Online Ticketing System

Project Report

Software Engineering

University Of Madeira

Authors:
Iachim Cristian Serbicean Alexandru

Professor:
Leonel Nóbrega

May 23, 2025

Contents

1 Introduction

1.1 Project Description

The Online Ticketing System (Ticksy) is a web application developed in ASP.NET Core that enables complete event management and online ticket sales. The system was designed to serve three main types of users: administrators, event organizers, and customers.

The platform offers an integrated solution that covers everything from event creation and management to ticket purchasing, payment processing, and user notifications. The system was developed following Clean Architecture principles and implements various design patterns to ensure code maintainability, scalability, and testability.

1.2 Objectives and Features

1.2.1 Main Objectives

- Create an intuitive platform for event management
- Facilitate online ticket purchasing
- Implement an efficient notification system
- Ensure transaction security
- Provide different access levels according to user type

1.2.2 Implemented Features

For Administrators:

- Administrative dashboard with system statistics
- User management (organizers and customers)
- Event and transaction supervision
- System parameter configuration

For Organizers:

- Event creation and editing
- Capacity and pricing management
- Image upload for events
- Sales tracking and statistics
- Seat map management

For Customers:

- Event browsing and search with advanced filters
- Filtering system by location, price, and date

- Event sorting by different criteria
- Ticket purchasing with seat selection
- Purchase history
- Personalized notification system

2 Proposal Analysis

2.1 Requirements

2.1.1 Functional Requirements

1. **FR01 - User Management**
    - The system must allow registration of different user types
    - Must implement authentication and authorization
    - Must manage user profiles
2. **FR02 - Event Management**
    - Organizers must be able to create, edit, and delete events
    - Events must have information such as title, description, date, location, price, and capacity
    - Must support image upload
3. **FR03 - Ticketing System**
    - Customers must be able to purchase tickets
    - Must implement seat selection
    - Must process payments
    - Must generate digital tickets
4. **FR04 - Search and Filtering System**
    - Must allow text search
    - Must implement filters by location, price, and date
    - Must allow sorting by different criteria
5. **FR05 - Notification System**
    - Must notify users about relevant events
    - Must allow configuration of notification preferences
    - Must support different types of notifications

2.1.2 Non-Functional Requirements

1. **NFR01 - Performance**
   - The system must respond in less than 3 seconds
   - Must support multiple simultaneous users
2. **NFR02 - Security**
   - Sensitive data must be encrypted
   - Must implement protection against common attacks
3. **NFR03 - Usability**
   - Interface must be intuitive and responsive
   - Must work on different devices
4. **NFR04 - Maintainability**
   - Code must follow best practices
   - Must implement appropriate design patterns

2.2 Domain Model

The system's domain model consists of the following main entities:

2.2.1 Core Entities

User (Base User)

- Id: Unique identifier
- Email: User email
- PasswordHash: Password hash
- FirstName, LastName: User name
- CreatedAt: Creation date

Administrator

- Inherits from User
- Role: Administrative role
- IsActive: Active status

Organizer

- Inherits from User
- CompanyName: Company name
- ContactNumber: Contact number

- IsVerified: Verification status

Customer

- Inherits from User
- DateOfBirth: Date of birth
- PhoneNumber: Phone number
- Address: Address

Event

- Id: Unique identifier
- Title: Event title
- Description: Description
- StartDate, EndDate: Event dates
- Location: Location
- TicketPrice: Ticket price
- Capacity: Capacity
- ImageUrl: Image URL
- OrganizerId: Organizer reference
- IsActive: Active status

Ticket

- Id: Unique identifier
- EventId: Event reference
- CustomerId: Customer reference
- SeatNumber: Seat number
- PurchaseDate: Purchase date
- Status: Ticket status
- QRCode: QR code

Payment

- Id: Unique identifier
- TicketId: Ticket reference
- Amount: Amount
- PaymentMethod: Payment method
- TransactionId: Transaction ID
- Status: Payment status
- ProcessedAt: Processing date

2.2.2 Relationships

- One Organizer can create multiple Events (1:N)
- One Event can have multiple Tickets (1:N)
- One Customer can purchase multiple Tickets (1:N)
- Each Ticket has an associated Payment (1:1)
- One Event can have one SeatMap (1:1)
- One Customer can have multiple NotificationPreferences (1:N)

2.3 System Architecture

The system was developed following Clean Architecture principles, organizing code into well-defined layers:

2.3.1 Presentation Layer (TicketingSystem.Web)

- Razor Pages for web interface
- Controllers for APIs
- ViewModels for data transfer
- Middleware for request handling

2.3.2 Application Layer (TicketingSystem.Application)

- Services for business logic
- DTOs for data transfer
- Interfaces for service contracts
- Handlers for commands and queries

2.3.3 Domain Layer (TicketingSystem.Core)

- Entities with business rules
- Value Objects
- Domain Services
- Repository interfaces

2.3.4 Infrastructure Layer (TicketingSystem.Infrastructure)

- Repository implementations
- Database context (Entity Framework)
- External services (email, payments)
- Persistence configurations

3 Development

3.1 Module Decomposition

The system was decomposed into the following main modules:

3.1.1 Authentication and Authorization Module

- User session management
- Role-based access control
- Authentication middleware

3.1.2 Event Management Module

- Event CRUD operations
- Image upload and management
- Advanced search and filtering system
- Event data validation

3.1.3 Ticketing Module

- Ticket purchase process
- Seat and capacity management
- QR code generation
- Ticket validation

3.1.4 Payment Module

- Transaction processing
- Payment gateway integration
- Payment status management

3.1.5 Notification Module

- Email sending system
- User preference management
- Notification templates

3.1.6 Administrative Module

- Dashboard with statistics
- User management
- System reports

3.2 Implemented Design Patterns

3.2.1 Repository Pattern

Implemented to abstract data access, allowing:

- Decoupling between business logic and persistence
- Ease of unit testing
- Flexibility for persistence technology changes

Listing 1: Repository Pattern Example

```
public interface IEventRepository
{
    Task<Event> GetByIdAsync(Guid id);
    Task<IEnumerable<Event>> GetAllAsync();
    Task<IEnumerable<Event>> SearchAsync(string searchTerm);
    Task AddAsync(Event entity);
    Task UpdateAsync(Event entity);
    Task DeleteAsync(Guid id);
}
```

3.2.2 Service Layer Pattern

Used to encapsulate business logic:

- Clear separation between presentation and business
- Logic reuse across different interfaces
- Ease of maintenance

3.2.3 Dependency Injection

Implemented through ASP.NET Core's DI container:

- Reduced coupling
- Ease of testing
- Automatic object lifecycle management

3.2.4 Model-View-ViewModel (MVVM)

Applied in Razor Pages:

- Separation between presentation logic and interface
- Automatic data binding
- Model validation

3.2.5 Strategy Pattern

Used in the event filtering system:

- Different filtering strategies (price, date, location)
- Extensibility for new filter types
- Cleaner and more organized code

3.3 Other Development Considerations

3.3.1 Data Validation

- Data Annotations for model validation
- Custom validation for specific business rules
- Validation on both client and server side

3.3.2 Error Handling

- Global middleware for exception capture
- Structured logging with different levels
- Custom error pages

3.3.3 Security

- Password hashing with salt
- CSRF protection
- Input validation to prevent XSS
- Claims-based authorization

3.3.4 Performance

- Lazy loading for related entities
- Caching of frequently accessed data
- Pagination for large lists
- Database query optimization

4 Installation/Execution Guide

4.1 Git Repository

The project source code is available at:

`[Git repository URL]`

4.2 Prerequisites

- .NET 6.0 SDK or higher
- SQL Server or SQL Server Express
- Visual Studio 2022 or Visual Studio Code
- Git for version control

4.3 Installation Steps

4.3.1 1. Clone the Repository

```
git clone [https://github.com/IachimCristian/Ticketing-System]
```

4.3.2 2. Configure the Database

1. Edit the `appsettings.json` file in the `TicketingSystem.Web` folder
2. Configure the connection string for the database
3. Run migrations:

```
dotnet ef database update --project TicketingSystem.Infrastructure
--startup-project TicketingSystem.Web
```

4.3.3 3. Restore Dependencies

```
dotnet restore
```

4.3.4 4. Run the Application

```
cd TicketingSystem.Web
dotnet run
```

The application will be available at `https://localhost:5232`

4.4 Test Users

4.4.1 Administrator

- **Email:** admin@ticksy.com
- **Password:** Admin123!

4.4.2 Organizer

- **Email:** organizer@ticksy.com
- **Password:** Organizer123!

4.4.3 Customer

- **Email:** customer@ticksy.com
- **Password:** Customer123!

4.5 Additional Configurations

4.5.1 Email

To test the notification system, configure SMTP settings in `appsettings.json`:

```
{
  "EmailSettings": {
    "SmtpServer": "smtp.gmail.com",
    "SmtpPort": 587,
    "SenderEmail": "your-email@gmail.com",
    "SenderPassword": "your-password"
  }
}
```

4.5.2 Payments

For payment testing, the system is configured to use a simulator that accepts any card with number `4111111111111111`.

5 Usage Guide

5.1 System Access

The system can be accessed through a web browser at `https://localhost:5001`. The home page displays featured events and navigation options.

5.2 Features by User Type

5.2.1 For Customers

1. Registration and Login

- Access the registration page through the menu
- Fill in personal details
- Confirm email (if configured)
- Login with credentials

2. Event Search and Filtering

- Use the search bar to find specific events
- Apply filters by:
  - Location (dropdown with available locations)
  - Price range (Free, $0-25, $25-50, $50-100, $100+)
  - Date (Today, Tomorrow, This Week, This Month, Next Month)
- Sort results by:
  - Date (earliest first/latest first)
  - Price (low to high/high to low)
  - Name (A-Z/Z-A)

3. Ticket Purchase

- Select desired event
- View event details
- Choose ticket quantity
- Select seats (if applicable)
- Proceed to payment
- Receive email confirmation

4. Profile Management

- Update personal information
- View purchase history
- Configure notification preferences
- Manage payment methods

5.2.2 For Organizers

1. **Event Management**

   - Access organizer dashboard
   - Create new event with:
     - Title and description
     - Date and time
     - Location
     - Price and capacity
     - Image upload
   - Edit existing events
   - Activate/deactivate events

2. **Sales Tracking**

   - View sales statistics
   - Participant list
   - Revenue reports
   - Available seat management

3. **Seat Management**

   - Configure seat map
   - Define different seat types
   - Manage pricing by section

5.2.3 For Administrators

1. **Administrative Dashboard**

   - System overview
   - User statistics
   - Event and sales metrics
   - System alerts

2. **User Management**

   - List of all users
   - Activate/deactivate accounts
   - Verify organizers

- Manage permissions

3. Event Supervision

- Approve/reject events
- Monitor activity
- Manage inappropriate content

4. System Configuration

- Configure global parameters
- Manage email templates
- Configure payment methods
- Backup and maintenance

5.3 Main Workflows

5.3.1 Ticket Purchase Flow

1. Customer searches for events
2. Applies filters as needed
3. Selects event of interest
4. Views event details
5. Chooses ticket quantity
6. Selects seats (if applicable)
7. Enters payment details
8. Confirms purchase
9. Receives digital ticket by email

5.3.2 Event Creation Flow

1. Organizer logs in
2. Accesses dashboard
3. Clicks "Create Event"
4. Fills in event information
5. Uploads image
6. Configures prices and capacity
7. Defines seat map (optional)
8. Submits for approval
9. Event becomes available after approval

6 Conclusions

6.1 Achieved Objectives

The development of the Online Ticketing System (Ticksy) successfully achieved the proposed objectives:

- **Complete Platform**: An integrated solution was developed covering the entire event lifecycle, from creation to ticket sales
- **Intuitive Interface**: The web interface is responsive and easy to use, providing a good user experience
- **Advanced Filtering System**: Implementation of filters by location, price, and date, with sorting by multiple criteria
- **Robust Architecture**: Application of Clean Architecture principles ensured well-structured and maintainable code
- **Security**: Implementation of adequate security measures to protect sensitive data

6.2 Challenges Faced

During development, various challenges were encountered:

- **State Management**: Maintaining data consistency between different simultaneous operations
- **Performance**: Query optimization for large data volumes
- **Responsive Interface**: Ensuring the application works well on different devices
- **System Integration**: Coordination between different system modules

6.3 Lessons Learned

- **Importance of Design**: Good architectural design significantly facilitates development and maintenance
- **Testing**: Implementing tests from the beginning would have saved time in bug detection
- **Documentation**: Keeping updated documentation is crucial for team projects
- **Design Patterns**: Correct application of patterns improves code quality

6.4 Future Work

For future versions of the system, the following improvements are suggested:

- **Mobile Application**: Development of a native app for iOS and Android
- **Social Media Integration**: Allow event sharing and social login
- **Recommendation System**: Algorithm to suggest events based on user history
- **Advanced Analytics**: Dashboard with more detailed metrics for organizers
- **Public API**: Provide API for third-party system integration
- **Multi-language Support**: Application internationalization
- **Review System**: Allow customers to rate events

6.5 Final Considerations

The Online Ticketing System project represented an excellent opportunity to apply theoretical Software Engineering knowledge in a practical context. The implementation of design patterns, clean architecture, and development best practices resulted in a robust and scalable application.

The developed system demonstrates the importance of careful requirements analysis, well-thought design, and disciplined implementation. The implemented features meet the needs of different user types and provide a solid foundation for future expansions.

The experience gained during this project will be valuable for future developments, especially regarding complex project management, teamwork, and application of software development methodologies.