Capstone Project: Building a Ticket Booking System with ASP.NET Core Web API and Angular Using Clean Architecture

1. Overview

The Ticket Booking System should allow users to book tickets for events, movies, or travel, view available seats, make payments, and receive booking confirmations. Administrators manage events, pricing, and availability in real time.

2. Technology Stack

• Backend: ASP.NET Core Web API
• Frontend: Angular
• Database: SQL Server
• ORM: Entity Framework Core
• Authentication: JWT Authentication
• Exception Handling: Global exception handling middleware
• Architecture: Clean Architecture

3. Clean Architecture Implementation

a) Domain Layer (Core Business Logic)

• Defines entities such as User, Event, Booking, Payment, etc.
• Contains business rules and validation logic.
• Models:
- User (Id, Name, Email, PhoneNumber, Role)
- Event (Id, Name, Description, Date, Venue, AvailableSeats, Price, EventType)
- Booking (Id, UserId, EventId, SeatNumber, BookingDate, Status)
- Payment (Id, BookingId, Amount, PaymentMethod, Status, PaymentDate)

b) Application Layer (Use Cases and Services)

• Implements interfaces for business logic.
• Contains services such as BookingService, PaymentService, EventService, and NotificationService.
• Interfaces:
- IBookingRepository
- IEventRepository
- IPaymentRepository
- INotificationRepository

c) Infrastructure Layer (Database and External Services)

• Implements repositories for data access using Entity Framework Core.
• Integrates third-party services for payment processing and notifications.

d) Presentation Layer (API Controllers)

• Exposes endpoints for the Angular frontend.
• Uses MediatR for CQRS (Command Query Responsibility Segregation).
• Controllers:
 - BookingsController
 - EventsController
 - PaymentsController
 - NotificationsController

4. Backend Implementation (ASP.NET Core Web API)

• Authentication & Authorization: Implement using JWT tokens.
• Controllers & Endpoints: RESTful API with endpoints for booking, events, payments, and notifications.
• Dependency Injection: Ensure loose coupling.

5. Exception Handling in Web API

• Custom Exceptions: Defines exceptions such as EventNotFoundException, SeatUnavailableException, PaymentProcessingException.

6. Frontend Implementation (Angular)

• Angular Modules: Organized into feature-based modules such as BookingModule, EventModule, PaymentModule, NotificationModule.
• Services: HTTP services for API communication.
• Error Handling: Implement interceptors to handle HTTP errors globally.
• Angular Services:
 - BookingService
 - EventService
 - PaymentService
 - NotificationService

7. Database Design

• Tables: Users, Events, Bookings, Payments, Notifications.
• Relationships:
 - One-to-many between Users and Bookings
 - One-to-many between Events and Bookings
 - One-to-one between Bookings and Payments