

## Project Overview

The **Event Management System** is a web-based application designed to help organizations and individuals create, manage, and attend events. It allows event organizers to create and publish events, manage registrations, and track attendees. The system supports role-based authentication (admin, organizer, attendee) with various levels of access and functionality. This project integrates **ASP.NET Core MVC**, **ASP.NET Core Web API**, **Entity Framework Core**, **ASP.NET Core Identity**, and **Role-based Authorization** to manage users, events, and registrations effectively.

The Event Management System is designed to be highly flexible, enabling organizers to create both online and offline events, set event details (date, location, etc.), and interact with attendees. The system also supports email notifications, event status tracking, and payment gateway integration for paid events.

---

## Key Features

### 1. User Registration and Role Management:

- **ASP.NET Core Identity** is used for user registration, login, and role management.
- **Admin** role has full access to the system.
- **Organizer** role can create, edit, and manage events.
- **Attendee** role can register for events and track event details.
- Role-based authentication ensures appropriate access levels for different users.

### 2. Event Creation and Management:

- **Organizers** can create events, specify event details (date, time, location, description, etc.), and set the maximum number of attendees.
- Event details can include categories (e.g., conference, workshop, seminar) and event status (e.g., scheduled, completed, canceled).
- Organizers can manage attendee registrations and view event statistics.

### 3. Event Registration:

- **Attendees** can browse events, view event details, and register for events.
- Attendees can view the events they've registered for and manage their registrations (e.g., cancel registration).
- Support for paid events with payment gateway integration (e.g., PayPal, Stripe).

### 4. Email Notifications:

- **Event reminders** and **confirmation emails** are sent to attendees upon successful registration.

- **Email notifications** are sent when event details are updated (e.g., date change, cancellation).

#### 5. Dashboard and Analytics:

- **Admin** and **Organizer** roles can access dashboards that provide key event statistics, such as the number of registered attendees, event status, and payment information.
- Admins can view the list of all events and manage event creation and deletion.

#### 6. Role-based Access Control:

- Different access levels are granted based on the role of the user.
- **Admins** can manage users, events, and registrations.
- **Organizers** can create and manage their own events.
- **Attendees** can view, register, and cancel their event registrations.

#### 7. API Endpoints:

- The system exposes a set of RESTful APIs for event management and user registration.
- API endpoints support event creation, registration, and notifications.

#### 8. Payment Gateway Integration:

- Integration with a payment provider (e.g., **Stripe** or **PayPal**) for paid events, allowing users to make secure payments during event registration.

---

### Technologies Used

- **ASP.NET Core MVC:** For building the front-end user interface for event management and interaction.
- **ASP.NET Core Web API:** To handle backend services, such as event creation, registration, and email notifications.
- **Entity Framework Core:** For database interaction and CRUD operations on events, users, and registrations.
- **ASP.NET Core Identity:** For managing user authentication, authorization, and roles (admin, organizer, attendee).
- **SignalR (Optional):** For real-time updates on event status or registration confirmations.
- **Payment Gateway Integration:** Using a service like **Stripe** or **PayPal** for handling payments for paid events.

- **Email Services:** For sending event registration confirmations, reminders, and updates.
  - **SQL Server:** For data storage (events, users, and registrations).
- 

## System Architecture

### 1. Frontend (ASP.NET Core MVC):

- The MVC architecture handles user interactions such as viewing events, registering, and managing events.
- The user interface is organized with views that allow users to create events, register, view event details, and manage registrations.

### 2. Backend (ASP.NET Core Web API):

- The API handles CRUD operations for events, registrations, and user management.
- The backend communicates with the database to store user data, event details, and registration information.
- The API also manages email notifications and payment transactions for paid events.

### 3. Database (Entity Framework Core):

- EF Core is used to manage the relationships between users, events, and registrations.
- The system uses a relational database (e.g., SQL Server) to persist event details, user registrations, and user profiles.

### 4. Email Notification System:

- Integrate an email service provider (e.g., SendGrid, SMTP) to send registration confirmations, reminders, and updates.

### 5. Payment Gateway Integration:

- Implement integration with **Stripe** or **PayPal** to handle payments for events that require a fee.
  - The payment system will trigger event registration once the payment is confirmed.
- 

## User Stories

### 1. User Registration and Role Management

- **As a new user,** I want to register for an account so that I can manage or attend events.
- **As a registered user,** I want to log in securely using my credentials.

- **As an admin**, I want to manage user roles (admin, organizer, attendee) and ensure users have appropriate access.

## 2. Event Creation and Management

- **As an organizer**, I want to create events, set event details (date, time, location), and publish them for attendees to view.
- **As an organizer**, I want to track the number of attendees and manage event registrations.
- **As an admin**, I want to view, manage, and delete events created by organizers.

## 3. Event Registration and Payment

- **As an attendee**, I want to browse events, register for events, and receive confirmation upon registration.
- **As an attendee**, I want to cancel my registration if I cannot attend.
- **As an attendee**, I want to pay for an event if it requires a fee using a secure payment gateway (e.g., Stripe, PayPal).

## 4. Email Notifications

- **As an attendee**, I want to receive email confirmations for my event registrations.
- **As an attendee**, I want to receive reminders and updates about the event (e.g., date changes, cancellations).
- **As an admin**, I want to send email updates to attendees for changes or announcements about an event.

## 5. Dashboard and Analytics

- **As an organizer**, I want to view event analytics, such as the number of attendees, registrations, and status (scheduled, completed, canceled).
- **As an admin**, I want to see an overview of all events and users.

---

## Database Design

### Entities:

#### 1. User:

- Id (Primary Key)
- Username
- Email
- PasswordHash

- Role (Admin, Organizer, Attendee)

**2. Event:**

- Id (Primary Key)
- Title
- Description
- DateTime
- Location
- MaxAttendees
- OrganizerId (Foreign Key referencing User)
- Status (Scheduled, Completed, Canceled)
- PaymentRequired (Boolean)

**3. Registration:**

- Id (Primary Key)
- UserId (Foreign Key referencing User)
- EventId (Foreign Key referencing Event)
- PaymentStatus (Paid, Pending, Failed)

**4. Category:**

- Id (Primary Key)
- Name

**5. PaymentTransaction:**

- Id (Primary Key)
- UserId (Foreign Key referencing User)
- EventId (Foreign Key referencing Event)
- Amount
- PaymentStatus (Success, Failed)
- TransactionDate

**6. Notification:**

- Id (Primary Key)
- UserId (Foreign Key referencing User)

- EventId (Foreign Key referencing Event)
  - Message
  - DateTime
- 

## APIs and Endpoints

- **POST /api/auth/register:** Register a new user.
  - **POST /api/auth/login:** Log in a user.
  - **GET /api/events:** Get all events.
  - **GET /api/events/{id}:** Get details of a specific event.
  - **POST /api/events:** Create a new event (for organizer).
  - **PUT /api/events/{id}:** Update an event (for organizer).
  - **DELETE /api/events/{id}:** Delete an event (for admin).
  - **POST /api/registrations:** Register a user for an event.
  - **GET /api/registrations/{userId}:** View events registered by a specific user.
  - **POST /api/payments:** Process payment for an event.
  - **POST /api/notifications:** Send notifications about an event (for admin and organizer).
- 

## Payment Gateway Integration

- **Stripe/PayPal:** Integrate a payment gateway for events that require payment. The payment process involves:
  - Attendees selecting an event.
  - Proceeding to payment (via Stripe or PayPal).
  - Upon successful payment, registration for the event is confirmed.