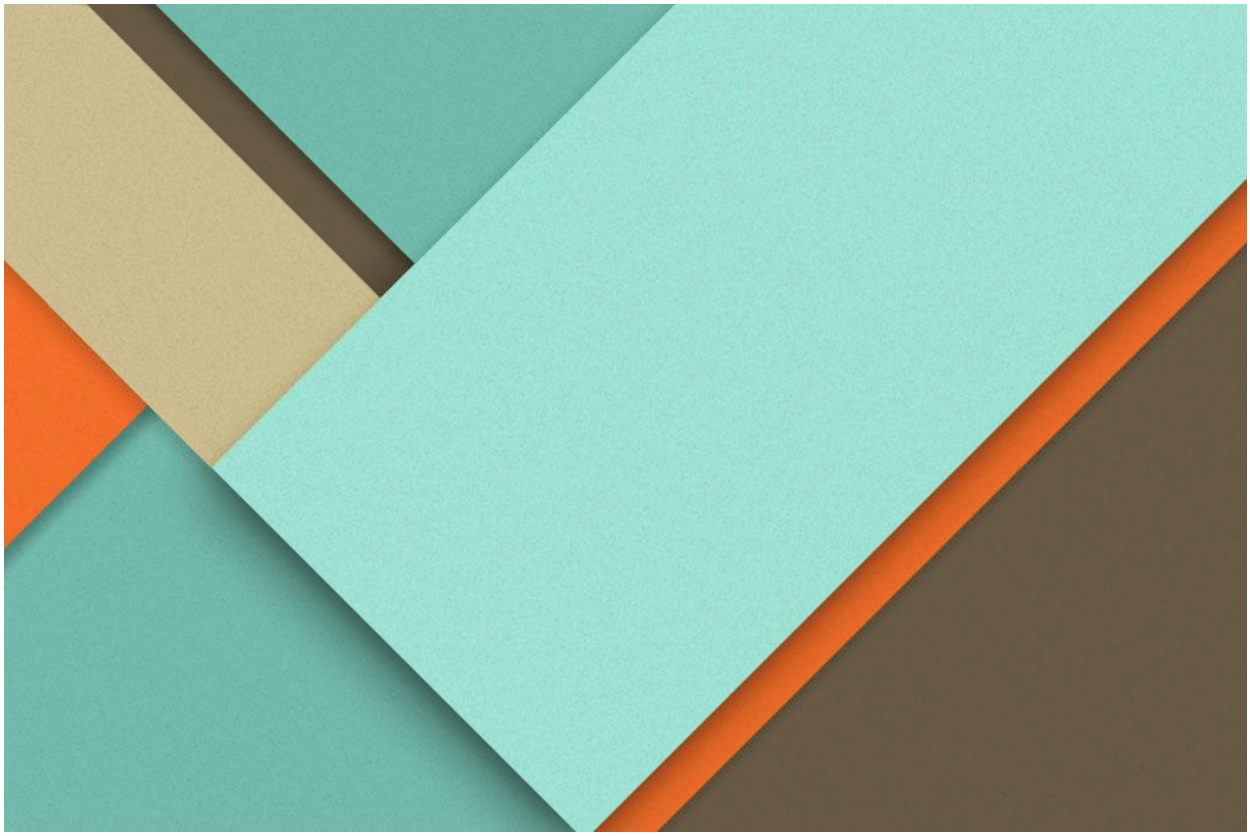


ACTIVIDAD 4: ACCESO A BD CON ENTITY FRAMEWORK

26/10/2025

—

Ángel Molina Rodríguez
campusfp
getafe
DWES

ÍNDICE

INTRODUCCIÓN

Para esta actividad he desarrollado una aplicación web de gestión de videojuegos. La idea es tener un catálogo donde puedes ver información sobre diferentes juegos, sus desarrolladoras, géneros y las plataformas donde están disponibles. También he añadido una sección de estadísticas para ver un resumen del catálogo.

¿Qué hace la aplicación?

Básicamente es como una mini base de datos de videojuegos donde puedes:

1. Ver todos los juegos con su info básica (título, precio, desarrolladora...)
2. Buscar juegos por género
3. Ver los detalles completos de cada juego, incluidas las plataformas
4. Consultar estadísticas como el precio medio o qué desarrolladora tiene más juegos

Tecnologías que he usado

He trabajado con ASP.NET Core 7.0 usando el patrón MVC. Para la base de datos he usado Entity Framework Core con SQLite (en lugar de SQL Server, porque estoy en Mac y LocalDB no funciona aquí). Para el diseño he tirado de Bootstrap 5 que ya viene preparado y es responsive.

Escribe aquí tu texto Escribe aquí tu texto Escribe aquí tu texto Escribe aquí tu texto Escribe aquí tu texto Escribe aquí tu texto Escribe aquí tu texto Escribe aquí tu texto.

ANÁLISIS DE TECNOLOGÍAS DE ACCESO A DATOS

ADO.NET vs Entity Framework

Cuando empecé a investigar cómo conectar la aplicación a una base de datos, vi que hay dos formas principales de hacerlo en .NET:

ADO.NET es la forma más tradicional. Aquí tienes que escribir todas las consultas SQL a mano y conectarte directamente a la base de datos con clases como SqlConnection. Te da control total pero es bastante tedioso porque tienes que mapear todo manualmente.

Entity Framework Core (que es lo que he usado) funciona como un ORM. Básicamente trabajas con objetos C# normales y él se encarga de convertir todo a SQL por ti. Es mucho más rápido de usar y el código queda más limpio.

¿Por qué elegí Entity Framework?

La verdad es que al principio dudé un poco porque había oído que EF Core podía ser más lento, pero para este proyecto me pareció mejor opción porque:

- Es mucho más rápido desarrollar. En lugar de escribir SQL para cada consulta, uso LINQ
- Las migraciones son automáticas, así que si cambio algo en los modelos, la base de datos se actualiza sola
- Gestiona las relaciones entre tablas de forma automática, que con tantas relaciones como tengo (1:N, N:M) me ahorra mucho trabajo
- El código es más fácil de leer y mantener

Además, como el proyecto no es muy grande ni va a tener millones de consultas, el tema del rendimiento no me preocupa mucho.

DISEÑO DE LA BASE DE DATOS

Estructura general

He diseñado la base de datos con 5 tablas que están relacionadas entre sí:

- Videojuegos: La tabla principal con toda la info de cada juego
- Desarrolladoras: Las empresas que hacen los juegos
- Géneros: Acción, RPG, Aventura, etc.
- Plataformas: PS5, Xbox, Switch, PC...
- VideojuegoPlataformas: Tabla intermedia para la relación muchos a muchos

Relaciones

He implementado dos tipos de relaciones:

1:N (Uno a muchos):

- Una desarrolladora puede tener muchos videojuegos, pero cada juego solo tiene una desarrolladora
- Lo mismo con los géneros: un género agrupa varios juegos

N:M (Muchos a muchos):

- Un videojuego puede estar en varias plataformas (por ejemplo GTA V está en PS5, Xbox y PC)
- Y una plataforma tiene muchos juegos diferentes
- Para esto he tenido que crear la tabla intermedia VideojuegoPlataformas

Decisiones que tomé

Al principio pensaba usar int para los IDs y poner autoincrement, porque es lo más sencillo. Para los precios he usado decimal para no perder precisión (aunque luego tuve que convertir a double para calcular el promedio en SQLite, pero eso lo cuento más adelante en problemas).

También decidí normalizar bien la base de datos para no repetir información. Por ejemplo, en lugar de poner el nombre de la desarrolladora en cada juego, pongo solo el ID y hago la relación.

IMPLEMENTACIÓN

Estructura del proyecto

He seguido el patrón MVC que divide todo en tres partes:

Models: Aquí están las clases que representan las tablas (Videojuego, Desarrolladora, etc.) y el contexto de la base de datos

Controllers: Gestiona las peticiones. Por ejemplo, cuando alguien pide ver la lista de juegos, el VideojuegosController se encarga de obtener los datos y enviarlos a la vista

Views: Las páginas que ve el usuario, hechas con Razor que mezcla HTML con C#

Los modelos

Cada modelo es una clase C# que representa una tabla.

Lo interesante son las propiedades de navegación. Por ejemplo, en Videojuego tengo:

- public Desarrolladora Desarrolladora { get; set; }
- public Genero Genero { get; set; }

Esto hace que Entity Framework cargue automáticamente la desarrolladora y el género cuando consulto un videojuego. Muy cómodo.

El contexto (ApplicationDbContext)

El contexto es como el puente entre el código y la base de datos. Aquí defino los DbSet (las "colecciones" de cada tabla) y configuro las relaciones más complejas.

Lo más importante que configuré fue la relación N:M con las plataformas. Tuve que usar el método OnModelCreating para decirle a EF que la clave de VideojuegoPlataformas es compuesta (VideojuegoId + PlataformaId).

Los controladores

He hecho dos controladores principales:

VideojuegosController tiene estos métodos:

- Index(): Muestra todos los juegos
- Details(id): Muestra los detalles de un juego concreto con sus plataformas
- PorGenero(generoId): Filtra los juegos por género
- Estadisticas(): Calcula y muestra las estadísticas

DesarrolladorasController:

- Index(): Lista todas las desarrolladoras con el número de juegos que tiene cada una

Consultas LINQ que he usado

Para traer los datos he usado LINQ, que es como escribir consultas pero en C#. Algunos ejemplos:

Para traer todos los juegos con sus relaciones:

```
var videojuegos = await _context.Videojuegos
    .Include(v => v.Desarrolladora)
    .Include(v => v.Genero)
    .ToListAsync();
```

Para filtrar por género:

```
var videojuegos = await _context.Videojuegos
    .Where(v => v.GeneroId == generoId)
    .Include(v => v.Desarrolladora)
    .ToListAsync();
```

Para las estadísticas:

```
var total = await _context.Videojuegos.CountAsync();
var promedio = await _context.Videojuegos.AverageAsync(v => (double)v.Precio);
```

Las vistas

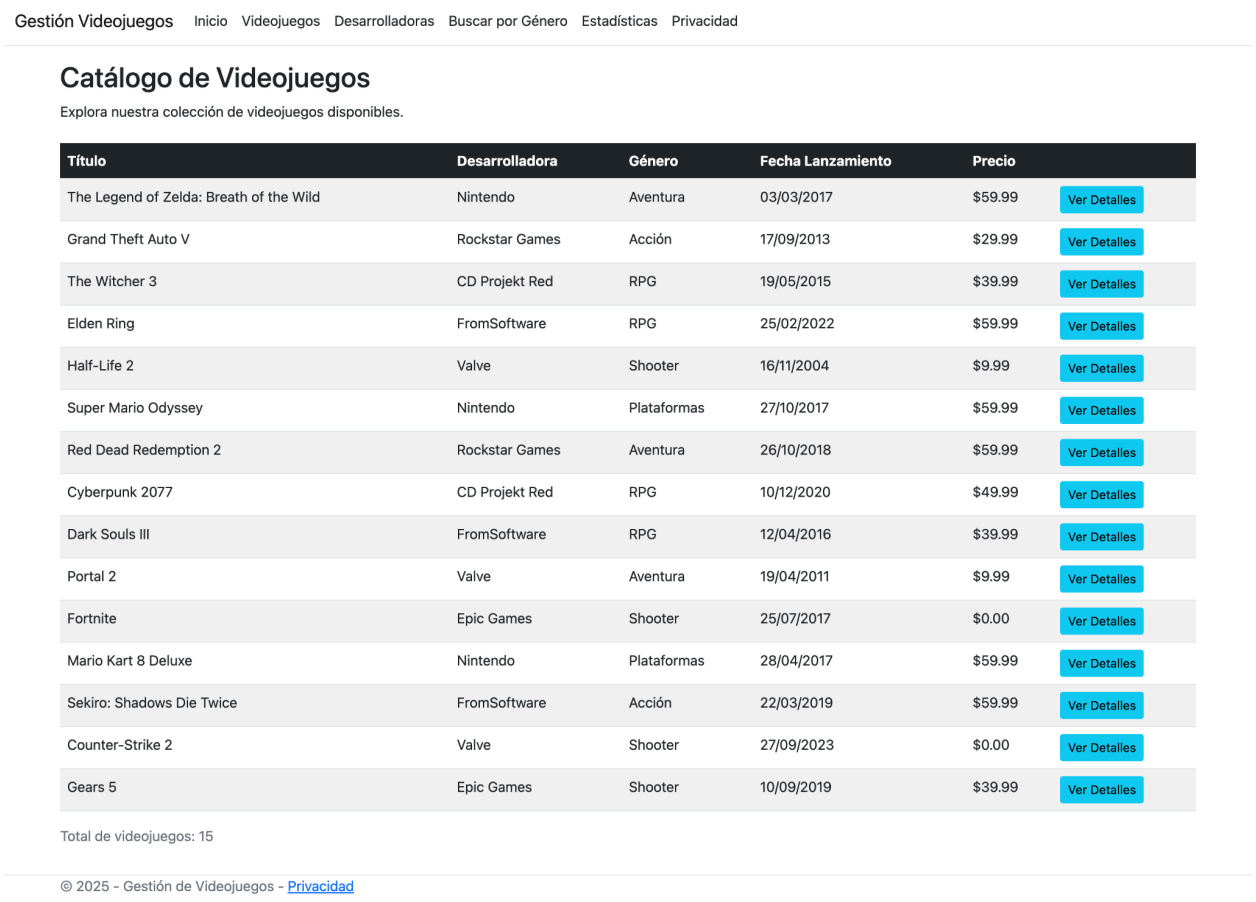
Todas las vistas usan tipado fuerte con @model. Esto significa que la vista “sabe” qué tipo de datos va a recibir, así que tienes autocompletado y comprobación de errores.

He usado Bootstrap para que quede decente visualmente sin tener que pelearme mucho con CSS. Las tablas, botones y demás vienen con estilos predefinidos que se adaptan a móvil y tablet.

PRUEBAS

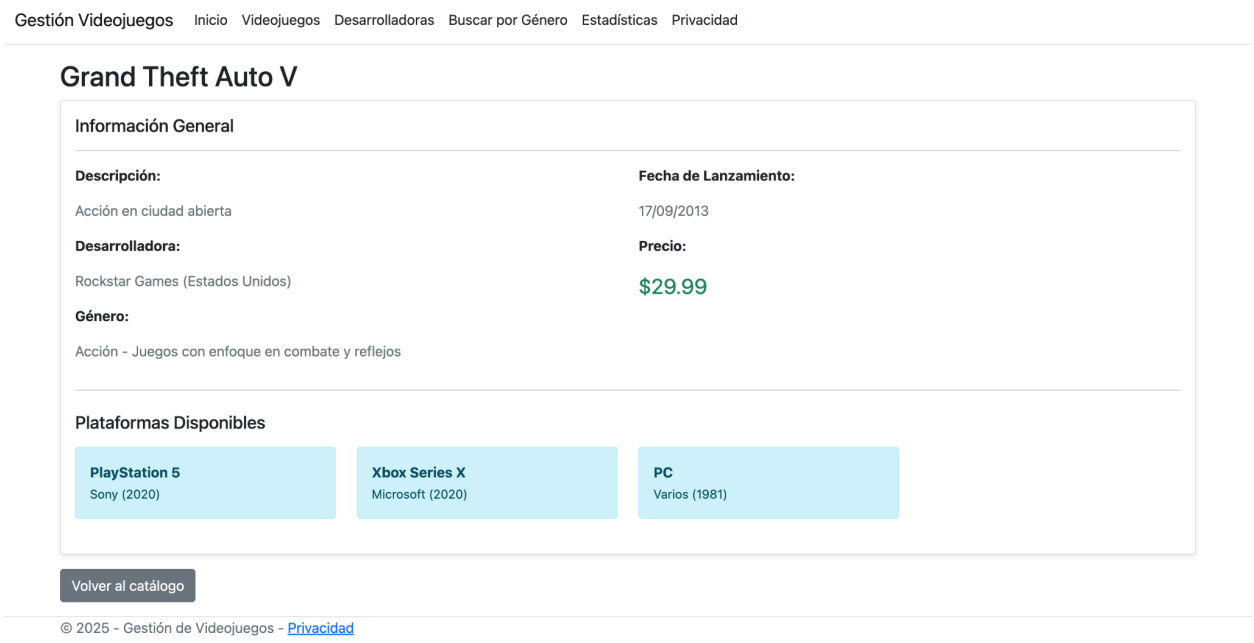
Prueba 1: Listado completo

He probado que la página principal muestra todos los videojuegos correctamente. Se ve una tabla con el título, desarrolladora, género, fecha y precio de cada uno. Funciona bien.



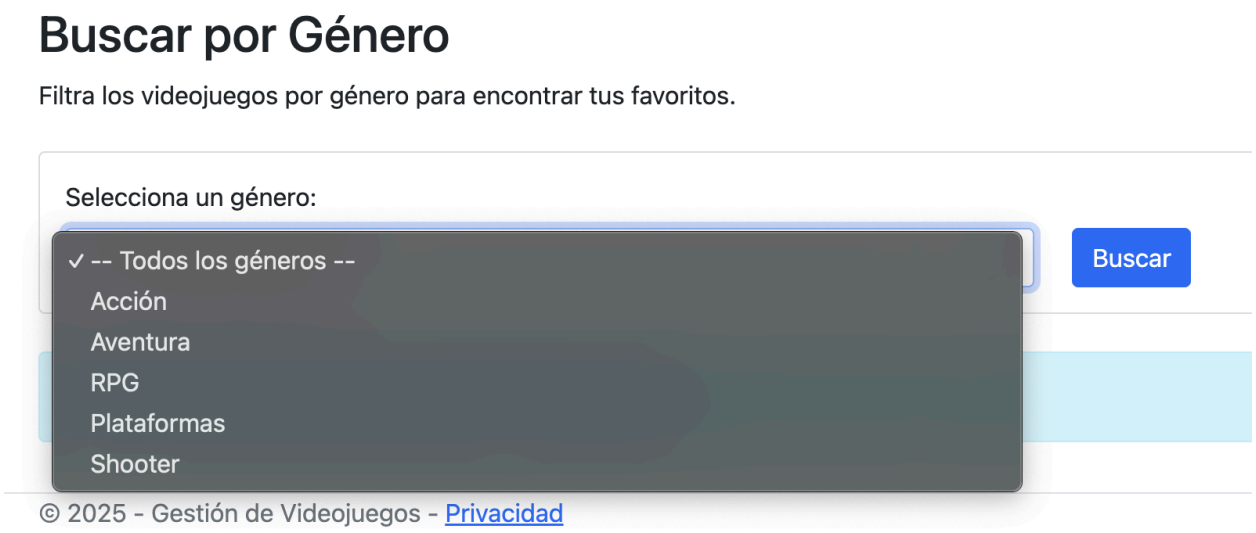
Prueba 2: Detalles de un juego

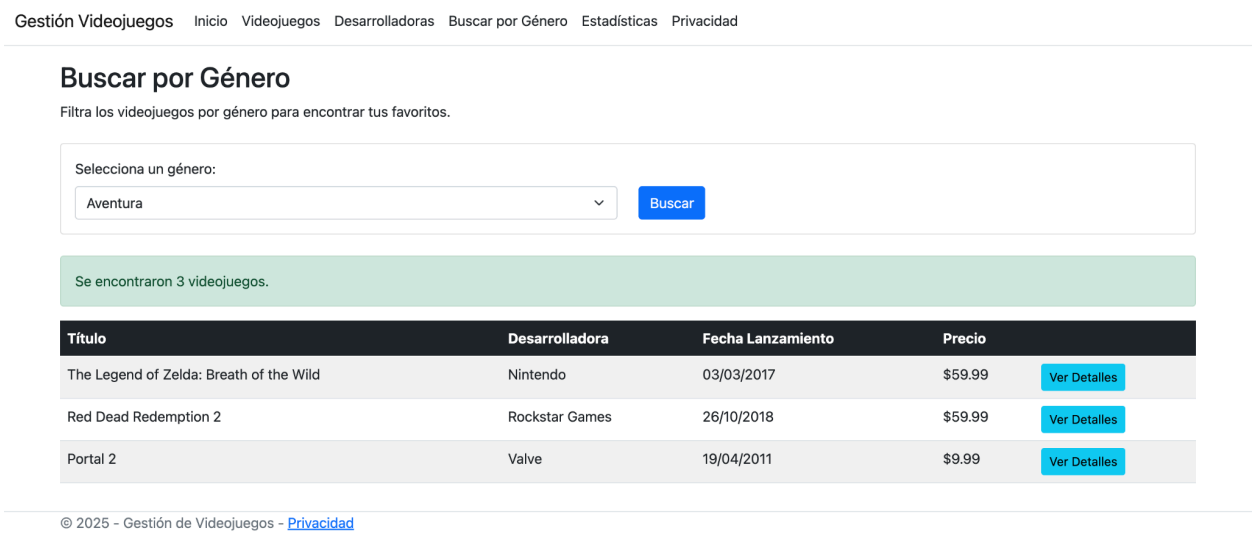
Al hacer clic en “Detalles” de cualquier juego, se abre una página con toda la información, incluida la lista de plataformas donde está disponible. Aquí usé Include con ThenInclude para traer las plataformas, y funcionó.



Prueba 3: Filtro por género

En la página “Por Género” hay un selector donde eliges el género y te muestra solo esos juegos. He probado con varios géneros y filtra correctamente.



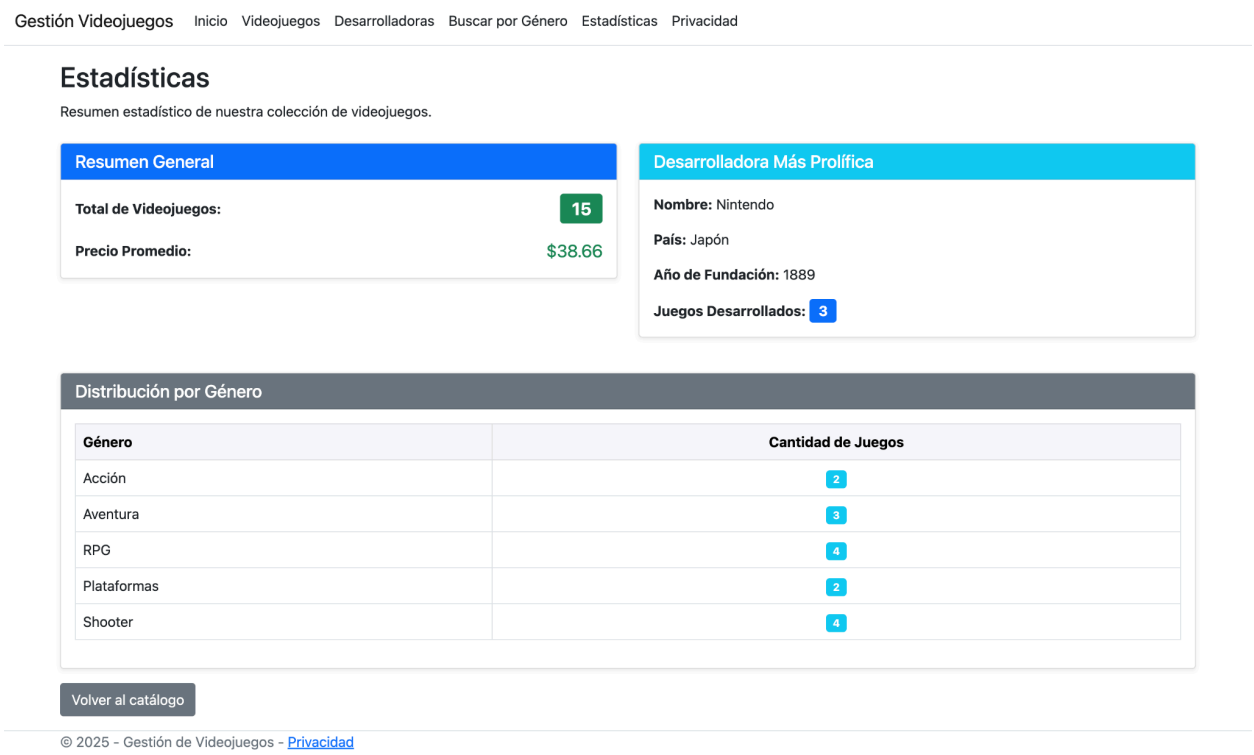


Prueba 4: Estadísticas

Esta página muestra:

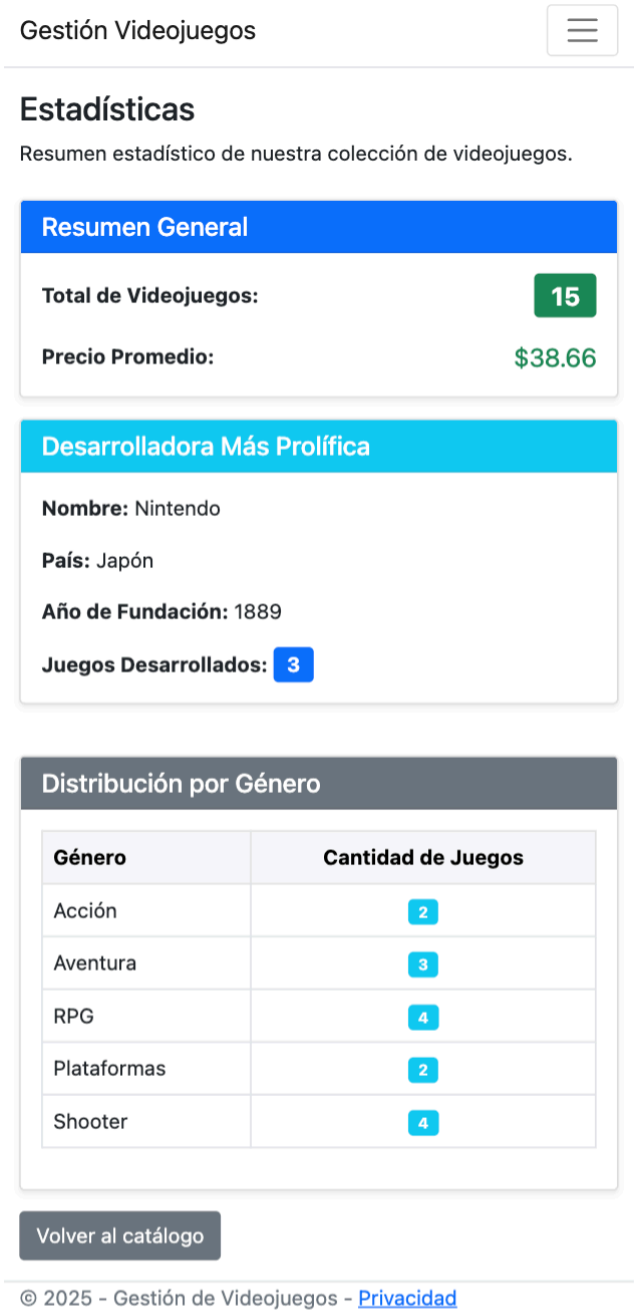
- Número total de videojuegos (15 en mi caso)
- Precio promedio
- Cuántos juegos hay de cada género
- Qué desarrolladora tiene más juegos

Todo se calcula correctamente con las consultas agregadas de LINQ.



Prueba 5: Responsive

He redimensionado la ventana del navegador para simular tablet y móvil. Bootstrap hace que todo se adapte automáticamente. En móvil las tablas se ven bien y los botones son más grandes, que está bien para tocar con el dedo.



PROBLEMAS QUE TUVE Y CÓMO LOS RESOLVÍ

Problema 1: LocalDB no funciona en Mac

Al principio configuré todo para usar SQL Server con LocalDB, pero cuando intenté ejecutar me dio error de que LocalDB no está soportado en macOS. Busqué alternativas y vi que SQLite funciona en cualquier plataforma.

Solución: Cambié el paquete NuGet de EntityFrameworkCore.SqlServer por EntityFrameworkCore.Sqlite, cambié la cadena de conexión en appsettings.json y listo. Incluso es más sencillo porque SQLite crea un archivo .db y no necesitas instalar nada.

Problema 2: SQLite y los decimales

Cuando intenté calcular el precio promedio con AverageAsync(), me saltó un error de que SQLite no puede hacer Average sobre decimal. Esto fue un poco frustrante.

Solución: Tuve que hacer un cast a double en la consulta: AverageAsync(v => (double)v.Precio) . No es muy elegante pero funciona perfectamente.

Problema 3: La relación N:M

Al principio Entity Framework no me generaba bien la tabla intermedia para videojuegos y plataformas. Después de leer la documentación entendí que tenía que configurarla manualmente con Fluent API.

Solución: En OnModelCreating configuré la clave compuesta y las relaciones con HasKey y HasOne/WithMany. Una vez entendido, es bastante lógico.

CONCLUSIONES

BIBLIOGRAFÍA

Otto, M., & Thornton, J. (s/f). *Get started with bootstrap*. Getbootstrap.com. Recuperado el 26 de octubre de 2025, de https://getbootstrap.com/docs/5.3/

SamMonoRT. (s/f). *SQLite database provider - EF core*. Microsoft.com. Recuperado el 26 de octubre de 2025, de https://learn.microsoft.com/en-us/ef/core/providers/sqlite/

(S/f). Stackoverflow.com. Recuperado el 26 de octubre de 2025, de https://stackoverflow.com/questions/50763108/vs-code-and-macos-error-localdb-is-not-supported-on-this-platform

Anexo

El código no se me pega con color como con visual studio code, me imagino que sera por que es el visual studio

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=videojuegos.db"
  }
}
```

Program.cs

```
using Microsoft.EntityFrameworkCore;
using videojuegosev.Models;

var builder = WebApplication.CreateBuilder(args);

// Configurar Entity Framework Core con SQLite
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddControllersWithViews();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Videojuegos}/{action=Index}/{id?}");

app.Run();
```

ApplicationDbContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace videojuegosev.Models
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        // DbSets - representan las tablas
        public DbSet<Videojuego> Videojuegos { get; set; }
        public DbSet<Desarrolladora> Desarrolladoras { get; set; }
        public DbSet<Genero> Generos { get; set; }
        public DbSet<Plataforma> Plataformas { get; set; }
        public DbSet<VideojuegoPlataforma> VideojuegoPlataformas { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // Configurar relación N:M entre Videojuego y Plataforma
            modelBuilder.Entity<VideojuegoPlataforma>()
                .HasKey(vp => new { vp.VideojuegoId, vp.PlataformaId });

            modelBuilder.Entity<VideojuegoPlataforma>()
                .HasOne(vp => vp.Videojuego)
```

```
            .WithMany(v => v.VideojuegoPlataformas)
            .HasForeignKey(vp => vp.VideojuegoId);

        modelBuilder.Entity<VideojuegoPlataforma>()
            .HasOne(vp => vp.Plataforma)
            .WithMany(p => p.VideojuegoPlataformas)
            .HasForeignKey(vp => vp.PlataformaId);

        // Configurar precisión para el precio
        modelBuilder.Entity<Videojuego>()
            .Property(v => v.Precio)
            .HasPrecision(18, 2);
    }
  }
}
```

Videojuego.cs

```
namespace videojuegosev.Models
{
    public class Videojuego
    {
        public int Id { get; set; }
        public string Titulo { get; set; } = string.Empty;
        public string Descripcion { get; set; } = string.Empty;
        public DateTime FechaLanzamiento { get; set; }
        public decimal Precio { get; set; }

        // Claves foráneas
        public int DesarrolladoraId { get; set; }
        public int GeneroId { get; set; }

        // Propiedades de navegación
        public Desarrolladora Desarrolladora { get; set; } = null!;
        public Genero Genero { get; set; } = null!;
        public ICollection<VideojuegoPlataforma> VideojuegoPlataformas { get; set; }
            = new List<VideojuegoPlataforma>();
    }
}
```

Desarrolladora.cs

```
namespace videojuegosev.Models
{
    public class Desarrolladora
    {
        public int Id { get; set; }
        public string Nombre { get; set; } = string.Empty;
        public string PaisOrigen { get; set; } = string.Empty;
        public int AñoFundacion { get; set; }

        public ICollection<Videojuego> Videojuegos { get; set; } = new List<Videojuego>();
    }
}
```

Genero.cs

```
namespace videojuegosev.Models
{
    public class Genero
    {
        public int Id { get; set; }
        public string Nombre { get; set; } = string.Empty;
        public string Descripcion { get; set; } = string.Empty;

        public ICollection<Videojuego> Videojuegos { get; set; } = new List<Videojuego>();
    }
}
```

Plataforma.cs

```
namespace videojuegosev.Models
{
    public class Plataforma
    {
        public int Id { get; set; }
        public string Nombre { get; set; } = string.Empty;
        public string Fabricante { get; set; } = string.Empty;
        public int AñoLanzamiento { get; set; }

        public ICollection<VideojuegoPlataforma> VideojuegoPlataformas { get; set; }
            = new List<VideojuegoPlataforma>();
    }
}
```

VideojuegoPlataforma.cs

```
namespace videojuegosev.Models
{
    public class VideojuegoPlataforma
    {
        public int VideojuegoId { get; set; }
        public Videojuego Videojuego { get; set; } = null!;

        public int PlataformaId { get; set; }
        public Plataforma Plataforma { get; set; } = null!;
    }
}
```

VideojuegosController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using videojuegosev.Models;

namespace videojuegosev.Controllers
{
    public class VideojuegosController : Controller
    {
        private readonly ApplicationDbContext _context;

        public VideojuegosController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Videojuegos
        public async Task<IActionResult> Index()
        {
            var videojuegos = await _context.Videojuegos
                .Include(v => v.Desarrolladora)
                .Include(v => v.Genero)
                .ToListAsync();

            return View(videojuegos);
        }
        // GET: Videojuegos/Details/5
        public async Task<IActionResult> Details(int? id)
        {
```

```
    if (id == null)
    {
        return NotFound();
    }
    var videojuego = await _context.Videojuegos
        .Include(v => v.Desarrolladora)
        .Include(v => v.Genero)
        .Include(v => v.VideojuegoPlataformas)
            .ThenInclude(vp => vp.Plataforma)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (videojuego == null)
    {
        return NotFound();
    }
    return View(videojuego);
}
// GET: Videojuegos/PorGenero
public async Task<IActionResult> PorGenero(int? generoId)
{
    ViewBag.Generos = await _context.Generos.ToListAsync();

    if (generoId == null)
    {
        return View(new List<Videojuego>());
    }
    var videojuegos = await _context.Videojuegos
        .Include(v => v.Desarrolladora)
        .Include(v => v.Genero)
        .Where(v => v.GeneroId == generoId)
```

```
            .ToListAsync();

        return View(videojuegos);
    }
    // GET: Videojuegos/Estadisticas
    public async Task<IActionResult> Estadisticas()
    {
        var totalJuegos = await _context.Videojuegos.CountAsync();
        var precioPromedio = await _context.Videojuegos.AverageAsync(v =>
(double)v.Precio);
        var juegosPorGenero = await _context.Generos
            .Select(g => new
            {
                Genero = g.Nombre,
                Cantidad = g.Videojuegos.Count()
            })
            .ToListAsync();
        var desarrolladoraMasProlifica = await _context.Desarrolladoras
            .Include(d => d.Videojuegos)
            .OrderByDescending(d => d.Videojuegos.Count)
            .FirstOrDefaultAsync();

        ViewBag.TotalJuegos = totalJuegos;
        ViewBag.PrecioPromedio = precioPromedio;
        ViewBag.JuegosPorGenero = juegosPorGenero;
        ViewBag.DesarrolladoraMasProlifica = desarrolladoraMasProlifica;
        return View();
    }
  }
}
```

DesarrolladorasController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using videojuegosev.Models;
namespace videojuegosev.Controllers
{
    public class DesarrolladorasController : Controller
    {
        private readonly ApplicationDbContext _context;
        public DesarrolladorasController(ApplicationDbContext context)
        {
            _context = context;
        }
        // GET: Desarrolladoras
        public async Task<IActionResult> Index()
        {
            var desarrolladoras = await _context.Desarrolladoras
                .Include(d => d.Videojuegos)
                .ToListAsync();

            return View(desarrolladoras);
        }
    }
}
```

Views/Videojuegos/Index.cshtml

```
@model IEnumerable<Videojuego>
@{
    ViewData["Title"] = "Catálogo de Videojuegos";
}
```

```
<h1>@ViewData["Title"]</h1>
<table class="table table-striped">
    <thead>
        <tr>
            <th>Título</th>
            <th>Desarrolladora</th>
            <th>Género</th>
            <th>Fecha Lanzamiento</th>
            <th>Precio</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.Titulo</td>
                <td>@item.Desarrolladora.Nombre</td>
                <td>@item.Genero.Nombre</td>
                <td>@item.FechaLanzamiento.ToString("dd/MM/yyyy")</td>
                <td>@item.Precio.ToString("C")</td>
                <td>
                    <a asp-action="Details" asp-route-id="@item.Id"
                       class="btn btn-sm btn-primary">Detalles</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Views/Videojuegos/Details.cshtml

```
@model Videojuego

@{
    ViewData["Title"] = "Detalles - " + Model.Titulo;
}

<h1>@Model.Titulo</h1>

<div class="row">
    <div class="col-md-6">
        <dl class="row">
            <dt class="col-sm-4">Descripción:</dt>
            <dd class="col-sm-8">@Model.Descripcion</dd>

            <dt class="col-sm-4">Desarrolladora:</dt>
            <dd class="col-sm-8">
                @Model.Desarrolladora.Nombre (@Model.Desarrolladora.PaisOrigen)
            </dd>

            <dt class="col-sm-4">Género:</dt>
            <dd class="col-sm-8">@Model.Genero.Nombre</dd>

            <dt class="col-sm-4">Fecha Lanzamiento:</dt>
            <dd class="col-sm-8">@Model.FechaLanzamiento.ToString("dd/MM/yyyy")</dd>

            <dt class="col-sm-4">Precio:</dt>
            <dd class="col-sm-8">@Model.Precio.ToString("C")</dd>
        </dl>
```

```
        <h4>Plataformas disponibles:</h4>
        <ul>
            @foreach (var vp in Model.VideojuegoPlataformas)
            {
                <li>
                    <strong>@vp.Plataforma.Nombre</strong> -
                    @vp.Plataforma.Fabricante (@vp.Plataforma.AñoLanzamiento)
                </li>
            }
        </ul>
    </div>
</div>

<a asp-action="Index" class="btn btn-secondary">Volver al listado</a>
```

Views/Shared/_Layout.cshtml

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Catálogo Videojuegos</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white
border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-controller="Videojuegos" asp-action="Index">
                    Catálogo Videojuegos
                </a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
                        data-bs-target=".navbar-collapse">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark"
                               asp-controller="Videojuegos" asp-action="Index">
                                Videojuegos
                            </a>
                        </li>
```

```
                <li class="nav-item">
                    <a class="nav-link text-dark"
                       asp-controller="Desarrolladoras" asp-action="Index">
                        Desarrolladoras
                    </a>
                </li>
                <li class="nav-item">
                    <a class="nav-link text-dark"
                       asp-controller="Videojuegos" asp-action="PorGenero">
                        Buscar por Género
                    </a>
                </li>
                <li class="nav-item">
                    <a class="nav-link text-dark"
                       asp-controller="Videojuegos" asp-action="Estadisticas">
                        Estadísticas
                    </a>
                </li>
            </ul>
        </div>
    </div>
</nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>
```

```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2025 - Catálogo de Videojuegos
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```