Concert Ticket Management System

Domain Features:

IVenueService.cs
IVenueService defines the contract for managing **venues** in the ticketing system. It abstracts the business logic related to venue creation, retrieval, updating, deletion.

```
Task<List<Venue>> GetAllAsync();
Task<Venue?> GetByIdAsync(Guid id);
Task<Venue> CreateAsync(CreateVenueRequest request);
Task<bool> UpdateAsync(Guid id, UpdateVenueRequest request);
Task<bool> DeleteAsync(Guid id);
```

P.S : Availability of the venue is managed while adding events (in Event Service) to the venue, by comparing against the current events that are scheduled at that venue.

IEventService.cs
IEventService defines the contract for managing **events** in the ticketing system. It handles the creation, updating, retrieval, and ticket-type management for events associated with venues.

```
Task<ServiceResult<IEnumerable<Event>>> GetAllEventsAsync();
Task<ServiceResult<Event>> CreateEventAsync(CreateEventRequest request);
Task<ServiceResult<Event?>> GetEventByIdAsync(Guid eventId);
Task<ServiceResult<bool>> UpdateEventAsync(Guid eventId,
UpdateEventRequest request);
Task<ServiceResult<bool>> PublishEventAsync(Guid eventId);
Task<ServiceResult<bool>> SetTicketTypesAsync(Guid eventId,
List<TicketTypeRequest> ticketTypes);
```

ITicketService.cs
ITicketService defines the contract for managing **ticket reservations, purchases, cancellations**, and **availability tracking** within the ticketing system.

```
Task<ServiceResult<Ticket?>> ReserveTicketAsync(Guid eventId, Guid
ticketTypeId, int seatQuantity = 1);
Task<ServiceResult<Ticket?>> ConfirmTicketAsync(Guid ticketId);
Task<ServiceResult<bool>> CancelReservationAsync(Guid ticketId);
Task<ServiceResult<TicketAvailabilityResponse>> GetAvailableTicketsAsync(Guid
eventId);
```

Ticket Service implementation details:
The user can reserve the tickets for a time frame (that is defined in the code). Before this time frame expires, the users should confirm their purchase. If the purchase isn't confirmed or the payment fails, the tickets are released to the public, for booking.
In-memory solution was implemented and locking was implemented to support concurrent operations and to make the code thread-safe.
A timer was implemented to clear the cache at an interval (defined in the code) to remove the expired reservation and release the pool of tickets.

General Coding Practices:

1. Interface driven design to promote loose coupling and dependency injection. This also enables other ways to implement the logic. For example, in-memory implementation is shown in the project, another implementation can be added to read/write from a database.
2. Thread safety was taken care of, by using locks while reading from and writing to shared resources.
3. Validation - DTO validation is implemented to check for the data validation errors before business logic is executed. Business logic validations are implemented in the service layer.
4. Error Handling - errors are gracefully handled using try/catch.
   ServiceResult class is defined to send the success/fail boolean, error list and the data to the calling code.
5. ILogger is used for logging.

Testing:
Tested the endpoints locally using Postman