessary steps and validating their inputs.

This project effectively demonstrates object-oriented principles in Python, including inheritance, encapsulation, and method overriding, while providing a practical tool for managing events and bookings.

4. FUNCTIONALITIES :

The Event Management and Booking System offers a range of functionalities designed to simplify the management of events and facilitate ticket bookings.

User Management:

- **Registration**: Users can register as either organizers or attendees, specifying their personal details (name, email, password) and their role.

- **Authentication**: The system includes secure login and authentication mechanisms to verify user credentials, ensuring that only registered users can access their respective functionalities.

Event Creation and Customization:

- **Event Details**: Organizers can create events by providing key details such as event name, date, and venue.

- **Venue Association**: Events are linked to specific venues, allowing organizers to specify where the event will take place, including the venue's address and seating capacity.

Venue Management:

- **Venue Information**: Organizers can input and update venue details, including its name, address, and capacity.

- **Capacity Constraints**: The system enforces venue capacity limits, ensuring that the number of attendees does not exceed the venue's seating capacity.

Ticketing System:

- **Booking Tickets**: Attendees can book tickets for events, choosing their preferred seat numbers. The system ensures that tickets are only issued if seats are available.

- **Ticket Management**: Tickets are created with detailed information linking the attendee, event, and seat number. This includes methods for retrieving and displaying ticket information.

Attendee Management:

- **Add/Remove Attendees**: Organizers can add or remove attendees from an event based on ticket bookings or cancellations.

- **Attendee List**: The system provides functionality to list all attendees for a particular event, facilitating attendee tracking and management.

Booking Management:

- **Canceling Tickets**: Attendees have the option to cancel their tickets, which updates the event's attendee list and frees up the seat for others.

- **Handling Overbooking**: The system prevents overbooking by checking the event's capacity before confirming a ticket booking.

User Role Differentiation:

- **Role-Based Access**: Different functionalities are available based on user roles (organizer vs. attendee). Organizers have access to event creation and management features, while attendees can only book and manage their tickets.

Error Handling and Feedback:

- **Input Validation**: The system validates user inputs to prevent invalid or erroneous data entries, such as incorrect email formats or exceeding venue capacities.

- **Error Messaging**: Users receive informative error messages in cases of failed login attempts, booking failures, or invalid actions, guiding them through corrective steps.

Event Availability Checking:

- **Current Events**: Attendees can view and book tickets for currently available events. The system checks the availability of seats in real-time to ensure accurate booking options.

Secure Data Handling:

- **Password Protection**: User passwords are securely managed and authenticated, with encryption practices ensuring that sensitive information is protected.

Scalability and Extendability:

- **Future Enhancements**: The system is designed with a modular structure, making it scalable and extendable for future enhancements such as adding new features, integrating payment gateways, or supporting additional user roles.

5. INPUT VERSATILITY WITH ERROR HANDLING AND EXCEPTION HANDLING:

Input Validation:

- **Type Checking**: The code includes mechanisms for checking the type of user inputs. For instance, when selecting options or entering numbers, the system ensures that inputs are of the correct type (e.g., integers for option selections).

- **Range Checking**: Input validation includes checking whether the entered values fall within acceptable ranges, such as seat numbers or venue capacities.

Exception Handling:

- **ValueError Handling**: The system catches ValueError exceptions that arise from invalid input types (e.g., non-integer input where an integer is expected). This prevents crashes and prompts the user to provide valid input.

- **Custom Exception Handling**: The code includes custom exceptions for specific scenarios, such as trying to book a ticket when the event is fully booked (Exception for overbooking) or attempting to cancel a ticket that does not exist (ValueError for ticket cancellation failures).

Error Messaging:

- **User-Friendly Messages**: The system provides clear and informative error messages when users enter invalid data or encounter issues. For example, if a user inputs an incorrect password or email, the system notifies them with appropriate feedback.

- **Guided Input Correction**: Users are guided to correct their inputs based on the error messages, improving the overall user experience and reducing the likelihood of repeated mistakes.

Error Handling in Booking System:

- **Capacity Checks**: Before booking a ticket, the system checks if there are available seats in the event. If the event is fully booked, an exception is raised, and the user is informed about the unavailability.

- **Ticket Existence**: During ticket cancellation, the system checks if the ticket exists in the list of bookings. If the ticket is not found, a ValueError is raised, and the user is informed that the cancellation could not be processed.

Error Handling in Authentication:

- **Login Failures**: The system handles incorrect login attempts by checking if the provided credentials match any registered user. If no match is found, it returns None, prompting the user to re-enter correct details.

- **Registration Failures**: While registering a new user, the system does not directly handle failures but ensures that valid data is collected. Users are prompted to enter all required information correctly.

Handling Invalid Actions:

- **Option Selection**: The system verifies user choices for actions like booking or canceling tickets. If an invalid option is selected, users are informed and prompted to choose a valid option, preventing incorrect operations.

Secure Handling of Sensitive Data:

- **Password Management**: While not explicitly detailed in the code, the system assumes secure handling of passwords, avoiding exposure of sensitive information. Passwords are checked against encrypted or securely stored values.

6. CODE IMPLEMENTATION:

```
# Base class to represent any entity with a name (could be Venue or Event)
class Entity:
    def __init__(self, name):
        self.__name = name  # Common name for entities

    def get_name(self):
        return self.__name

# Class to represent a Venue, inheriting from Entity
class Venue(Entity):
    def __init__(self, name, address, capacity):
        super().__init__(name)  # Call to Entity's constructor
        self.__address = address    # Private variable for venue address
        self.__capacity = capacity # Private variable for venue capacity

    def get_address(self):
        return self.__address

    def get_capacity(self):
        return self.__capacity

# Base class to represent a Person (for User, Organizer, Attendee)
class Person:
    def __init__(self, name, email):
        self.__name = name          # Private variable for person's name
        self.__email = email        # Private variable for person's email
```

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

    def get_email(self):
        return self.__email

# Class to represent a User, inheriting from Person
class User(Person):
    def __init__(self, name, email, password, role):
        super().__init__(name, email)  # Call to Person's constructor
        self.__password = password     # Private variable for user's password
        self.__role = role             # Private variable for user's role ('organizer' or 'attendee')

    def get_role(self):
        return self.__role

    def authenticate(self, password):
        return self.__password == password

# Class to represent an Event, inheriting from Entity
class Event(Entity):
    def __init__(self, name, date, venue, organizer):
        super().__init__(name)  # Call to Entity's constructor
        self.__date = date      # Private variable for event date
        self.__venue = venue    # Private variable for event venue (Venue object)
        self.__organizer = organizer  # Private variable for event organizer (User object)
        self.__attendees = []         # Private list for event attendees (User objects)
```

```
    def get_date(self):
        return self.__date

    def get_venue(self):
        return self.__venue

    def get_organizer(self):
        return self.__organizer

    def add_attendee(self, user):
        if len(self.__attendees) < self.__venue.get_capacity():
            self.__attendees.append(user)
        else:
            raise Exception("Event is fully booked")

    def remove_attendee(self, user):
        if user in self.__attendees:
            self.__attendees.remove(user)
        else:
            raise ValueError("User not found in attendees")

    def list_attendees(self):
        return [attendee.get_name() for attendee in self.__attendees]

# Class to represent a Ticket
class Ticket:
    def __init__(self, event, user, seat_number):
```

```
        self.__event = event            # Private variable for the event
        self.__user = user              # Private variable for the user who booked the ticket
        self.__seat_number = seat_number # Private variable for the seat number

    def get_event(self):
        return self.__event

    def get_user(self):
        return self.__user

    def get_seat_number(self):
        return self.__seat_number

# Class to handle bookings for events
class Booking:
    def __init__(self):
        self.__bookings = []  # List to store all ticket bookings

    def book_ticket(self, event, user, seat_number):
        if len(event.list_attendees()) < event.get_venue().get_capacity():
            ticket = Ticket(event, user, seat_number)
            event.add_attendee(user)
            self.__bookings.append(ticket)
            return ticket
        else:
            raise Exception("Cannot book ticket: Event is fully booked")

    def cancel_ticket(self, event, user):
```

```
        for ticket in self.__bookings:
            if ticket.get_event() == event and ticket.get_user() == user:
                event.remove_attendee(user)
                self.__bookings.remove(ticket)
                return True
        return False

# Class to handle user authentication
class Authentication:
    def __init__(self):
        self.__users = []  # List to store registered users

    def register(self, name, email, password, role):
        user = User(name, email, password, role)
        self.__users.append(user)
        return user

    def login(self, email, password):
        for user in self.__users:
            if user.get_email() == email and user.authenticate(password):
                return user
        return None

def main():
    authen_sys = Authentication()
    booking_system = Booking()
    event = None
```

```
while True:
    print("Welcome to our Organization :)\nWe request you to register in the user platform")
    print("1.Organizer")
    print("2.Attendee")

    try:
        choice = int(input("Select '1' if you are an organizer, '2' if you're an attendee: "))
    except ValueError:
        print("Invalid input. Please enter 1 or 2.")
        continue

    if choice == 1:
        print("Please fill the respective details to make Registration.....")
        name = input("Enter Your Name: ")
        email = input("Enter Your Email: ")
        password = input("Create a Password: ")

        organizer = authen_sys.register(name, email, password, 'organizer')

        print("SIGN IN PROCESS")
        email = input("Enter your Email: ")
        password = input("Enter your Password: ")

        sign_in = authen_sys.login(email, password)

        if sign_in:
            print("Venue Details....")
            place = input("Enter the Name of the Venue: ")
```

```
            address = input("Enter the Address: ")

            capacity = int(input("Enter the Capacity: "))


            venue = Venue(place, address, capacity)

            print("Event Details....")
            event_name = input("Enter the Name of the Event: ")
            date = input("Enter the Date of the Event: ")

            event = Event(event_name, date, venue, organizer)
            print(f"Event '{event_name}' created successfully!")
        else:
            print("Invalid login credentials.")

    elif choice == 2:
        if not event:
            print("No events available. Please wait for the organizer to create an event.")
            continue

        print("Enter the option from the below :)")
        print("1. Book Tickets")
        print("2. Cancel Tickets")

        try:
            option = int(input("Enter the option number that you've chosen: "))
        except ValueError:
```

```
            print("Invalid input. Please enter a valid option number.")
            continue

        if option == 1:
            print("Please fill in the details to book tickets.....")
            name = input("Enter Your Name: ")
            email = input("Enter Your Email: ")
            password = input("Create a Password: ")

            attendee = authen_sys.register(name, email, password, 'attendee')

            try:
                seat_number = input("Enter the seat number you wish to book: ")
                ticket = booking_system.book_ticket(event, attendee, seat_number)
                print(f"Ticket booked successfully for {attendee.get_name()} at seat
{seat_number}!")
            except Exception as e:
                print(str(e))

        elif option == 2:
            print("Please fill in the details to cancel your ticket.....")
            email = input("Enter Your Email: ")
            password = input("Enter Your Password: ")

            user = authen_sys.login(email, password)
            if user:
                if booking_system.cancel_ticket(event, user):
                    print("Ticket cancelled successfully.")
                else:
```

```
                print("Ticket not found or could not be cancelled.")
            else:
                print("Invalid login credentials.")

        else:
            print("You've entered an invalid option :(")

    else:
        print("You've entered an invalid option :(")

if __name__ == "__main__":
    main()
```

7. RESULTS AND OUTCOMES:

Entity Base Class:

Purpose:

- The Entity class is a base class that other classes like Venue and Event inherit from. It holds a common attribute, name, which applies to any entity that has a name.

Outcome:

- Any derived class (like Venue or Event) will inherit the name attribute and the get_name() method, allowing retrieval of the name.
- This abstraction reduces code duplication.

Venue Class:

Purpose:

- The Venue class represents a venue where events are hosted. It inherits the name from Entity and adds additional properties like address and capacity.

Outcome:

- When a venue is created, it can store details like the venue's name, address, and capacity.
- This allows checking if the event can accommodate a given number of attendees.

Person Base Class:

Purpose:

- The Person class serves as a base for any person involved in the system, like users. It encapsulates details like name and email.

Outcome:

- Any derived class (like User) will inherit the name and email attributes, reducing redundancy.
- The get_name() and get_email() methods help access this data for all types of users.

User Class:

Purpose:

- The User class inherits from Person and adds password and role (either 'organizer' or 'attendee'). It has an authenticate() method for verifying passwords.

Outcome:

- When a user is created, they have a role, and they can be authenticated by matching their stored password with the input password.
- This class helps manage user access to event organization (organizers) and booking (attendees).

Event Class

Purpose:

- The Event class represents an event created by an organizer. It has attributes like date, venue, and a list of attendees (users).

Outcome:

- When an event is created, it links the event's name, date, venue, and organizer.
- The add_attendee() method allows users to be added to the event, as long as the venue's capacity is not exceeded.
- The remove_attendee() method allows users to cancel their attendance.
- The list_attendees() method returns a list of all users attending the event, providing a clear overview.

Ticket Class:

Purpose:

- The Ticket class represents a ticket booked for an event. It holds a reference to the event, the user, and the seat number.

Outcome:

- Every time a ticket is booked, an instance of the Ticket class is created, linking the user to the event and the specific seat number.
- This ensures that every attendee has an assigned seat for the event.

8. Future Enhancements:

To further enhance the system, consider the following improvements:

- **Multi-Event Management**: Allow organizers to manage multiple events simultaneously, offering attendees the option to select from various events.
- **User Interface Improvements**: Develop a graphical user interface (GUI) or a web application to make the system more accessible and visually appealing.
- **Data Persistence**: Implement a database to store user, event, and ticket data permanently, allowing for retrieval and analysis over time.
- **Notifications and Reminders**: Integrate email or SMS notifications for upcoming events and ticket confirmations, improving user engagement.
- **Analytics and Reporting**: Add features for generating reports on ticket sales, attendee demographics, and event success metrics, aiding organizers in decision-making.

9. CONCLUSION :

- The Event Management System serves as a robust foundation for managing events, balancing user needs with operational efficiency. By continuously evolving the system with new features and enhancements, it can adapt to changing requirements and improve the overall event experience for both organizers and attendees.