Link Publicacion

<https://dev.to/ebarrioscode/base-de-datos-en-memoria-con-asp-net-core-3-0-mvc-entity-framework-core-3950>

Link al repositorio en GitHub  
<https://github.com/EbarriosCode/Base-de-Datos-en-Memoria-Net-Core-3.0-MVC-EFCore>

Base de Datos en memoria con ASP.NET Core 3.0 MVC + Entity Framework Core

[[ebarrioscode profile image](https://dev.to/ebarrioscode) Eduardo Barrios](https://dev.to/ebarrioscode)Nov 7 ・4 min read

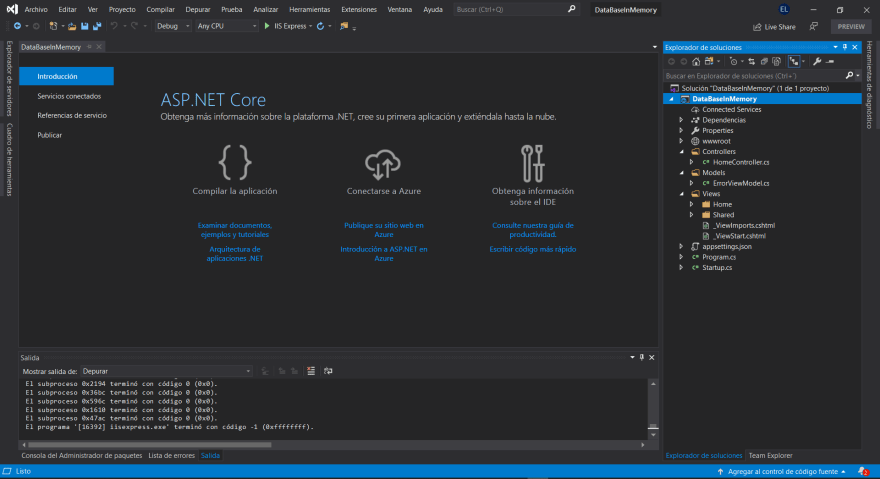
[**#csharp**](https://dev.to/t/csharp) [**#dotnet**](https://dev.to/t/dotnet) [**#database**](https://dev.to/t/database)

Utilizar una base de datos en memoria es muy útil cuando necesitamos probar componentes utilizando un escenario muy parecido al que nos provee una conexión a una base de datos real, obviamente esto sin el consumo de recursos y la sobrecarga de las operaciones reales a la base de datos, también es útil cuando no se tiene acceso a la base de datos real que posiblemente se encuentre en un servidor al que no tenemos acceso.

Cabe mencionar que las bases de datos en memoria están diseñadas para ser una base de datos de propósito general únicamente para pruebas, por lo que no deberíamos tratar de imitar a una base de datos relacional ni tratar de llevar este tipo de técnica a un ambiente productivo.

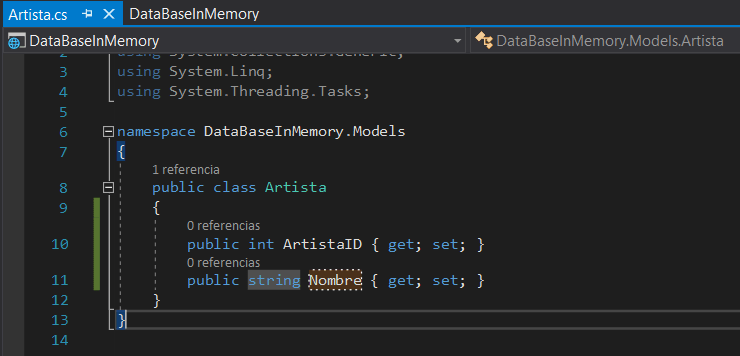
Sabiendo lo anterior empezamos creando el proyecto en Visual Studio 2019.

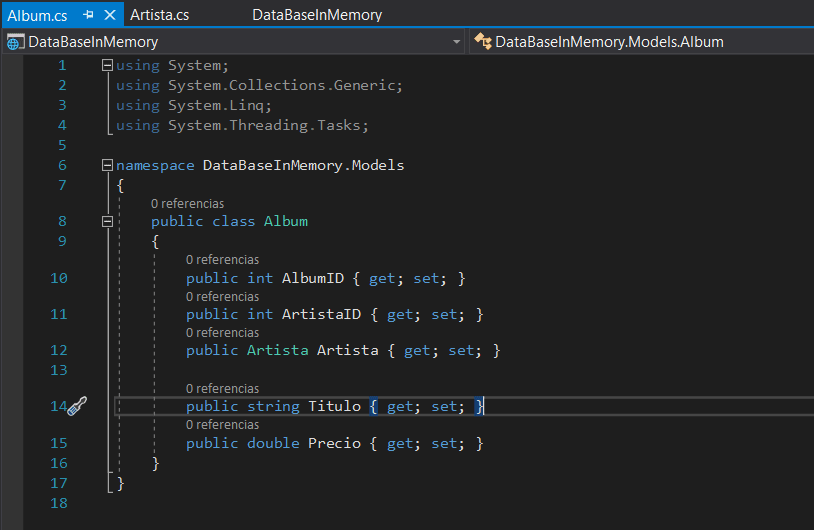
Una vez creado el proyecto de ASP.NET Core 3.0 MVC

[](https://res.cloudinary.com/practicaldev/image/fetch/s--dpYCCX_B--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/qq2b5rmde5hurpgwa0ub.PNG)

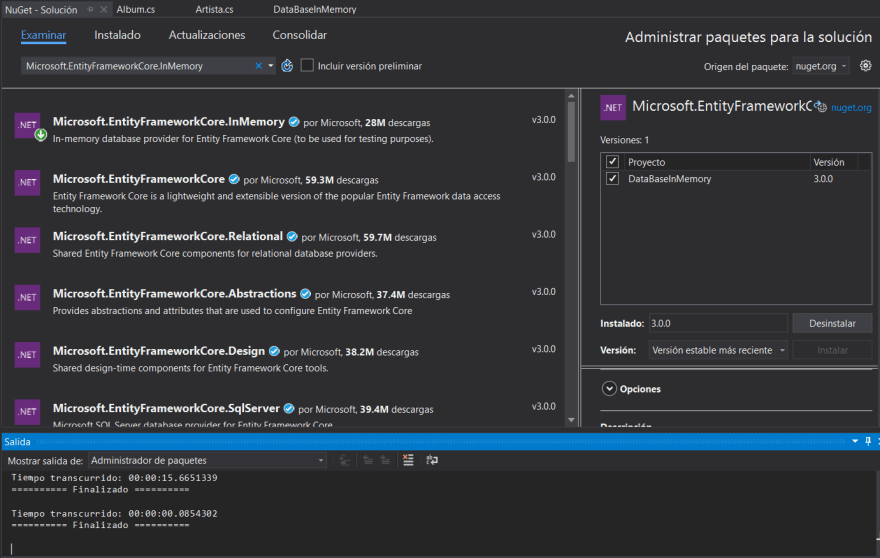
El siguiente paso es crear los modelos que necesitaremos para logar persistencia de información en la base de datos en memoria.

Para este ejemplo utilizaremos la relación de modelos Álbum y Género donde un Álbum puede contener un Artista y un Artista puede pertenecer a múltiples Álbumes, vamos a utilizar el enfoque Code First este es el enfoque que nos conviene en este escenario.

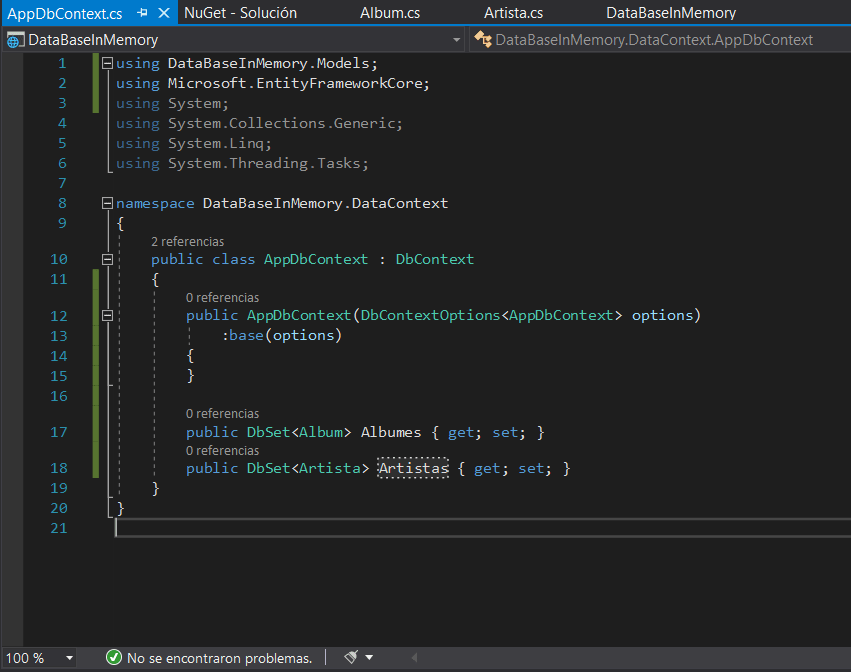
[](https://res.cloudinary.com/practicaldev/image/fetch/s--f5uW-9RD--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/q9brsi2s4u2ve4qvrd9f.PNG)

[](https://res.cloudinary.com/practicaldev/image/fetch/s--JoX19rdO--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/7jidf4ezvbtjntwzkg2z.PNG)

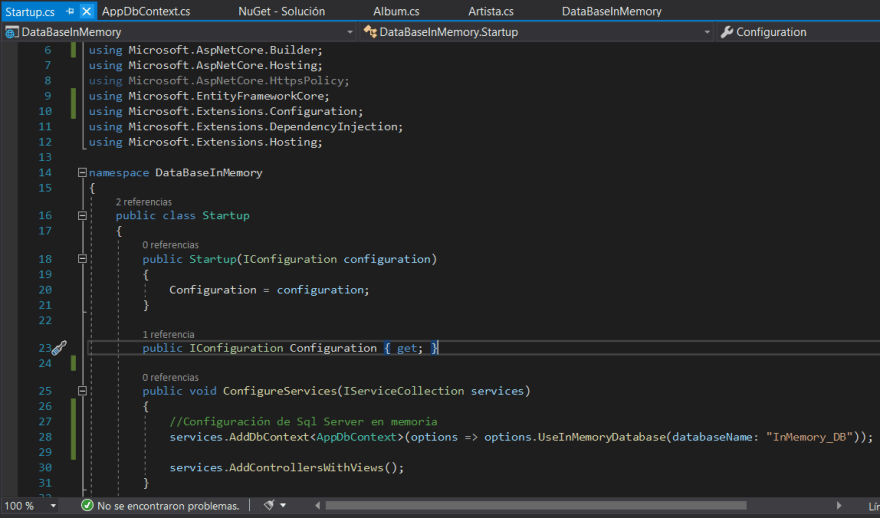
Después de la creación de los modelos y enfocarnos en el código primero vamos a instalar el nuget de Entity Framework Core InMemory para después crear el Contexto de Datos y decirle a Entity Framework Core que proceda a mapear nuestros modelos.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--RJLEJ31K--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/qc67ffx1hovc66xh2pjc.PNG)

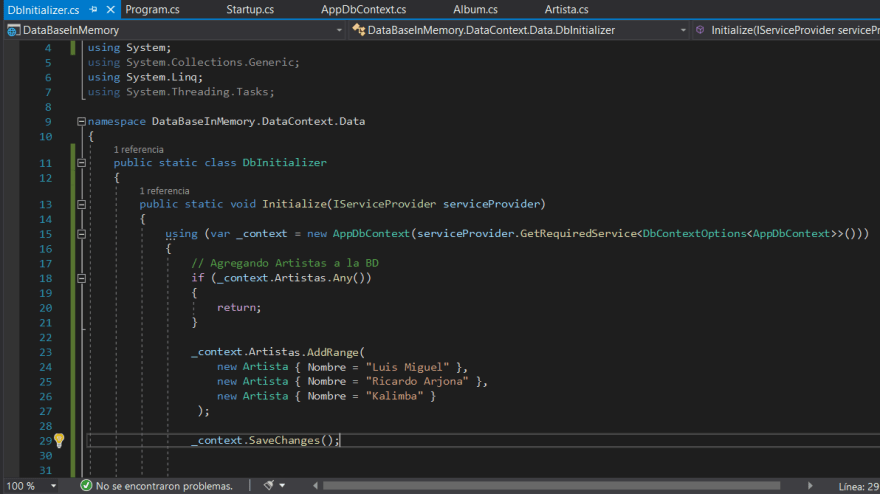
Ahora creamos el Contexto, creamos una carpeta llamada DataContext y dentro de esta creamos una nueva clase llamada AppDbContext la cuál hereda de la clase DbContext que está definida dentro del nuget de Entity Framework Core claro sin olvidar agregar la referencia a Microsoft.EntityFrameworkCore.

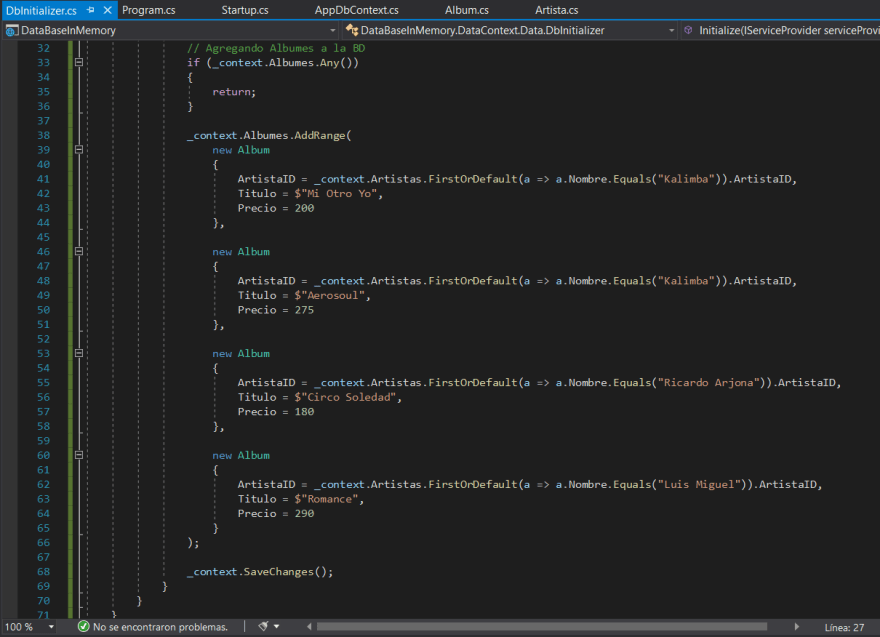
[](https://res.cloudinary.com/practicaldev/image/fetch/s--EK889LZN--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/3bshu75q36rstvpfzqn4.PNG)

El siguiente paso es dar de alta el servicio en el método ConfigureServices de la clase Startup.cs aquí debemos decirle al servicio que utilize el contexto que acabamos de crear y decirle que la base de datos será en memoria.

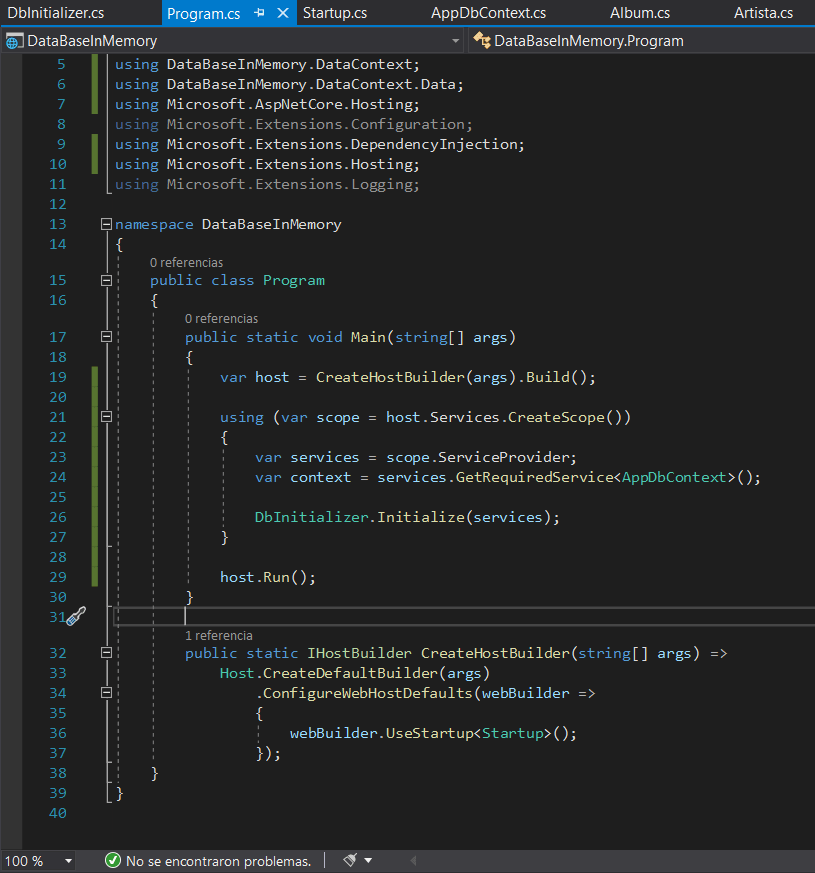
[](https://res.cloudinary.com/practicaldev/image/fetch/s--3wbzX56---/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/548kjfhkf2kaqwuj419p.PNG)

Luego debemos crear una clase static y un método static void para inicializar la base de datos. Debido a que no podemos hacerlo por medio de sql y el enfoque para este escenario es code first debemos apoyarnos de Entity Framework Core para persistir de datos para que cuando se cree la base datos en memoria seguidamente empiece el proceso de inicializar y poblar de información las tablas de la base de datos.  
Situados dentro de la carpeta DataContext crearemos una nueva carpeta llamada Data y dentro de Data vamos a crear una nueva clase llamada DbInitializer no olvidar definirla como static y seguidamente crear el método public static void Initialize(IServiceProvider serviceProvider) el parámetro que recibe este método es una interfaz que utilizaremos para inyectar la dependencia y abstraer la implementación para utilizar el contexto de datos, aprovecharemos que ASP.NET Core admite el patrón de diseño de software de inyección de dependencias, técnica para lograr la Inversión de Control (IoC) entre las clases y sus dependencias.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--tfeB1YNu--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/ugqmpdunj4blyq6640qo.PNG)

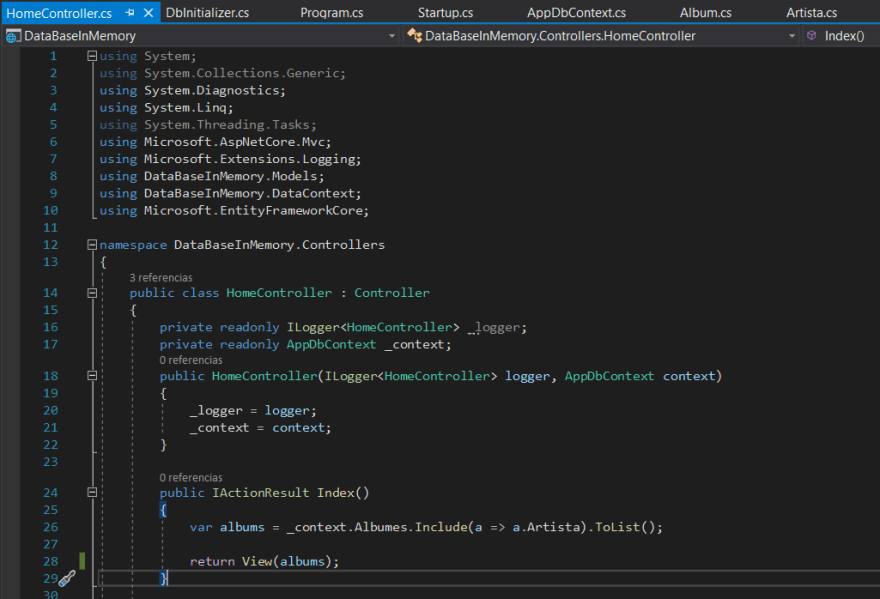
[](https://res.cloudinary.com/practicaldev/image/fetch/s--xa9RztVg--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/sqnidygw38vnzki34re7.PNG)

Seguidamente vamos a modificar el código del método Main en la clase Program.cs necesitamos obtener el IWebhost que alojará la aplicación, lo asignamos a un objeto anónimo, luego necesitamos acceder a un servicio con alcance en el inicio (Startup) para ejecutar tareas de inicialización y por último debemos crear la instancia del contexto en la capa de servicios e invocar al método DbInitializer.Initialize(services) enviando como parámetro el servicio de inicialización.

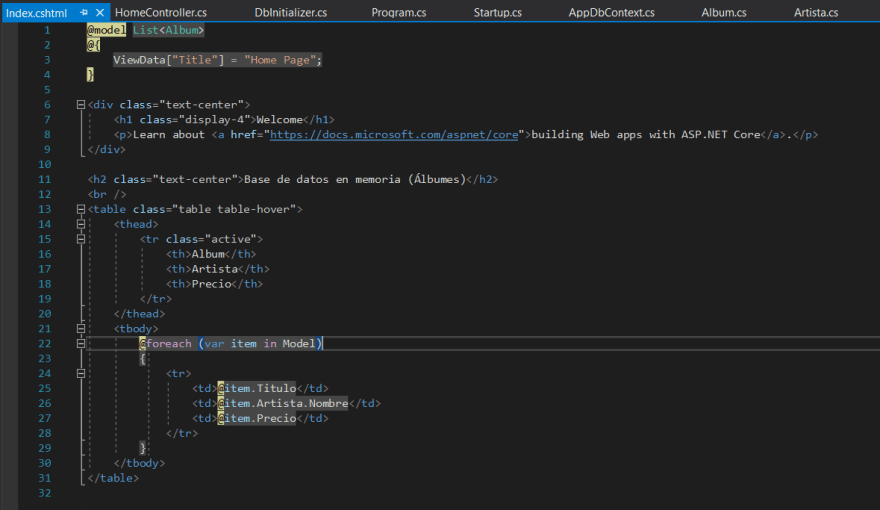
[](https://res.cloudinary.com/practicaldev/image/fetch/s--39RKlcX_--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/nvydox0k9vvslpvf1bg4.PNG)

Ya estamos cerca del final, tenemos la configuración lista para acceder a los datos de una base de datos en memoria, vamos al HomeController y utilizaremos el contexto de datos para listar los datos y mostrarlos en la vista Index.

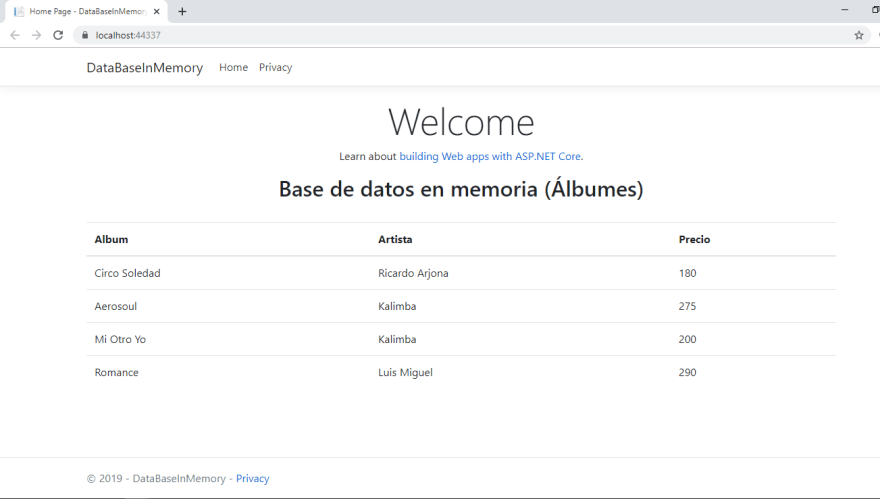
Creamos una propiedad de lectura de tipo AppDbContext, inyectamos la dependencia en el constructor del HomeController y recuperamos la lista de Álbumes y su relación con el modelo Artista asignando el resultado a un objeto anónimo que le pasaremos a la vista.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--u1nmhQmJ--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/iqsfq90deklfm6ck0c6a.PNG)

Tipamos nuestra vista Index para que reciba una lista de Álbumes y luego agregaremos este código para mostrar en una tabla de html las propiedades de nuestra lista de Álbumes.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--a04GctAS--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/eyjnsiijzc7hymtba885.PNG)

Al compilar y ejecutar nuestra aplicación web el resultado es el siguiente.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--lEror2kg--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/r9dubaggme6z72wv1uzn.PNG)