

Step 2: Create the database context	8
Debería añadirle justo antes de “Logging” lo siguiente:	38
En el .yml file me recomienda poner User Id en vez de solo User, así lo hago:.....	38
PART1-Test connection from your host aka verify connection from your host machine.....	41
PART2-let's check that your backend container can reach your SQL Server container (db – service name-) over the internal Docker network. (using Sserver=db, siendo db el service name del container_name sqlserver) AKA Verify connection (to sql server container) from inside the backend container	45
Connected to sql server from inside its own container (not that backend can talk to it over docker network).....	48
Así que tambié he conseguido PART 2 – Verify connection (to sql server container) from inside the backend container.	51
Paso 3. es Create a simple booking model and database context (backend), esto lo hacemos inside our ASP.NET Core Web API project.	51
Step 4: Creating your first API endpoints in your ASP.NET Core backend (so you can read/write data from/to SQL Server).....	52
Mi ReservasController.cs empieza así.	54
CRUD – HTTP Methods correspondencia	56
Make sure AppDbContext is registered, cosa que se comprueba en Program.cs de mi backend	58
You now need to replace that Program.cs default code so your app can	60
Connecting to my SQL Server container in DBeaver pre-running migrations con comandos dotnet	63
Time to run migrations with dotnet to create my ReservasDb	68
Exito en ejecutar con windows cmd estando en ReservasApi folder el dotnet build	70
Ahora (post ejecutar los dos comandos dotnet ef creadores de la bd) en Dbeaver refresco Databases y exitósamente aparece la nueva bd con sus tablas	75
Primero voy a testear endpoints vía dotnet run osease con backend corriendo en local... ..	76
Mi Swagger con backend corriendo en local osease corriendo con dotnet run	77
A continuación, toca probar algunos endpoints con backend corriendo en su docker container.....	95

Entonces doy por hecho el haber testeado algunos endpoints con swagger y primero con backend corriendo en local con dotnet run y después (lo que acabamos de hacer) con backend corriendo en su docker container..... 111

Semana 2 goals..... 112

Índice de contenidos pendiente de mejora.

Documentación pendiente de ser resumida.

Let's walk through the process of creating my first backend and frontend endpoints step by step so it is clear what an endpoint is, how to build one in ASP.NET Core (backend), and how to connect to it from Angular (frontend).

In web dev, an **endpoint** is a URL in your backend
that the frontend (or another service) can call to send or receive data.

Por ejemplo:

GET <https://api.yourapp.com/api/bookings> might return a list of all bookings (entiendo que esto sería receive data).

Think of it (of an endpoint) as a door into your backend:

Your Angular frontend knocks on the door (**HTTP request**),
and the backend opens it and replies (**HTTP response**).

First let's **create a model file in my ASP.NET Core Web API project** (que es mi backend project). Para esto lo primero es tener claro where (will the .cs file be stored) and how (con qué ide o code editor) to create that model file.

A **model** in ASP.NET Core **represents the shape of your data** –
– (the model is) basically a C# class that matches
what you'll store in the database and (what you'll) send/receive through your API.

Ese "what" viene siendo la data.

Para crear este modelo, emplearemos preferiblemente VS 2022 Community Edition o VS Code.

Según Chatgpt, ReservasApi is my “actual Web API project folder”.

... Usuarios > iago.cadavidgonzalez > Proyectos > reservas-app > backend > ReservasApi >

Ordenar Ver ...

Nombre	Fecha de modificación	Tipo	Tamaño
bin	03/11/2025 10:28	Carpeta de archivos	
obj	03/11/2025 10:28	Carpeta de archivos	
Properties	03/11/2025 10:10	Carpeta de archivos	
appsettings.Development.json	03/11/2025 10:10	Archivo de origen JSON	1 KB
appsettings.json	03/11/2025 10:10	Archivo de origen JSON	1 KB
Program.cs	03/11/2025 10:10	Archivo de origen C#	2 KB
ReservasApi.csproj	03/11/2025 10:10	Archivo de origen C# Project	1 KB
ReservasApi.http	03/11/2025 10:10	Archivo HTTP	1 KB

F

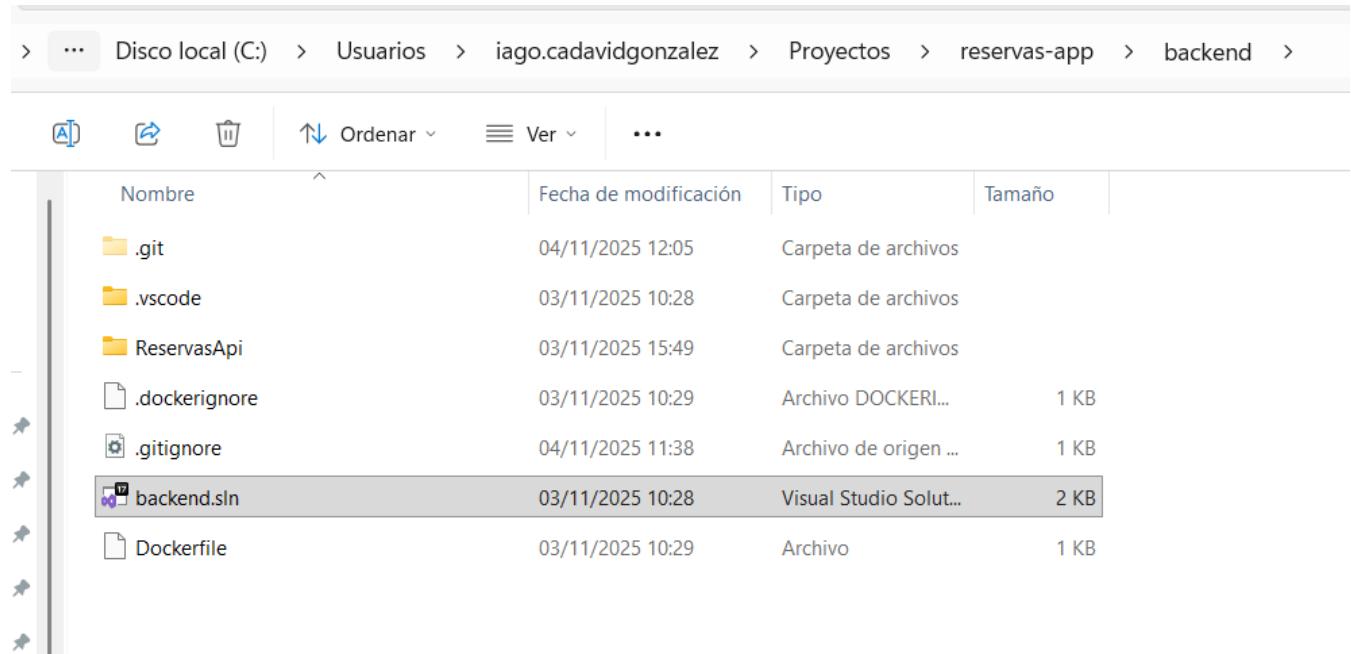
En otras palabras “the code that runs lives inside ReservasApi folder.

Where to create my model file (que Chatgpt inicialmente llamó Bookings.cs):

Inside ReservasApi folder create a new folder named Models y aquí dentro creo el Booking.cs file.

Si voy a hacerlo con VS IDE, how exactly should I open the backend project?

Since you already have a .sln file (backend.sln)



Nombre	Fecha de modificación	Tipo	Tamaño
.git	04/11/2025 12:05	Carpeta de archivos	
.vscode	03/11/2025 10:28	Carpeta de archivos	
ReservasApi	03/11/2025 15:49	Carpeta de archivos	
.dockerignore	03/11/2025 10:29	Archivo DOCKERI...	1 KB
.gitignore	04/11/2025 11:38	Archivo de origen ...	1 KB
backend.sln	03/11/2025 10:28	Visual Studio Solut...	2 KB
Dockerfile	03/11/2025 10:29	Archivo	1 KB

That includes your ReservasApi project (I hope it does), **you'll want to open that solution** so Visual Studio can properly manage and build everything for you. To do this:

En VS IDE start window, click “**Open a project or solution**” i.e. “**abre un archivo .sln o proyecto de Visual Studio local**”. Y lo que abro es el .sln file hijo de ‘backend’ folder.

Resultado de abrirlo: NEXT PAGE

The screenshot shows the Visual Studio IDE interface. The top menu bar includes Archivo, Editar, Ver, Git, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda, and a search bar. Below the menu is a toolbar with various icons. The main area has tabs for Debug, Any CPU, https://, and ReservasApi. The Output window (Salida) at the bottom left shows numerous errors related to missing language attributes in ASP.NET snippets. The Solution Explorer window on the right shows a single project named "backend" with one item: "ReservasApi".

```

Salida
Mostrar salida de: Resultado del fragmento de código
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\checkbox.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\detailsview.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\dynamiccontrol.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\dynamicdatamanager.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\form.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\gridview.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\hyperlink.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\image.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\label.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\listbox.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\listitem.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\listview.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\loginname.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\logonstatus.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\loginview.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\multiview.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\panel.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\placeholder.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\radiobutton.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\register.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\registeraspx.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\repeater.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\requiredfieldvalidator.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\runat.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\scriptreference.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\scripts.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\servicereference.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\sitemappath.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\sm.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\smpl.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\sqldatasource.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\textbox.snippet: Falta o no se especificó el atributo de lenguaje
C:\Program Files\Microsoft Visual Studio\2022\Community\Web\Snippets\HTML\3082\ASP.NET\updatepanel.snippet: Falta o no se especificó el atributo de lenguaje

```

“Salida” pone mil veces “Falta o no se especificó el atributo de lenguaje”.

The screenshot shows the Windows File Explorer interface. The address bar shows the path: Proyectos > reservas-app > backend > ReservasApi >. The main pane displays the contents of the ReservasApi folder, which includes a bin folder, an obj folder, a Properties folder, an appsettings.Development.json file, an appsettings.json file, a Program.cs file, a ReservasApi.csproj file, a ReservasApi.csproj.user file, and a ReservasApi.http file. The files are listed in descending order of modification date.

Nombre	Fecha de modificación	Tipo	Tamaño
bin	03/11/2025 10:28	Carpeta de archivos	
obj	03/11/2025 10:28	Carpeta de archivos	
Properties	03/11/2025 10:10	Carpeta de archivos	
appsettings.Development.json	03/11/2025 10:10	Archivo de origen JSON	1 KB
appsettings.json	03/11/2025 10:10	Archivo de origen JSON	1 KB
Program.cs	03/11/2025 10:10	Archivo de origen C#	2 KB
ReservasApi.csproj	03/11/2025 10:10	Archivo de origen C# Project	1 KB
ReservasApi.csproj.user	05/11/2025 8:35	Per-User Project Options File	1 KB
ReservasApi.http	03/11/2025 10:10	Archivo HTTP	1 KB

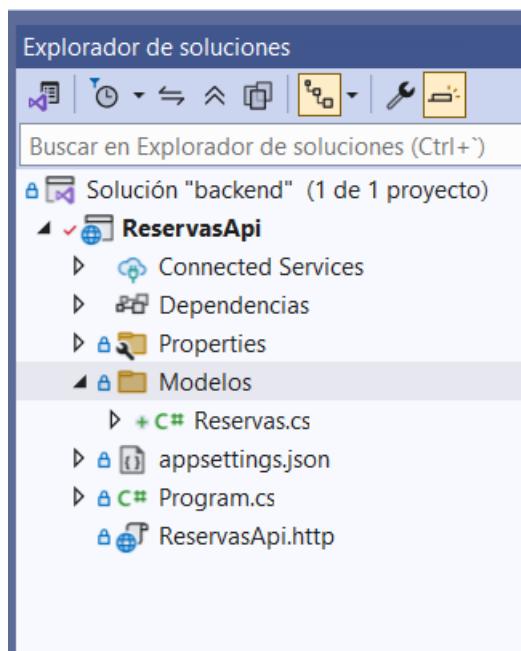
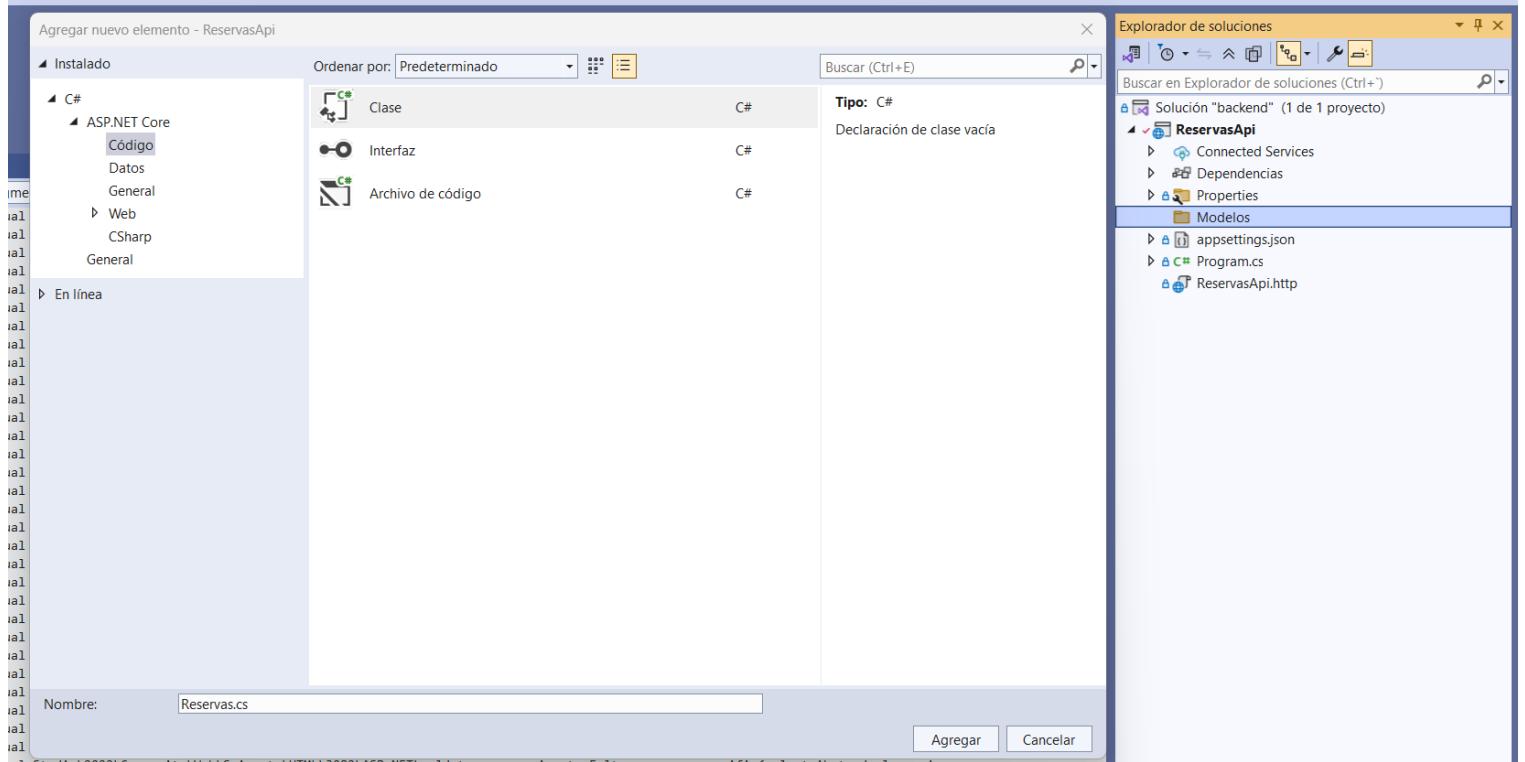
Nótese que en Solution explorer de VS IDE no aparece ni bin ni obj folders. Tampoco aparece appsettings.Development.json, pero sí appsettings.json, tmb aparece Program.cs, pero no ReservasApi.csproj ni el csproj.user. Finalmente, sí aparece ReservasApi.http.

Y a mayores de lo que veo en windows explorer, en solution explorer de vs ide aparecen Connected Services y Dependencias, ambos son desplegables.

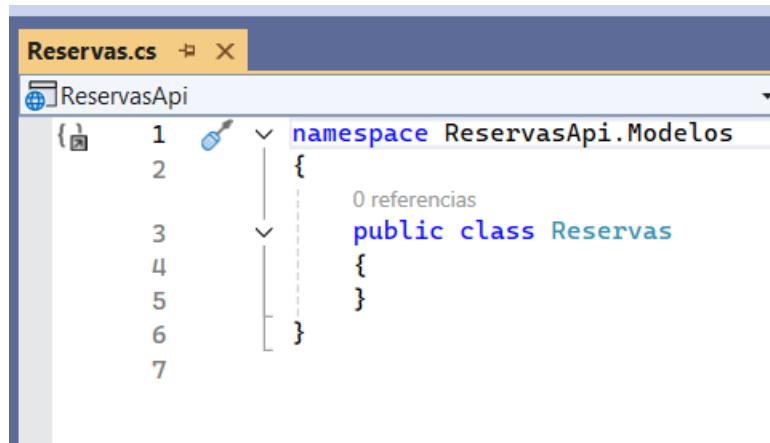
En 1io, el ver esas cosas confirma que VSIDE recognizes my ReservasApi (ASP.NET Core) Web API project.

Ahora en el Solution Explorer de vside hago clic derecho en **ReservasApi** project (el que tiene un globe icon), doy en **Add->New Folder->name it Models**.

Y ahora clic derecho en esa nueva carpeta + **Add + Class** y name it **Reservas.cs** y clica Add.

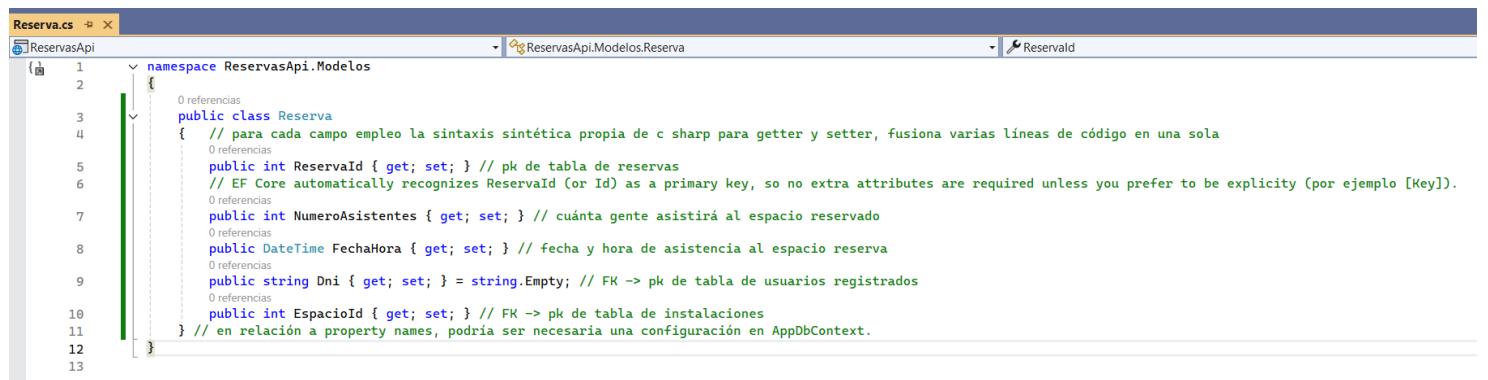


VSIDE will create the file and open it for you with some code:



```
Reservas.cs + X
ReservasApi
namespace ReservasApi.Modelos
{
    public class Reservas
}
```

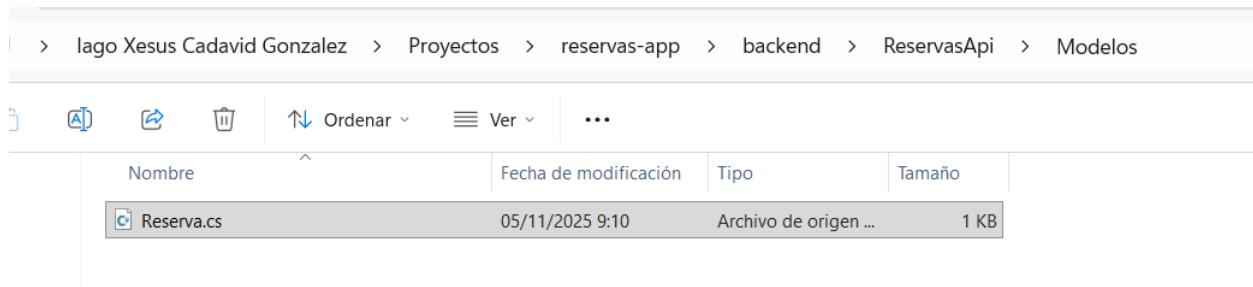
Replace the contents with (cambié Reservas a Reserva):



```
Reserva.cs + X
ReservasApi
namespace ReservasApi.Modelos
{
    public class Reserva
    {
        // para cada campo empleo la sintaxis sintética propia de c sharp para getter y setter, fusiona varias líneas de código en una sola
        public int ReservaId { get; set; } // pk de tabla de reservas
        // EF Core automatically recognizes ReservaId (or Id) as a primary key, so no extra attributes are required unless you prefer to be explicitly (por ejemplo [Key]).
        public int NumeroAsistentes { get; set; } // cuánta gente asistirá al espacio reservado
        public DateTime FechaHora { get; set; } // fecha y hora de asistencia al espacio reserva
        public string Dni { get; set; } = string.Empty; // FK -> pk de tabla de usuarios registrados
        public int EspacioId { get; set; } // FK -> pk de tabla de instalaciones
    }
}
```

Ese código está adaptado a mi diseño bd.

Recordatorio de dónde tengo ese .cs file:



Iago Xesus Cadavid Gonzalez > Proyectos > reservas-app > backend > ReservasApi > Modelos

Nombre	Fecha de modificación	Tipo	Tamaño
Reserva.cs	05/11/2025 9:10	Archivo de origen ...	1 KB

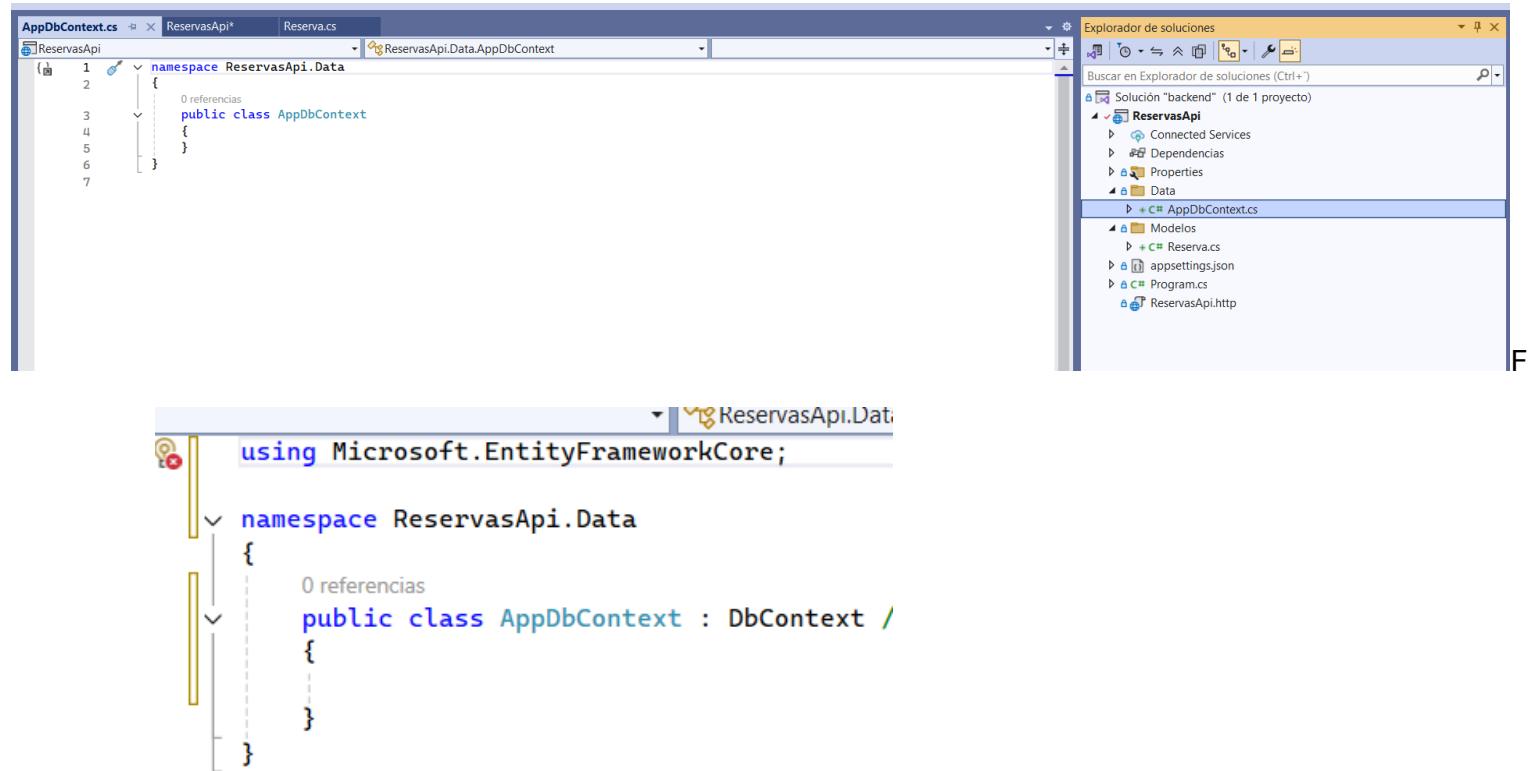
(NEXT PAGE)

Step 2: Create the database context

File path: ReservasApi/Data/AppDbContext.cs (**create a new folder named Data inside your ReservasApi project**, just like you did for Models)

[No, you don't need to install sql server locally to use EFC (because) using the sql server docker image is completely sufficient –EFC just needs a **connection string** that points to a running **SQL Server instance.**]

Dentro de Data folder creo ApplicationDbContext.cs, que viene con este código por defecto:



```
AppDbContext.cs  ReservasApi*  Reserva.cs
ReservasApi
1  namespace ReservasApi.Data
2  {
3      0 referencias
4      public class AppDbContext
5      {
6      }
7 }

Explorador de soluciones
ReservasApi
    Solución "backend" (1 de 1 proyecto)
        ReservasApi
            Connected Services
            Dependencias
            Properties
            Data
                + C# AppDbContext.cs
            Modelos
                + C# Reserva.cs
                appsettings.json
            Program.cs
            ReservasApi.http

ReservasApi.Data
    + C# AppDbContext.cs

F
```

```
ReservasApi.Data
using Microsoft.EntityFrameworkCore;

namespace ReservasApi.Data
{
    0 referencias
    public class AppDbContext : DbContext
    {
    }
}
```

Cuando escribo using Microsoft.EntityFrameworkCore y DbContext, there is no intellisense, según chatgpt porque “EF Core hasn't been added to my project yet, VSIDE is telling you that there is not reference to that library in your project, that's because EFC is an external library, not part of .NET by default, so you need to install it via NuGet”,

para ello en Solution Explorer de VSIDE hago clic derecho en mi project name ReservasApi y elijo Manage NuGet Packages, doy en Browse tab y busco e instalo estas tres:

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

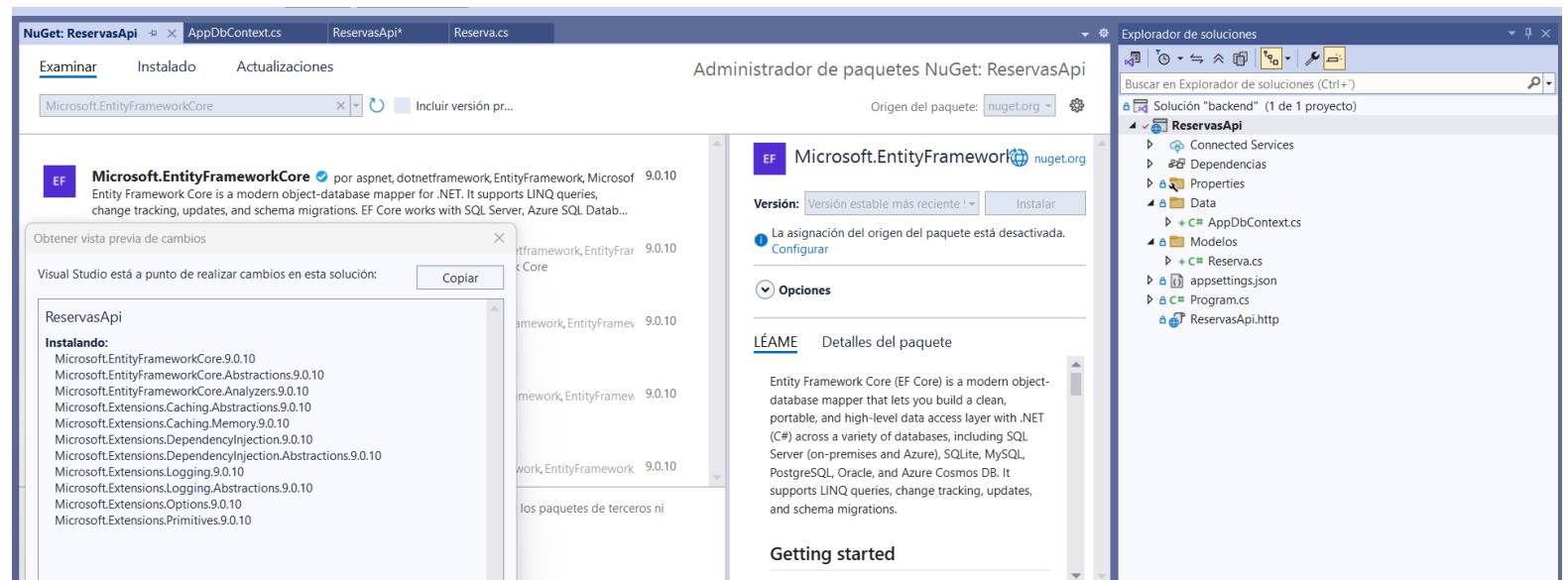
Microsoft.EntityFrameworkCore.Tools

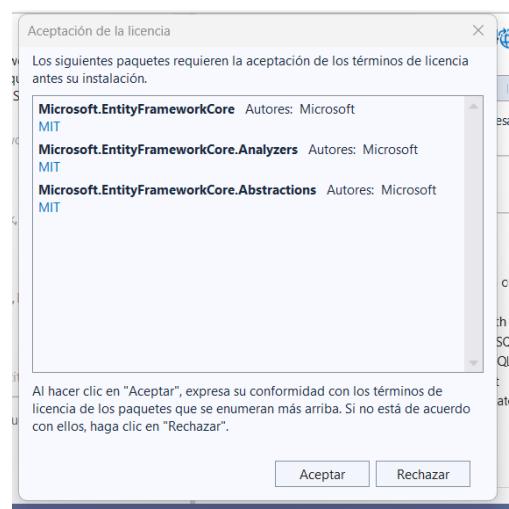
You should now see them listed under the Installed tab.

To confirm installation, open your ReservasApi.csproj file and make sure you see entries like:

```
<Item Group>
<PackageReference instalacion1/>
<PackageReference instalacion2/>
<PackageReference instalacion3/>
</Item Group>
```

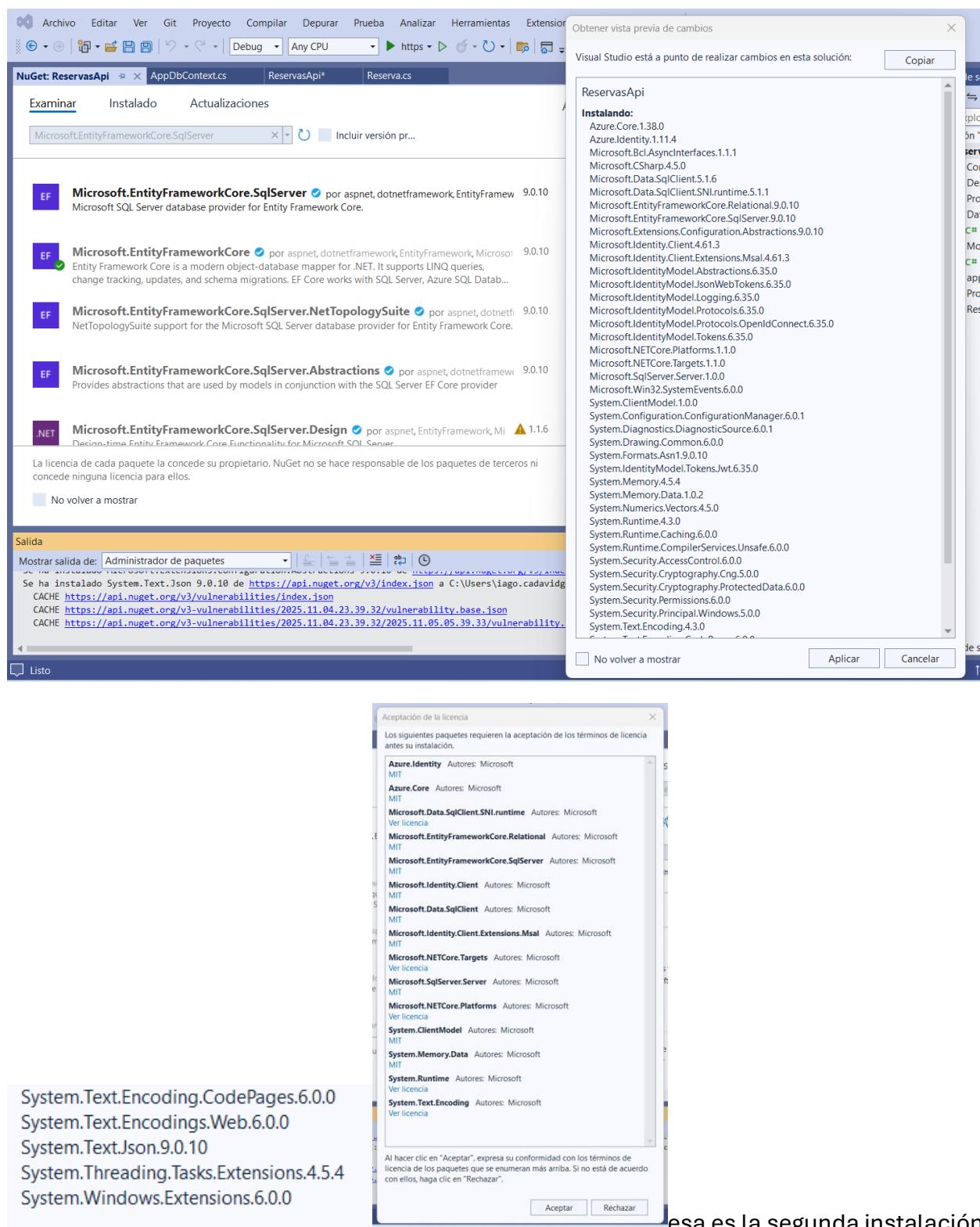
Los instalo:

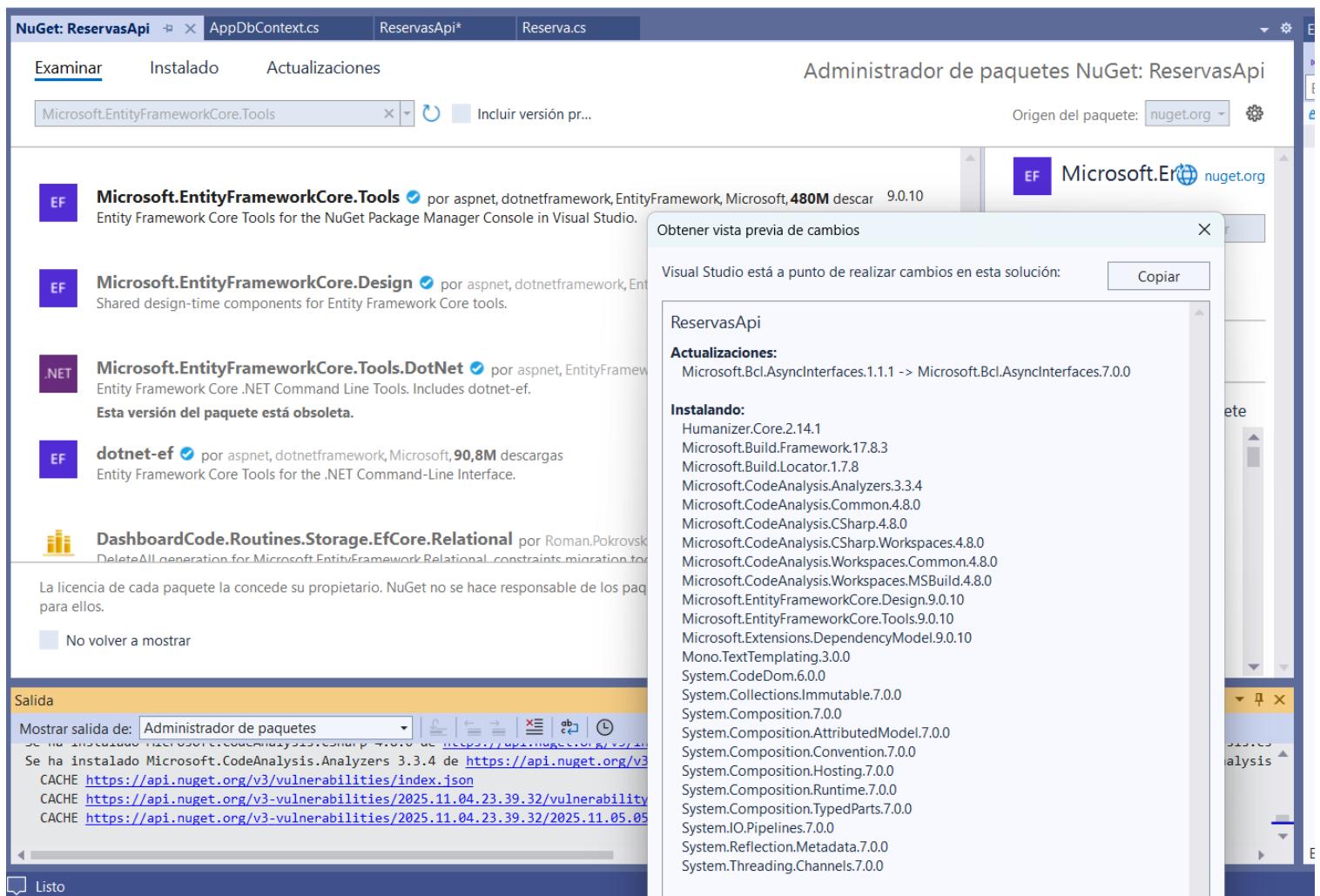




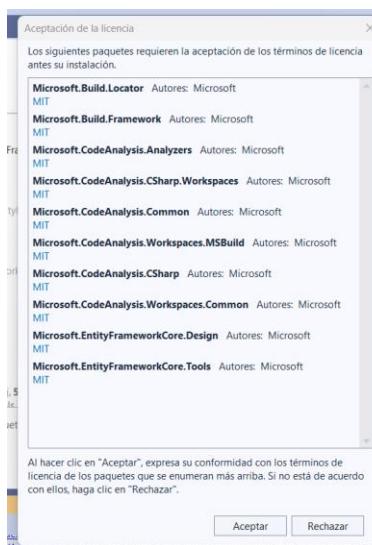
Esa es la primera instalación, que es la **core EF library**, which gives you the essential building blocks, including DbContext base class and DbSet<T> class, among other things.

2nda instalación:





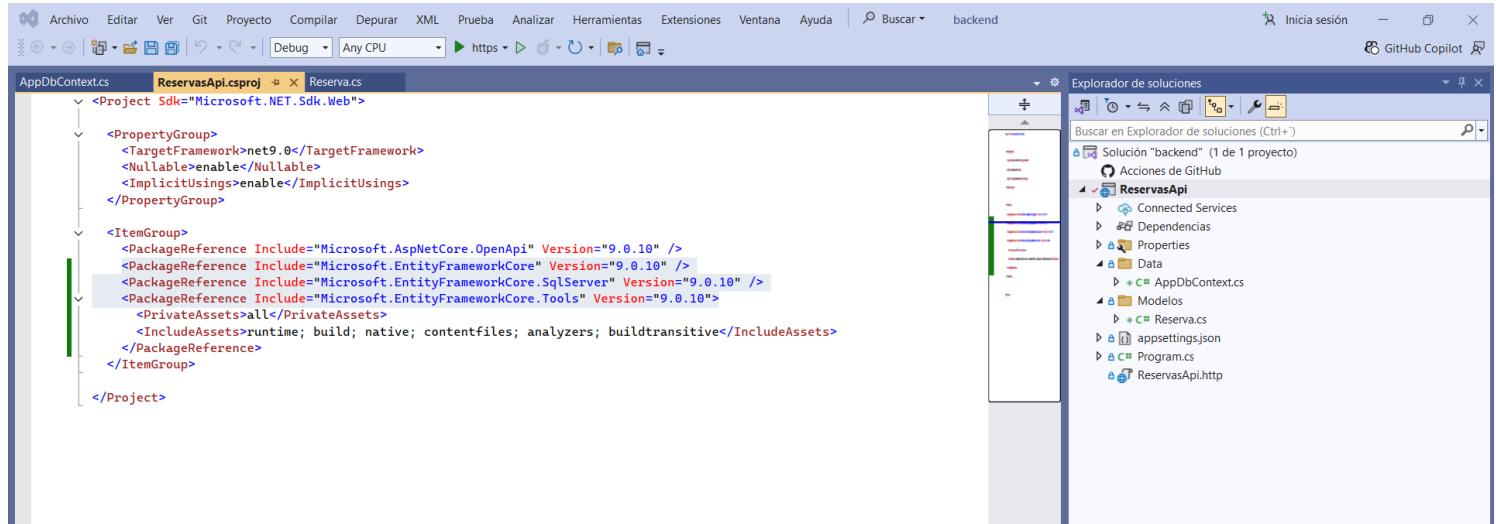
Y la tercera instalación (encima y debajo):



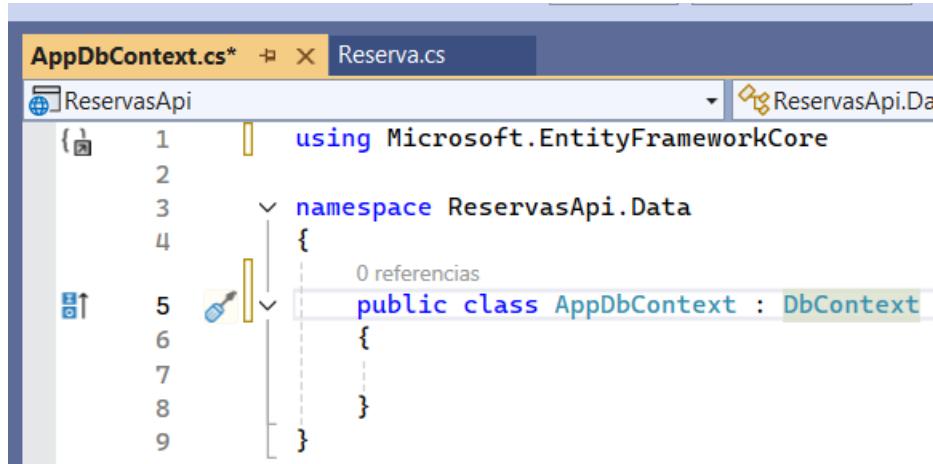
Hasta ahí 3era instalación de EFC pa que lo reconozca intellisense.

After installing those 3 packages, restart and verify IntelliSense, save all files, close and reopen VSIDE (or rebuild the project). Now when you type using Microsoft.EntityFrameworkCore; you should see IntelliSense suggestions (y al escribir DbContext tmb).

Hago el reopen vside y ya aparezco aquí:



El intellisense reconoce el using Microsoft.EFC y el DbContext, pero para que reconozca este último primero tuve que poner el using.

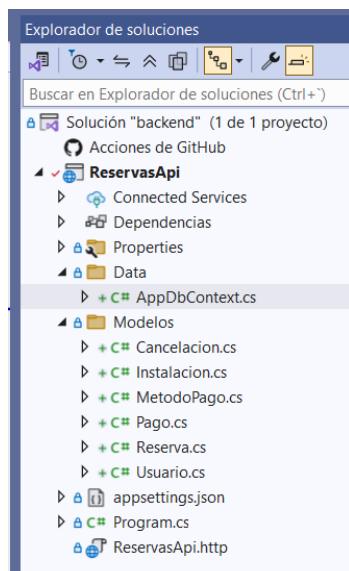


(página siguiente)

The screenshot shows a code editor window with the title bar "AppDbContext.cs". The code is written in C# and defines a DbContext class named "AppDbContext". The code includes using statements for Microsoft.EntityFrameworkCore and ReservasApi.Modelos, and a namespace declaration for ReservasApi.Data. The class has a constructor that takes an DbContextOptions<AppDbContext> parameter and calls base(options). It contains properties for DbSet<Reserva>, DbSet<Usuario>, DbSet<Instalacion>, DbSet<Pago>, and DbSet<Cancelacion>. The code editor interface includes a status bar at the bottom.

```
1  using Microsoft.EntityFrameworkCore;
2  using ReservasApi.Modelos;
3
4
5  namespace ReservasApi.Data
6  {
7      public class AppDbContext : DbContext // represents the bridge between the database and the application
8      {
9          public AppDbContext(DbContextOptions<AppDbContext> options)
10             : base(options) { }
11
12         // Cada DbSet se corresponde con una model class (definida en el archivo .cs)
13         // El primer nombre (el que está entre <>) es el nombre de la tabla en la base de datos
14         public DbSet<Reserva> Reservas { get; set; } // Reserva es la clase que define la tabla
15
16         public DbSet<Usuario> Usuarios { get; set; }
17
18         public DbSet<Instalacion> Instalaciones { get; set; }
19
20         public DbSet<Pago> Pagos { get; set; }
21
22         public DbSet<Cancelacion> Cancelaciones { get; set; }
23     }
24 }
```

Lo que va entre <> en captura encima desta línea son los Modelos:




```

namespace ReservasApi.Modelos
{
    5 referencias
    public class Reserva
    {   // para cada campo empleo la sintaxis sintética propia de c sharp para getter y setter, fusiona varias líneas de código en una sola
        0 referencias
        public int ReservaId { get; set; } // pk de tabla de reservas
        // EF Core automatically recognizes ReservaId (or Id) as a primary key, so no extra attributes are required unless you prefer to be expl
        0 referencias
        public int NumeroAsistentes { get; set; } // cuánta gente asistirá al espacio reservado
        0 referencias
        public DateTime FechaHora { get; set; } // fecha y hora de asistencia al espacio reserva
        0 referencias
        public string Dni { get; set; } = string.Empty; // FK -> pk de tabla de usuarios registrados
        0 referencias
        public int EspacioId { get; set; } // FK -> pk de tabla de instalaciones
        // en relación a property names, podría ser necesaria una configuración en AppDbContext.
    }
}

```

```

namespace ReservasApi.Modelos
{
    1 referencia
    public class Usuario
    {
        0 referencias
        public string UsuarioId { get; set; } = string.Empty; // el dni es la primary key referenciada por fk en tabla de reservas
        0 referencias
        public string NombreCompleto { get; set; } = string.Empty;
        0 referencias
        public string Email { get; set; } = string.Empty;
        0 referencias
        public string Telefono { get; set; } = string.Empty;
        0 referencias
        public string Domicilio { get; set; } = string.Empty;
        // fechas:
        0 referencias
        public DateTime FechaNacimiento { get; set; }
        0 referencias
        public DateTime FechaRegistro { get; set; }
        // contraseña:
        0 referencias
        public string Contrasinal { get; set; } = string.Empty;

        // propiedad de navegación: un usuario registrado puede haber realizado de cero a varias reservas (0,N)
        0 referencias
        public ICollection<Reserva>? Reservas { get; set; } // la propiedad es nullable osease puede tener un valor null -> a user ca
    }
}

```

```

namespace ReservasApi.Modelos
{
    0 referencias
    public class Instalacion // cada objeto será o bien un espacio deportivo o bien una sala de estudio.
    {
        0 referencias
        public int InstalacionId { get; set; } // pk referenciada por una fk de tabla de reservas
        0 referencias
        public string Localizacion { get; set; } = string.Empty;
        0 referencias
        public int CapacidadMaxima { get; set; }
        0 referencias
        public double PrecioHora { get; set; }
        0 referencias
        public bool Tipo { get; set; } // true será espacio deportivo y false será sala de estudio.

        // Propiedad de navegación: Una instalación puede estar reservada de cero a varias veces:
        0 referencias
        public ICollection<Reserva>? Reservas { get; set; } // cardinalidad mínima y máxima de entidad Instalacion respecto a Reserva es (0,N)
    }
}

```



```
// DateTime viene de un using System que parece ser que no tengo necesidad de poner

namespace ReservasApi.Modelos
{
    1 referencia
    public class Pago
    {
        0 referencias
        public int PagoId { get; set; } // clave primaria de la tabla de pagos
        0 referencias
        public int ReservaId { get; set; } // fk -> pk de tabla de reservas
        0 referencias
        public double Monto { get; set; } // cuánto paga el usuario en total por la reserva
        0 referencias
        public MetodoPago Metodo { get; set; } // MetodoPago es una ENUM class donde cada instancia es un método de pago (por ejemplo CONTARJETA).
        0 referencias
        public DateTime FechaHora { get; set; } // fecha y hora del pago (¿coincidirá con la fecha y hora en que se realizó la solicitud de reserva?).

        // Propiedad de navegación: cada pago siempre paga una sola reserva, ni más ni menos osease (1,1) es la cardinalidad mínima y máxima de la entidad Pago
        0 referencias
        public Reserva? Reserva { get; set; }
    }
}
```

```
namespace ReservasApi.Modelos
{
    1 referencia
    public enum MetodoPago
    {
        Tarjeta, // sea de crédito o de débito
        Transferencia, // bancaria
        Otro // por ejemplo Bizum o PayPal. Podríamos desglosar esta última instancia en varias.
    }
}
```

```
// DateTime viene de un using System que parece ser que no tengo necesidad de poner

namespace ReservasApi.Modelos
{
    1 referencia
    public class Cancelacion
    {
        0 referencias
        public int CancelacionId { get; set; } // clave primaria de tabla de cancelaciones
        0 referencias
        public int ReservaId { get; set; } // fk -> pk de tabla de reservas
        0 referencias
        public DateTime Fecha { get; set; } // fecha de la cancelación de la reserva
        0 referencias
        public string Motivo { get; set; } = string.Empty; // motivo o razón de la cancelación, podría ser un ENUM si se me ocurren las opciones posibles.

        // Propiedad de navegación: cada cancelación siempre cancela una sola reserva, ni más ni menos (1,1):
        0 referencias
        public Reserva? Reserva { get; set; }
    }
}
```

Teniendo eso, me dice chatgpt “you’ve done a lot of the foundational backend work already, so now it is the perfect moment to move towards those **first working API endpoints** (that your Angular frontend will soon call)”.

Siéndole más concreto a chatgpt con lo que tengo (reservas-app root-of-it-all folder y dentro frontend folder con lo básico de un proyecto Angular, backend folder con lo básico de un ASP.NET Core Web API project, una 3era carpeta para el servicio cron –esta está todavía vacía- y el docker-compose.yml file que entre otras cosas tiene db: image: .../server:2022-latest que es mi “database image, the official SQL Server image”), me dice “you’ve got a clean **Docker-based full-stack setup**”.

Dice “let’s now make sure everything connects properly –because this is the key foundation before your first endpoint works end-to-end.”

Según chatgpt, el **.yml file** “does all the hard wiring for you:

- **Starts sql server** (docker image)
- **Starts** your asp.net core **backend**
- **Starts** your angular **frontend**
- **Connects them** through a shared Docker network”

Assuming your docker-compose.yml is good, here is exactly what you should do in order to **continue your journey toward having your first working backend+frontend endpoint**.

Lo de los endpoints tiene que ver con “creating your first end-to-end flow (frontend->backend->database) for your booking app”. “You are right at the point where everything starts coming together – **your models, database and API (backend) will soon connect to your Angular frontend**” En 7 pasos:

steps to create my first frontend-backend endpoints for a booking app where the frontend is developed with Angular, the backend is developed as an ASP.NET Core Web API project and the database is a Docker Hub image, the official one of SQL Server. Take into account that I already created the frontend and backend projects and I also have a docker-compose.yml file, I have three containers in Docker Desktop, one for the frontend, another for backend and another for the database. And in relation to all this I also have a Docker Desktop Volume.



Perfect — you’re in a great place to start wiring things up!

Let’s walk step-by-step through creating your **first end-to-end flow** (frontend → backend → database) for your **booking app**.

We’ll go through this in 7 steps:



F

1. Confirm your setup i.e. confirm you already have the following:
 - a. Angular frontend (fex running on port 4200)

Docker Desktop interface showing the 'Containers' list. The table includes columns for Name, Container ID, Image, Port(s), CPU (%), Last started, and Actions. The 'frontend-1' container is highlighted.

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	vibrant_carson	5f1347c184de	hello-world	-	N/A	21 days ago	... ⋮ trash
<input type="checkbox"/>	reservas-app	-	-	-	N/A	21 hours ago	... ⋮ trash
<input type="checkbox"/>	frontend-1	3b4ebf8eb7d9	reservas-app-frontend	4200:80	N/A	21 hours ago	... ⋮ trash
<input type="checkbox"/>	backend-1	7b0568d2e9de	reservas-app-backend	5000:8080	N/A	21 hours ago	... ⋮ trash
<input type="checkbox"/>	sqlserver	ef940a47d551	mssql/server:2022-latest	1433:1433	N/A	21 hours ago	... ⋮ trash

Vemos en esa captura que en **Port(s)** del **frontend-1** pone **4200:80**, lo cual significa que

Port 80 inside the container is being mapped to port 4200 on my host machine.

So **inside Docker (the internal network)**, my frontend container listens on port 80.

Outside Docker (on my computer), I access it (el frontend) via <http://localhost:4200>

Docker Desktop interface showing the 'Containers' list. The table includes columns for Name, Container ID, Image, Port(s), CPU (%), Last started, and Actions. The 'frontend-1' container is highlighted.

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	reservas-app	-	-	-	N/A	0 seconds ago	... ⋮ trash
<input type="checkbox"/>	frontend-1	3b4ebf8eb7d9	reservas-app-frontend	4200:80	N/A	0 seconds ago	... ⋮ trash
<input type="checkbox"/>	backend-1	7b0568d2e9de	reservas-app-backend	5000:8080	N/A	1 second ago	... ⋮ trash
<input type="checkbox"/>	sqlserver	ef940a47d551	mssql/server:2022-latest	1433:1433	N/A	1 second ago	... ⋮ trash

Showing 6 items

Debido a que no metí un Angular build en el frontend container y debido tamb a que en dockerfile de frontend puse from nginx:alpine, si uso esa URL (<http://localhost:4200>) o si clico en 4200:80 ahí en Docker Desktop, entonces voy a la default welcome web pag del servidor web nginx:

localhost:4200

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

(PÁGINA SIGUIENTE)

En otras palabras, external (host) -> <http://localhost:4200>

internal (Docker network)->otros contenedores (como el backend)
would see it as <http://frontend:80>.

“Your backend container and frontend container communicate inside the same Docker network – – they do not use localhost to reach each other.”

● backend-1 7b0568d2e9de [reservas-app-backend](#) [5000:8080 ↗](#)

Inside the backend container, my ASP.NET Core (Web API) app is **listening on port 8080**.

That internal port is mapped to port 5000 on my host machine.

So inside the container (internal network), the backend listens on port 8080 (<http://backend:8080>),

Outside Docker (my machine), *Docker forwards traffic* from host port 5000 to container port 8080,

url outside docker es <http://localhost:5000>

Entonces, vemos que hay un patrón que es el siguiente:

en p1:p2,

p2 es el puerto “inside the container” (internal, Docker network),

mientras que p1 es el puerto “on my host machine” (external).

● sqlserver ef940a47d551 [mssql/server:2022-latest](#) [1433:1433 ↗](#)

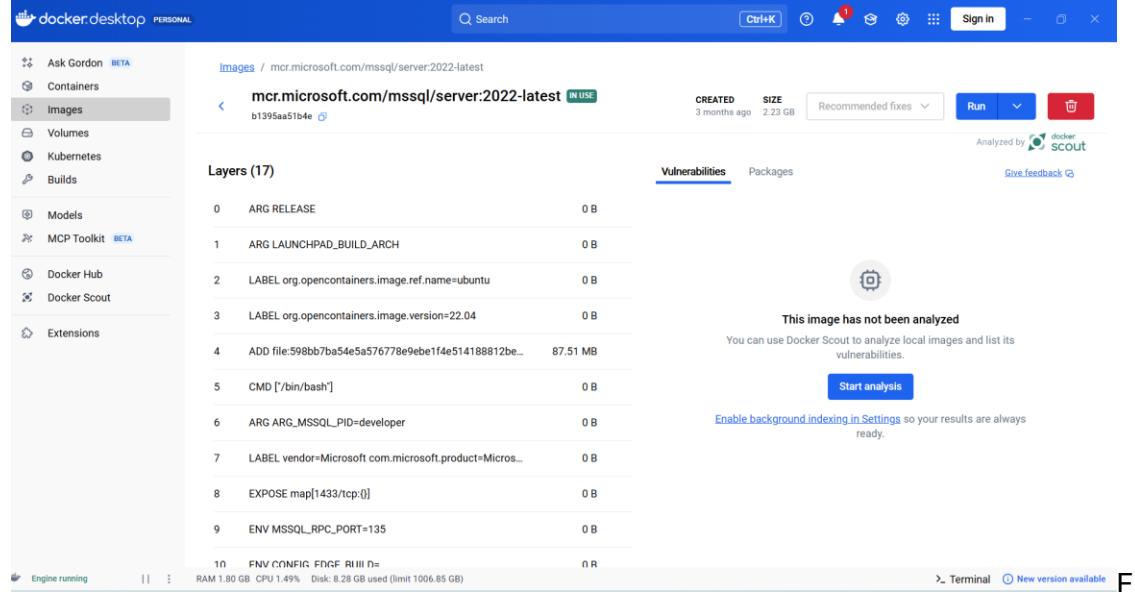
My ‘**sqlserver**’ container is based on Microsoft’s official SQL Server for Linux image, pulled from Docker Hub.

Con respecto a Linux, recordemos que cuando creé el Dockerfile del backend usando VSC, escogí Linux como SO.

(el dockerfile del frontend no lo creé vía la command palette, sino que escribí yo el código del frontend dockerfile)

The full image name es mcr.microsoft.com/mssql/server:2022-latest

En efecto, si clico en mssql/server:2022-latest, voy a:



That image gives you **SQL Server 2022 running in a Linux environment** – – totally fine for development or even production if configured properly.

Lo siguiente sobre “confirm your Setup” es confirmar que “you already have” **Docker Desktop volume (persisten storage for SQL)**.

The screenshot shows the Docker Desktop interface. On the left, a sidebar menu includes 'Ask Gordon BETA', 'Containers', 'Images', 'Volumes' (which is selected), 'Kubernetes', 'Builds', 'Models', 'MCP Toolkit BETA', 'Docker Hub', 'Docker Scout', and 'Extensions'. The main area displays the 'Volumes' section with a sub-section for 'reservas-app_sql_data'. This volume was created 2 days ago and is currently in use. It contains four stored data entries: '.system' (83.3 kB, 2 days ago, drwxr-xr-x), 'data' (134.3 MB, 2 days ago, drwxr-xr-x), 'log' (1.1 MB, 26 minutes ago, drwxr-xr-x), and 'secrets' (44 Bytes, 2 days ago, drwxr-xr-x). There are tabs for 'Stored data', 'Container in-use', and 'Exports'.

[Volumes](#) / [reservas-app_sql_data](#)

This screenshot shows the 'Container in-use' tab for the 'reservas-app_sql_data' volume. It lists one container named 'sqlserver' which is using the 'mcr.microsoft.com/mssql/server:2022-latest' image. The port mapping is 1433 to /var/opt/mssql. There are tabs for 'Stored data', 'Container in-use', and 'Exports'.

Según Chatgpt, “that setup **is exactly how** a persistent SQL Server volume **should look**.

'reservas-app_sql_data' es un nombre **that was automatically generated by Docker Compose based on your Compose project name (**reservas-app**)**

and **the volume name defined in your docker-compose.yml (sql_data)**.

The screenshot shows a file explorer interface with a breadcrumb navigation bar: Iago Xesus Cadavid Gonzalez > Proyectos > reservas-app >. The main area displays a list of files and folders: 'backend', 'frontend', 'serviciocron', and 'docker-compose.yml'. The 'docker-compose.yml' file is listed as an 'Archivo de origen Yaml' with a size of 1 KB. The interface includes standard file operations like 'Nuevo', 'Borrar', 'Ordenar', and 'Ver'.


```
volumes:  
|   sql_data:
```

En docker-compose.yml.

Según chatgpt, en mi caso el Compose project name reservas-app procede del “name of the directory that contains docker-compose.yml (que recordemos “defines all of your services (frontend, backend, SQL Server, volumes, networks, etc.)”. “docker-compose.yml brings them all up together”, donde them se refiere a Angular frontend, ASP.NET Core Web API backend, SQL Server container (official image from Docker Hub) y Docker Desktop volume (persisnte storage for SQL).

Finally (sobre 1. Confirm your Setup), **make sure your containers can talk to each other**

through Docker's internal network (they will share a default network defined by Compose).

When you run **docker compose up** for the first time, Docker Compose looks at your docker-compose.yml file and:

- Parses all the defined services (frontend, backend, sqlserver, etc.)
- Creates any needed networks, volumes you declared (reservas-app_sql_data) and containers for each service (fex reservas-app_backend_1) that don't already exist.
- Starts all those containers (building images first if needed)
- **Creates the default network** (fex. reservas-app_default)

Bien, ahora toca el **paso 2**, que es **Connect ASP.NET Core (Web API) Backend to SQL Server (in Docker)**

✳️ 1. When you run `docker compose up` for the first time

That's the key moment.

When you execute:

```
bash  
  
docker compose up
```

Docker Compose looks at your **Compose file** (`docker-compose.yml`) and then:

1. Parses all the defined services (frontend, backend, sqlserver, etc.)
2. Creates any needed **networks, volumes, and containers** that don't already exist
3. Starts all those containers (building images first if needed)

So the **first** `docker compose up` command triggers the creation of:

- The **default network** (e.g. `reservas-app_default`)
- The **containers** for each service (e.g. `reservas-app_backend_1`)
- The **volumes** you declared (e.g. `reservas-app_sql_data`)

Docker-compose.yml defines your multi-container application – all the services, networks and volumes that make up your app.

💡 2. How to check that your containers share the same network

You can confirm this directly with Docker Desktop or the CLI.

🔍 Option 1 – Docker Desktop

1. Open Docker Desktop.
2. Go to the **Containers** tab.
3. Expand your stack (e.g., `reservas-app`).
4. Click on one container → **Inspect** → **Networks** section.

You'll see something like:

```
arduino                                ⌂ Copy code

"Networks": {
  "reservas-app_default": {
    "IPAddress": "172.20.0.2",
    ...
  }
}
```

5. Check that all your containers (frontend, backend, SQL Server) list the same network name — for example `reservas-app_default`.

If they all share the same network, they can talk to each other by service name (e.g.

`http://backend:8080, Server=sqlserver`).



NEXT PAGE

Sqlserver container:

```
"Networks": {  
    "reservas-app_default": {  
        "IPAMConfig": null,  
        "Links": null,  
        "Aliases": [  
            "sqlserver",  
            "db"  
        ],  
        "MacAddress": "96:5d:df:d3:4f:55",  
        "DriverOpts": null,  
        "GwPriority": 0,  
        "NetworkID": "5365434570b4cf97625c5f2df75fab717f629aac92569e7ce721d22aac0fc1ba",  
        "EndpointID": "21231bf991a65748afa294196b07aabbbf0337fc31ef5ada44dfdafce180fc61",  
        "Gateway": "172.18.0.1",  
        "IPAddress": "172.18.0.2",  
        "IPPrefixLen": 16,  
        "IPv6Gateway": "",  
        "GlobalIPv6Address": "",  
        "GlobalIPv6PrefixLen": 0,  
        "DNSNames": [  
            "sqlserver",  
            "db",  
            "ef940a47d551"  
        ]  
    }  
}
```

Backend container:

```
"Networks": {  
    "reservas-app_default": {  
        "IPAMConfig": null,  
        "Links": null,  
        "Aliases": [  
            "reservas-app-backend-1",  
            "backend"  
        ],  
        "MacAddress": "52:25:d0:c0:63:a1",  
        "DriverOpts": null,  
        "GwPriority": 0,  
        "NetworkID": "5365434570b4cf97625c5f2df75fab717f629aac92569e7ce721d22aac0fc1ba",  
        "EndpointID": "0865028abd80f3fd1926264f94e832ee250fd3e42e9cd7f54f83377be7a4142b",  
        "Gateway": "172.18.0.1",  
        "IPAddress": "172.18.0.3",  
        "IPPrefixLen": 16,  
        "IPv6Gateway": "",  
        "GlobalIPv6Address": "",  
        "GlobalIPv6PrefixLen": 0,  
        "DNSNames": [  
            "reservas-app-backend-1",  
            "backend",  
            "7b0568d2e9de"  
        ]  
    }  
}
```


Frontend container:

```
"Networks": {  
    "reservas-app_default": {  
        "IPAMConfig": null,  
        "Links": null,  
        "Aliases": [  
            "reservas-app-frontend-1",  
            "frontend"  
        ],  
        "MacAddress": "2e:8a:c1:bf:e2:f4",  
        "DriverOpts": null,  
        "GwPriority": 0,  
        "NetworkID": "5365434570b4cf97625c5f2df75fab717f629aac92569e7ce721d22aac0fc1ba",  
        "EndpointID": "197beba53da90b71cf1d2fd554fa2c896170cd3c12825494d2698b6e38347da8",  
        "Gateway": "172.18.0.1",  
        "IPAddress": "172.18.0.4",  
        "IPPrefixLen": 16,  
        "IPv6Gateway": "",  
        "GlobalIPv6Address": "",  
        "GlobalIPv6PrefixLen": 0,  
        "DNSNames": [  
            "reservas-app-frontend-1",  
            "frontend",  
            "3b4ebf8eb7d9"  
        ]  
    }  
}
```

Según esos resultados, chatgpt dice “perfect, that is exactly what you wanna see, your containers’ inspection results confirm that **all three containers** –sqlserver,backend-1 and frontend-1- are **correctly connected to the same internal Docker network** (reservas-app_default), which means: They **can communicate directly using their service names**” defined in your ‘docker-compose.yml’.

The **service names** are used by Docker as hostnames inside the internal Compose network (reservas-app_default).

NEXT PAGE

✳️ 1. What "service name" means in a Compose file

In your `docker-compose.yml`, every top-level key under `services:` defines one service.

The key name itself is the service name.

Example

```
yaml                                         ⌂ Copy code

version: "3.8"

services:
  frontend:
    build: ./frontend
    ports:
      - "4200:80"
    depends_on:
      - backend

  backend:
    build: ./backend
    ports:
      - "5000:8080"
    depends_on:
      - sqlserver

  sqlserver:
    image: mcr.microsoft.com/mssql/server:2022-latest
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=YourStrong!Passw0rd
    ports:
      - "1433:1433"
    volumes:
      - sql_data:/var/opt/mssql

volumes:
  sql_data:
```

In that example, the service names are:

```
nginx                                         ⌂ Copy code

frontend
backend
sqlserver
↓
```

How Docker uses them: inside the internal Docker network (`reservas-app_default`),
those (service) names become DNS hostnames that all containers can *resolve* automatically.
En mi caso: NEXT PAGE

Service name	What it represents	How other containers use it
Frontend	Angular frontend container	http://frontend:80 reservas-app-frontend 4200:80 🔗
Backend	ASP.NET Core Web API container	http://backend:8080 reservas-app-backend 5000:8080 🔗
sqlserver	SQL Server container	Server=db;...

Important: You never use localhost between containers – always the service names.
 Eso chatgpt lo puso porque pensó que en mi **.yml file** ponía **sqlserver: image: mcr...**
 Pero en verdad pone **db: image: mcr...**
 Entonces al saber esto último, chatgpt dice que entonces **your service name is ‘db’, not sqlserver.**
 So **inside Docker, the hostname to reach SQL Server is ‘db’, not sqlserver.**
 “**inside your internal Docker network** (reservas-app_default),
any other container can reach this one (el container que llamé ‘sqlserver’ en el .yml file al poner **container_name=sqlserver** dentro de db:) **by hostname ‘db’** since en mi caso **‘db’ is the service name** in Docker Compose (dato que puse db justo debajo de services en el y.yml file”).
“from your backend container, you can connect to SQL Server at Server=db;...”.
 “even though you gave it a container_name: sqlserver, other containers still refer to it by the service name db”
 Osease “other services still resolve the service name ‘db’, not the container name sqlserver.
 Con respecto a eso de **Server=db;** “Context: your backend needs to connect to SQL Server, your ASP.NET Core Web API (backend) needs a **connection string** to talk to your SQL Server database”.
 The backend container can reach the SQL Server container using the **hostname db** (el service name).
 “your backend’s connection string should be
Server=db;Database=BookingDB;User Id=sa;Password=YourStrong!Passw0rd;” **de dónde sale todo esto?**
When your backend (the asp.net core web api) **connects to sql server,**
it needs a connection string – a single line that tells .NET where is sql server, what database, what credenciales should I use.
 The connection string format is a standard one defined by Microsoft SQL Server drivers, here is the generic pattern:
Server=<host or IP>;
Database=<db_name>;
User Id=<username>;
Password=<password>;
 [Optional settings...]
De dónde proceden my specific values for all that?
‘db’ is the **service name** you defined for the sql server container

```
db:
  image: mcr.microsoft.com/mssql/server:2022-latest
  container_name: sqlserver
```

 ‘db’ is under ‘services’ en ese .yml file.
 Inside Docker, other containers can reach it (the sqlserver container) using that name.
 Database=**BookingDB** “you choose this, it is **the logical database name** you’ll create inside SQL Server,
 It can be anything you want (reservasDB, BookingDB, etc.).
 If does not exist yet, **you will create it with a migration or SQL script.**

User Id=sa comes from SQL server defaults, the sa user (**System Admin**) is the built-in SQL Server admin account.
Password=YourStrong!Passw0rd comes from your Compose environment variables, **you set it under db service with SA_PASSWORD=YourStrong!Pass0rd** en el .yml file.

```
db:
  image: mcr.microsoft.com/mssql/server:2022-latest
  container_name: sqlserver
  environment:
    - ACCEPT_EULA=Y
    - SA_PASSWORD=dockEr!8
```

Esto está en mi .yml file.

TrustServerCertificate=True is Optional .NET flag which avoids SSL certificate validation errors during local/Docker dev. (Recommended in dev environments **only**).

Sobre el connection string, notemos que en mi .yml file lo tengo:

```
docker-compose.yml x
C: > Users > iago.cadavidgonzalez > Proyectos > reservas-app > docker-compose.yml
  3   services:
  4     |     ▷ Run Service
  5     |     backend:
  6     |       build:
  7     |         context: ./backend
  8     |       ports:
  9     |         - "5000:8080"
 10     |       environment:
 11     |         - ASPNETCORE_ENVIRONMENT=Development
 12     |         - ConnectionStrings_DefaultConnection=Server=db;Database=ReservasDb;User=sa;Password=dockEr!8;TrustServerCertificate=True;
 13     |       depends_on:
 14     |         - db
```

“This environment variable tells .NET that the connection string named DefaultConnection should be Server=db;...”.

Recordar: even though your container is named **sqlserver**,
your **backend** should connect using **Server=db** because db is the Compose service name,
which Docker uses for container-to-container DNS.

Según chatgpt, como en .yml tengo esto:

```
backend:
  build:
    context: ./backend
  ports:
    - "5000:8080"
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ConnectionStrings_DefaultConnection=Server=db;Database=ReservasDb;User=sa;Password=dockEr!8;TrustServerCertificate=True;
  depends_on:
    - db
```


Entonces a mi appsettings.json, que actualmente es:

```
1  {
2      "Logging": {
3          "LogLevel": {
4              "Default": "Information",
5              "Microsoft.AspNetCore": "Warning"
6          }
7      },
8      "AllowedHosts": "*"
9  }
10
```

Debería añadirle justo antes de “Logging” lo siguiente:

```
1  {
2      "ConnectionStrings": {
3          "DefaultConnection": "Server=localhost,1433;Database=ReservasDb;User Id=sa;Password=dockEr!8;TrustServerCertificate=True;"
4      },
5
6      "Logging": {
7          "LogLevel": {
8              "Default": "Information",
9              "Microsoft.AspNetCore": "Warning"
10         }
11     },
12     "AllowedHosts": "*"
13 }
14
```

(ojo porque en .yml file no puse User Id sino solo User; osease **en .yml file puse User=sa;**).

¿Por qué añadir eso? Pues porque “without the “ConnectionStrings” section in appsettings.json, your backend will only know about the connection string if it is provided via environment variable (like in your docker-compose.yml). That’s fine if you only ever plan to run your backend in Docker. But most devs also run their backend locally (fex with dotnet run), so it is best practice to define a default fallback connection string in appsettings.json”

En el .yml file me recomienda poner User Id en vez de solo User, así lo hago:

NEXT PAGE

The screenshot shows a code editor with two tabs: 'docker-compose.yml' and 'appsettings.json'. The 'docker-compose.yml' tab contains the following configuration:

```

C: > Users > iago.cadavidgonzalez > Proyectos > reservas-app > docker-compose.yml
3 services:
  backend:
    build:
      context: ./backend
    ports:
      - "5000:8080"
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ConnectionStrings_DefaultConnection=Server=db;Database=ReservasDb;User Id=sa;Password=dockEr!8;TrustServerCertificate=True;
    depends_on:
      - db

```

Para mí el segundo paso de los 7 (pa lograr tener mis primeros endpoints) ya está hecho (pero falta confirmar unas cosillas, lo cual hago después desta captura de pantalla):

2. Connect ASP.NET Core Backend to SQL Server (in Docker)

Example connection string (in `appsettings.json`):

```
json
Copy code

{
  "ConnectionStrings": {
    "DefaultConnection": "Server=db;Database=BookingDB;User Id=sa;Password=YourStrong!Passw0rd;"
  }
}
```

⚠ Important:

- “`Server=db` → this should match the service name of your SQL container in `docker-compose.yml`.”
- “SQL authentication must use the credentials defined in your compose file.”



Dice chatgpt que por haber cambiado de User a User Id en el .yml file, que pa ese cambio surta efecto he de abrir Docker Desktop, ir a Containers tab, ir al project stack (reservas-app), stop el backend container, then start it again (lo voy hacer con todo el reservas-app).

Confirming the connection in both environments (from your host and from your backend container)

will 100% **verify that your backend <-> SQL Server wiring is correct**

Before you even try to run EF migrations or API endponits.

PART 1 – Verify connection from your host machine.

1.1. Confirm **SQL Server container is running** (docker ps en cmd, da igual el currenti dir):

```
C:\Windows\System32>docker ps
CONTAINER ID  IMAGE                               COMMAND                  CREATED     STATUS      PORTS          NAMES
3b4ebf8e87d9  Reservas-app-frontend             "/docker-entrypoint..."  2 days ago   Up 6 minutes  0.0.0.0:4200->80/tcp   reservas-app-frontend-1
7b568d2e9de  Reservas-app-backend              "dotnet ReservasApi..."  2 days ago   Up 6 minutes  0.0.0.0:5000->8080/tcp   reservas-app-backend-1
ef940a47d551  mcr.microsoft.com/mssql/server:2022-latest  "/opt/mssql/bin/laun..."  2 days ago   Up 6 minutes  0.0.0.0:1433->1433/tcp   sqlserver

C:\Windows\System32>
```


La última línea means **SQL Server is listening on your host's port 1433**. Visto en Docker Desktop:

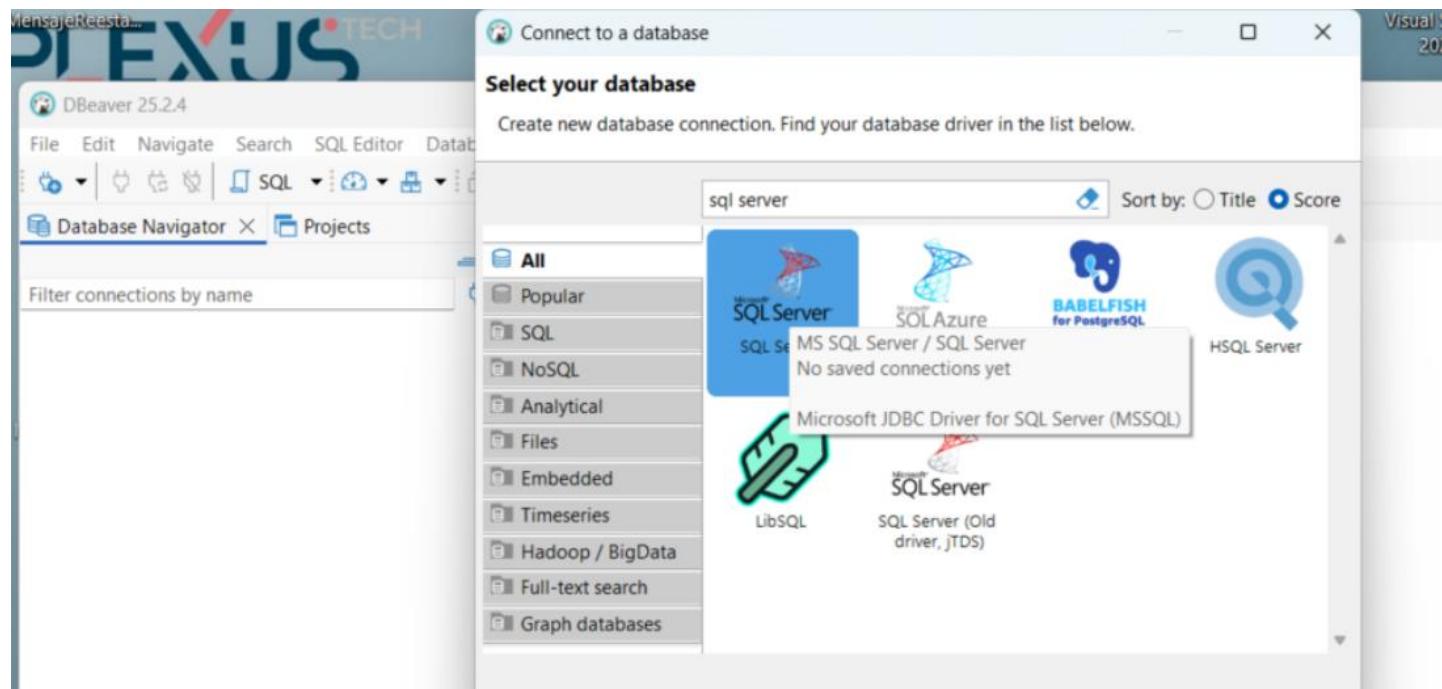
sqlserver	ef940a47d551	mssql/server:2022-latest 1433:1433 ↗	6.34% 7 minutes ago
-----------	--------------	--------------------------------------	---------------------	------------------	------------------	------------------

PART1-Test connection from your host aka verify connection from your host machine

Se puede hacer con SQL Server Management Studio o con Azure Data Studio, si no tienes ninguno tmb lo puedes testear con el recomendado por Victo i.e. **DBeaver**.

Create a new connection, para esto [Open DBeaver](#),

click **Database -> New Database Connection, Select SQL Server**

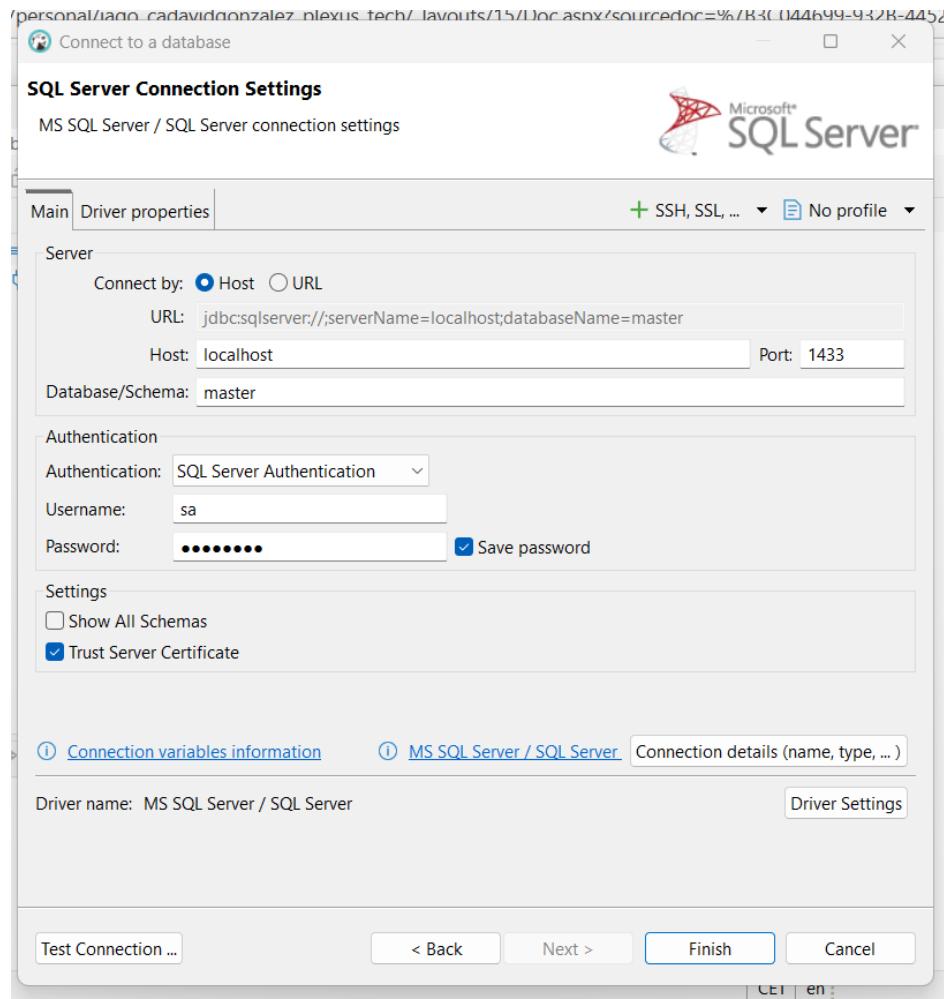


Enter connection details (pero antes de finalizar este diálogo, los containers (por lo menos el que llamé sqlserver) debería tenerlos running, cosa de la que me puedo asegurar fácilmente con DockerDestkop en sección Containers.

“The sql server container must be up and running, and if you want to test connectivity from the backend container, that container must also be running.” Yo pongo todo running (tmb el frontend) dándole al play en reservas-app:

...	...	reservas-app	...	N/A 0 seconds ago
...	...	frontend-1	3b4ebf8eb7d9	reservas-app-frontend 4200:80 ↗ N/A 0 seconds ago
...	...	backend-1	7b0568d2e9de	reservas-app-backend 5000:8080 ↗ N/A 1 second ago
...	...	sqlserver	ef940a47d551	mssql/server:2022-latest 1433:1433 ↗ N/A 1 second ago

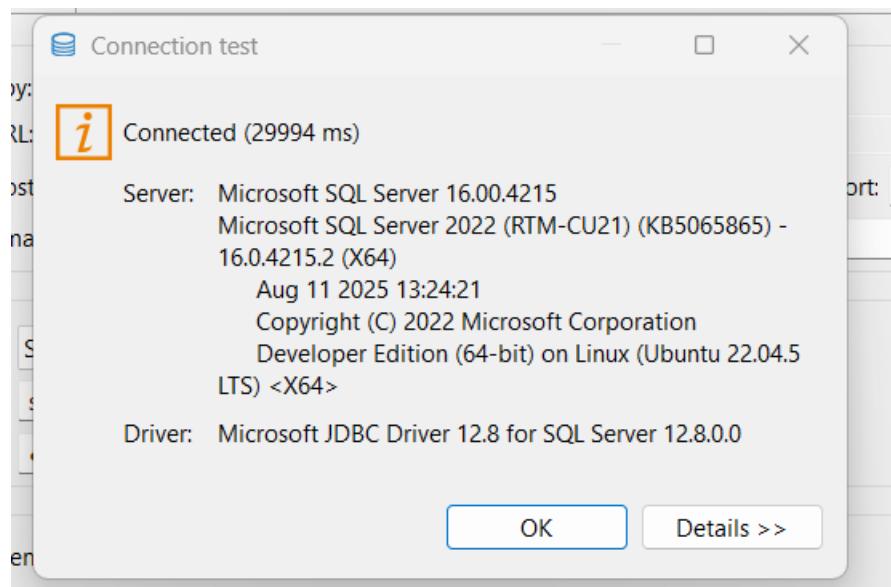
Showing 6 items



Puse la pw del .yml file, sa tmb de ahí.

Doy en **Test Connection**.

Le di a Download en el pop-up de Download sql server driver files y entonces



si doy en Details:

Properties for "master"	
Property	Value
Driver	
Driver Type	SQL Server
Driver	
Driver Files	
0	C:\Users\iago.cadavidgonzalez\AppData\Roaming\dbeaverData\drivers\maven\maven-central\com.microsoft.sqlserver\mssql-jdbc-12.8.0.jre11.jar
1	C:\Users\iago.cadavidgonzalez\AppData\Roaming\dbeaverData\drivers\maven\maven-central\com.microsoft.sqlserver\mssql-jdbc_auth-12.8.0.x64.dll
2	C:\Users\iago.cadavidgonzalez\AppData\Roaming\dbeaverData\drivers\remote\microsoft\sql-server-samples\master\license.txt
Main	
ID	microsoft-19a58b2ae98-63554e6fdb8019a6
Name	master
Description	
Origin	local
Connect Time	
Connect Type	Development
Server	
Address	localhost:1433
Database	master
URL	jdbc:sqlserver://;serverName=localhost;databaseName=master
Server	
Details	

"If the connection test says "Connected", it means DBeaver successfully reached your SQL Server container. Now click on Finish on the Connect to a database dialogue.

The screenshot shows the DBeaver 25.2.4 interface. In the Database Navigator, a connection to 'master' on 'localhost:1433' is selected. The tree view shows the 'master' database structure, including 'dbo' schema with various objects like Tables, External Tables, Views, Indexes, Procedures, Sequences, Synonyms, Triggers, and Data Types. A tooltip on the right provides instructions for creating a database named 'ReservasDb':

"(ReservasDb should appear once created)" when does it get created though ?

Good catch — ReservasDb doesn't exist automatically when you just spin up the SQL Server container. It gets created in one of two ways:

- Automatically by your backend (Entity Framework / migrations)**

If your ASP.NET Core backend uses Entity Framework Core and is configured to apply migrations on startup, it can create the database automatically.

 - This usually happens in `Program.cs` or `Startup.cs`, for example:

```
charrp
using (var scope = app.Services.CreateScope())
{
    var db = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
    db.Database.Migrate(); // creates the database and applies migrations
}
```
- Manually**

If you don't have migrations set up yet, you need to create the database yourself:

 - Open DBeaver.
 - Connect to SQL Server.
 - Run this SQL command in the query editor:

```
sql
CREATE DATABASE ReservasDb;
```

4. Refresh the database list — ReservasDb will now appear.

Tip: Most tutorials with Docker + ASP.NET Core + SQL Server assume EF Core will handle database creation automatically. But if you haven't set that up yet, the manual step ensures it exists.

I can guide you on how to check if your backend is configured to create the database automatically so you know whether to do it manually or not. Do you want me to do that?

Entonces eso en 1º debería valer en cuanto a **verify connection from your host machine** i.e. **test connection from your host**. Pero es lo mismo que **comprobar que backend application running on my host i.e. outside Docker can connect to sql server container?** Al respecto desto chatgpt viene a decir que **sí** porque:

When you run your ASP.NET Core backend outside docker, that backend app will also be running on your host - Just like DBeaver. So if DBeaver could reach the SQL Server container using localhost:1433, then your backend app **can too**, as long as it uses the same connection string, la de appsettings.json del backend project:

```
{} appsettings.json 1 X  
C: > Users > iago.cadavidgonzalez > Proyectos > reservas-app > backend > ReservasApi > {} appsettings.json > ...  
1  {  
2    "ConnectionStrings":{  
3      "DefaultConnection": "Server=localhost,1433;Database=ReservasDb;User Id=sa;Password=docker18;TrustServerCertificate=True;"  
4    },  
5  } ,
```

PART 2 – Verify connection from inside the backend container.

Now let's **confirm the backend container can reach SQL Server over the internal Docker network** (using **Server=db**, donde db es el service name cuyo container name es sqlserver)

Make sure DD is running (lo está), then type

docker exec -it nombredelcontenedor /bin/bash

[Containers](#) / reservas-app-backend-1

reservas-app-backend-1

En inspector tab dese container:

"Name": "/reservas-app-backend-1",

Entonces

docker exec -it reservas-app-backend-1 /bin/bash

```
app@7b0568d2e9de:/app  
Microsoft Windows [Versión 10.0.26200.6901]  
(c) Microsoft Corporation. Todos los derechos reservados.  
C:\Windows\System32>docker exec -it reservas-app-backend-1 /bin/bash  
app@7b0568d2e9de:/app$
```

Ese resultado means:

You are inside your backend container (reservas-app-backend-1),

You're logged in as the user 'app',

Your current directory inside the container is /app (the folder where your backend is running)

Next step: **test connectivity to SQL Server**:

PART2-let's check that your backend container can reach your SQL Server container (db –service name-) over the internal Docker network. (using Sserver=db, siendo db el service name del container_name sqlserver) AKA Verify connection (to sql server container) from inside the backend container

[esta primera opción no me vale porque no reconoce /dev/tcp/db/1433

Una opción para chequear eso es “simple TCP connection test (no tools needed)”:

“run this inside the container shell (right after the \$):

bash -c “</dev/tcp/db/1433” && echo “Connected to SQL Server” || echo “Conexion fallida”

Bash -c ... tells Docker to run a Bash command inside the (backend) container.

The -c flag means “execute the following string as a command” siendo el string lo que va entre comillas dobles.

< is an **input redirection operator**, /dev/tcp/db/1433 is a virtual file that represents a TCP socket.

So Bash tries to open /dev/tcp/db/1433, which triggers an internal call to open a TCP connection/socket to host db on port 1433.

&& significa "if the previous command succeeded, then run this next command".

|| significa "if the previous command (</dev/tcp/db/1433) failed, then run this one instead".

/dev/tcp/<host>/<port> can be used to test a TCP connection.ç

If the connection succeeds, the file open operation (</dev/tcp/db/1433 means try to connect to host db on port 1433) returns success whereas if the connection fails, it returns failure.

File does not exist me dice a mi la ejecución del comando en relación a la ruta entera

]

Una (pseudo)alternativa para hacer ese chequeo es “run the test from the sql server container (built-in sqlcmd)”

Since the SQL Server container already has Microsoft’s client tools, (run this on cmd –curr dir does not matter-):

docker exec -it sqlserver /opt/mssql-tools18/bin/sqlcmd -S db -U sa -P “mipw” -C

docker exec tells Docker “run a command inside an already running container”.

Unlike docker run (which creates a new container), docker exec attaches to an existing one.

It’s like remote-controlling a container that has already started.

-it is two flags combined: **-i** means **interactive** (lets you type commands inside the container)

-t means **tty** (makes it feel like a normal shell)

-it means you’ll see a prompt and can type interactively (like when you run /bin/bash).

sqlserver is the **container name** (where) you want to execute the command in. It matches what I defined in my docker-compose.yml under container_name: sqlserver. Docker uses that name to find the right container instance.

/opt/mssql-tools18/bin/sqlcmd is the path to the **sqlcmd** program inside the SQL Server container. Sqlcmd is Microsoft’s official command-line client for SQL Server –like a terminal-based SQL query tool.

Así que estamos diciendo “run the sqlcmd tool that is already installed inside the sql server container”.

-S db : -S stands for Server. Here you are saying: Connect to the SQL Server instance whose host name is ‘db’.

‘db’ comes from service name ‘db’ que puse en docker-compose.yml file (each service name becomes a host name).

From one container to another, Server=db points to your SQL Server container.

-U means **User name**, so **-U sa** means you’re **logging in as** the sa (**System Administrator**) **SQL Server account**.

User Id=sa lo puse en mi docker-compose.yml.

-P “mipw” : -P means password for SQL user. You defined this same password in your docker-compose.yml en environment: **- SA_PASSWORD=...** dentro del db sevice.

-C trust the server certificate. SQL Server in Docker uses a self-signed TLS certificate, and without -C you might get an SSL trust error. Adding -C skips that warning. Mi .yml contiene TrustServerCertificate=True

NEXT PAGE

 What does this page say about "1>"?

Let's break it down:

- `docker exec -it sqlserver` → opens an interactive session inside your **SQL Server container** (whose name is `sqlserver`).
- `/opt/mssql-tools18/bin/sqlcmd` → runs the **Microsoft SQL Server command-line client** that's already installed in the official SQL Server image.
- `-S db` → tells it to connect to the hostname `db` (the name your backend uses inside Docker).
- `-U sa` and `-P "MyStrong!Passw0rd"` → use your SQL Server credentials.
- `-c` → trusts the server certificate (needed for Docker SSL).

What you should see

If the connection succeeds, you'll enter the SQL prompt:

shell

 Copy code

1>



That means you're connected to SQL Server from inside its own container 

Connected to sql server from inside its own container (not that backend can talk to it over docker network)

What that command actually does

bash

 Copy code

```
docker exec -it sqlserver /opt/mssql-tools18/bin/sqlcmd -S db -U sa -P "MyStrong!Passw0rd" -C
```

This **does not** test whether your **backend container** can reach your SQL Server container.

Instead, it runs inside your **SQL Server container** (`sqlserver`), and then tries to connect to the hostname `db`.

Since you're *already inside* the SQL Server container, that only checks whether the SQL Server container can reach something named `db` — which, in most setups, would just point back to itself.

So yes, it confirms that **SQL Server itself is running and accepting connections**,
but **not that your backend can talk to it over the Docker network**.

To **confirm the backend container can reach the SQL Server container**,
you need to run a connection test from inside the backend container, not the SQL (Server) one.
To test this, one option is the following:

Run this on your host (en cmd):

docker exec -it reservas-app-backend-1 /bin/bash

Once inside (app@...:/app\$ prompt):

Run:

apt-get update && apt-get install -y netcat

nc -zv db 1433

If you see:

db (172.18.0.2:1433) open

Then your backend container can reach SQL Server successfully.

apt-get update = update the list of available software packages and their versions,

así evitamos un fail al hacer el apt-get install por package not found.

&& means only run the next command (apt-get install -y netcat) if the previous one succeeded (apt-get update).

Apt-get in stasll -y netcat: it downloads and installs the netcat tool –a small networking utility that can test open ports, send/receive data, etc.

nc -zv db 1433: once netcat is installed, you use it to **test connectivity**.

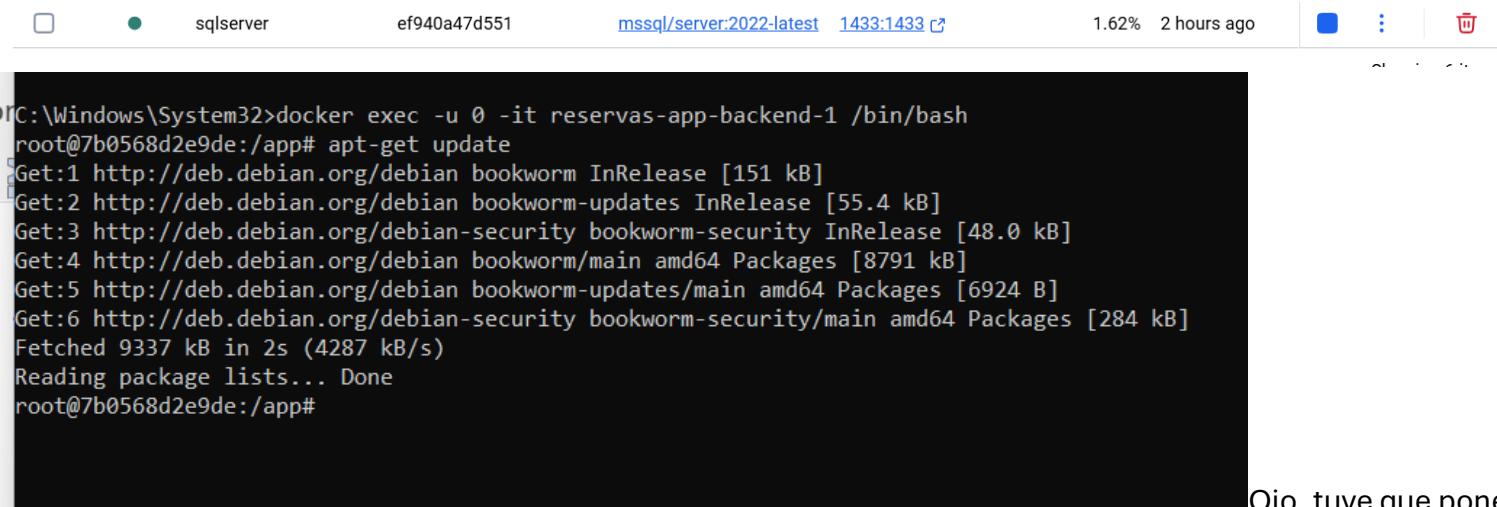
nc runs netcat

-z es zero I/O mode, just **checks if the port is open** (does not send data)

-v verbose –prints detailed output

db es the hostname you want to connect to (el service name correspondiente en .yml file)

1433 is the port number SQL Server listens on



The screenshot shows a terminal window within a Docker container. The container details at the top are: sqlserver, ef940a47d551, mssql/server:2022-latest, 1433:1433, 1.62%, 2 hours ago. The terminal output is as follows:

```
C:\Windows\System32>docker exec -u 0 -it reservas-app-backend-1 /bin/bash
root@7b0568d2e9de:/app# apt-get update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8791 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [6924 B]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [284 kB]
Fetched 9337 kB in 2s (4287 kB/s)
Reading package lists... Done
root@7b0568d2e9de:/app#
```

Ojo, tuve que poner

-u 0 para poder ejecutar con éxito el apt-get update, -u 0 means **run as root/superuser (UID 0)**, which **gives full admin permissions**.

NEXT PAGE

```
root@7b0568d2e9de:/app# apt-get install -y netcat-openbsd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
 libbsd0
The following NEW packages will be installed:
 libbsd0 netcat-openbsd
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 159 kB of archives.
After this operation, 317 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main amd64 libbsd0 amd64 0.11.7-2 [117 kB]
Get:2 http://deb.debian.org/debian bookworm/main amd64 netcat-openbsd amd64 1.219-1 [41.5 kB]
Fetched 159 kB in 0s (892 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libbsd0:amd64.
(Reading database ... 6600 files and directories currently installed.)
Preparing to unpack .../libbsd0_0.11.7-2_amd64.deb ...
Unpacking libbsd0:amd64 (0.11.7-2) ...
Selecting previously unselected package netcat-openbsd.
Preparing to unpack .../netcat-openbsd_1.219-1_amd64.deb ...
Unpacking netcat-openbsd (1.219-1) ...
Setting up libbsd0:amd64 (0.11.7-2) ...
Setting up netcat-openbsd (1.219-1) ...
update-alternatives: using /bin/nc.openbsd to provide /bin/nc (nc) in auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/nc.1.gz because associated
c_openbsd.1.gz (of link group nc) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/man1/netcat.1.gz because associa
n1/nc_openbsd.1.gz (of link group nc) doesn't exist
Processing triggers for libc-bin (2.36-9+deb12u13) ...
root@7b0568d2e9de:/app#
```

Now test your connection to the SQL Server container running:

nc -zv db 1433

```
root@7b0568d2e9de:/app# nc -zv db 1433
Connection to db (172.18.0.2) 1433 port [tcp/*] succeeded!
root@7b0568d2e9de:/app#
```

Según chatgpt así hemos confirmado que **the backend container can reach the sql server container.**

Así que tambien he conseguido PART 2 – Verify connection (to sql server container) from inside the backend container.

Habiendo comprobado lo de PART1 y PART2,
“you are ready to run EF migrations or hit API endpoints”

Volviendo sobre los 7 pasos, **considero haber hecho ya el 2. Connect ASP.NET Core Backend to SQL Server (in Docker)** habiendo comprobado lo de PART1 y PART2 (en especial lo de PART2).

Paso 3. es Create a simple booking model and database context (backend), esto lo hacemos inside our ASP.NET Core Web API project.

Pero esto es precisamente lo que hice ayer.

Según chatgpt, al tener definido mi appdbcontext y los .cs files de los modelos de las tablas, “you’ve already done 90 per cent of the work, now let’s walk through

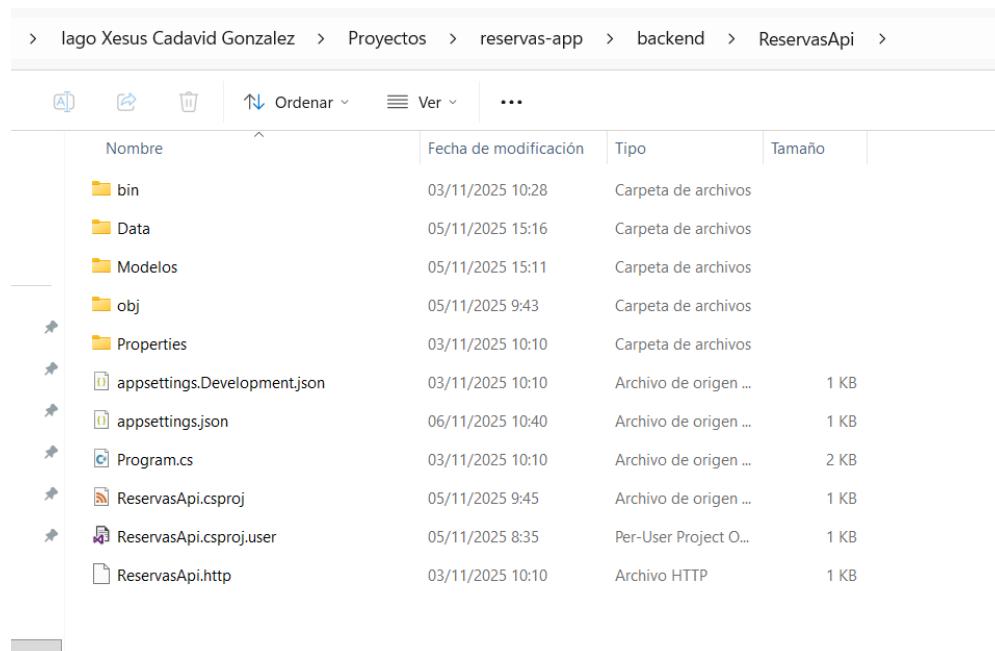
Step 4: Creating your first API endpoints in your ASP.NET Core backend (so you can read/write data from/to SQL Server).

We gonna make the first working REST API endpoint, for example:

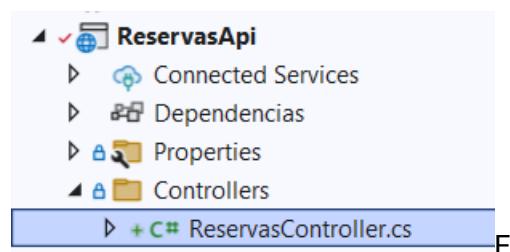
ACTION	HTTP Verb	URL	Description
GET	GET	/api/reservas	Get all bookings/reservations
POST	POST	/api/reservas	Create a new booking

Your **backend will respond to these requests** –and later our **Angular frontend can “hit” them** to show and save data.

ReservasApi es mi root of my ASP.NET Core backend porque es quien tiene por hijos a Program.cs, appsettings.json y ReservasApi.csproj.



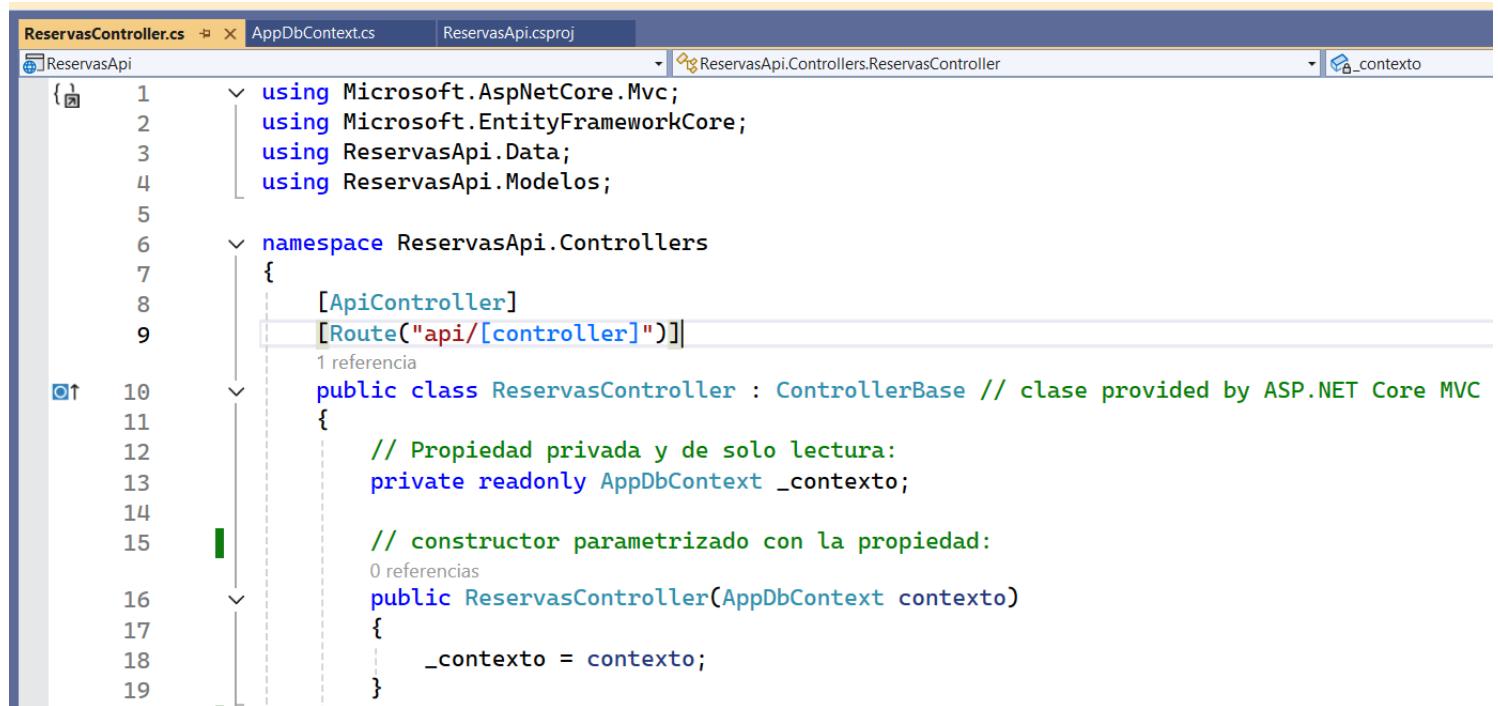
Pues tamb como hijo de ReservasApi folder he de tener a Controllers folder, la cual creo ahora y dentro le creo ReservasController.cs :



Con lo cual acabo de hacer lo de “create a new file named ReservasController.cs inside the Controllers folder, ahora toca meterle el código pertinente que en mi caso será el siguiente:

ControllerBase is a class provided by **ASP.NET Core MVC**.

Mi ReservasController.cs empieza así.



The screenshot shows the Visual Studio IDE with the ReservasController.cs file open. The code is as follows:

```
1  using Microsoft.AspNetCore.Mvc;
2  using Microsoft.EntityFrameworkCore;
3  using ReservasApi.Data;
4  using ReservasApi.Modelos;
5
6  namespace ReservasApi.Controllers
7  {
8      [ApiController]
9      [Route("api/[controller]")]
10     public class ReservasController : ControllerBase // clase provided by ASP.NET Core MVC
11     {
12         // Propiedad privada y de solo lectura:
13         private readonly AppDbContext _contexto;
14
15         // constructor parametrizado con la propiedad:
16         public ReservasController(AppDbContext contexto)
17         {
18             _contexto = contexto;
19         }
20     }
21 }
```

Al respecto de:

[ApiController]

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

Chatgpt dice “those two attributes at the top of your controller

are essential to how ASP.NET Core turns your C# code into a real REST API.”

[ApiController] tells ASP.NET Core that **the class is an ApiController** –NOT an MVC controller that returns HTML views.

It automatically enables a bunch of features designed for APIs.

[ApiController] = “treat this class as a **REST API endpoint provider** -handle JSON, not HTML.”

[Route("api/[controller]")] defines the URL pattern (route) for your controller’s endpoints.

“api/[controller]” is a route template.

The [controller] token automatically gets replaced with the controller class name WITHOUT the “Controller” sufix.

So, since your class is named:

Public class **ReservasController** : ControllerBase

Then the route becomes: /api/**reservas**

(recordar que estamos viendo step 4: Creating your first REST API endpoints in your ASP.NET Core backend so you can read/write data from/to SQL Server; your backend will respond to “these” requests and later your Angular Frontend can “hit” them to show and save data)

Ahora el código de ReservasController.cs prosigue y termina con los POST, GET, PUT, DELETE respecto de tabla reservas (chatgpt no me dio más a pesar de que le pasé mi código de todos los modelos, no solo del de reservas; supongo que no me dio más porque yo le pedí que me guará en hacer mis *primeros* endpoints)

De momento no tengo tiempo para pararme a entender al super detalle ese código del párrafo anterior (pero sí puse comentarios en el propio código al respecto de lo esencial).

(

CRUD – HTTP Methods correspondencia

CRUD stands for **Create**, **Read**, **Update** and **Delete**, which are the four basic operations you can perform on data in a database.

Esas cuatro operaciones corresponden a las HTTP methods **POST**, **GET**, **PUT** and **DELETE**, respectivamente.

)

Según chatgpt, “you normally have one controller per model in a well-structured REST API.”

The default, recommended pattern

ASP.NET Core's Web API template (and most REST APIs) follow a “**one controller per entity**” pattern because it's clean, predictable, and consistent:

Entity / Model	Typical Controller	Example Routes
Reserva	ReservasController	/api/reservas , /api/reservas/5
Usuario	UsuariosController	/api/usuarios , /api/usuarios/12345678X
Instalacion	InstalacionesController	/api/instalaciones , /api/instalaciones/3
Pago	PagosController	/api/pagos , /api/pagos/7
Cancelacion	CancelacionesController	/api/cancelaciones , /api/cancelaciones/12

Each controller then handles:

- CRUD for its own table
- Business logic directly related to that entity

That's the pattern EF Core and tools like Swagger/OpenAPI expect.

Esa tabla se

corresponde con mi caso.

Entonces, a continuación, procedo a crear **UsuariosController.cs**.

A este respecto, chatgpt me dice que “since you already have your AppDbContext and your Usuario model, you’re totally ready to create the corresponding UsuariosController that exposes CRUD endpoints for users”.

Después escribo el código de **InstalacionesController** the same way we did UsuariosController, pero en este caso (InstalacionesController) fully aligned with my Instalacion model and AppDbContext.

This controller will expose clean CRUD endpoints for managing my “instalaciones” (study rooms or sport spaces).

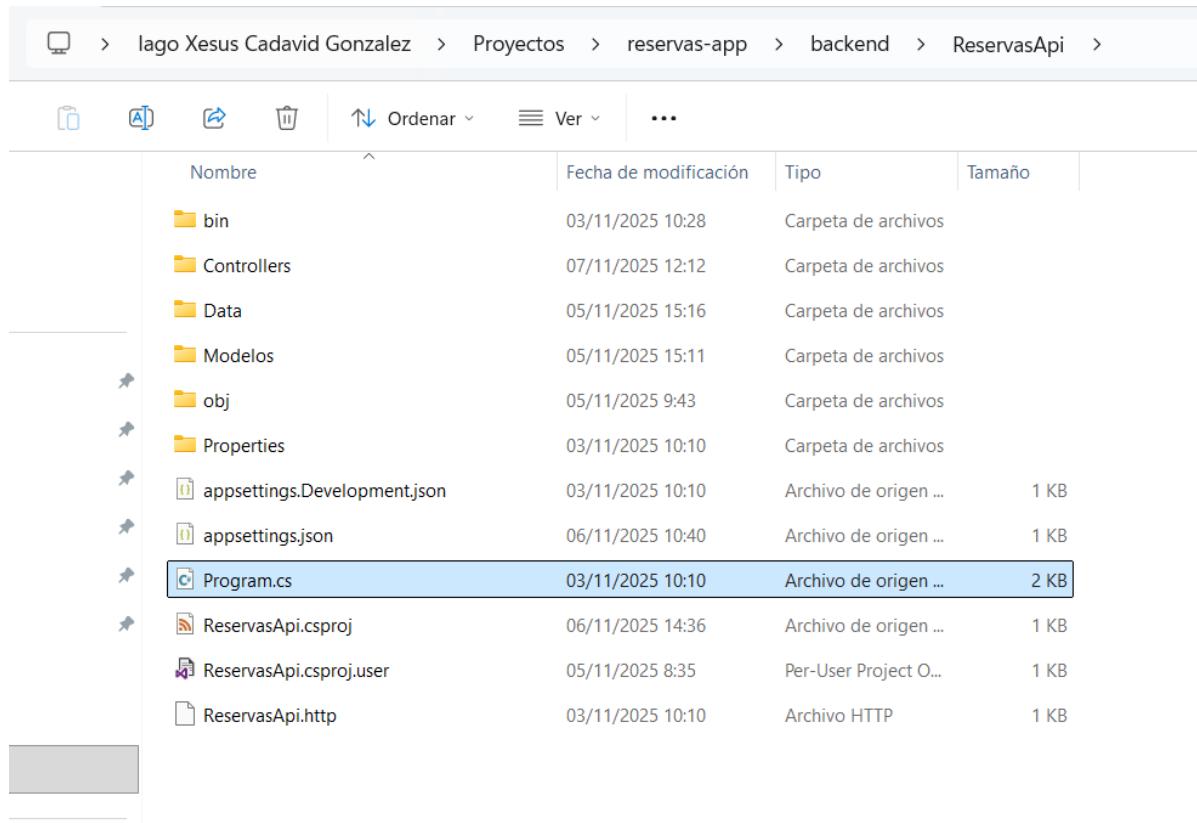
A continuación, codifico el **PagosController**.

Y finalmente **CancelacionesController.cs**.

A todos estos .cs les he puesto un código análogo con excepción de que en PagosController.cs añadimos algo de código to “also handle the navigation property (public Reserva? Reserva { get; set; }) to make sure EFCore behaves correctly”; y en CancelacionesController.cs tamb añadimos algo de código que es some logic to ensure that cancellations refer to valid reservations (recordemos que mi modelo de cancelaciones tiene una fk que apunta a la pk de la tabla de reservas y en relación a esto tamb tiene la propiedad de navegación public Reserva? Reserva { get; set; }).

En su momento chatgpt me dijo que después de escribir todo ese código, lo siguiente era:

Make sure AppDbContext is registered, cosa que se comprueba en Program.cs de mi backend



The screenshot shows a file explorer interface with the following navigation path: Home > Iago Xesus Cadavid Gonzalez > Proyectos > reservas-app > backend > ReservasApi >. The main area displays a list of files and folders:

	Nombre	Fecha de modificación	Tipo	Tamaño
📁	bin	03/11/2025 10:28	Carpeta de archivos	
📁	Controllers	07/11/2025 12:12	Carpeta de archivos	
📁	Data	05/11/2025 15:16	Carpeta de archivos	
📁	Modelos	05/11/2025 15:11	Carpeta de archivos	
📁	obj	05/11/2025 9:43	Carpeta de archivos	
📁	Properties	03/11/2025 10:10	Carpeta de archivos	
📄	appsettings.Development.json	03/11/2025 10:10	Archivo de origen ...	1 KB
📄	appsettings.json	06/11/2025 10:40	Archivo de origen ...	1 KB
📄	Program.cs	03/11/2025 10:10	Archivo de origen ...	2 KB
🔗	ReservasApi.csproj	06/11/2025 14:36	Archivo de origen ...	1 KB
👤	ReservasApi.csproj.user	05/11/2025 8:35	Per-User Project O...	1 KB
📄	ReservasApi.http	03/11/2025 10:10	Archivo HTTP	1 KB

Next page

```

1 var builder = WebApplication.CreateBuilder(args);
2
3 // Add services to the container.
4 // Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
5 builder.Services.AddOpenApi();
6
7 var app = builder.Build();
8
9 // Configure the HTTP request pipeline.
10 if (app.Environment.IsDevelopment())
11 {
12     app.MapOpenApi();
13 }
14
15 app.UseHttpsRedirection();
16
17 var summaries = new[]
18 {
19     "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
20 };
21
22 app.MapGet("/weatherforecast", () =>
23 {
24     var forecast = Enumerable.Range(1, 5).Select(index =>
25         new WeatherForecast
26         (
27             DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
28             Random.Shared.Next(-20, 55),
29             summaries[Random.Shared.Next(summaries.Length)])
30         )
31         .ToArray();
32     return forecast;
33 })
34 .WithName("GetWeatherForecast");
35
36 app.Run();
37
38 record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary)
39 {
40     public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
41 }

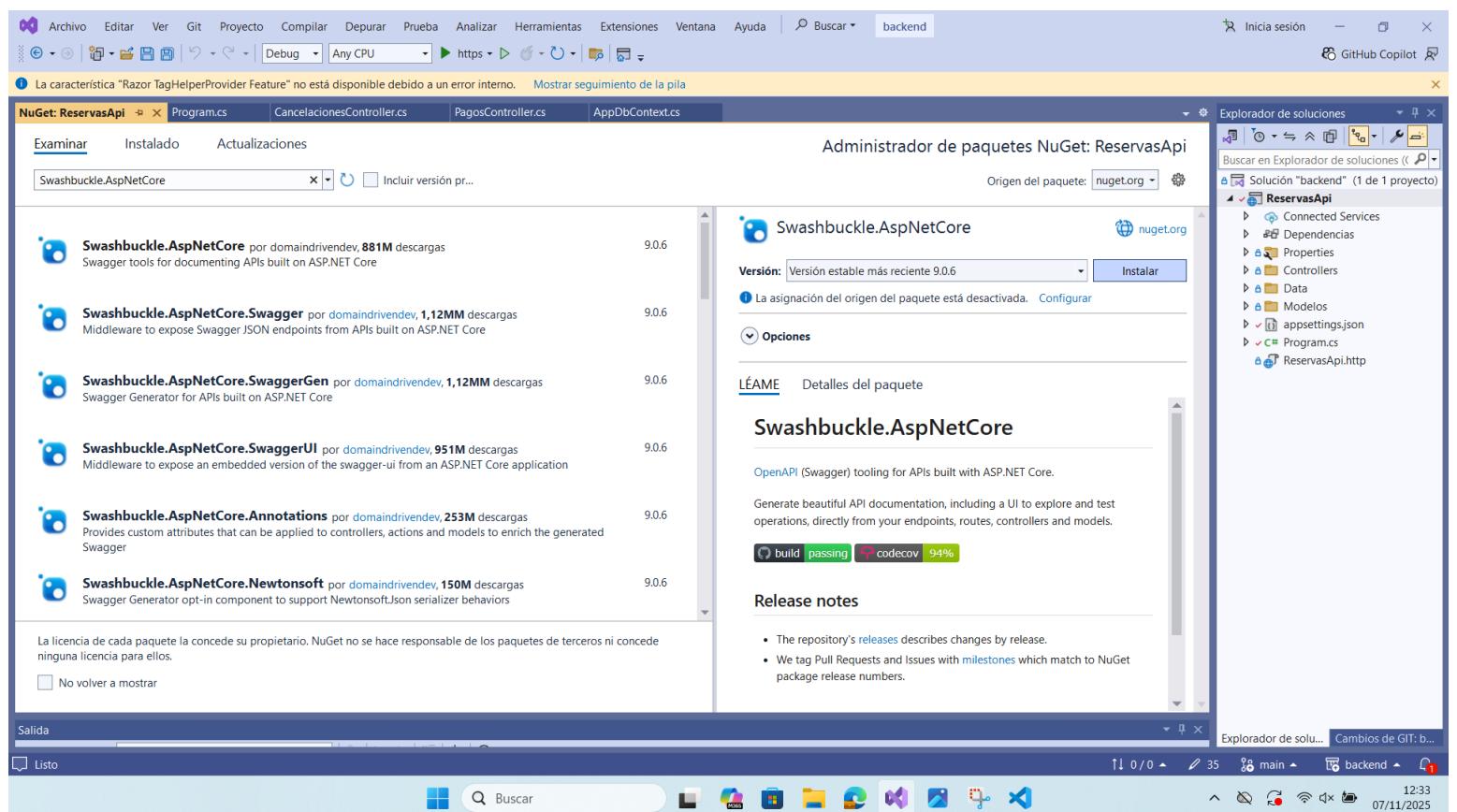
```

Según chatgpt, ese es “the new minimal API template that .NET creates automatically when you make a new Web Api project (it’s the weather forecast demo)”.

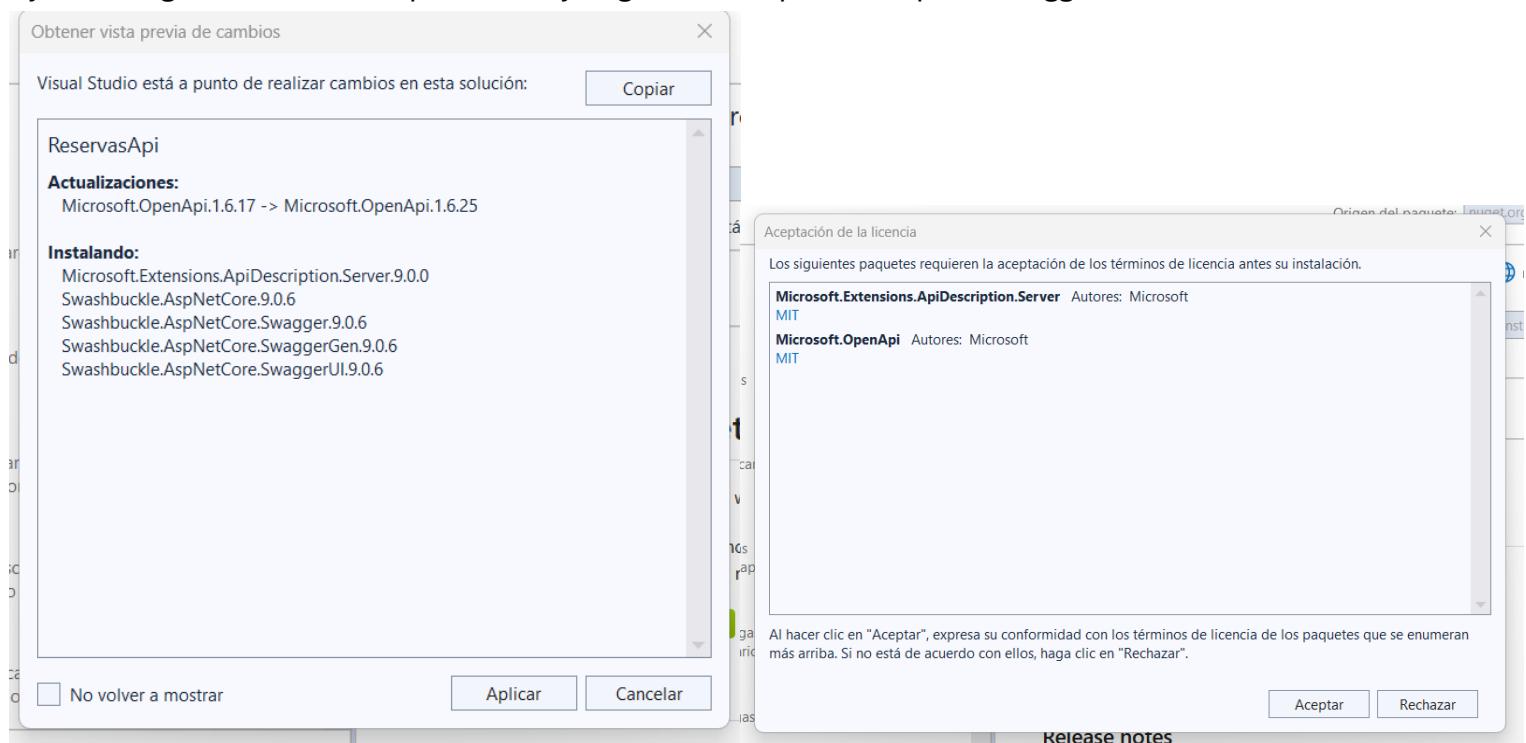
You now need to replace that Program.cs default code so your app can

- **Register** your **AppDbContext** (so EFCore can talk to SQL Server).
- **Enable** your **controllers** (ReservasController, UsuariosController, etc.). Recordar que tengo uno por modelo.
- **Configure Swagger** (for testing your endpoints easily)

- Set up HTTPS, routing and the basic pipeline.



By installing Swashbuckle.AspNetCore, you get the complete setup for Swagger in ASP.NET Core.



NuGet: ReservasApi

Program.cs CancelacionesController.cs PagosController.cs AppDbContext.cs

Examinar **Instalado** Actualizaciones

Swashbuckle.AspNetCore

Paquetes de nivel superior (1)

Swashbuckle.AspNetCore por domaindrivendev 9.0.6
Swagger tools for documenting APIs built on ASP.NET Core

Paquetes transitivos (3)

Swashbuckle.AspNetCore.Swagger por domaindrivendev 9.0.6
Middleware to expose Swagger JSON endpoints from APIs built on ASP.NET Core

Swashbuckle.AspNetCore.SwaggerGen por domaindrivendev 9.0.6
Swagger Generator for APIs built on ASP.NET Core

Swashbuckle.AspNetCore.SwaggerUI por domaindrivendev 9.0.6
Middleware to expose an embedded version of the swagger-ui from an ASP.NET Core application

Ahora VS IDE ya me reconoce `AddSwaggerGen();` cosa que voy poner en Program.cs.

Connecting to my SQL Server container in DBeaver pre-running migrations con comandos dotnet

DBeaver 25.2.4 - <master> Script

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator Projects

master localhost:1433

Databases master

Schemas dbo

Tables Views Indexes Procedures Sequences Synonyms Triggers Data Types

Security Administrator

Script

SELECT name FROM sys.databases;

Results 1 ×

SELECT name FROM sys.data

name
master
tempdb
model
msdb

Value master

started Actions

ays ago

utes ago

utes ago

utes ago

utes ago

Showing 6 items

File - General

Name Bookmarks Dashboards Diagrams Scripts

DataSource

Record

Refresh Save Cancel Export data 200

4 row(s) fetched - 0.0s, on 2025-11-07 at 12:54:40

CET en Writable Smart Insert 1:32:31

Docker Desktop interface showing the Containers list. The sidebar on the left includes options like Ask Gordon, Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit, Docker Hub, Docker Scout, and Extensions. The main area displays container statistics: Container CPU usage (7.14% / 800%) and Container memory usage (1.23GB / 15.11GB). A search bar and a filter for 'Only show running containers' are present. The table lists six containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
	vibrant_carson	5f1347c184de	hello-world	-	0%	22 days ago	▶ ⋮ 🗑
	reservas-app	-	-	-	7.14%	7 minutes ago	▶ ⋮ 🗑
	frontend-1	3b4ebf8eb7d9	reservas-app-frontend	4200:80	0%	7 minutes ago	▶ ⋮ 🗑
	backend-1	7b0568d2e9de	reservas-app-backend	5000:8080	0.01%	7 minutes ago	▶ ⋮ 🗑
	sqlserver	ef940a47d551	mssql/server:2022-latest	1433:1433	7.13%	7 minutes ago	▶ ⋮ 🗑

Show charts

Si eso no está levantado, o por lo menos el sqlserver container, entonces al intentar desplegar en DBeaver:

DBeaver Database Navigator interface showing a connection to 'master' on localhost:1433.

Te salta este error:

Connect to 'master'

⚠ Connect to 'master'

The TCP/IP connection to the host localhost, port 1433 has failed. Error: "Connection refused: getsockopt. Verify the connection properties. Make sure that an instance of SQL Server is running on the host and accepting TCP/IP connections at the port. Make sure that TCP connections to the port are not blocked by a firewall."

Details >> OK

"make sure that an instance of sql server (en mi caso el sqlserver container) is running on the host."

Lo que acabo de comprobar es lo siguiente: NEXT PAGE

Before running migrations (your current expected state)

When you connect to your SQL Server container in DBeaver —
you should see:

1. The server connection itself works

- In DBeaver's Database Navigator:
 - You can expand your connection (e.g. `localhost,1433` or `db,1433`).
 - You see "Databases" listed under that connection.
 - That proves the SQL Server container is **running and reachable**.

2. Possibly a few system databases

These are **always present** in any SQL Server instance:

- `master`
- `model`
- `msdb`
- `tempdb`



"the sql server container is running and reachable"

Para ver master model msdb tempdb tuve que ir a:

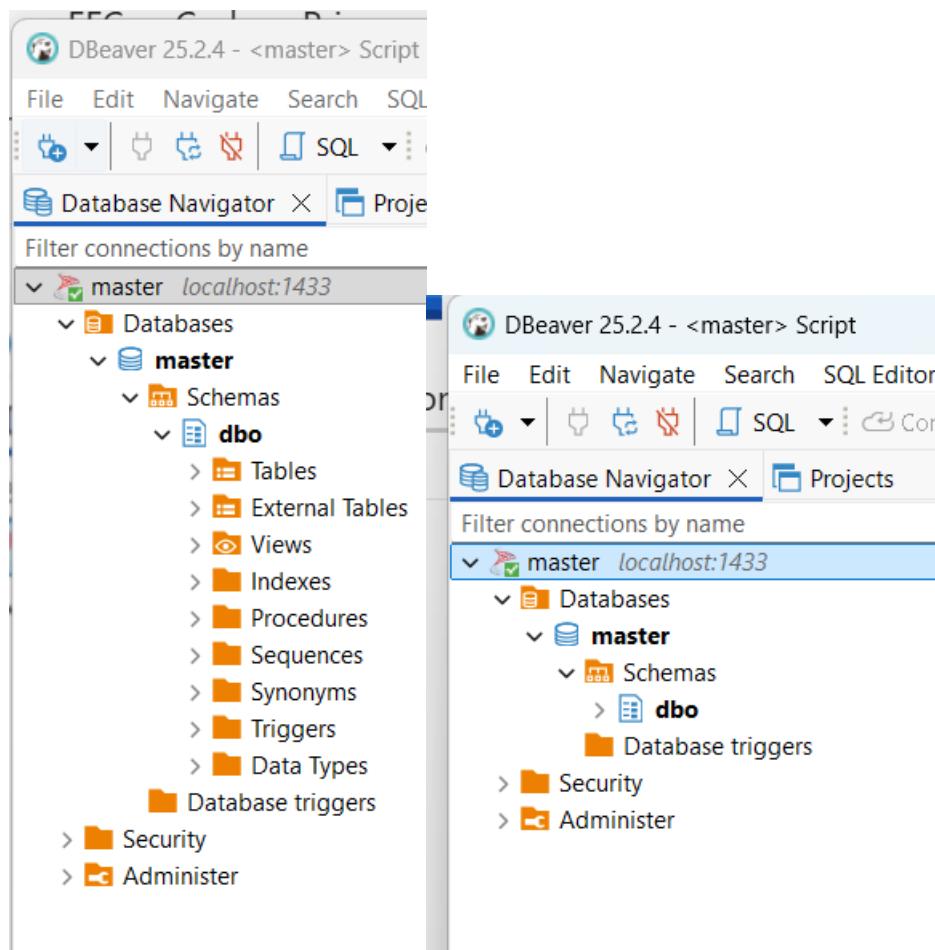
The screenshot shows the DBeaver interface. The Database Navigator pane on the left has a tree view. Under the master connection, the 'master' database is expanded, showing its schema and objects. The 'tempdb' entry is highlighted with a blue selection bar. The SQL Editor pane on the right displays the following SQL command:

```
SELECT name FROM sys.databases;
```

The results grid shows the following data:

name
master
tempdb
model
msdb

Dice chatgpt que como aún no fixen las migraciones, entonces “you should NOT yet see your custom database ReservasDb”.



sobre 'dbo' dice chatgpt que it stands for

Database Owner y que it is **the default schema that SQL Server users** for any objects (like tables, views, stored procedures, etc.) unless you specify another schema.

When SQL Server creates a new database (like 'master') -el que me aparece en negrita- or later –postmigrations– your ReservasDb, **it automatically creates one default schema called 'dbo'**.

Y hasta ahí lo relativo a qué debo poder ver ya en DBeaver pre-running migrations.

Ahora vamos con lo de running migrations que parece que es lo que va a lograr que aparezca la database ReservasDb. Recordar que este nombre para la database procede de / lo puse aquí: NEXT PAGE

```
{} appsettings.json 1 X docker-compose.yml
C: > Users > iago.cadavidgonzalez > Proyectos > reservas-app > backend > ReservasApi > {} appsettings.json > {} ConnectionStrings > DefaultConnection
1 {
2     "ConnectionStrings": {
3         "DefaultConnection": "Server=localhost,1433;Database=ReservasDb;User Id=sa;Password=dockEr!8;TrustServerCertificate=True;"
4     },
5
6     "Logging": {
7         "LogLevel": {
8             "Default": "Information",
9             "Microsoft.AspNetCore": "Warning"
10        }
11    },
12    "AllowedHosts": "*"
13}
14
```

Y aquí:

```
File Edit Selection View Go Run Search
{} appsettings.json 1 docker-compose.yml X
C: > Users > iago.cadavidgonzalez > Proyectos > reservas-app > docker-compose.yml
1 version: "3.9"
2
3     ▷Run All Services
4     services:
5         ▷ Run Service
6             frontend:
7                 build:
8                     context: ./frontend
9                     ports:
10                        - "4200:80"
11                     depends_on:
12                         - backend
13
14             ▷ Run Service
15             backend:
16                 build:
17                     context: ./backend
18                     ports:
19                         - "5000:8080"
20                     environment:
21                         - ASPNETCORE_ENVIRONMENT=Development
22                         - ConnectionStrings_DefaultConnection=Server=db;Database=ReservasDb;User Id=sa;Password=dockEr!8;TrustServerCertificate=True;
23                     depends_on:
24                         - db
```

NEXT PAGE

Time to run migrations with dotnet to create my ReservasDb

4 Run migrations (only if you haven't already)

In your terminal (in the `ReservasApi` folder, where the `.csproj` file is), run:

bash

```
dotnet ef migrations add InitialCreate  
dotnet ef database update
```

This will:

- Generate SQL migration scripts for your model classes.
 - Create the database `ReservasDb` and its tables inside your SQL Server container.
- You should now see your tables (`Reservas`, `Usuarios`, etc.) inside DBeaver.

Y más en concreto

supongo que deberán estar dentro del dbo de ReservasDb.

Sobre “in your terminal” dice que se refiere a

any command-line interface **where you can run ‘.NET CLI’ commands (‘dotnet...’).**

On Windows, that can be one of several options

–all work the same as long as the dotnet command is available in your PATH.

dotnet ef migrations add `InitialCreate` (is just the name you give to the migration)

is a key command in the EFC workflow. It is the command that creates a migration, which is EFC’s way of saying:

“generate a C# file that describes all the database tables and columns needed to match my model classes.”

So EFC looks at my model classes (Reserva, Usuario, Instalacion, etc.) and my AppDbContext, and it figures out what SQL would be required to create the database schema that fits those classes.

En cuanto a **dotnet ef database update**, “this is the next step after creating your migration,

and it’s where your code changes actually *hit the database*.

This command tells EFC to take all the migrations I’ve created so far –including InitialCreate–

and apply them to the actual SQL Server database.

En otras palabras, it **runs the SQL** that creates my tables, fks and indexes in the real db my app connects to.

NEXT PAGE

```
C:\Windows\System32\cmd.e > dotnet ef migrations add PrimeraMigracion
Microsoft Windows [Versión 10.0.26200.6901]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet ef migrations add PrimeraMigracion
No se pudo ejecutar porque no se encontró el comando o archivo especificado.
Entre las posibles razones para esto se incluyen:
* Escribió de manera incorrecta un comando dotnet integrado.
* Tenía previsto ejecutar un programa .NET, pero dotnet-ef no existe.
* Tuvo la intención de ejecutar una herramienta global, pero no se encontró un ejecutable con el prefijo dotnet con este nombre en la ruta.
```

Según chatgpt, this means the **Entity Framework CLI tool (dootnet-ef)** is NOT installed or NOT available globally.

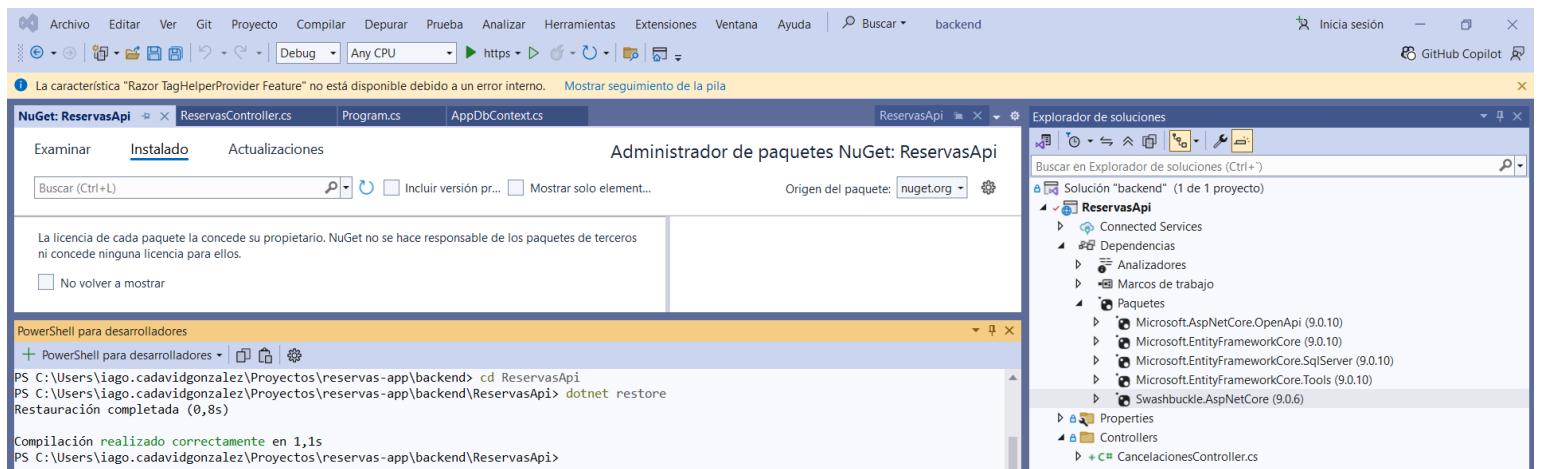
To install the efc cli globally on your system, run the following (from any directory) in your windows command prompt or powershell:

dotnet tool install --global dotnet-ef

```
C:\Windows\System32\cmd.e > dotnet tool install --global dotnet-ef
C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet tool install --global dotnet-ef
Puede invocar la herramienta con el comando siguiente: dotnet-ef
La herramienta "dotnet-ef" (versión '9.0.10') se instaló correctamente.
```

F

Gracias a este **dotnet restore**, se piró el triángulo amarillo con exclamación blanca que me aparecía en la dependencia del swashbuckle y que puede que fuese razón de problemas con dotnet build



Exito en ejecutar con windows cmd estando en ReservasApi folder el dotnet build

```
C:\Windows\System32\cmd.e > dotnet build
C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet build
Restauración completada (0,3s)
  ReservasApi realizado correctamente (7,9s) → bin\Debug\net9.0\ReservasApi.dll

Compilación realizado correctamente en 8,9s

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>
```

F


```
C:\Windows\System32\cmd.e X + V

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet build
Restauración completada (0,3s)
  ReservasApi realizado correctamente (7,9s) → bin\Debug\net9.0\ReservasApi.dll
Compilación realizado correctamente en 8,9s

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet ef migrations add PrimeraMigracion
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>
```

That means my migrations system is working perfectly
and EFCore has generated the code needed to create my database schema.

Ahora toca ejecutar **dotnet ef database update** tmb con curr dir en ReservasApi folder y teniendo en cuenta que database is NOT a literal name of any database, it is just the subcommand name in the Entity Framework CLI; the general command pattern is: **dotnet ef <category> <action>**

Ef tells .NET to use the Entity Framework CLI tool.

database is the category of commands that deal with databases.

Update is the action, meaning “apply migrations to the database”.

```
C:\Windows\System32\cmd.e X + V

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet build
Restauración completada (0,3s)
  ReservasApi realizado correctamente (7,9s) → bin\Debug\net9.0\ReservasApi.dll
Compilación realizado correctamente en 8,9s

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet ef migrations add PrimeraMigracion
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet ef database update
Build started...
Build succeeded.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
  Executed DbCommand (374ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
    CREATE DATABASE [ReservasDb];
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
  Executed DbCommand (249ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
    IF SERVERPROPERTY('EngineEdition') >> 5
    BEGIN
      ALTER DATABASE [ReservasDb] SET READ_COMMITTED_SNAPSHOT ON;
    END;
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
  Executed DbCommand (6ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT 1
info: Microsoft.EntityFrameworkCore.Migrations[20411]
  Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
  Executed DbCommand (20ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
  DECLARE @result int;
  EXEC @result = sp_getapplock @Resource = '_EFMigrationsLock', @LockOwner = 'Session', @LockMode = 'Exclusive';
  SELECT @result
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
  Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
  IF OBJECT_ID('[_EFMigrationsHistory]') IS NULL
  BEGIN
    CREATE TABLE [_EFMigrationsHistory] (
      [MigrationId] nvarchar(150) NOT NULL,
      [ProductVersion] nvarchar(32) NOT NULL,
      CONSTRAINT [PK__EFMigrationsHistory] PRIMARY KEY ([MigrationId])
    );
  END;
```



```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT OBJECT_ID(N'[_EFMigrationsHistory]');
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (18ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT [MigrationId], [ProductVersion]
      FROM [__EFMigrationsHistory]
      ORDER BY [MigrationId];
info: Microsoft.EntityFrameworkCore.Migrations[20402]
      Applying migration '20251107130944_PrimeraMigracion'.
      Applying migration '20251107130944_PrimeraMigracion'.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Instalaciones] (
          [InstalacionId] int NOT NULL IDENTITY,
          [Localizacion] nvarchar(max) NOT NULL,
          [CapacidadMaxima] int NOT NULL,
          [PrecioHora] float NOT NULL,
          [Tipo] bit NOT NULL,
          CONSTRAINT [PK_Instalaciones] PRIMARY KEY ([InstalacionId])
      );
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (4ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Usuarios] (
          [UsuarioId] nvarchar(450) NOT NULL,
          [NombreCompleto] nvarchar(max) NOT NULL,
          [Email] nvarchar(max) NOT NULL,
          [Telefono] nvarchar(max) NOT NULL,
          [Domicilio] nvarchar(max) NOT NULL,
          [FechaNacimiento] datetime2 NOT NULL,
          [FechaRegistro] datetime2 NOT NULL,
          [Contrasinal] nvarchar(max) NOT NULL,
          CONSTRAINT [PK_Usuarios] PRIMARY KEY ([UsuarioId])
      );
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Reservas] (
          [ReservaId] int NOT NULL IDENTITY,
          [NumeroAsistentes] int NOT NULL,
          [FechaHora] datetime2 NOT NULL,
          [Dni] nvarchar(max) NOT NULL,
          [EspacioId] int NOT NULL,
```

Sigue en next page

```

CREATE TABLE [Reservas] (
    [ReservaId] int NOT NULL IDENTITY,
    [NumeroAsistentes] int NOT NULL,
    [FechaHora] datetime2 NOT NULL,
    [Dni] nvarchar(max) NOT NULL,
    [EspacioId] int NOT NULL,
    [InstalacionId] int NULL,
    [UsuarioId] nvarchar(450) NULL,
    CONSTRAINT [PK_Reservas] PRIMARY KEY ([ReservaId]),
    CONSTRAINT [FK_Reservas_Instalaciones_InstalacionId] FOREIGN KEY ([InstalacionId]) REFERENCES [Instalaciones] ([InstalacionId]),
    CONSTRAINT [FK_Reservas_Usuarios_UsuarioId] FOREIGN KEY ([UsuarioId]) REFERENCES [Usuarios] ([UsuarioId])
);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (4ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [Cancelaciones] (
    [CancelacionId] int NOT NULL IDENTITY,
    [ReservaId] int NOT NULL,
    [Fecha] datetime2 NOT NULL,
    [Motivo] nvarchar(max) NOT NULL,
    CONSTRAINT [PK_Cancelaciones] PRIMARY KEY ([CancelacionId]),
    CONSTRAINT [FK_Cancelaciones_Reservas_ReservaId] FOREIGN KEY ([ReservaId]) REFERENCES [Reservas] ([ReservaId]) ON DELETE CASCADE
);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [Pagos] (
    [PagoId] int NOT NULL IDENTITY,
    [ReservaId] int NOT NULL,
    [Monto] float NOT NULL,
    [Metodo] int NOT NULL,
    [FechaHora] datetime2 NOT NULL,
    CONSTRAINT [PK_Pagos] PRIMARY KEY ([PagoId]),
    CONSTRAINT [FK_Pagos_Reservas_ReservaId] FOREIGN KEY ([ReservaId]) REFERENCES [Reservas] ([ReservaId]) ON DELETE CASCADE
);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (4ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_Cancelaciones_ReservaId] ON [Cancelaciones] ([ReservaId]);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (6ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_Pagos_ReservaId] ON [Pagos] ([ReservaId]);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (8ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_Reservas_InstalacionId] ON [Reservas] ([InstalacionId]);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_Reservas_UsuarioId] ON [Reservas] ([UsuarioId]);

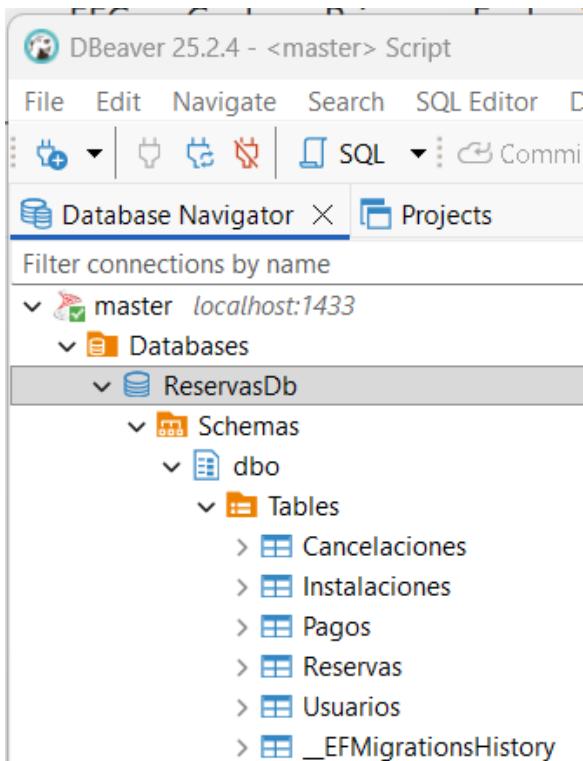
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_Reservas_UsuarioId] ON [Reservas] ([UsuarioId]);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (11ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES ('N'20251107130944_PrimeraMigracion', N'9.0.10');
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
DECLARE @result int;
EXEC @result = sp_releaseapplock @Resource = '__EFMigrationsLock', @LockOwner = 'Session';
SELECT @result
Done.

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>

```

Ahora (post ejecutar los dos comandos dotnet ef creadores de la bd) en Dbeaver refresco Databases y exitósamente aparece la nueva bd con sus tablas

NEXT PAGE



F

En 1o now it's time to [test your API endpoints](#) using **Swagger UI**, which will act as your friendly [in-browser API client](#).

Al igual que cuando testeé la conexión al sql server container, ahora tmb voy a hacer dos tests, uno con el dotnet run que seica es con el backend corriendo en local y el otro con el backend corriendo en su docker container.

Primero voy a testear endpoints vía dotnet run osease con backend corriendo en local

Component	Run locally or in Docker?	Should be running?
SQL Server	Docker	<input checked="" type="checkbox"/> Yes
Backend (ASP.NET API)	Locally (<code>dotnet run</code>)	<input checked="" type="checkbox"/> Yes
Frontend	Docker (later)	<input checked="" type="checkbox"/> No (optional for now)

Para ello chatgpt y yo estamos de acuerdo en que el único contenedor que necesito tener corriendo es el del sqlserver.

NEXT PAGE

The screenshot shows the Docker Desktop interface. On the left, there's a sidebar with various options like Ask Gordon, Containers (which is selected), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit, Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' with a 'Give feedback' link. It displays container usage statistics: 'Container CPU usage' at 2.36% / 800% (8 CPUs available) and 'Container memory usage' at 786.2MB / 15.11GB. A 'Show charts' button is also present. Below these stats is a search bar and a filter option 'Only show running containers'. The main table lists six containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
	vibrant_carson	5f1347c184de	hello-world	-	0%	22 days ago	
	reservas-app	-	-	-	1.55%	4 minutes ago	
	frontend-1	3b4ebf8eb7d9	reservas-app-frontend	4200:80	0%	3 hours ago	
	backend-1	7b0568d2e9de	reservas-app-backend	5000:8080	0%	3 hours ago	
	sqlserver	ef940a47d551	mssql/server:2022-latest	1433:1433	1.55%	4 minutes ago	

Showing 6 items

F

```
C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet run
Usando la configuración de inicio de C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi\Properties\launchSettings.json...
Compilando...
info: Microsoft.Hosting.Lifetime[14]
  Now listening on: http://localhost:5184
info: Microsoft.Hosting.Lifetime[0]
  Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
  Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
  Content root path: C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi
```

Según chatgpt, ese output (del dotnet run ejecutado desde la carpeta padre del .csproj de mi backend) significa que “perfect -> that means **your backend API is up and running locally and listening at <http://localhost:5184>**”, so now you are just one step away from testing your endpoints through Swagger UI.

1. Open Swagger in browser: go to: <http://localhost:5184/swagger>

Mi resultado a continuación:

NEXT PAGE

Mi Swagger con backend corriendo en local osease corriendo con dotnet run

localhost:5184/swagger/index.html

Swagger. Supported by SMARTBEAR

Select a definition ReservasApi v1

ReservasApi 1.0 OAS 3.0

http://localhost:5184/swagger/v1/swagger.json

Cancelaciones

POST /api/Cancelaciones

GET /api/Cancelaciones

GET /api/Cancelaciones/{id}

PUT /api/Cancelaciones/{id}

DELETE /api/Cancelaciones/{id}

Instalaciones

POST /api/Instalaciones

GET /api/Instalaciones

GET /api/Instalaciones/{id}

PUT /api/Instalaciones/{id}

DELETE /api/Instalaciones/{id}

Pagos

POST /api/Pagos

GET /api/Pagos

GET /api/Pagos/{id}

PUT /api/Pagos/{id}

DELETE /api/Pagos/{id}

NEXT PAGE

Reservas

POST	/api/Reservas	▼
GET	/api/Reservas	▼
GET	/api/Reservas/{id}	▼
PUT	/api/Reservas/{id}	▼
DELETE	/api/Reservas/{id}	▼

Usuarios

POST	/api/Usuarios	▼
GET	/api/Usuarios	▼
GET	/api/Usuarios/{id}	▼
PUT	/api/Usuarios/{id}	▼
DELETE	/api/Usuarios/{id}	▼

F

Schemas	^
Cancelacion >	
Instalacion >	
MetodoPago >	
Pago >	
Reserva >	
Usuario >	

Nótese que con Schemas Swagger se refiere a los models c sharp (lo sé porque pone MetodoPago) de las tablas:

NEXT PAGE

```
▲ 🔒 Modelos
  ▷ + C# Cancelacion.cs
  ▷ + C# Instalacion.cs
  ▷ + C# MetodoPago.cs
  ▷ + C# Pago.cs
  ▷ + C# Reserva.cs
  ▷ + C# Usuario.cs
```

Metodopago no es realmente una tabla, sino un tipo de un campo de la tabla/modelo Pago.cs.

"If everything is wired correctly, you will see Swagger's interactive interface –with sections for each controller" (recordar que hice un controller por cada modelo de tabla)

```
▲ 🔒 Controllers
  ▷ + C# CancelacionesController.cs
  ▷ + C# InstalacionesController.cs
  ▷ + C# PagosController.cs
  ▷ + C# ReservasController.cs
  ▷ + C# UsuariosController.cs
```

Eso que vengo de comprobar se corresponde con esto que me dijo chatgpt:

When you run your backend locally:

- Your backend app is **not inside Docker** — it's running directly on your host.
- It needs to talk to your **SQL Server**, which *is* inside Docker.
- Your SQL container exposes port **1433** to your host (thanks to your `ports`:
"1433:1433" in docker-compose).
- So your backend connects to it via the connection string:

```
json
Copy code
"Server=localhost,1433;Database=ReservasDb;User Id=sa;Password=MyStrong!Passw
```

That's all you need.

Your backend will connect through `localhost`  `.1433`, and it'll work perfectly.

Backend container and frontend containers stopped for this test (de los endpoints con backend corriendo en local).

Y tmb con ese mismo test se corresponde esto:

Okay, my sqlserver container is running and from ReservasApi folder (which in my case is the father of the .csproj file of my backend) I've just run "dotnet run" and got the following output:

```
Usando la configuración de inicio de
C:\Users\iago.cadavidgonzalez\Proyectos\reservas-
app\backend\ReservasApi\Properties\launchSettings.json...
Compilando...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5184
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path:
C:\Users\iago.cadavidgonzalez\Proyectos\reservas-
app\backend\ReservasApi
```

Perfect 🎉 — that means your backend API is **up and running locally** and listening at

👉 <http://localhost:5184> ↴

“So now you’re just one step away from testing your endpoints through Swagger UI, para ello:”

1 Open Swagger in your browser

Go to:

```
bash
```

 Copy code

```
http://localhost:5184/swagger
```

If everything is wired correctly, you'll see Swagger's interactive interface — with sections for each controller (e.g. Reservas, Usuarios, etc.).

2 Expand one controller (for example, Reservas)

Click the "GET /api/reservas" endpoint to expand it.

You should see a blue "Try it out" button.

3 Click "Try it out" → then click "Execute"

Swagger will send a real HTTP GET request to your API at:

```
bash
```

 Copy code

```
http://localhost:5184/api/reservas
```

If your database is empty (fresh after migrations), the response will be:

```
json
```

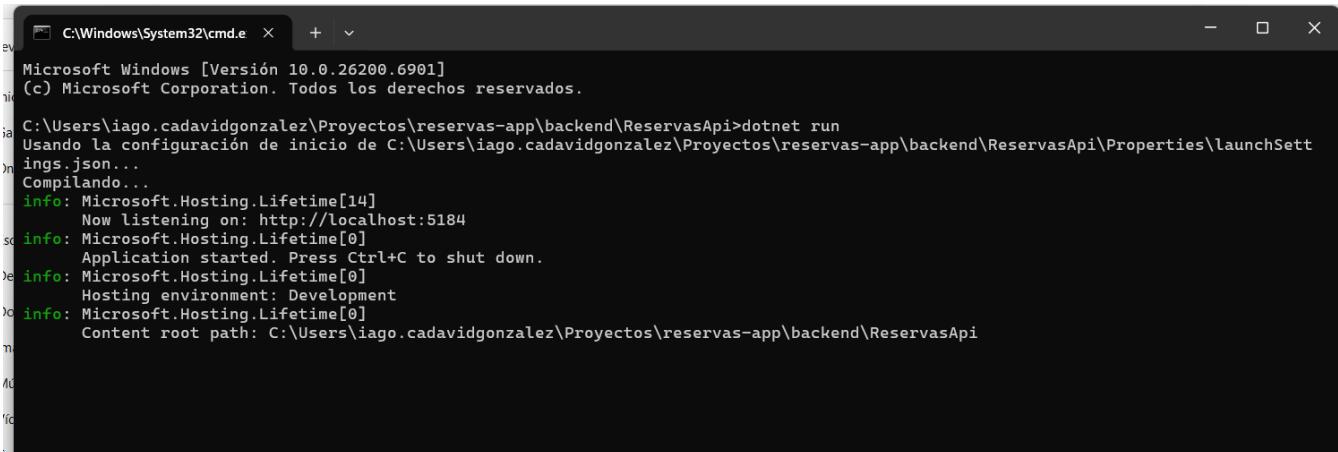
 Copy code

```
[]
```



F

<input type="checkbox"/>	<input checked="" type="radio"/>	reservas-app	-	-	-	N/A	1 second ago			
<input type="checkbox"/>	<input type="radio"/>	frontend-1	3b4ebf8eb7d9	reservas-app-frontend	4200:80	N/A	3 days ago			
<input type="checkbox"/>	<input type="radio"/>	backend-1	7b0568d2e9de	reservas-app-backend	5000:8080	N/A	3 days ago			
<input type="checkbox"/>	<input checked="" type="radio"/>	sqlserver	ef940a47d551	mssql/server:2022-latest	1433:1433	N/A	1 second ago			



```
C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Versión 10.0.26200.6901]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi>dotnet run
Usando la configuración de inicio de C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi\Properties\launchSettings.json...
Compilando...
info: Microsoft.Hosting.Lifetime[14]
  Now listening on: http://localhost:5184
info: Microsoft.Hosting.Lifetime[0]
  Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
  Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
  Content root path: C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi
```

blinking el cursor justo después del último info.

Queda

The screenshot shows a Swagger UI interface for a REST API. At the top, the URL is localhost:5184/swagger/index.html. Below the header, there's a blue bar with the method 'GET' and the endpoint '/api/Reservas'. To the right of the endpoint is a 'Try it out' button. Underneath the bar, there's a section for 'Parameters' which says 'No parameters'. Then there's a section for 'Responses' with a table. The table has columns for 'Code', 'Description', and 'Links'. A single row is shown for status code 200 with the description 'OK'. In the 'Links' column for this row, it says 'No links'. Below the table, there's a dropdown for 'Media type' set to 'text/plain', with a note that it controls the Accept header. There are also 'Example Value' and 'Schema' buttons. A large black box covers the response body area, which contains a JSON array of objects representing reservation data.

```
[{"reservaId": 0, "numeroAsistentes": 0, "fechaHora": "2025-11-10T07:12:49.429Z", "dni": "string", "espacioId": 0}]
```

Al clicar en **Try it out** me aparece botón azul que pone **Execute** y que chatgpt me recomienda clicar en él, cause that is what actually **sends a live HTTP request to your backend API** –in your case, to:

GET <http://localhost:5184/api/reservas>

Una vez clico en Execute, se ve esto:

NEXT PAGE

localhost:5184/swagger/index.html

GET /api/Reservas

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5184/api/Reservas' \
-H 'accept: text/plain'
```

Request URL

<http://localhost:5184/api/Reservas>

Server response

Code	Details
200	<p>Response body</p> <pre>[]</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Mon,10 Nov 2025 07:15:13 GMT server: Kestrel transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	OK	No links

Media type

[text/plain](#) ▾

Controls Accept header.

Example Value | Schema

```
[
  {
    "reservaId": 0,
    "numeroAsistentes": 0,
    "fechahora": "2025-11-10T07:15:14.068Z",
    "dni": "string",
    "espacioId": 0
  }
]
```

Insisto, esa captura es post-clicar en Execute, pero nótese que Execute sigue apareciendo aun habiendo ya clicado en él.

Lo importante es que el response body es el esperado por chatgpt osease []

En concreto me dijo “if your database is empty (fresh after migrations), the response will be []”.

Ese [] means the endpoint works and your backend successfully connected to SQL Server.

NEXT PAGE

4 Test a POST /api/reservas

Expand that section, click "Try it out", and you'll see a JSON body example.

You can edit it to create a new reservation, for example:

```
json Copy code
{
  "numeroAsistentes": 3,
  "fechaHora": "2025-11-07T10:00:00",
  "dni": "12345678A",
  "espacioId": 1
}
```

Then click "Execute" — if it succeeds, you'll get a `201 Created` response with the new object (including `reservaId`).

↓

Él en ese ejemplo de Edit NO pone Reservald, pero a mí sí me aparece, como en breves mostraré.

```
// POST: api/reservas
// CreateReserva(Reserva nuevareserva)
// The following is the "create" (una nueva reserva) operation. Create reserva.
[HttpPost] // matches POST /api/reservas
0 referencias
public async Task<ActionResult<Reserva>> CrearReserva(Reserva nuevareserva)
{ // the body of the request must contain a JSON representation of a Reserva (de la que quieres crear, supongo).
    _contexto.Reservas.Add(nuevareserva); // EFCore adds it (la nueva reserva) to the database (Add + SaveChangesAsync()).
    await _contexto.SaveChangesAsync();

    return CreatedAtAction(nameof(GetReserva), new { id = nuevareserva.ReservaId }, nuevareserva);
} // returns '201 Created' with the new object (la nueva reserva) and a Location header pointing to /api/reservas/{id}
```

NEXT PAGE

localhost:5184/swagger/index.html

POST /api/Reservas

Parameters

No parameters

Request body

application/json

```
{ "reservaId": 0, "numeroAsistentes": 0, "fechaHora": "2025-11-10T07:35:54.654Z", "dni": "string", "espacioId": 0 }
```

Execute

Responses

Code	Description	Links
200	OK	No links

Media type

text/plain

Controls Accept header.

Example Value | Schema

```
{ "reservaId": 0, "numeroAsistentes": 0, "fechaHora": "2025-11-10T07:35:54.659Z", "dni": "string", "espacioId": 0 }
```

Captura encima desta línea es justo después de dar en Try it out en el Post de tabla de reservas.

Edit Value | Schema

```
{  
  "reservaId": 1,  
  "numeroAsistentes": 3,  
  "fechaHora": "2025-11-10T07:35:54.654Z",  
  "dni": "12345678A",  
  "espacioId": 1  
}
```

NEXT PAGE

Doy en Execute y da error porque seica no puedo insertar explícitamente el id de la reserva:

localhost:5184/swagger/index.html

POST /api/Reservas

Parameters

No parameters

Request body

application/json

Edit Value | Schema

```
{  
    "reservaId": 1,  
    "numeroAsistentes": 3,  
    "fechaHora": "2025-11-10T07:35:54.654Z",  
    "dni": "12345678A",  
    "espacioId": 1  
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \  
'http://localhost:5184/api/Reservas' \  
-H 'accept: text/plain' \  
-H 'Content-type: application/json' \  
-d '{  
    "reservaId": 1,  
    "numeroAsistentes": 3,  
    "fechaHora": "2025-11-10T07:35:54.654Z",  
    "dni": "12345678A",  
    "espacioId": 1  
}'
```

Request URL

http://localhost:5184/api/Reservas

Sigue en NEXT PAGE

Server response

Code	Details
500 Undocumented	Error: Internal Server Error

Response body

```
Microsoft.EntityFrameworkCore.DbUpdateException: An error occurred while saving the entity changes. See the inner exception for details.
---> Microsoft.Data.SqlClient.SqlException (0x80131904): Cannot insert explicit value for identity column in table 'Reservas' when IDENTITY_INSERT is set to OFF.
   at Microsoft.Data.SqlClient.SqlCommand.<>c.<ExecuteDbDataReaderAsync>b_211_0(Task`1 result)
   at System.Threading.Tasks.ContinuationResultFromTask`2.InnerInvoke()
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object state)
--- End of stack trace from previous location ---
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object state)
   at System.Threading.Tasks.Task.ExecuteWithThreadLocal(Task& currentTaskSlot, Thread threadPoolThread)
--- End of stack trace from previous location ---
   at Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteNonQuery(RelationalCommandParameterObject parameterObject, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteNonQuery(RelationalCommandParameterObject parameterObject, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Update.ReaderModificationCommandBatch.ExecuteNonQuery(IRelationalConnection connection, CancellationToken cancellationToken)
ClientConnectionId:c029c525-f6bf-4900-b74d-0b92955008c8
Error Number:544,State::1,Class::16
--- End of inner exception stack trace ---
   at Microsoft.EntityFrameworkCore.Update.ReaderModificationCommandBatch.ExecuteNonQuery(IRelationalConnection connection, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Update.Internal.SqlServerModificationCommandBatch.ExecuteNonQuery(IRelationalConnection connection, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Update.Internal.BatchExecutor.ExecuteNonQuery(IEnumerable`1 commandBatches, IRelationalConnection connection, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Update.Internal.BatchExecutor.ExecuteNonQuery(IEnumerable`1 commandBatches, IRelationalConnection connection, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Update.Internal.BatchExecutor.ExecuteNonQuery(IEnumerable`1 commandBatches, IRelationalConnection connection, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Storage.RelationalDatabase.SaveChangesSync(IList`1 entries, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.ChangeTracking.Internal.StateManager.SaveChangesSync(IList`1 entriesToSave, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.ChangeTracking.Internal.StateManager.SaveChangesSync(IList`1 entriesToSave, Boolean acceptAllChangesOnSuccess, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.SqlServer.Storage.Internal.SqlServerExecutionStrategy.ExecuteAsync(IState,TResult)(IState state, Func`4 operation, Func`4 verifySucceeded, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.Update.ChangeTrackingStrategy.SaveChangesAsync(Boolean acceptAllChangesOnSuccess, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.DbContext.SaveChangesAsync(Boolean acceptAllChangesOnSuccess, CancellationToken cancellationToken)
   at Microsoft.EntityFrameworkCore.DbContext.SaveChangesAsync(Boolean acceptAllChangesOnSuccess, CancellationToken cancellationToken)
   at ReservasApi.Controllers.ReservasController.CrearReserva(Reserva nuevareserva) in C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi\Controllers\ReservasController.cs:line 32
```

Download

Response headers

```
content-type: text/plain; charset=utf-8
date: Mon, 10 Nov 2025 07:40:20 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	No links

Media type

text/plain

Controls Accept header.

Example Value | Schema

```
{
  "reservaId": 0,
  "numeroAsistentes": 0,
  "fechaHora": "2025-11-10T07:40:20.174Z",
  "dni": "string",
  "espacioId": 0
}
```

En captura superior es donde pone lo de que **no puedo insertar explícitamente id de reserva**.

Response body

```
at Microsoft.EntityFrameworkCore.DbContext.SaveChangesAsync(Boolean acceptAllChangesOnSuccess, CancellationToken cancellationToken)
at Reservasapi.Controllers.ReservasController.CrearReserva(Reserva nuevareserva) in C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app\backend\ReservasApi\Controllers\ReservasController.cs:line 32
at lambda_method28(Closure, Object)
at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.AwaitableObjectResultExecutor.Execute(ActionContext actionContext, IActionResultTypeMapper mapper, ObjectMethodExecutor executor, Object controller, Object[] arguments)
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeActionMethodAsync>g__Awaited|12_0(ControllerActionInvoker invoker, ValueTask`1 actionResultValueTask)
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeNextActionFilterAsync>g__Awaited|10_0(ControllerActionInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted)
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Rethrow(ActionExecutedContext& failedContext)
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State`1 next, Scope`1 scope, Object`1 state, Boolean`1 isCompleted)
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeInnerFilterAsync>g__Awaited|13_0(ControllerActionInvoker invoker, Task lastTask, State next, Scope scope, Object`1 state, Boolean`1 isCompleted)
at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeFilterPipelineAsync>g__Awaited|20_0(ResourceInvoker invoker, Task lastTask, State next, Scope scope, Object`1 state, Boolean`1 isCompleted)
at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Awaited|17_0(ResourceInvoker invoker, Task task, IDisposable scope)
at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Awaited|17_0(ResourceInvoker invoker, Task task, IDisposable scope)
at Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invoke(HttpContext context)
at Swashbuckle.AspNetCore.SwaggerUI.SwaggerUIMiddleware.Invoke(HttpContext httpContext)
at Swashbuckle.AspNetCore.Swagger.Middleware.Invoke(HttpContext httpContext, ISwaggerProvider swaggerProvider)
at Microsoft.AspNetCore.Authentication.AuthenticationMiddleware.Invoke(HttpContext context)
at Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)
```

HEADERS

```
=====
Accept: text/plain
Connection: keep-alive
Host: localhost:5184
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.0.0 Safari/537.36 Edg/142.0.0.0
```

Response headers

```
content-type: text/plain; charset=utf-8
date: Mon, 10 Nov 2025 07:40:20 GMT
server: Kestrel
transfer-encoding: chunked
```

Download

F

HEADERS

```
=====
Accept: text/plain
Connection: keep-alive
Host: localhost:5184
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36 Edg/142.0.0.0
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en,en-US;q=0.9,en-GB;q=0.8,es;q=0.7,es-ES;q=0.6
Content-Type: application/json
Origin: http://localhost:5184
Referer: http://localhost:5184/swagger/index.html
Content-Length: 128
sec-ch-ua-platform: "Windows"
sec-ch-ua: "Chromium";v="142", "Microsoft Edge";v="142", "Not_A_Brand";v="99"
sec-ch-ua-mobile: ?0
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
```



Download

Pruebo entonces el post api reservas sin explicitar id de la reserva:

Edit Value | Schema

```
{
    "numeroAsistentes": 3,
    "fechaHora": "2025-11-10T07:35:54.654Z",
    "dni": "12345678A",
    "espacioId": 1
}
```

Doy en Execute

Resultado parece ser satisfactoro porque ahora aparece el código 201 (que según chatgpt es el que aparece si todo ha ido bien):

Responses**Curl**

```
curl -X 'POST' \
'http://localhost:5184/api/Reservas' \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-d '{
    "numeroAsistentes": 3,
    "fechaHora": "2025-11-10T07:35:54.654Z",
    "dni": "12345678A",
    "espacioId": 1
}'
```

**Request URL**

<http://localhost:5184/api/Reservas>

Server response

Code	Details
201	Response body

Undocumented

```
{
    "reservaId": 1,
    "numeroAsistentes": 3,
    "fechaHora": "2025-11-10T07:35:54.654Z",
    "dni": "12345678A",
    "espacioId": 1
}
```



Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon, 10 Nov 2025 07:45:33 GMT
location: http://localhost:5184/api/Reservas/1
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
------	-------------	-------

200 OK

No links

Media type

[text/plain](#) ▾

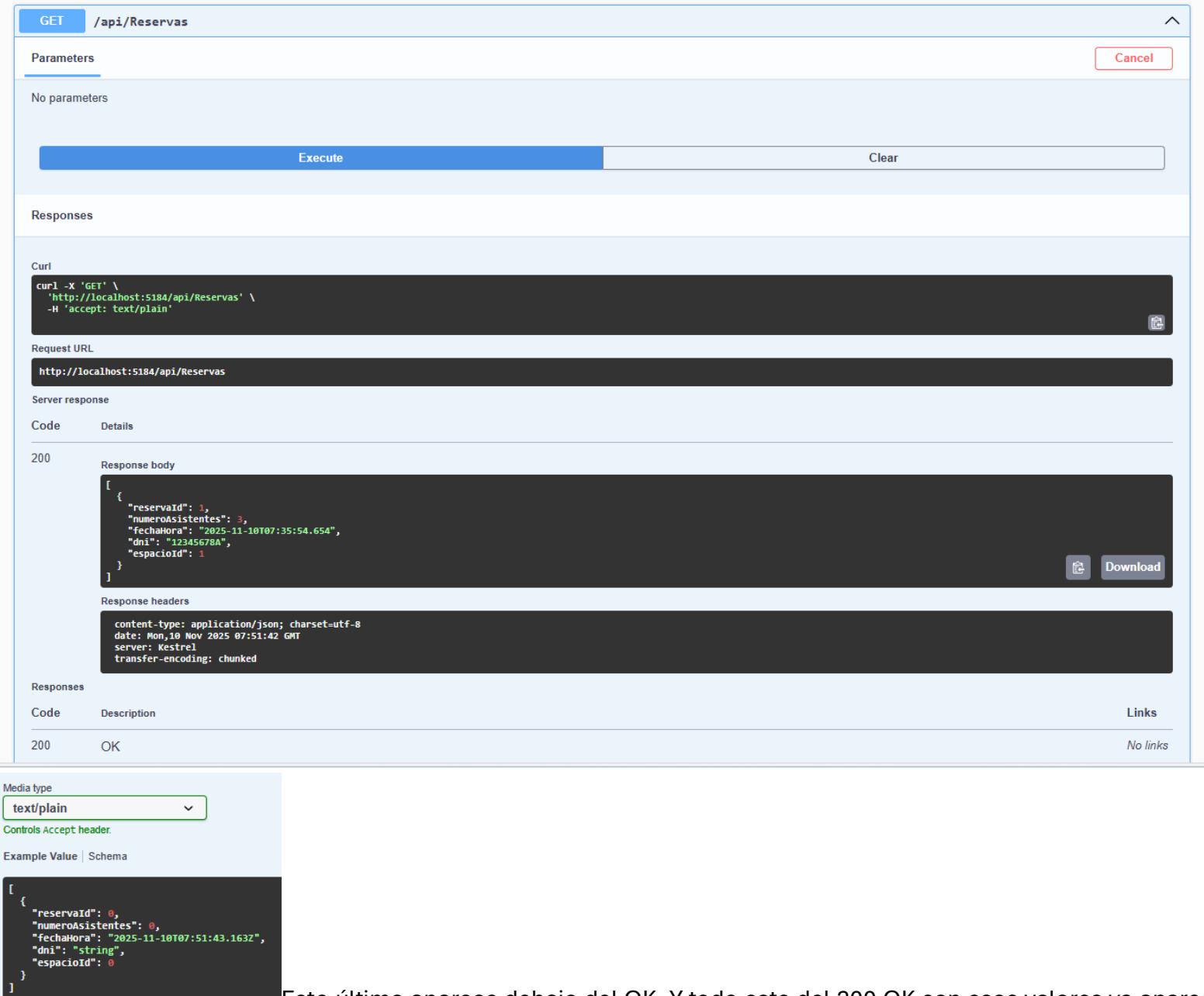
Controls Accept header.

Example Value | Schema

```
{
    "reservaId": 0,
    "numeroAsistentes": 0,
    "fechaHora": "2025-11-10T07:45:33.734Z",
    "dni": "string",
    "espacioId": 0
}
```

F

Fijarse que cuando explicité el id de reserva, devolvía Code 500 y Details: Error: Internal Server Error, VS ahora al no explicitar id de reserva (osease al dejar que la máquina lo ponga automáticamente), me da Code 201 y Details: “Response body”. Puso como idreserva el 1.
 (Lo del code 200 ok lo pone en ambos casos y con los mismos valores pa los campos, así que a esto no lo presto atención.)
 Finalmente, dice chatgpt “go back and run GET /api/reservas again, now you should see your new record returned in the list.” Vamos a verlo:



The screenshot shows a REST API testing interface with the following details:

- Method:** GET
- Endpoint:** /api/Reservas
- Parameters:** No parameters
- Buttons:** Execute (highlighted in blue), Clear, Cancel
- Responses:** Curl command and Request URL.

 - Curl:** curl -X 'GET' \ 'http://localhost:5184/api/Reservas' \ -H 'accept: text/plain'
 - Request URL:** http://localhost:5184/api/Reservas

- Server response:**
 - Code:** 200
 - Details:** Response body
 - Content:** A JSON array with one element representing a reservation. The JSON is:


```
[
            {
              "reservaId": 1,
              "numeroAsistentes": 3,
              "fechaHora": "2025-11-10T07:35:54.654",
              "dni": "12345678A",
              "espacioId": 1
            }
          ]
```
 - Headers:** content-type: application/json; charset=utf-8
date: Mon, 10 Nov 2025 07:51:42 GMT
server: Restrel
transfer-encoding: chunked
 - Buttons:** Copy, Download
- Responses:**

Code	Description	Links
200	OK	No links
- Media type:** text/plain
- Example Value:** A JSON object representing a reservation with all fields set to 0.

```
[
  {
    "reservaID": 0,
    "numeroAsistentes": 0,
    "fechaHora": "2025-11-10T07:51:43.163Z",
    "dni": "string",
    "espacioID": 0
  }
]
```

Esto último aparece debajo del OK. Y todo esto del 200 OK con esos valores ya aparecía antes cuando el resultado era solo [], así que no le presto atención a esto del 200 OK.

Faltaría comprobar toooodo lo demás, pero tardaría demasiado y además chatgpt no va más allá de las pruebas que acabo de hacer, así que doy por exitosa la prueba de los endpoints empleando swagger y con backend corriendo en local con dotnet run, así que ahora paro el dotnet run (que ejecuté en el cmd estando situado en ReservasApi folder, por ser padre del .csproj file de mi proyecto backend) con Control+C.

A continuación, toca probar algunos [endpoints con backend corriendo en su docker container](#)

“now that your API (i.e. mi ASP.NET Core backend API) runs correctly with dotnet run, you can test it inside its Docker container.”

Lo primero es asegurarnos de que tanto el container backend como el de sqlserver are running in Docker Desktop.

□	▼	●	reservas-app	-	-	-	1.81%	0 seconds ago	■	⋮	☒
□	○	frontend-1	3b4ebf8eb7d9	reservas-app-frontend	4200:80	0%	3 days ago	▶	⋮	☒	
□	●	backend-1	7b0568d2e9de	reservas-app-backend	5000:8080 ↗	0%	0 seconds ago	■	⋮	☒	
□	●	sqlserver	ef940a47d551	mssql/server:2022-latest	1433:1433 ↗	1.81%	52 minutes ago	■	⋮	☒	

Simplemente di en el play del backend (el sqlserver ya lo tenía corriendo de la prueba anterior)

Otra opción pa arrancar ambas cosas según chatgpt es:

“start them all at once from your docker-compose.yml file by running:

Docker compose up --build” pero supongo que esto tmb arrancaría el frontend, cosa que no debería necesitar pa este test.

En todo caso, ese docker compose up --build should be run in the root folder of my project –the one that has the docker-compose.yml file; en mi caso el padre dese file es reservas-app folder:

Nombre	Fecha de modificación	Tipo
backend	05/11/2025 8:35	Carpeta de archivos
frontend	04/11/2025 10:26	Carpeta de archivos
serviciocron	04/11/2025 12:25	Carpeta de archivos
docker-compose.yml	06/11/2025 10:45	Archivo de origen Yaml

A continuación, “**verify backend started correctly**; once it’s up, [look for a log like this in DockerDesktop: nginx](#)

Now listening on: <http://0.0.0.0:8080>

Or maybe:

nginx

Now listening on: <http://+:8080>

That means **your backend is running inside the container on port 8080.**"

Para acceder a ese log, **clico en el contenedor** (en docker desktop's container section) to open a detailed view for that container y **voy a Logs tab** en top bar: you'll now see your **container's real-time console output**, exactly what you'd see if you were running `dotnet run` locally.

Lo más parecido que a mí me aparece (con respecto al log que chatgpt dice que yo debería ver) es esto:

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:5000 ↗
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

Lo que chatgpt dice que yo debería ver es esto que muestro a continuación:

🔍 What you're looking for in those logs

Scroll through and look for something like:

pgsql

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://0.0.0.0:8080
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

This means:

- Your backend is **running successfully** inside the container.
- It's listening on **port 8080 inside Docker**.
- Because of your Docker port mapping (e.g., `8080:8080`), it's reachable from your host at:

👉 `http://localhost:8080/swagger`



Le paso a chatgpt todos los logs del backend container, chatgpt concluye lo siguiente:

NEXT PAGE

✳️ What your logs are saying

The key line is:

```
pgsql
```

warn: Microsoft.AspNetCore.Hosting.Diagnostics[15]
Overriding HTTP_PORTS '8080' and HTTPS_PORTS ''. Binding to values defined by URLs instead 'http://'
info: Microsoft.Hosting.Lifetime[14]
Now listening on: http://[::]:5000

Copy code

- That means your backend is **running fine inside the container**
- But it's **not listening on port 8080 anymore — it's listening on port 5000.**

ASP.NET Core is choosing `http://+:5000` because of environment variables (usually `ASPNETCORE_URLS` or `HTTP_PORTS`) that Docker passes in or overrides.

👉 What this means for you

Right now your backend is reachable **inside Docker** on port 5000.

If your `docker-compose.yml` maps that port like this:

```
yaml
```

ports:
- "8080:5000"

Copy code

Then you can access it from your host at:

👉 <http://localhost:8080/swagger>

But if your compose file says:



('`http://+:5000`' es lo que pone al final de la "key line")

But if your compose file says:

yaml

ports:

- "5000:5000"

Then you should open:

<http://localhost:5000/swagger>

Lo que mi docker-compose.yml pone a ese respecto es lo siguiente:

```
backend:
  build:
    context: ./backend
  ports:
    - "5000:8080"
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ConnectionStrings_DefaultConnectionString=Server=db;Database=ReservasDb;User Id=sa;Password=dockEr!8;TrustServerCertificate=True;
  depends_on:
    - db
```

Osease justo al revés de lo que en pág anterior dijo chatgpt. Se lo hago saber y me responde:

In your docker-compose.yml, you have:

Ports:

- "5000:8080"

The format is always:

<HOST_PORT>:<CONTAINER_PORT>

So your configuration means:

5000 is the **port** on your **host machine** (outside Docker)

8080 is the **port** inside the **container** (*where your app listens*)

⚠ The issue

According to your container logs, your ASP.NET Core app is **listening on port 5000 inside the container**:

```
nginx
```

 Copy code

```
Now listening on: http://[::]:5000
```

But your compose file maps **host:5000 → container:8080**,

which means Docker is forwarding traffic to the wrong internal port (8080, where nothing is listening).

Result: ❌ You won't reach Swagger from your host, because port 8080 inside the container is closed.

✓ The fix

You have **two options**, depending on what you prefer to expose.

❤ Option A (Recommended)

Keep host port 5000 and fix the mapping to match the container's internal port:

```
yaml
```

 Copy code

```
ports:
```

```
- "5000:5000"
```

Then Swagger will be available at:

👉 <http://localhost:5000/swagger> ↴



F

Pero antes de modificar nada, le pido a chatgpt que se asegure de que realmente a un issue, responde “let's analyze calmly whether there is actually a mismatch between your (backend container) logs and your docker-compose.yml, or if everything might already be fine”.

Your (backend) container **logs** say:

Now listening on: [http://\[::\]:5000](http://[::]:5000)

Logs Inspect Bind mounts Exec Files Stats

```
warn: Microsoft.AspNetCore.Hosting.Diagnostics[15]
      Overriding HTTP_PORTS '8080' and HTTPS_PORTS ''. Binding to values defined by URLs instead 'http://+:5000'.
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:5000 🔗
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

That means inside the container, ASP.NET Core is listening on port 5000. (the app inside the container listens on 5000)

Your **docker-compose.yml** says (in ‘backend’ service):

ports:

- “**5000:8080**”

Which means:

5000 -> host port (the one you will use in the browser, fex <http://localhost:5000>)

8080 -> container port (the port inside the container that you’re forwarding to)

⚠ The potential mismatch

Right now:

- The app inside the container listens on 5000.
 - Docker is trying to forward your host’s port 5000 to port 8080 inside the container.
 - But nothing inside the container is actually listening on 8080.
- That’s why we *suspect* this setup won’t work.

In other words, when you go to:

👉 <http://localhost:5000/swagger>

your browser connects to port 5000 on your host,

Docker forwards that to port 8080 *inside the container*,

but your ASP.NET Core app isn’t listening there — only on 5000.

So it’s like knocking on the wrong door 🚫 😞

✓ How to verify (no need to edit yet)

You can absolutely **test right now** to confirm:

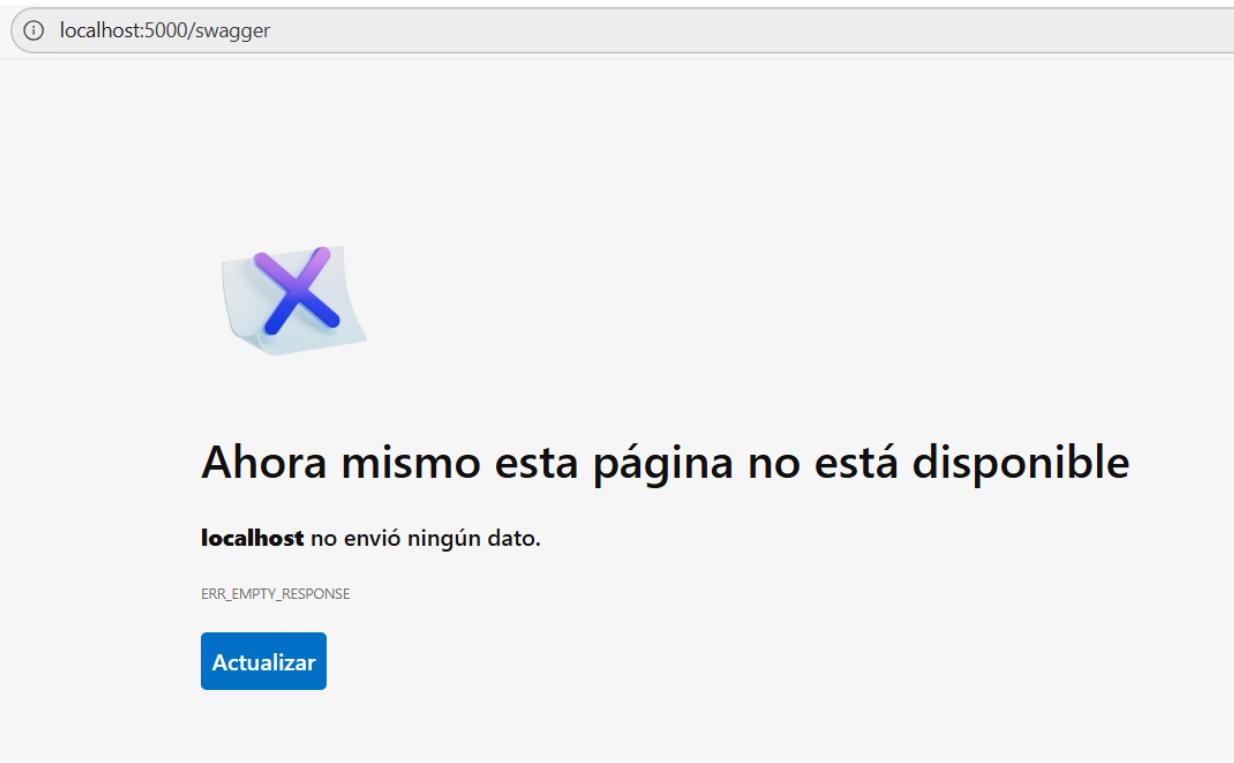
1. Make sure your containers are running (`docker ps`).
2. Open your browser and go to:

```
bash
http://localhost:5000/swagger
Copy code
```

3. Observe:

- If Swagger UI **loads** → great! It means your app *is* bound to 8080 somehow (maybe environment variables override it).
You’re good — no change needed.
- If you get “**This site can’t be reached**” → th ↴ confirms the mismatch we suspected.
In that case, you should fix the mapping to “`5000:5000`”.

“then it confirms the mismatch we suspected”



Es lo que me sale al

poner <http://localhost:5000/swagger>

Así que procedo a ir a mi docker-compose.yml y pongo 5000:**5000** en ports del backend service:

```
▶ Run Service
backend:
  build:
    context: ./backend
  ports:
    - "5000:5000"
```

Because you changed your docker-compose.yml, you MUST **recreate the containers** for Docker to apply the [new port mapping](#). Para ello, in the folder where your docker-compose.yml is located (reservas-app en mi caso), run these commands in your terminal or powershell:

docker-compose down

docker-compose up -build

That will [stop and remove all containers](#) (including your old backend), [rebuild the backend image and start them](#) (the containers) [again using the new configuration](#).

Corré el docker-compose down sin parar los contenedores en docker desktop, ahora la sección containers es:

NEXT PAGE

The screenshot shows the Docker Desktop interface. On the left, there's a sidebar with various options like Ask Gordon, Containers (selected), Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit, Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' with a 'Search' bar and a 'Give feedback' link. It shows 'Container CPU usage' and 'Container memory usage' both indicating 'No containers are running'. There's a 'Show charts' button. A search bar and a 'Only show running containers' toggle are also present. A table lists the two containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	dreamy_haibt	94b9b0895494	ubuntu		N/A	25 days ago	▶ ⋮ ✖
<input type="checkbox"/>	vibrant_carson	5f1347c184de	hello-world		N/A	25 days ago	▶ ⋮ ✖

At the bottom right, it says 'Showing 2 items'.

Osease el down se cargó reservas-app y sus tres containers (frontend, backend y sqlserver)

El cmd muestra esto:

```
Microsoft Windows [Versión 10.0.26200.6901]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app>docker-compose down
time="2025-11-10T10:00:00+01:00" level=warning msg="C:\\\\Users\\\\iago.cadavidgonzalez\\\\Proyectos\\\\reservas-app\\\\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 4/4
 ✓ Container reservas-app-frontend-1  Removed          0.1s
 ✓ Container reservas-app-backend-1  Removed          0.6s
 ✓ Container sqlserver              Removed          10.5s
 ✓ Network reservas-app_default    Removed          0.4s
```

Ahora hago el docker-compose up --build :

NEXT PAGE

```

C:\Users\iago.cadavidgonzalez\Proyectos\reservas-app>docker-compose up --build
time="2025-11-10T10:02:10+01:00" level=warning msg="C:\\\\Users\\\\iago.cadavidgonzalez\\\\Proyectos\\\\reservas-app\\\\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 22.0s (16/31)
=> [internal] load local bake definitions 0.0s
=> reading from stdin 1.11kB 0.0s
=> [backend internal] load build definition from Dockerfile 0.0s
=> transferring dockerfile: 760B 0.0s
=> [frontend internal] load build definition from Dockerfile 0.0s
=> transferring dockerfile: 4.66kB 0.0s
=> [backend internal] load metadata for mcr.microsoft.com/dotnet/aspnet:9.0 0.5s
=> [backend internal] load metadata for mcr.microsoft.com/dotnet/sdk:9.0 0.5s
=> [frontend internal] load metadata for docker.io/library/node:22 2.7s
=> [frontend internal] load metadata for docker.io/library/nginx:alpine 1.8s
=> [backend internal] load .dockerrignore 0.0s
=> transferring context: 383B 0.0s
=> [backend build 1/7] FROM mcr.microsoft.com/dotnet/sdk:9.0@sha256:81f6d22fe21ed9d31375167f62a3538ff4d6835f9d5e6da9c2defa8a84b7687 0.0s
=> => resolve mcr.microsoft.com/dotnet/sdk:9.0@sha256:81f6d22fe21ed9d31375167f62a3538ff4d6835f9d5e6da9c2defa8a84b7687 0.0s
=> [backend base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:9.0@sha256:3dcba33395722d14c80d19107158293ed677b2c07841100d51df07275ae2b2682 0.0s
=> => resolve mcr.microsoft.com/dotnet/aspnet:9.0@sha256:3dcba33395722d14c80d19107158293ed677b2c07841100d51df07275ae2b2682 0.0s
=> [backend internal] load build context 0.0s
=> transferring context: 55.82kB 0.0s
=> CACHED [backend build 2/7] WORKDIR /src 0.0s
=> [backend build 3/7] COPY [ReservasApi/ReservasApi.csproj, ReservasApi/] 0.0s
=> [backend build 4/7] RUN dotnet restore "ReservasApi/ReservasApi.csproj" 19.2s
=> => # Determining projects to restore...
=> [frontend internal] load .dockerrignore 0.0s
=> => transferring context: 2B 0.0s
=> [frontend build 1/6] FROM docker.io/library/node:22@sha256:dcf06103a9d4087e3244a51697adbbb85331dc7b161dbe994ca1cd07dd32e2a5 17.2s
=> => resolve docker.io/library/node:22@sha256:dcf06103a9d4087e3244a51697adbbb85331dc7b161dbe994ca1cd07dd32e2a5 0.0s
=> => sha256:6b274eeda21aa0cdc47c9ed16f68ba6c165239f31549c7a662081f71ded3ccf 448B 0.3s
=> => sha256:16577378d212b933c9ce32521b23fff847e38654535553b5d219450a9282cf15 1.25MB / 1.25MB 0.8s
=> => sha256:72f02c574aecfb602f943861f4542638f1abc42e8fe5bd499df48c494dbe17f 23.07MB / 58.32MB 16.9s
=> => sha256:4f3de9c3ceb0d12af8a779e4a3c88f987a371632fcbe5e776976e46cfe39710 3.32kB / 3.32kB 0.6s
=> => sha256:2123190679e81d983648da92f1bb9dc74383512edb00ad64f93d24d00d8807a 14.68MB / 64.40MB 16.7s
=> => sha256:32885a2b0a589e832bf6b250bd35a528b268360f166af2cd7094d3a14993fc1 42.99MB / 211.45MB 16.3s
=> => sha256:bb445e472b1bad54f5a28edd51b11aef79eca8513394866a261891be9da6a343 13.63MB / 24.03MB 16.2s
=> [frontend stage-1 1/2] FROM docker.io/library/nginx:alpine@sha256:b3c656d55d7ad751196f21b7fd2e8d4da9cb430e32f646adcf92441b72f82b14 0.0s
=> => resolve docker.io/library/nginx:alpine@sha256:b3c656d55d7ad751196f21b7fd2e8d4da9cb430e32f646adcf92441b72f82b14 0.0s
=> [frontend internal] load build context 3.4s
=> => transferring context: 1.79MB 3.0s

```

Ese output está incompleto, todavía estaba en ejecución cuando saqué la captura.

La última línea de output dice “Recovery is complete. This is an informational message only. No user action is required.” It comes from SQL Server, meaning your database container is fully up and ready.

Si después de que termine, voy a DockerDesktop's containers section, veo esto:

<input type="checkbox"/>	<input checked="" type="radio"/> reservas-app	-	-	-	-	1.73%	4 minutes ago	<input type="checkbox"/> <input type="radio"/> <input type="button" value="Delete"/>
<input type="checkbox"/>	<input checked="" type="radio"/> sqlserver	1420e741c56c	mssql/server:2022-latest	1433:1433	1433:1433	1.72%	4 minutes ago	<input type="checkbox"/> <input type="radio"/> <input type="button" value="Delete"/>
<input type="checkbox"/>	<input checked="" type="radio"/> backend-1	38e8d672d5fd	reservas-app-backend	5000:5000	5000:5000	0.01%	4 minutes ago	<input type="checkbox"/> <input type="radio"/> <input type="button" value="Delete"/>
<input type="checkbox"/>	<input checked="" type="radio"/> frontend-1	ea1af1c102a9	reservas-app-frontend	4200:80	4200:80	0%	4 minutes ago	<input type="checkbox"/> <input type="radio"/> <input type="button" value="Delete"/>

NEXT PAGE

Backend container logs now say the following:

```
warn: Microsoft.AspNetCore.Hosting.Diagnostics[15]
Overriding HTTP_PORTS '8080' and HTTPS_PORTS ''. Binding to values
defined by URLs instead 'http://+:5000'.
info: Microsoft.Hosting.Lifetime[14]
Now listening on: http://[::]:5000
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: /app
```

Perfect — this is exactly what we want to see ✅

[Containers](#) / reservas-app-backend-1

reservas-app-backend-1

 38e8d672d5fd ⏪  [reservas-app-backend:latest](#)
[5000:5000](#) ↗

STATUS
Running (6 minutes ago)



[Logs](#) [Inspect](#) [Bind mounts](#) [Exec](#) [Files](#) [Stats](#)

```
warn: Microsoft.AspNetCore.Hosting.Diagnostics[15]
Overriding HTTP_PORTS '8080' and HTTPS_PORTS ''. Binding to values defined by URLs instead 'http://+:5000'.
info: Microsoft.Hosting.Lifetime[14]
Now listening on: http://\[::\]:5000 ↗
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: /app
```



◆ What to do next

1. Since your `docker-compose.yml` now maps `5000:5000`, you can access the backend from your host at:

bash

 [Copy code](#)

```
http://localhost:5000/swagger
```

2. Open your browser and check if the **Swagger UI** appears with your controllers (`Reservas`, `Usuarios`, etc.).

<http://localhost:5000/swagger>

The screenshot shows the ReservasApi v1 Swagger UI. At the top, it says "Select a definition" and "ReservasApi v1". Below that, the title "ReservasApi 1.0 OAS 3.0" is displayed, along with the URL "http://localhost:5000/swagger/v1/swagger.json".

Cancelaciones

- POST /api/Cancelaciones
- GET /api/Cancelaciones
- GET /api/Cancelaciones/{id}
- PUT /api/Cancelaciones/{id}
- DELETE /api/Cancelaciones/{id}

Instalaciones

- POST /api/Instalaciones
- GET /api/Instalaciones

Para el frontend container y vuelvo a probar, mismo resultado osease swagger ui test satisfactorio de backend running in its container:

The screenshot shows the ReservasApi v1 Swagger UI. At the top, it says "Select a definition" and "ReservasApi v1". Below that, the title "ReservasApi 1.0 OAS 3.0" is displayed, along with the URL "http://localhost:5000/swagger/v1/swagger.json".

Cancelaciones

- POST /api/Cancelaciones
- GET /api/Cancelaciones
- GET /api/Cancelaciones/{id}
- PUT /api/Cancelaciones/{id}
- DELETE /api/Cancelaciones/{id}

Instalaciones

- POST /api/Instalaciones

Pero falta probar algunos endpoints (tal como hicimos antes cuando backend corría en local con dotnet run):

You should see all your controllers listed (recordar que creé un controller por cada model i.e. tabla), en efecto veo los 5 (cancelaciones, instalaciones pagos, reservas, usuarios)

Antes de testear ningún endpoint, vamos a DBeaver, conectamos con mi sqlserver container, look under Databases to see if ReservasDb is still there with tables.

DBeaver 25.2.4

File Edit Navigate Search SQL Editor Database Window

Database Navigator × Projects

Filter connections by name

master localhost:1433

Databases

ReservasDb

Schemas

dbo

Tables

Cancelaciones

Instalaciones

Pagos

Reservas

Usuarios

_EFMigrationsHistory

Como todavía está reservasdb con las tablas, dice chatgpt que

entonces you do NOT need to rerun migrations (el par de dotnet ef commands que en su momento hice para que se me crease la db y así me apareciese como ahora la veo en DBeaver).

Dice chatgpt que el problema está aquí:

```
C: > Users > iago.cadavidgonzalez > Proyectos > reservas-app > backend > ReservasApi > appsettings.json > {} ConnectionStrings > DefaultConnection
1 {
2     "ConnectionStrings": {
3         "DefaultConnection": "Server=localhost,1433;Database=ReservasDb;User Id=sa;Password=dockEr!8;TrustServerCertificate=True"
4     },
5
6     "Logging": {
7         "LogLevel": {
8             "Default": "Information",
9             "Microsoft.AspNetCore": "Warning"
10        }
11    },
12    "AllowedHosts": "*"
13 }
14
```

Me dice de ponerle '**db**' en lugar de 'localhost,1433' (lo cual pone porque en su momento el propio chatgpt me lo recomendó poner dese modo, **acaso tengo que cambiarlo dependiente de si backend está corriendo en local vs inside its container? Desde luego tiene toda la pinta de que así es.**)

'db' is the service name of my sqlserver container.

Hecho ese cambio, hay que volver hacer docker-compose down y docker-compose up --build

This ensures the backend container picks up the updated environment variable.

Then tests again the GET /api/reservas endpoint in Swagger.

El último docker compose up build me dejó en la terminal sin ponerme ruta dir actual, di control + c dos veces y entonces se paró los containers que estaban corriendo en docker desktop (estaban corriendo el backend y el sqlserver).

Entonces hago esa modificación en appsettings.json file de mi proyecto backend y docker-compose down seguido de docker-compose up --build, paro el frontend container, voy a DBeaver y veo que sigue la db con sus tablas, pongo en browser <http://localhost:5000/swagger> y doy enter, sale la swagger UI con mis 5 controllers/modelos/tablas, testeo el get api reservas endpoint (dando en el desplegable+try it out+execute):

NEXT PAGE

localhost:5000/swagger/index.html

Reservas

POST /api/Reservas

GET /api/Reservas

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5000/api/Reservas' \
-H 'accept: text/plain'
```

Request URL

<http://localhost:5000/api/Reservas>

Server response

Code	Details
200	<p>Response body</p> <pre>[{"reservaId": 1, "numeroAsistentes": 3, "fechaHora": "2025-11-10T07:35:54.654", "dni": "12345678A", "espacioId": 1}]</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Mon, 10 Nov 2025 09:42:23 GMT server: Kestrel transfer-encoding: chunked</pre>

Sale la reserva que había creado cuando testeé el post con backend corriendo en local.

“Perfect! That means everything is wired up correctly inside Docker:

- Your backend container can reach the sql server container via Server=db”
- The data you previously inserted (from when the backend ran locally) is still there –that is because your sqlserver container uses a **persistent Docker volume** (sql_data –en mi caso el volumen se llama **reservas-api_sql_data**), so the database survived the container recreation (de la cual hice varias después de crear la reserva)
- Swagger is successfully calling the API endpoints inside the container, and responses are correct (200 OK with existing bookings –lo del 200 ok me aparece siempre regardless of si me da o no error).

NEXT PAGE

"Then test the GET /api/reservas endpoint in Swagger again." I've just done that test (with backend running in its container), the result is 'code 200' and under 'Details' it says 'response body' which contains the booking we created when we tested POST with backend running in local.

Perfect! That means everything is wired up correctly inside Docker:

- Your backend container can reach the SQL Server container via `Server=db`.
- The data you previously inserted (from when the backend ran locally) is still there — that's because your SQL Server container uses a persistent Docker volume (`sql_data`), so the database survived the container recreation.
- Swagger is successfully calling the API endpoints inside the container, and responses are correct (`200 OK` with existing bookings).

At this point, you can proceed to:

1. Test all other CRUD endpoints (Usuarios, Instalaciones, Pagos, Cancelaciones) in the same way using Swagger.
2. Verify that POST, PUT, DELETE all persist data in SQL Server container, and you can see changes in DBeaver.
3. Use your Angular frontend to call these endpoints, now that backend is running in Docker.

Everything you've done so far confirms that:

- The internal Docker networking is working.
- Your backend container is properly configured to talk to the SQL Server container.
- Swagger is fully functional with the containerized backend.

You're in a solid position to now test the full flow end-to-end with frontend → backend → database, all in Docker.

If you want, I can guide you step by step through testing all controllers/endpoints in Swagger with the containerized backend, just like we did before with `dotnet run`.

Entonces doy por hecho el haber testeado algunos endpoints con swagger y primero con backend corriendo en local con `dotnet run` y después (lo que acabamos de hacer) con backend corriendo en su docker container

NEXT PAGE

"Then test the GET /api/reservas endpoint in Swagger again." I've just done that test (with backend running in its container), the result is 'code 200' and under 'Details' it says 'response body' which contains the booking we created when we tested POST with backend running in local.

Perfect! That means everything is wired up correctly inside Docker:

- Your backend container can reach the SQL Server container via `Server=db`.
- The data you previously inserted (from when the backend ran locally) is still there — that's because your SQL Server container uses a persistent Docker volume (`sql_data`), so the database survived the container recreation.
- Swagger is successfully calling the API endpoints inside the container, and responses are correct (`200 OK` with existing bookings).

At this point, you can proceed to:

1. Test all other CRUD endpoints (Usuarios, Instalaciones, Pagos, Cancelaciones) in the same way using Swagger.
2. Verify that POST, PUT, DELETE all persist data in SQL Server container, and you can see changes in DBeaver.
3. Use your Angular frontend to call these endpoints, now that backend is running in Docker.

Everything you've done so far confirms that:

- The internal Docker networking is working.
- Your backend container is properly configured to talk to the SQL Server container.
- Swagger is fully functional with the containerized backend.

You're in a solid position to now test the full flow end-to-end with frontend → backend → database, all in Docker.

If you want, I can guide you step by step through testing all controllers/endpoints in Swagger with the containerized backend, just like we did before with `dotnet run`.

En esto de que frontend llame a endpoints no me voy meter todavía en esta semana 2, que es la del backend.

Juan Carlos Martinez Perez jueves 15:19 99 ...

 #3 Semana 2: Backend y Base de Datos

- CRUD de entidades principales en .NET (usuarios, reservas, espacios)
- Implementación de autenticación básica y control de permisos
- Pruebas unitarias en backend y validaciones de negocio esenciales
- Publicación del evento "ReservaCreada" en Kafka dummy/real
- Documentación técnica de endpoints API

Semana 2 goals

- **CRUD de entidades principales en .NET (usuarios, reservas, espacios –y algunas tablas más en mi caso-)** Esto lo hice en VS IDE al crear las clases y redactar el código relativo a los modelos (que se corresponden con las tablas de la bd), los controller (uno por cada modelo) y el appdbcontext (que declara los dbsets correspondientes a las tablas de la bd). En las clases controller va el código de los **CRUD endpoints**.
- **Documentación técnica de endpoints api:** este documento, aunque algo extenso demás y faltaría probablemente retocar algo el índice de contenidos en plan añadirle más puntos.
- **Pruebas unitarias en backend:** si con esto nos referimos a probar los endpoints uno por uno tanto con backend corriendo en local (dotnet run) como con backend corriendo en su contenedor (y en ambos casos con sqlserver container corriendo y frontend container parado), yo hice unas cuantas pruebas que figuran en la documentación, pero **no probé absolutamente todos los endpoints**, con lo cual puede ser que algunos me den problemas.
- **Validaciones de negocio esenciales:** me suena un poco abstracto demás, agradecería una mayor concreción, pero entiendo que estamos hablando de implementar cosas como el controlar que los datos introducidos por el usuario para su registro son válidos (por ejemplo que el dni cumple con su formato de 8 letras + 1 número). **Tales validaciones yo no he hecho ninguna todavía.**
- **Implementación de autenticación básica y control de permisos**
- **Publicación del evento “ReservaCreada” en Kafka dummy/real**