**Backend app creation:**

**1. Crea la solución y el proyecto principal (API)**

**# Crea la solución**

dotnet new sln -n PruebaBackend

**# Crea el proyecto Web API**

dotnet new webapi -n PruebaBackend.Api

**# Agrega el proyecto a la solución**

dotnet sln add .\PruebaBackend.Api\PruebaBackend.Api.csproj

**2. Crea el proyecto de pruebas**

**# Crea el proyecto de pruebas XUnit**

dotnet new xunit -n PruebaBackend.Tests

**# Agrega el proyecto de pruebas a la solución**

dotnet sln add .\PruebaBackend.Tests\PruebaBackend.Tests.csproj

**# Agrega referencia del proyecto de pruebas al proyecto principal**

dotnet add .\PruebaBackend.Tests\PruebaBackend.Tests.csproj reference .\PruebaBackend.Api\PruebaBackend.Api.csproj

**3. Ahora vamos con el siguiente paso: agregar Entity Framework Core y el proveedor de PostgreSQL a tu proyecto principal ([PruebaBackend.Api](vscode-file://vscode-app/c:/Users/mario/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ")).**

**1. Instala los paquetes necesarios**

dotnet add .\PruebaBackend.Api\PruebaBackend.Api.csproj package Microsoft.EntityFrameworkCore

dotnet add .\PruebaBackend.Api\PruebaBackend.Api.csproj package Microsoft.EntityFrameworkCore.Design

dotnet add .\PruebaBackend.Api\PruebaBackend.Api.csproj package Npgsql.EntityFrameworkCore.PostgreSQL

**(Opcional) Instala la herramienta de migraciones globalmente**

Esto te permitirá crear migraciones fácilmente:

dotnet tool install --global dotnet-ef

**4. Ahora vamos a crear el modelo y el DbContext para Entity Framework.**

**Crea la carpeta Models y el modelo MarcaAuto**

**Crea el archivo MarcaAuto.cs dentro de esa carpeta con este contenido:**

namespace PruebaBackend.Api.Models

{

    /// <summary>

    /// Representa una marca de auto.

    /// </summary>

    public class MarcaAuto

    {

        public int Id { get; set; }

        public string Nombre { get; set; } = string.Empty;

    }

}

**5. Crea la carpeta Data y el DbContext**

**Luego, crea el archivo ApplicationDbContext.cs dentro de esa carpeta con este contenido:**

using Microsoft.EntityFrameworkCore;

using PruebaBackend.Api.Models;

namespace PruebaBackend.Api.Data

{

    /// <summary>

    /// DbContext principal para la aplicación.

    /// </summary>

    public class ApplicationDbContext : DbContext

    {

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

            : base(options)

        {

        }

        public DbSet<MarcaAuto> MarcasAutos { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            base.OnModelCreating(modelBuilder);

            // Seed de datos iniciales

            modelBuilder.Entity<MarcaAuto>().HasData(

                new MarcaAuto { Id = 1, Nombre = "Toyota" },

                new MarcaAuto { Id = 2, Nombre = "Ford" },

                new MarcaAuto { Id = 3, Nombre = "Volkswagen" }

            );

        }

    }

}

**Ahora vamos a configurar la conexión a PostgreSQL y a registrar el `AppDbContext` en tu proyecto.**

Abre el archivo [appsettings.json](vscode-file://vscode-app/c:/Users/mario/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") y agrega una sección como esta:

{

  "ConnectionStrings": {

    "DefaultConnection": "Host=localhost;Port=5432;Database=pruebabackenddb;Username=postgres;Password=postgres"

  },

  // ...existing code...

}

**Registra el DbContext en Program.cs**

Abre [Program.cs](vscode-file://vscode-app/c:/Users/mario/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") y agrega el registro del DbContext usando PostgreSQL:

using Microsoft.EntityFrameworkCore;

using PruebaBackend.Api.Data;

var builder = WebApplication.CreateBuilder(args);

// Agrega el DbContext y la cadena de conexión

builder.Services.AddDbContext<ApplicationDbContext>(options =>

    options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));

// ...existing code...

var app = builder.Build();

// ...existing code...

Ya tienes configurado el DbContext y la conexión a PostgreSQL.  
Ahora toca crear la migración y aplicar la base de datos.

**1. Crea la migración inicial**

En la terminal, ejecuta:

dotnet ef migrations add InitialCreate --project .\PruebaBackend.Api\ --startup-project .\PruebaBackend.Api\

**2. Aplica la migración para crear la base de datos**

dotnet ef database update --project .\PruebaBackend.Api\ --startup-project .\PruebaBackend.Api\

**Ahora vamos a crear el controlador para exponer el endpoint que devuelve todas las marcas de autos.**

### 1. Crea la carpeta `Controllers` (si no existe)

### 2. Crea el archivo `MarcasAutosController.cs` con el siguiente contenido:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using PruebaBackend.Api.Data;

using PruebaBackend.Api.Models;

namespace PruebaBackend.Api.Controllers

{

[ApiController]

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

public class MarcasAutosController : ControllerBase

{

private readonly AppDbContext \_context;

public MarcasAutosController(AppDbContext context)

{

\_context = context;

}

/// <summary>

/// Obtiene todas las marcas de autos.

/// </summary>

[HttpGet]

public async Task<ActionResult<IEnumerable<MarcaAuto>>> GetMarcasAutos()

{

var marcas = await \_context.MarcasAutos.ToListAsync();

return Ok(marcas);

}

}

}

[HttpPost]

public async Task<ActionResult<MarcaAuto>> PostMarcaAuto(MarcaAuto marca)

{

if (string.IsNullOrWhiteSpace(marca.Nombre))

return BadRequest("El nombre es obligatorio.");

\_context.MarcasAutos.Add(marca);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetMarcasAutos), new { id = marca.Id }, marca);

}

/// <summary>

/// Actualiza una marca de auto existente.

/// </summary>

[HttpPut("{id}")]

public async Task<IActionResult> PutMarcaAuto(int id, MarcaAuto marca)

{

if (id != marca.Id)

return BadRequest("El id no coincide.");

var existe = await \_context.MarcasAutos.AnyAsync(m => m.Id == id);

if (!existe)

return NotFound();

\_context.Entry(marca).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Elimina una marca de auto.

/// </summary>

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteMarcaAuto(int id)

{

var marca = await \_context.MarcasAutos.FindAsync(id);

if (marca == null)

return NotFound();

\_context.MarcasAutos.Remove(marca);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

Dotnet run para corer la aplicación : <https://localhost:5001/swagger> or <https://localhost:5001/api/marcaauto>

**Unit tests**

Dentro de unittests.cs

using Xunit;

using PruebaBackend.Api.Controllers;

using PruebaBackend.Api.Data;

using PruebaBackend.Api.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

using System.Collections.Generic;

using System.Linq;

namespace PruebaBackend.Tests

{

    public class MarcasAutosControllerTests

    {

        private AppDbContext GetDbContext()

        {

            var options = new DbContextOptionsBuilder<AppDbContext>()

                .UseInMemoryDatabase(databaseName: "TestDb")

                .Options;

            var context = new AppDbContext(options);

            // Seed data

            context.MarcasAutos.AddRange(

                new MarcaAuto { Id = 1, Nombre = "Toyota" },

                new MarcaAuto { Id = 2, Nombre = "Ford" },

                new MarcaAuto { Id = 3, Nombre = "Volkswagen" }

            );

            context.SaveChanges();

            return context;

        }

* The goal is to **verify that the GetMarcasAutos endpoint returns all the expected car brands** (3 seeded entries in this case).
*  Uses **InMemoryDatabase** to create a test database in memory — no need for a real SQL Server.
*  Seeds 3 MarcaAuto entries to simulate existing data.
*  Returns the prepared AppDbContext for testing.

        [Fact]

        public async Task GetMarcasAutos\_ReturnsAllMarcas()

        {

            // Arrange

            var context = GetDbContext();

            var controller = new MarcasAutosController(context);

            // Act

            var result = await controller.GetMarcasAutos();

            // Assert

            var okResult = Assert.IsType<OkObjectResult>(result.Result);

            var marcas = Assert.IsAssignableFrom<IEnumerable<MarcaAuto>>(okResult.Value);

            Assert.Equal(3, marcas.Count());

        }

    }

}

 **Arrange**:

* Sets up the controller with the in-memory context.

 **Act**:

* Calls GetMarcasAutos() to retrieve the list.

 **Assert**:

* Verifies the result is an OkObjectResult (HTTP 200).
* Ensures the returned value is an IEnumerable<MarcaAuto>.
* Confirms that exactly **3 brands** are returned.

    public async Task GetMarcasAutos\_ReturnsEmptyList\_WhenNoData()

        {

            // Arrange

            var context = GetDbContext(withData: false);

            var controller = new MarcasAutosController(context);

            // Act

            var result = await controller.GetMarcasAutos();

            // Assert

            var okResult = Assert.IsType<OkObjectResult>(result.Result);

            var marcas = Assert.IsAssignableFrom<IEnumerable<MarcaAuto>>(okResult.Value);

            Assert.Empty(marcas);

        }

**Crear un archivo docker-compose.yml que configure dos servicios:  
PostgreSQL para la base de datos y otro para la API REST.  
o Asegurarse de que la API REST pueda conectarse correctamente a la base de  
datos PostgreSQL.**

Crear el archivo Docker-compose.yml

version: '3.8'

services:

  db:

    image: postgres:16

    environment:

      POSTGRES\_DB: pruebabackenddb

      POSTGRES\_USER: postgres

      POSTGRES\_PASSWORD: yournewsecurepassword

    ports:

      - "5432:5432"

    volumes:

      - postgres\_data:/var/lib/postgresql/data

  api:

    build:

      context: ./PruebaBackend.Api

    environment:

      ASPNETCORE\_ENVIRONMENT: Development

      ConnectionStrings\_\_DefaultConnection: Host=db;Port=5432;Database=pruebabackenddb;Username=postgres;Password=yournewsecurepassword

    ports:

      - "5000:80"

    depends\_on:

      - db

volumes:

  postgres\_data:

**Crea un Dockerfile para tu API (si no tienes uno)**

En la carpeta [PruebaBackend.Api](vscode-file://vscode-app/c:/Users/mario/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html):

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base

WORKDIR /app

EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build

WORKDIR /src

COPY . .

RUN dotnet restore

RUN dotnet publish -c Release -o /app/publish

FROM base AS final

WORKDIR /app

COPY --from=build /app/publish .

ENTRYPOINT ["dotnet", "PruebaBackend.Api.dll"]

**docker-compose up –build** y tener docker instalado y corriendo en tu computadora