**GESTIÓN DE EVENTOS**

**Apertura:** viernes, 17 de mayo de 2024, 00:00

**Cierre:** jueves, 13 de junio de 2024, 00:00

Desarrollar una aplicación tipo SPA para gestionar Eventos. A continuación se detalla el alcance y la forma de evaluación

**Alcance**

1. Gestión de cuentas de usuario
   1. Crear cuenta
   2. Iniciar/cerrar Sesión
2. Portal de eventos disponibles
   1. Visualizar eventos disponibles
   2. Inscripción a un evento
      1. Obtener comprobante de registro por email
3. Administrar eventos
   1. Los eventos tienen información:
      1. Nombre del evento
      2. Detalles/Descripción
      3. Ubicación/Lugar
      4. Información de contacto
      5. Fecha y hora de realización
      6. Pueden ser uno o varios días
   2. Crear Eventos para personas individuales, los datos de los participantes:
      1. Nombre
      2. Dirección
      3. Fecha de nacimiento
      4. Correo
      5. Nro de Tel/Cel
      6. Organización a la que pertenece
      7. Profesión
      8. Cargo
   3. Crear Eventos para equipos, los equipos tienen la sgte información:
      1. Nombre de equipo
      2. Institución/Organización a la cual representa
      3. Los equipos tienen un representante
      4. Pueden ser uno o más participantes
      5. Los participantes tienen la misma información que individual
4. Reportes
   1. Historial de eventos realizados y los que participó el usuario
   2. Certificados de participación individual
   3. Certificados de participación por equipo
   4. Lista de Registrados en evento

**Evaluación**

* Documentación, la misma que debe contener:
  + Carátula
  + Introducción
  + Objetivos
  + Alcance
  + Desarrollo
    - Requerimientos (historias de usuario)
    - Modelo de dominio
    - Modelo de base de datos
    - Código de los algoritmos principales
  + Características de seguridad
* Software funcionando
* Software de acuerdo al diseño
* Buenas prácticas y orientado a objetos

**-----------------**  
 **Historias de Usuario**

1. **Gestión de Cuentas de Usuario**

**1. Crear Cuenta de Usuario**

- Como: usuario no registrado

- Quiero: poder crear una cuenta en la aplicación

- Para: poder acceder a todas las funcionalidades disponibles, como la inscripción a eventos y la administración de los mismos.

***Criterios de Aceptación:***

- El formulario de registro debe solicitar el nombre, dirección de correo electrónico, contraseña, y confirmación de contraseña.

- Debe haber validación de datos para asegurarse de que todos los campos obligatorios están completos y son correctos.

- Al completar el registro, el usuario debe recibir un correo de confirmación.

**2. Iniciar/Cerrar Sesión**

- Como: usuario registrado

- Quiero: poder iniciar y cerrar sesión en la aplicación

- Para: acceder a mi cuenta de manera segura y gestionar mis eventos y participaciones.

***Criterios de Aceptación:***

- Debe haber una opción clara para iniciar sesión desde la página principal.

- Debe haber validación de datos al iniciar sesión (correo electrónico y contraseña).

- Después de iniciar sesión, debe haber una opción para cerrar sesión en cualquier momento.

1. **Portal de Eventos Disponibles**

**3. Visualizar Eventos Disponibles**

- Como: usuario registrado o no registrado

- Quiero: poder ver una lista de todos los eventos disponibles

- Para: decidir a cuál evento me gustaría inscribirme.

***- Criterios de Aceptación:***

- La lista de eventos debe mostrar el nombre del evento, la descripción, la ubicación, la fecha y hora, y la información de contacto.

- Los eventos deben estar organizados de manera clara y fácil de navegar.

4. Inscripción a un Evento

- Como: usuario registrado

- Quiero: poder inscribirme en un evento desde el portal de eventos disponibles

- Para: asegurar mi participación en el evento.

Criterios de Aceptación:

- Debe haber un botón de inscripción visible en la página del evento.

- Al inscribirse, debe solicitar la información necesaria para el tipo de inscripción (individual o equipo).

- Debe recibir un comprobante de registro por correo electrónico después de completar la inscripción.

5. Obtener Comprobante de Registro por Email

- Como: usuario registrado

- Quiero: recibir un comprobante de registro por correo electrónico después de inscribirme en un evento

- Para: tener una confirmación y recordatorio de mi inscripción.

Criterios de Aceptación:

- El comprobante debe incluir los detalles del evento y la información de inscripción.

- Debe enviarse automáticamente después de la inscripción exitosa.

1. Administrar Eventos

6. Crear Evento

- Como: organizador de eventos

- Quiero: poder crear nuevos eventos en la aplicación

- Para: permitir que los usuarios se inscriban y participen en los eventos.

Criterios de Aceptación:

- El formulario de creación de eventos debe solicitar el nombre del evento, descripción, ubicación, información de contacto, fecha y hora de realización.

- Debe permitir especificar si el evento se lleva a cabo en uno o varios días.

7. Inscribir Participantes Individuales

- Como: organizador de eventos

- Quiero: poder inscribir participantes individuales a un evento

- Para: gestionar la asistencia de personas individuales.

- Criterios de Aceptación:

- El formulario de inscripción individual debe solicitar el nombre, dirección, fecha de nacimiento, correo, número de teléfono, organización y profesión/cargo.

- Debe permitir guardar y editar la información de los participantes.

8. Inscribir Equipos

- Como: organizador de eventos

- Quiero: poder inscribir equipos a un evento

- Para: gestionar la asistencia de equipos completos.

Criterios de Aceptación:

- El formulario de inscripción de equipos debe solicitar el nombre del equipo, institución/organización que representa, y la información del representante.

- Debe permitir agregar uno o más participantes con la misma información requerida para inscripciones individuales.

1. Reportes

9. Historial de Eventos Realizados y Participaciones del Usuario\*\*

- Como: usuario registrado

- Quiero: poder ver un historial de todos los eventos en los que he participado

- Para: tener un registro de mi actividad y participaciones en eventos.

Criterios de Aceptación:

- El historial debe listar todos los eventos pasados y futuros en los que el usuario ha estado inscrito.

- Debe mostrar detalles básicos de cada evento, como nombre, fecha y estado de participación.

10. Certificados de Participación Individual

- Como: usuario registrado

- Quiero: poder descargar certificados de participación en eventos individuales

- Para: tener una evidencia formal de mi participación.

Criterios de Aceptación:

- Debe haber una opción para descargar certificados desde el historial de eventos.

- Los certificados deben incluir el nombre del usuario, el nombre del evento y la fecha de participación.

11. Certificados de Participación por Equipo

- Como: representante de un equipo

- Quiero: poder descargar certificados de participación para mi equipo

- Para: tener una evidencia formal de la participación del equipo.

Criterios de Aceptación:

- Debe haber una opción para descargar certificados desde el historial de eventos del equipo.

- Los certificados deben incluir el nombre del equipo, los nombres de los participantes, el nombre del evento y la fecha de participación.

12. Lista de Registrados en un Evento

- Como: organizador de eventos

- Quiero: poder ver y descargar una lista de todos los registrados en un evento

- Para: gestionar y organizar mejor el evento.

Criterios de Aceptación:

- Debe haber una opción para ver la lista de registrados desde la página de administración del evento.

- La lista debe incluir toda la información relevante de los participantes y permitir su exportación en formato CSV o PDF.

**-----------------**

**DISEÑO SQL – DIAGRAMA SQL**

Tablas Principales

1. \*\*Usuarios\*\*

- `id`: INT, PRIMARY KEY, AUTO\_INCREMENT

- `nombre`: VARCHAR(100)

- `direccion`: VARCHAR(255)

- `fecha\_nacimiento`: DATE

- `correo`: VARCHAR(100), UNIQUE

- `telefono`: VARCHAR(20)

- `organizacion`: VARCHAR(100)

- `profesion`: VARCHAR(50)

- `cargo`: VARCHAR(50)

- `contraseña`: VARCHAR(255)

2. \*\*Eventos\*\*

- `id`: INT, PRIMARY KEY, AUTO\_INCREMENT

- `nombre`: VARCHAR(100)

- `descripcion`: TEXT

- `ubicacion`: VARCHAR(255)

- `informacion\_contacto`: VARCHAR(255)

- `fecha\_inicio`: DATETIME

- `fecha\_fin`: DATETIME

3. \*\*Equipos\*\*

- `id`: INT, PRIMARY KEY, AUTO\_INCREMENT

- `nombre`: VARCHAR(100)

- `institucion`: VARCHAR(100)

- `representante\_id`: INT, FOREIGN KEY REFERENCES `Usuarios(id)`

4. \*\*Participantes\*\*

- `id`: INT, PRIMARY KEY, AUTO\_INCREMENT

- `usuario\_id`: INT, FOREIGN KEY REFERENCES `Usuarios(id)`

- `evento\_id`: INT, FOREIGN KEY REFERENCES `Eventos(id)`

- `equipo\_id`: INT, FOREIGN KEY REFERENCES `Equipos(id)`, NULLABLE

5. \*\*Inscripciones\*\*

- `id`: INT, PRIMARY KEY, AUTO\_INCREMENT

- `evento\_id`: INT, FOREIGN KEY REFERENCES `Eventos(id)`

- `usuario\_id`: INT, FOREIGN KEY REFERENCES `Usuarios(id)`

- `equipo\_id`: INT, FOREIGN KEY REFERENCES `Equipos(id)`, NULLABLE

- `fecha\_inscripcion`: DATETIME

6. \*\*Certificados\*\*

- `id`: INT, PRIMARY KEY, AUTO\_INCREMENT

- `inscripcion\_id`: INT, FOREIGN KEY REFERENCES `Inscripciones(id)`

- `tipo`: ENUM('individual', 'equipo')

- `fecha\_emision`: DATETIME

- `archivo`: VARCHAR(255)

### Tablas de Auditoría e Historial

7. \*\*Historial\_Eventos\*\*

- `id`: INT, PRIMARY KEY, AUTO\_INCREMENT

- `usuario\_id`: INT, FOREIGN KEY REFERENCES `Usuarios(id)`

- `evento\_id`: INT, FOREIGN KEY REFERENCES `Eventos(id)`

- `fecha\_participacion`: DATETIME

### Relación entre Tablas

- Un `Usuario` puede crear múltiples `Eventos`.

- Un `Evento` puede tener múltiples `Participantes` y `Equipos`.

- Un `Equipo` puede tener múltiples `Participantes`.

- Un `Participante` puede inscribirse en múltiples `Eventos`, ya sea individualmente o como parte de un equipo.

- Un `Certificado` se emite para cada `Inscripcion`.

### Relación en Diagrama ER (Modelo Entidad-Relación)

1. \*\*Usuarios\*\*

- Un usuario puede ser participante o representante de un equipo.

- Relación uno a muchos con eventos (un usuario puede crear varios eventos).

2. \*\*Eventos\*\*

- Un evento puede tener múltiples inscripciones.

- Relación muchos a muchos con usuarios a través de la tabla `Inscripciones`.

3. \*\*Equipos\*\*

- Un equipo tiene un representante (un usuario) y puede estar inscrito en múltiples eventos.

- Relación uno a muchos con participantes.

4. \*\*Participantes\*\*

- Un participante puede ser un usuario individual o parte de un equipo.

5. \*\*Inscripciones\*\*

- Una inscripción registra la participación de un usuario o equipo en un evento.

6. \*\*Certificados\*\*

- Un certificado está vinculado a una inscripción y puede ser para una participación individual o de equipo.

### SQL para Creación de Tablas

```sql

CREATE TABLE Usuarios (

id INT PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(100),

direccion VARCHAR(255),

fecha\_nacimiento DATE,

correo VARCHAR(100) UNIQUE,

telefono VARCHAR(20),

organizacion VARCHAR(100),

profesion VARCHAR(50),

cargo VARCHAR(50),

contraseña VARCHAR(255)

);

CREATE TABLE Eventos (

id INT PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(100),

descripcion TEXT,

ubicacion VARCHAR(255),

informacion\_contacto VARCHAR(255),

fecha\_inicio DATETIME,

fecha\_fin DATETIME

);

CREATE TABLE Equipos (

id INT PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(100),

institucion VARCHAR(100),

representante\_id INT,

FOREIGN KEY (representante\_id) REFERENCES Usuarios(id)

);

CREATE TABLE Participantes (

id INT PRIMARY KEY AUTO\_INCREMENT,

usuario\_id INT,

evento\_id INT,

equipo\_id INT NULL,

FOREIGN KEY (usuario\_id) REFERENCES Usuarios(id),

FOREIGN KEY (evento\_id) REFERENCES Eventos(id),

FOREIGN KEY (equipo\_id) REFERENCES Equipos(id)

);

CREATE TABLE Inscripciones (

id INT PRIMARY KEY AUTO\_INCREMENT,

evento\_id INT,

usuario\_id INT,

equipo\_id INT NULL,

fecha\_inscripcion DATETIME,

FOREIGN KEY (evento\_id) REFERENCES Eventos(id),

FOREIGN KEY (usuario\_id) REFERENCES Usuarios(id),

FOREIGN KEY (equipo\_id) REFERENCES Equipos(id)

);

CREATE TABLE Certificados (

id INT PRIMARY KEY AUTO\_INCREMENT,

inscripcion\_id INT,

tipo ENUM('individual', 'equipo'),

fecha\_emision DATETIME,

archivo VARCHAR(255),

FOREIGN KEY (inscripcion\_id) REFERENCES Inscripciones(id)

);

CREATE TABLE Historial\_Eventos (

id INT PRIMARY KEY AUTO\_INCREMENT,

usuario\_id INT,

evento\_id INT,

fecha\_participacion DATETIME,

FOREIGN KEY (usuario\_id) REFERENCES Usuarios(id),

FOREIGN KEY (evento\_id) REFERENCES Eventos(id)

);

**-----------------**

**CONSULTAS SQL (QUERYS)**

Estas consultas SQL permiten cubrir las necesidades básicas de la aplicación para gestionar eventos, como crear cuentas de usuario, iniciar sesión, visualizar eventos, inscribirse en eventos, y obtener reportes y certificados de participación.

### Gestión de Cuentas de Usuario

\*\*1. Crear Cuenta de Usuario\*\*

```sql

INSERT INTO Usuarios (nombre, direccion, fecha\_nacimiento, correo, telefono, organizacion, profesion, cargo, contraseña)

VALUES ('Juan Perez', 'Calle Falsa 123', '1985-05-10', 'juan.perez@example.com', '1234567890', 'ABC Corp', 'Ingeniero', 'Jefe de Proyecto', 'hashedpassword');

```

\*\*2. Iniciar Sesión\*\*

```sql

SELECT id, nombre, correo

FROM Usuarios

WHERE correo = 'juan.perez@example.com' AND contraseña = 'hashedpassword';

```

\*\*3. Cerrar Sesión\*\*

- La sesión se maneja generalmente en el lado del servidor, eliminando la sesión del usuario en la aplicación.

### Portal de Eventos Disponibles

\*\*4. Visualizar Eventos Disponibles\*\*

```sql

SELECT id, nombre, descripcion, ubicacion, informacion\_contacto, fecha\_inicio, fecha\_fin

FROM Eventos

WHERE fecha\_inicio >= NOW();

```

\*\*5. Inscripción a un Evento\*\*

```sql

INSERT INTO Inscripciones (evento\_id, usuario\_id, fecha\_inscripcion)

VALUES (1, 2, NOW());

```

\*\*6. Obtener Comprobante de Registro por Email\*\*

- Esto implica una combinación de SQL y lógica de servidor. El SQL para obtener los detalles podría ser:

```sql

SELECT e.nombre, e.descripcion, e.ubicacion, e.fecha\_inicio, e.fecha\_fin, u.nombre AS usuario\_nombre, u.correo

FROM Inscripciones i

JOIN Eventos e ON i.evento\_id = e.id

JOIN Usuarios u ON i.usuario\_id = u.id

WHERE i.id = 1;

```

- El servidor luego enviaría el correo electrónico con estos detalles.

### Administrar Eventos

\*\*7. Crear Evento\*\*

```sql

INSERT INTO Eventos (nombre, descripcion, ubicacion, informacion\_contacto, fecha\_inicio, fecha\_fin)

VALUES ('Conferencia de Tecnología', 'Evento sobre las últimas tendencias en tecnología', 'Auditorio Principal', 'contacto@example.com', '2023-10-15 09:00:00', '2023-10-15 17:00:00');

```

\*\*8. Inscribir Participantes Individuales\*\*

```sql

INSERT INTO Participantes (usuario\_id, evento\_id)

VALUES (2, 1);

```

\*\*9. Inscribir Equipos\*\*

```sql

INSERT INTO Equipos (nombre, institucion, representante\_id)

VALUES ('Equipo A', 'Institución XYZ', 2);

INSERT INTO Participantes (usuario\_id, evento\_id, equipo\_id)

VALUES (3, 1, 1),

(4, 1, 1);

```

### Reportes

\*\*10. Historial de Eventos Realizados y Participaciones del Usuario\*\*

```sql

SELECT e.nombre, e.descripcion, e.fecha\_inicio, e.fecha\_fin

FROM Inscripciones i

JOIN Eventos e ON i.evento\_id = e.id

WHERE i.usuario\_id = 2;

```

\*\*11. Certificados de Participación Individual\*\*

```sql

SELECT c.archivo

FROM Certificados c

JOIN Inscripciones i ON c.inscripcion\_id = i.id

WHERE i.usuario\_id = 2 AND c.tipo = 'individual';

```

\*\*12. Certificados de Participación por Equipo\*\*

```sql

SELECT c.archivo

FROM Certificados c

JOIN Inscripciones i ON c.inscripcion\_id = i.id

WHERE i.equipo\_id = 1 AND c.tipo = 'equipo';

```

\*\*13. Lista de Registrados en un Evento\*\*

```sql

SELECT u.nombre, u.correo, u.telefono

FROM Inscripciones i

JOIN Usuarios u ON i.usuario\_id = u.id

WHERE i.evento\_id = 1;

```

**-----------------**

**INTEGRACION C#**

Para integrar la gestión de eventos con una base de datos SQL en una aplicación C#, se deben crear varias clases que representen las entidades del sistema y manejar la interacción con la base de datos. Aquí hay un ejemplo de las clases que se pueden crear:

### 1. Clases de Entidad

\*\*Usuario.cs\*\*

```csharp

public class Usuario

{

public int Id { get; set; }

public string Nombre { get; set; }

public string Direccion { get; set; }

public DateTime FechaNacimiento { get; set; }

public string Correo { get; set; }

public string Telefono { get; set; }

public string Organizacion { get; set; }

public string Profesion { get; set; }

public string Cargo { get; set; }

public string Contraseña { get; set; }

}

```

\*\*Evento.cs\*\*

```csharp

public class Evento

{

public int Id { get; set; }

public string Nombre { get; set; }

public string Descripcion { get; set; }

public string Ubicacion { get; set; }

public string InformacionContacto { get; set; }

public DateTime FechaInicio { get; set; }

public DateTime FechaFin { get; set; }

}

```

\*\*Equipo.cs\*\*

```csharp

public class Equipo

{

public int Id { get; set; }

public string Nombre { get; set; }

public string Institucion { get; set; }

public int RepresentanteId { get; set; }

}

```

\*\*Participante.cs\*\*

```csharp

public class Participante

{

public int Id { get; set; }

public int UsuarioId { get; set; }

public int EventoId { get; set; }

public int? EquipoId { get; set; }

}

```

\*\*Inscripcion.cs\*\*

```csharp

public class Inscripcion

{

public int Id { get; set; }

public int EventoId { get; set; }

public int UsuarioId { get; set; }

public int? EquipoId { get; set; }

public DateTime FechaInscripcion { get; set; }

}

```

\*\*Certificado.cs\*\*

```csharp

public class Certificado

{

public int Id { get; set; }

public int InscripcionId { get; set; }

public string Tipo { get; set; }

public DateTime FechaEmision { get; set; }

public string Archivo { get; set; }

}

```

### 2. Clase de Contexto para la Base de Datos

Para interactuar con la base de datos SQL, se puede utilizar Entity Framework (EF) Core, que facilita el mapeo de las clases de C# a las tablas de la base de datos.

\*\*AppDbContext.cs\*\*

```csharp

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext

{

public DbSet<Usuario> Usuarios { get; set; }

public DbSet<Evento> Eventos { get; set; }

public DbSet<Equipo> Equipos { get; set; }

public DbSet<Participante> Participantes { get; set; }

public DbSet<Inscripcion> Inscripciones { get; set; }

public DbSet<Certificado> Certificados { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("YourConnectionStringHere");

}

}

```

### 3. Repositorios para Acceso a Datos

Crear repositorios para encapsular la lógica de acceso a datos. Estos repositorios utilizarán `AppDbContext` para interactuar con la base de datos.

\*\*UsuarioRepository.cs\*\*

```csharp

public class UsuarioRepository

{

private readonly AppDbContext \_context;

public UsuarioRepository(AppDbContext context)

{

\_context = context;

}

public void CrearUsuario(Usuario usuario)

{

\_context.Usuarios.Add(usuario);

\_context.SaveChanges();

}

public Usuario ObtenerUsuarioPorCorreoYContraseña(string correo, string contraseña)

{

return \_context.Usuarios.FirstOrDefault(u => u.Correo == correo && u.Contraseña == contraseña);

}

}

```

\*\*EventoRepository.cs\*\*

```csharp

public class EventoRepository

{

private readonly AppDbContext \_context;

public EventoRepository(AppDbContext context)

{

\_context = context;

}

public IEnumerable<Evento> ObtenerEventosDisponibles()

{

return \_context.Eventos.Where(e => e.FechaInicio >= DateTime.Now).ToList();

}

public void CrearEvento(Evento evento)

{

\_context.Eventos.Add(evento);

\_context.SaveChanges();

}

}

```

\*\*InscripcionRepository.cs\*\*

```csharp

public class InscripcionRepository

{

private readonly AppDbContext \_context;

public InscripcionRepository(AppDbContext context)

{

\_context = context;

}

public void InscribirUsuario(Inscripcion inscripcion)

{

\_context.Inscripciones.Add(inscripcion);

\_context.SaveChanges();

}

public Inscripcion ObtenerInscripcionPorId(int id)

{

return \_context.Inscripciones

.Include(i => i.Evento)

.Include(i => i.Usuario)

.FirstOrDefault(i => i.Id == id);

}

}

```

\*\*CertificadoRepository.cs\*\*

```csharp

public class CertificadoRepository

{

private readonly AppDbContext \_context;

public CertificadoRepository(AppDbContext context)

{

\_context = context;

}

public IEnumerable<Certificado> ObtenerCertificadosPorUsuarioId(int usuarioId)

{

return \_context.Certificados

.Include(c => c.Inscripcion)

.Where(c => c.Inscripcion.UsuarioId == usuarioId)

.ToList();

}

}

```

### 4. Servicios para Lógica de Negocio

Crear servicios que usen los repositorios para implementar la lógica de negocio.

\*\*UsuarioService.cs\*\*

```csharp

public class UsuarioService

{

private readonly UsuarioRepository \_usuarioRepository;

public UsuarioService(UsuarioRepository usuarioRepository)

{

\_usuarioRepository = usuarioRepository;

}

public void RegistrarUsuario(Usuario usuario)

{

\_usuarioRepository.CrearUsuario(usuario);

}

public Usuario AutenticarUsuario(string correo, string contraseña)

{

return \_usuarioRepository.ObtenerUsuarioPorCorreoYContraseña(correo, contraseña);

}

}

```

\*\*EventoService.cs\*\*

```csharp

public class EventoService

{

private readonly EventoRepository \_eventoRepository;

public EventoService(EventoRepository eventoRepository)

{

\_eventoRepository = eventoRepository;

}

public IEnumerable<Evento> ObtenerEventosDisponibles()

{

return \_eventoRepository.ObtenerEventosDisponibles();

}

public void CrearEvento(Evento evento)

{

\_eventoRepository.CrearEvento(evento);

}

}

```

\*\*InscripcionService.cs\*\*

```csharp

public class InscripcionService

{

private readonly InscripcionRepository \_inscripcionRepository;

public InscripcionService(InscripcionRepository inscripcionRepository)

{

\_inscripcionRepository = inscripcionRepository;

}

public void InscribirUsuario(Inscripcion inscripcion)

{

\_inscripcionRepository.InscribirUsuario(inscripcion);

}

public Inscripcion ObtenerInscripcionPorId(int id)

{

return \_inscripcionRepository.ObtenerInscripcionPorId(id);

}

}

```

### 5. Controladores para la API

Crear controladores para manejar las solicitudes HTTP.

\*\*UsuarioController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class UsuarioController : ControllerBase

{

private readonly UsuarioService \_usuarioService;

public UsuarioController(UsuarioService usuarioService)

{

\_usuarioService = usuarioService;

}

[HttpPost("register")]

public IActionResult RegistrarUsuario([FromBody] Usuario usuario)

{

\_usuarioService.RegistrarUsuario(usuario);

return Ok();

}

[HttpPost("login")]

public IActionResult IniciarSesion([FromBody] LoginDto login)

{

var usuario = \_usuarioService.AutenticarUsuario(login.Correo, login.Contraseña);

if (usuario == null)

{

return Unauthorized();

}

return Ok(usuario);

}

}

public class LoginDto

{

public string Correo { get; set; }

public string Contraseña { get; set; }

}

```

\*\*EventoController.cs\*\*

```csharp

[ApiController]

[Route("api/[controller]")]

public class EventoController : ControllerBase

{

private readonly EventoService \_eventoService;

public EventoController(EventoService eventoService)

{

\_eventoService = eventoService;

}

[HttpGet]

public IActionResult ObtenerEventosDisponibles()

{

var eventos = \_eventoService.ObtenerEventosDisponibles();

return Ok(eventos);

}

[HttpPost]

public IActionResult CrearEvento([FromBody] Evento evento)

{

\_eventoService.CrearEvento(evento);

return Ok();

}

}

```

### 6. Configuración de Dependencias

Configurar la inyección de dependencias en el `Startup.cs`.

\*\*Startup.cs\*\*

```csharp

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddScoped<UsuarioRepository>();

services.AddScoped<EventoRepository>();

services.AddScoped<InscripcionRepository>();

services.AddScoped<CertificadoRepository>();

services.AddScoped<UsuarioService>();

services.AddScoped<EventoService>();

services.AddScoped<InscripcionService>();

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

```

Estas clases y configuraciones proporcionan una base sólida para gestionar la interacción con la base de datos y cumplir con los requisitos del sistema de gestión de eventos.

**-----------------**

DIAGRAMA DE CLASES

Para representar la integración de la aplicación de gestión de eventos en un diagrama de clases, incluiremos las entidades principales, sus relaciones y las clases de repositorio y servicio. Aquí tienes un diagrama de clases simplificado que incluye las entidades, repositorios, servicios y controladores:

Digrama:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Texto

Descripción generada automáticamente

### Descripción del Diagrama de Clases:

#### Entidades

1. \*\*Usuario\*\*: Representa los datos de los usuarios.

2. \*\*Evento\*\*: Contiene los detalles de los eventos.

3. \*\*Equipo\*\*: Almacena información sobre los equipos que participan.

4. \*\*Participante\*\*: Relaciona usuarios, eventos y equipos.

5. \*\*Inscripcion\*\*: Registra la inscripción de usuarios (individuales o como parte de equipos) en eventos.

6. \*\*Certificado\*\*: Almacena información sobre los certificados emitidos.

#### Repositorios

1. \*\*UsuarioRepository\*\*: Encapsula la lógica de acceso a datos para los usuarios.

2. \*\*EventoRepository\*\*: Encapsula la lógica de acceso a datos para los eventos.

3. \*\*EquipoRepository\*\*: Encapsula la lógica de acceso a datos para los equipos.

4. \*\*InscripcionRepository\*\*: Encapsula la lógica de acceso a datos para las inscripciones.

5. \*\*CertificadoRepository\*\*: Encapsula la lógica de acceso a datos para los certificados.

#### Servicios

1. \*\*UsuarioService\*\*: Provee la lógica de negocio relacionada con los usuarios.

2. \*\*EventoService\*\*: Provee la lógica de negocio relacionada con los eventos.

3. \*\*InscripcionService\*\*: Provee la lógica de negocio relacionada con las inscripciones.

#### Controladores

1. \*\*UsuarioController\*\*: Maneja las solicitudes HTTP relacionadas con los usuarios.

2. \*\*EventoController\*\*: Maneja las solicitudes HTTP relacionadas con los eventos.

### Notas:

- Los repositorios interactúan directamente con `AppDbContext` para realizar operaciones CRUD.

- Los servicios usan los repositorios para aplicar la lógica de negocio y manipular los datos.

- Los controladores exponen las APIs y utilizan los servicios para responder a las solicitudes HTTP.

Este diagrama proporciona una visión general de las clases principales y sus interacciones dentro de la arquitectura de la aplicación. Cada clase tiene responsabilidades claras y está diseñada para mantener la separación de preocupaciones, lo cual facilita el mantenimiento y la escalabilidad del sistema.

**-----------------**

**MODULARIZACION ANGULAR PARA INTERACCIONAR CON CONTROLADORES**

Para implementar una aplicación Angular por módulos que interactúe con los controladores, seguiremos una arquitectura modular que incluye:

1. \*\*Módulo principal (AppModule)\*\*

2. \*\*Módulo de autenticación (AuthModule)\*\*

3. \*\*Módulo de eventos (EventModule)\*\*

4. \*\*Servicios (Services)\*\*

5. \*\*Componentes (Components)\*\*

6. \*\*Rutas (Routing)\*\*

### Estructura del Proyecto

Texto

Descripción generada automáticamente

### Módulo Principal (AppModule)

\*\*app.module.ts\*\*

```typescript

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { AuthModule } from './auth/auth.module';

import { EventModule } from './events/event.module';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

HttpClientModule,

AppRoutingModule,

AuthModule,

EventModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

```

### Módulo de Autenticación (AuthModule)

\*\*auth.module.ts\*\*

```typescript

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { LoginComponent } from './login/login.component';

import { RegisterComponent } from './register/register.component';

import { AuthService } from './auth.service';

import { RouterModule } from '@angular/router';

@NgModule({

declarations: [

LoginComponent,

RegisterComponent

],

imports: [

CommonModule,

FormsModule,

ReactiveFormsModule,

RouterModule.forChild([

{ path: 'login', component: LoginComponent },

{ path: 'register', component: RegisterComponent }

])

],

providers: [AuthService]

})

export class AuthModule { }

```

\*\*login.component.ts\*\*

```typescript

import { Component } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { AuthService } from '../auth.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent {

loginForm: FormGroup;

constructor(

private fb: FormBuilder,

private authService: AuthService,

private router: Router

) {

this.loginForm = this.fb.group({

correo: ['', [Validators.required, Validators.email]],

contraseña: ['', Validators.required]

});

}

onSubmit() {

if (this.loginForm.valid) {

this.authService.login(this.loginForm.value).subscribe(() => {

this.router.navigate(['/']);

});

}

}

}

```

\*\*register.component.ts\*\*

```typescript

import { Component } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { AuthService } from '../auth.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-register',

templateUrl: './register.component.html',

styleUrls: ['./register.component.css']

})

export class RegisterComponent {

registerForm: FormGroup;

constructor(

private fb: FormBuilder,

private authService: AuthService,

private router: Router

) {

this.registerForm = this.fb.group({

nombre: ['', Validators.required],

correo: ['', [Validators.required, Validators.email]],

contraseña: ['', Validators.required]

});

}

onSubmit() {

if (this.registerForm.valid) {

this.authService.register(this.registerForm.value).subscribe(() => {

this.router.navigate(['/login']);

});

}

}

}

```

\*\*auth.service.ts\*\*

```typescript

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { User } from '../shared/models/user.model';

@Injectable({

providedIn: 'root'

})

export class AuthService {

private apiUrl = 'http://localhost:5000/api'; // Cambiar por tu URL de API

constructor(private http: HttpClient) { }

login(credentials: { correo: string, contraseña: string }): Observable<User> {

return this.http.post<User>(`${this.apiUrl}/Usuario/login`, credentials);

}

register(user: User): Observable<User> {

return this.http.post<User>(`${this.apiUrl}/Usuario/register`, user);

}

}

```

### Módulo de Eventos (EventModule)

\*\*event.module.ts\*\*

```typescript

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { EventListComponent } from './event-list/event-list.component';

import { EventCreateComponent } from './event-create/event-create.component';

import { EventService } from './event.service';

import { RouterModule } from '@angular/router';

@NgModule({

declarations: [

EventListComponent,

EventCreateComponent

],

imports: [

CommonModule,

FormsModule,

ReactiveFormsModule,

RouterModule.forChild([

{ path: 'events', component: EventListComponent },

{ path: 'events/create', component: EventCreateComponent }

])

],

providers: [EventService]

})

export class EventModule { }

```

\*\*event-list.component.ts\*\*

```typescript

import { Component, OnInit } from '@angular/core';

import { EventService } from '../event.service';

import { Event } from '../../shared/models/event.model';

@Component({

selector: 'app-event-list',

templateUrl: './event-list.component.html',

styleUrls: ['./event-list.component.css']

})

export class EventListComponent implements OnInit {

events: Event[] = [];

constructor(private eventService: EventService) {}

ngOnInit(): void {

this.eventService.getEvents().subscribe(events => {

this.events = events;

});

}

}

```

\*\*event-create.component.ts\*\*

```typescript

import { Component } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { EventService } from '../event.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-event-create',

templateUrl: './event-create.component.html',

styleUrls: ['./event-create.component.css']

})

export class EventCreateComponent {

eventForm: FormGroup;

constructor(

private fb: FormBuilder,

private eventService: EventService,

private router: Router

) {

this.eventForm = this.fb.group({

nombre: ['', Validators.required],

descripcion: ['', Validators.required],

ubicacion: ['', Validators.required],

informacionContacto: ['', Validators.required],

fechaInicio: ['', Validators.required],

fechaFin: ['', Validators.required]

});

}

onSubmit() {

if (this.eventForm.valid) {

this.eventService.createEvent(this.eventForm.value).subscribe(() => {

this.router.navigate(['/events']);

});

}

}

}

```

\*\*event.service.ts\*\*

```typescript

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Event } from '../shared/models/event.model';

@Injectable({

providedIn: 'root'

})

export class EventService {

private apiUrl = 'http://localhost:5000/api'; // Cambiar por tu URL de API

constructor(private http: HttpClient) { }

getEvents(): Observable<Event[]> {

return this.http.get<Event[]>(`${this.apiUrl}/Evento`);

}

createEvent(event: Event): Observable<Event> {

return this.http.post<Event>(`${this.apiUrl}/Evento`, event);

}

}

```

### Modelos Compartidos

\*\*user.model.ts\*\*

```typescript

export interface User {

id?: number;

nombre: string;

direccion?: string;

fechaNacimiento?: Date;

correo: string;

telefono?: string;

organizacion?: string;

profesion?: string;

cargo?: string;

contraseña: string;

}

```

\*\*event.model.ts\*\*

```typescript

export interface Event {

id?: number;

nombre: string;

descripcion: string;

ubicacion: string;

informacionContacto: string;

fechaInicio: Date;

fechaFin

: Date;

}

```

### Servicio HTTP Compartido (opcional)

\*\*http.service.ts\*\*

```typescript

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class HttpService {

constructor(private http: HttpClient) { }

get<T>(url: string): Observable<T> {

return this.http.get<T>(url);

}

post<T>(url: string, body: any): Observable<T> {

return this.http.post<T>(url, body);

}

// Otros métodos HTTP como put, delete, etc.

}

```

### Enrutamiento Principal

\*\*app-routing.module.ts\*\*

```typescript

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [

{ path: '', redirectTo: '/events', pathMatch: 'full' },

{ path: 'login', loadChildren: () => import('./auth/auth.module').then(m => m.AuthModule) },

{ path: 'register', loadChildren: () => import('./auth/auth.module').then(m => m.AuthModule) },

{ path: 'events', loadChildren: () => import('./events/event.module').then(m => m.EventModule) },

{ path: '\*\*', redirectTo: '/events' }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

```

### Componente Principal

\*\*app.component.ts\*\*

```typescript

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Event Management App';

}

```

\*\*app.component.html\*\*

```html

<nav>

<a routerLink="/events">Eventos</a>

<a routerLink="/login">Iniciar Sesión</a>

<a routerLink="/register">Registrar</a>

</nav>

<router-outlet></router-outlet>

```

### Resumen

Este esqueleto de aplicación Angular está diseñado para ser modular y fácil de mantener. Cada módulo (autenticación y eventos) tiene sus propios componentes, servicios y enrutamiento. Los servicios se encargan de la comunicación con la API backend y los componentes manejan la lógica de la interfaz de usuario.

**-----------------**

**REFERENCIA:**

Link CHAT-GPT: <https://chatgpt.com/share/3a38af00-008f-46c8-b2e6-39bdcafe31f2>