Trabajo Integrador: Escuchify

Grupo 16

---

1. Instalaciones Necesarias

Antes de escribir código, nos aseguramos de tener el entorno de desarrollo listo:

1. **.NET SDK:** escribir `dotnet --version` en tu terminal y ves un número (ej. 8.0.x), estás listo, sino es necesario realizar la instalacion.
2. **Extensiones de VS Code:**
   - **C# Dev Kit:** es la extensión oficial moderna. Instala también automáticamente la extensión base de C#.
   - **SQLite Viewer:** para ver los datos dentro de la base de datos directamente desde VS Code sin instalar programas externos.

---

2. Estructura del Proyecto

Para que el proyecto sea profesional y ordenado, no crearemos todo en una sola carpeta. Usaremos una estructura de **3 proyectos** dentro de una **Solución**:

1. **Shared:** una biblioteca de clases para guardar los Modelos (Cancion, Disco, Artista). ¿Por qué? para que tanto el Frontend como el Backend usen las mismas clases y no tengas que escribirlas dos veces.
2. **Api:** el Backend (ASP.NET Core WebAPI).
3. **Client:** el Frontend (Blazor WebAssembly).

Paso a paso para crear la estructura

Abrir una carpeta vacía en VS Code, luego abrir la terminal (`Ctrl + ñ`) y ejecutar estos comandos uno por uno:

```
# 1. Crear la solución vacía
dotnet new sln -n MusicaApp

# 2. Crear el proyecto Compartido (Modelos)
dotnet new classlib -o Shared

# 3. Crear el proyecto API (Backend)
dotnet new webapi -o Api

# 4. Crear el proyecto Blazor (Frontend)
dotnet new blazorwasm -o Client

# 5. Agregar los proyectos a la solución
dotnet sln add Shared
dotnet sln add Api
dotnet sln add Client

# 6. Referencias (el Cliente y la API deben conocer a Shared)
dotnet add Api reference Shared
dotnet add Client reference Shared
```

---

3. Definición de Modelos - Base de Datos

Vamos a la carpeta **Shared**. Aquí definirás cómo son tus datos. Crea tres clases: `Artista.cs`, `Disco.cs`, `Cancion.cs`.

---

4. Configuración del Backend (API + SQLite)

Ahora trabajaremos en la carpeta **Api**.

A. Instalar Entity Framework Core (Para la base de datos)

En la terminal, asegúrarse de estar en la carpeta de la API o usa el flag `--project`:

```
dotnet add Api package Microsoft.EntityFrameworkCore.Sqlite
dotnet add Api package Microsoft.EntityFrameworkCore.Tools
dotnet add Api package Microsoft.EntityFrameworkCore.Design
```

B. Crear el DbContext

Dentro de la carpeta **Api**, crear una carpeta `Data` y dentro un archivo `ApplicationDbContext.cs`:

```
using Microsoft.EntityFrameworkCore;
using Shared;

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

        public DbSet<Artista> Artistas { get; set; }
        public DbSet<Disco> Discos { get; set; }
        public DbSet<Cancion> Canciones { get; set; }
    }
}
```

C. Conexión en `Program.cs` y `appsettings.json`

1. Abrir `Api/appsettings.json` y agregar la conexión antes de "Logging":

   ```
   "ConnectionStrings": {
     "DefaultConnection": "Data Source=musica.db"
   },
   ```

2. Abrir `Api/Program.cs` y configurar el servicio y **CORS**, es muy importante para que Blazor pueda hablar con la API:

```
using Api.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// 1. Configurar SQLite
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

// 2. Configurar CORS (Permitir que Blazor llame a la API)
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowBlazor", policy =>
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader());
});

builder.Services.AddControllers();

var app = builder.Build();


// 3. Activar CORS (antes de Authorization/Controllers)
app.UseCors("AllowBlazor");

app.UseAuthorization();
app.MapControllers();

app.Run();
```

D. Generar la Base de Datos - Migraciones

En la terminal:

```
# Crear la migración inicial
dotnet ef migrations add InitialCreate --project Api --startup-project Api

# Aplicar los cambios y crear el archivo .db
dotnet ef database update --project Api --startup-project Api
```

Esto creará un archivo `musica.db` en la carpeta Api.

---

5. Creación de Controladores - API

En `Api/Controllers`, crear `ArtistasController.cs`, `DiscosController.cs`, `CancionesController.cs`. Aquí implementaremos el CRUD.

---

6. Configuración del Frontend (Blazor)

Ahora continuamos con la carpeta **Client**.

A. Conectar con la API

Abrir `Client/Program.cs`. Ahi encontraremos una línea que configura `HttpClient`. Nos aseguramos de que apunte a la dirección donde corre tu API (en el caso de nuestro proyecto `http://localhost:5235`).

```
// Asegúrarse de cambiar la URL por la que usa tu API al ejecutarse
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("http://localhost:5235") }
```

B. Instalar librería para Formularios

Para facilitar el envío de JSON, nos aseguramos de tener:

```
dotnet add Client package Microsoft.Extensions.Http
```

C. Crear una Página CRUD

En `Client/Pages`, creamos `Artistas.razor`, `Discos.razor`, `Canciones.razor`.

Resumen de Ejecución

Para probar todo, necesitamos correr ambos proyectos a la vez.

1. Abrimos una terminal y corremos la API:
   ```
   dotnet run --project Api
   ```
2. Abrimos otra terminal y corremos Blazor:
   ```
   dotnet run --project Client
   ```

Preguntas que surgieron durante el desarrollo:

1. ¿Qué es una API?

API (Interfaz de Programación de Aplicaciones) es un intermediario que permite que dos aplicaciones de software se comuniquen entre sí.

- **Analogía:** Es como un **camarero** en un restaurante; tú (el cliente) le pides algo, él lleva la orden a la cocina (el sistema) y te trae la respuesta (la comida), sin que tú necesites saber cómo funciona la cocina.

2. ¿Qué es un Framework?

Un **Framework** es un esquema o estructura base de trabajo que proporciona herramientas y código predefinido.

- **Función:** Te evita empezar desde cero. En lugar de fabricar los ladrillos, el framework te da los ladrillos, el cemento y los planos básicos para que tú construyas la casa (la

aplicación) más rápido.

3. ¿Qué es .NET?

.NET es una plataforma de desarrollo gratuita y de código abierto creada por **Microsoft**.

- **Uso:** Sirve para construir todo tipo de aplicaciones (Web, Móvil, Escritorio, Juegos, IoT) utilizando lenguajes como C#, F# o Visual Basic. Es el ecosistema donde corren tus programas.

4. ¿Qué es Blazor?

Blazor es una tecnología dentro de .NET que permite crear interfaces web interactivas usando **C#** en lugar de JavaScript.

- **Ventaja:** Permite compartir código entre el servidor y el cliente (navegador) y usar un solo lenguaje para todo el desarrollo web.

5. ¿Qué es DbContext?

El **DbContext** es la clase principal de **Entity Framework** (el ORM de .NET).

- **Función:** Actúa como un puente entre tu código C# y la base de datos. Es responsable de abrir la conexión, consultar datos y guardar cambios (insertar, actualizar o borrar registros).

6. ¿Qué es CORS?

CORS (Intercambio de Recursos de Origen Cruzado) es una medida de **seguridad del navegador**.

- **Función:** Por defecto, los navegadores bloquean peticiones entre sitios diferentes (ej: tu frontend está en `localhost:3000` y tu API en `localhost:5000`). Configurar CORS permite decirle al navegador: "Es seguro que este sitio A pida datos a este sitio B".
