**Event Booking System-Technical Documentation**

This application is a multi-layered platform that enables users to register, browse and book tickets for events. This application also provide the functionality for Admins to register or book an event on behalf of users. The entire users password is open to admin for easy of bookings.

It supports event management, ticket categorization, user booking, and waiting list handling when events are full.

**Architecture**

**Why the use of Layered Architecture:** is one of the most widely used software design patterns, especially in enterprise, web, and desktop applications (like your C# / .NET background).

The system follows a 3 layered architecture

1. **Presentation Layer**- Web API or UI that exposes endpoints and handles request
2. **Business Logic Layer(Service Layer)**: Implement core booking logic, validation and rules.
3. **Data Access Layer(Repository)**: Handle persistence using Entity Framework Core

**Database Schema Overview**

1. One Event can have multiple TicketTypes
2. One User can have multiple Bookings. Though was restricted to allow one booking per users in code to make things simple for now. If we want users to have multiple bookings then we can comment out the code in line 58-64.
3. A Booking reference one Event and one TicketType
4. A WaitingListEntry is created when a user tries to book for a full event**.**

**Entry Description**

1. **User**

Represents a registered attendee or admin who can book events or manage them.

|  |  |  |
| --- | --- | --- |
| **Fields** | **Type** | **Description** |
| **Id** | **Guid** | **Unique identifier for user** |
| **FullName** | **String** | **Users full name** |
| **Email** | **String** | **Unique email for login** |
| **PasswordHash** | **String** | **Password for authentication but not hash for simplicity** |
| **Role** | **String** | **Either Admin or User** |

**Relationships:**

1. One user can create many Bookings

2. One user can appear multiple times in WaitingListEntry but I have also restricted it to one user to make things simple.

2. **Event**

**Represent an event(e.g conference, concert, seminar) that users can book**

|  |  |  |
| --- | --- | --- |
| **Fields** | **Type** | **Description** |
| **Id** | **Guid** | **Unique identifier for user** |
| Description | String | Detailed description |
| StartDate | DateTime | Start date and time |
| EndDate | DateTime | End date and time |
| Venue | String | Venue or Location of the Event |
| Capacity | Int | Numbers that can be accommodated in an Event |
| Status | Int | Inprogress, Cancelled or Postponed |

**Relationships:**

1. One Event can have Multiple TicketTypes
2. One Event can have many Bookings
3. One Event can have a WaitingList for overflow users

3 **TicketType**

Defines ticket categories available for an Event(e.g Vip, Regular, Student)

|  |  |  |
| --- | --- | --- |
| **Fields** | **Type** | **Description** |
| **Id** | **Guid** | **Unique identifier for user** |
| Name | String | Ticket type name |
| Price | Decimal | Cost of a single ticket |
| QuantityAvailable | Int | Numbers of tickets available |
| EventId | Guid | Foreign key linking to the event |
| Event | Event | Navigation property to the Event |

**Relationship:**

1. **Many tickets belongs to one Event**
2. **Each Booking reference one TicketType**
3. **Bookings**

Represents a confirmed or pending users booking for an event.

|  |  |  |
| --- | --- | --- |
| **Fields** | **Type** | **Description** |
| **Id** | **Guid** | **Unique identifier for user** |
| UserId | Guid | Foreign key referencing user |
| EventId | Guid | Foreign key referencing event |
| TicketTypeId | Guid | Foreign key referencing TicketType |
| Quantity | Int | Numbers of ticket booked |
| BookingDate | DateTime | Date time booking was made |
| Status | BookingStatusEnum | Pending,Confirm,Cancelled |

**Business Rules**

1. Prevent duplicate bookings for same event and user
2. Deduct from TicketType.QuantityAvailable after a successful booking
3. If event is full , user is added to WaitingListEntry
4. **WaitingListEntry**

Represents users waiting for available seats in an event that has sold out.

|  |  |  |
| --- | --- | --- |
| **Fields** | **Type** | **Description** |
| **Id** | **Guid** | **Unique identifier for user** |
| UserId | Guid | Foreign key referencing user |
| EventId | Guid | Foreign key referencing event |
| RequestedAt | DateTime | Date and time added to the list |
| Notifed | Bool | Use to notify a user that a seat is automatically assigned to |

**Business Rules**

1. A user can join the waiting list only once per event
2. When tickets are released, the system notify waiting users

**Core Business Rules**

|  |  |
| --- | --- |
| **Rule** | **Description** |
| **Prevent Duplicate Bookings** | **A user cannot book the same event twice** |
| **Validate Event dates** | **End date must be after start date** |
| **Role Restriction** | **Only Admin can create or modify events/ticket** |
| **Ticket Availabilty** | **Prevent overbooking beyondQuantityAvailable** |
| **Waiting List Handling** | **If ticket are full, user is automatically added to the waiting list** |

**Entity Relationship (EF Core)**

Example EF core configuration

modelBuilder.Entity<Event>()  
    .HasMany(e => e.TicketTypes)  
    .WithOne(t => t.Event)  
    .HasForeignKey(t => t.EventId);  
  
modelBuilder.Entity<Booking>()  
    .HasOne(b => b.Event)  
    .WithMany(e => e.Bookings)  
    .HasForeignKey(b => b.EventId);  
  
modelBuilder.Entity<Booking>()  
    .HasOne(b => b.User)  
    .WithMany(u => u.Bookings)  
    .HasForeignKey(b => b.UserId);  
  
modelBuilder.Entity<WaitingListEntry>()  
    .HasOne(w => w.Event)  
    .WithMany(e => e.WaitingListEntries)  
    .HasForeignKey(w => w.EventId);

**Enumerations**

public enum BookingStatusEnum  
{  
    Pending = 0,  
    Confirmed = 1,  
    Cancelled = 2  
}

public enum EventStatusEnum  
{  
    Postpone,  
    InProgess,  
    NotStarted,  
    Cancelled  
}

public enum RoleTypeEnum  
{  
    Unknown = 0,  
    User = 1,  
    Admin=2  
}

**Security & Authentication**

* Uses **JWT Authentication** for login and authorization.
* Only **Admins** can create or manage events and ticket types.
* Regular **Users** can book and view their own bookings.