# Creating a .NET Core WebAPI Project with Code First Entity Framework

Note: If you are starting with an existing MVC project with the Entity Framework already setup, skip to the “Add Your Controllers” step.

## Create a .NET Core MVC Project

* File, New Project, Web, **ASP.NET Core Web Application** (or search in VS2019)
* Select **API** and uncheck “Configure for HTTPS”

## Add your EF models and DbContext.

* Add your entity model classes. Plain C# classes. (Product, Customer, Vendor, etc. At least add Product for this demo.)
* Add your database context class and add a constructor. (Code below assumes the context class is named “MyDB”.)

public class MyDB : DbContext

{

public DbSet<Product> Products { get; set; }

public DbSet<Customer> Customers { get; set; }

// constructor to allow .NET Core to do dependency injection

**public MyDB(DbContextOptions options) : base(options)**

**{ }**

}

## Edit Startup.cs:

* Add a using statement: using Microsoft.EntityFrameworkCore;
* In the ConfigureServices method add your connection string.

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

**var connection = @"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=DotNetCoreMVC.Models.MyDB;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=True;ApplicationIntent=ReadWrite;MultiSubnetFailover=False";**

**services.AddDbContext<Models.MyDB>**

**(options => options.UseSqlServer(connection));**

Note: Storing the connection string in the code is BAD! Store it in appsettings.json and retrieve with:  
 var connetion = Configuration["yourConnectionStringName"];

Note: If you copied your connection string from a “server” connection property, add “Initial Catalog=youNewDatabaseName;” to the connection string.

## Build the project

## Configure migrations:

Tools > NuGet Package Manager > Package Manager Console

Add-Migration InitialCreate  
Update-Database

## Add some sample data

Using the Visual Studio SQL Server Object Explorer add several records to your Products table.

## Add Your Controllers

Use: **API Controller with Actions, using Entity Framework**

## Build, Run and Test Your App

https://localhost:44304/api/products

## Add a WebAPI Tester

Add a new folder to the project: wwwroot

Right-click the new folder and add an “HTML Page”. Name the page “WebApiTester.HTML”.

Copy the text from the supplied WebApiTester.HTML file and replace the content of the new file with the copied text.

Edit the Startup.cs file.

Just before “app.UseMVC” add “app.UseStaticFiles();”

## Build, Run and Test Your App

Load the WebApiTester.HTML page:

https://localhost:44304/WebApiTester.HTML

In the URL box enter the URL to your service:

http(or https)://localhost:yourPort/api/products

Click the **Send** button and review the returned JSON.

Also test the POST, PUT and DELETE methods.

## Add ODATA Queries to Your Project

Note: At this time, Swagger and ODATA do not play well together…

1. Add the **Nuget Microsoft.AspNetCore.OData** package
2. Go to the Startup.cs file and add in this line of code in your ConfigureServices method:  
    services.AddOData();
3. Enable the dependency injection support for ALL HTTP routes. In Startup.cs, in the Configure method, add the following below any existing app.UseMvc() line:  
    app.UseMvc(routeBuilder => {  
    routeBuilder.EnableDependencyInjection();  
    routeBuilder.Expand().Select().OrderBy().Filter();  
    });
4. Go to your Products controller, and add the following line just before the GetProducts method:  
    [EnableQuery]  
   And add the needed using statement.
5. Test some of these queries:  
   ?$select=name,qty  
   ?$filter=qty gt 100  
   ?$filter=name eq 'boat'  
   ?$select=name,qty&$filter=qty gt 100&orderby=price