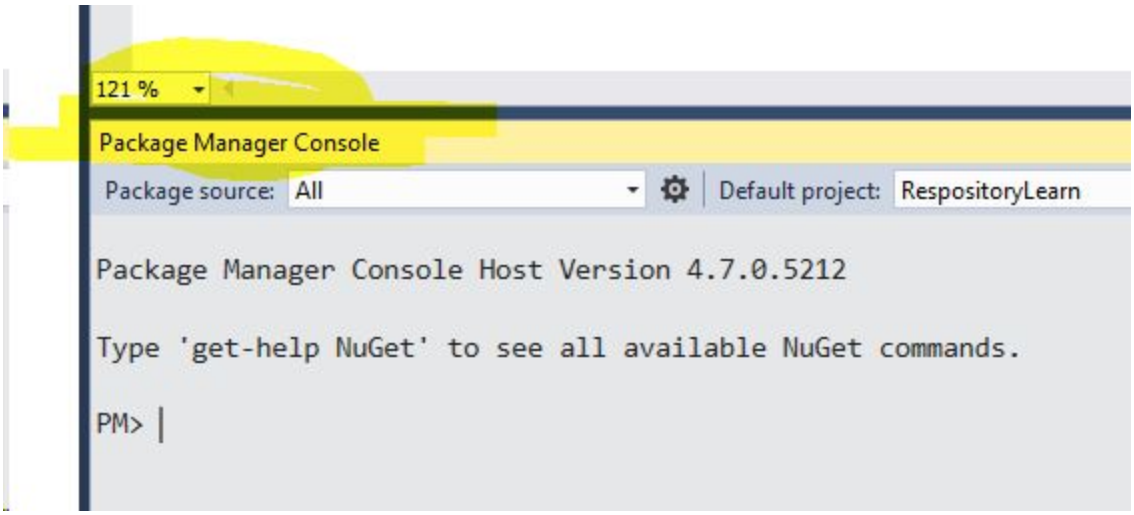
.Net core project using Repository design pattern (Generics)and DI

[Code first approach]

Read this before starting
https://docs.microsoft.com/en-us/ef/core/miscellaneous/configuring-dbcontext
And
https://jessedotnet.com/2016/12/29/asp-net-core-injecting-your-db-context-into-your-controllers/

Steps:

- Create a .Net Core Web Application project
- Install EF core 2 using following steps manually (can not install from nuget packages so use Package Manager console)
    - 

    - **run this command**
      ```
      dotnet add package Microsoft.EntityFrameworkCore.Design
      dotnet restore
      ```
    - **If it is looking at wrong project and display errors like**

```
121 %
Package Manager Console
Package source: All        Default project: Core
PM> dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet : Could not find any project in `C:\Users\sarbjit\source\repos\RespositoryLearn\`.
At line:1 char:1
+ dotnet add package Microsoft.EntityFrameworkCore.Design
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (Could not find ...ositoryLearn\`.:String) [], RemoteException
    + FullyQualifiedErrorId : NativeCommandError

Usage: dotnet add <PROJECT> package [options] <PACKAGE_NAME>
```

then run following

```
dotnet add <projectName> package
Microsoft.EntityFrameworkCore.Design

dotnet restore
```

- Right click on project and click on "Edit <projectName>.cproj" file. It will add following lines in it

```
Core.csproj   MyContext.cs   Customer.cs   ProductSold.cs   Sale.cs   Product.cs   Customer.cs
1  <Project Sdk="Microsoft.NET.Sdk.Web">
2    <PropertyGroup>
3      <TargetFramework>netcoreapp2.0</TargetFramework>
4    </PropertyGroup>
5    <ItemGroup>
6      <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.8" />
7      <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="2.1.2" />
8    </ItemGroup>
9    <ItemGroup>
10     <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.4" />
11   </ItemGroup>
12 </Project>
13
```

- **NOTE:** To add EF Core support to a project, install the database provider that you want to target. This tutorial uses SQL Server, and the provider package is Microsoft.EntityFrameworkCore.SqlServer. This package is included in the Microsoft.AspNetCore.All metapackage, so you don't have to install it

- **Read:** https://docs.microsoft.com/en-us/ef/core/miscellaneous/cli/dotnet

- Add models and create a Context file ex-MyContext which should inherit from dbcontext of EF

```
using System.Threading.Tasks;

namespace Core.Context
{
    public class MyContext  : DbContext
    {
        public MyContext(DbContextOptions<MyContext> options): base(options)
        {

        }

        DbSet<Customer> Customers { get; set; }
        DbSet<Product> Products { get; set; }
        DbSet<Sale> Sales { get; set; }
    }
}
```

NOTE:

1. *Dependency Injection is setup in the ConfigureServices method of Startup.cs This ConfigureServices method is responsible for setting the things that can be injected into our controller action constructors.*

2. Each services.Add extension method adds (and potentially configures) services. For example, services.AddMvc() adds the services MVC requires.

- Add the following line in start.cs:

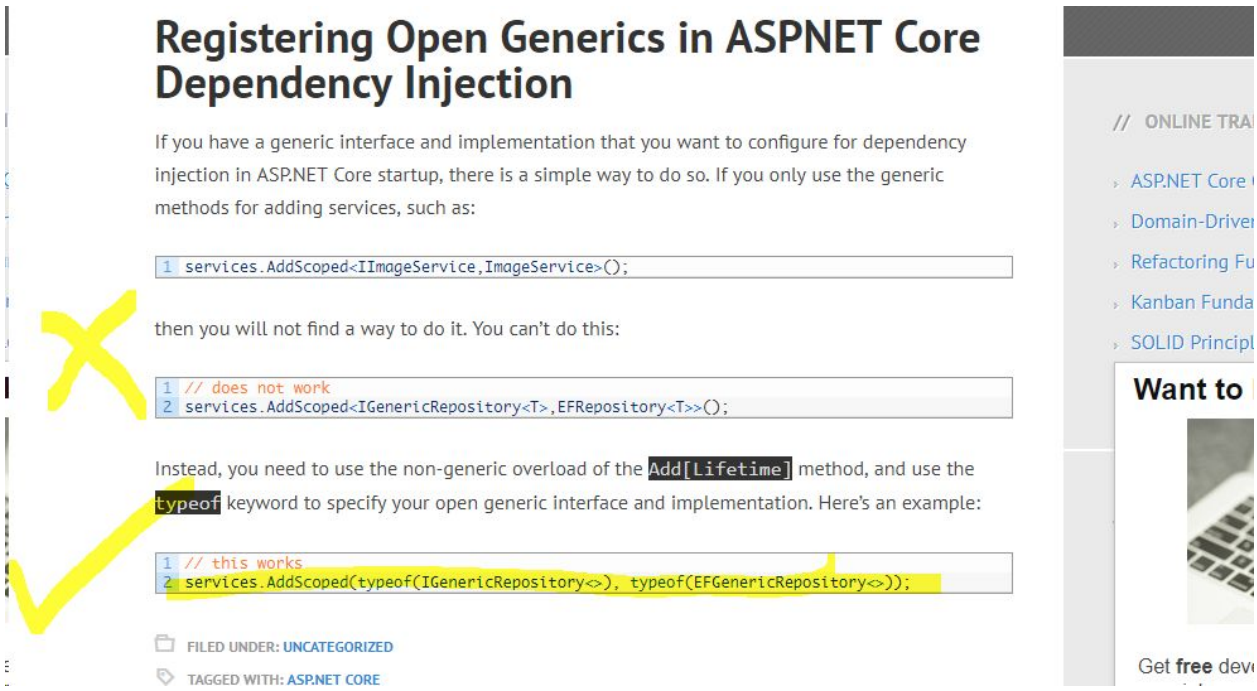
services.AddDbContext<MyContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DatabaseConnection")));

```
Lifecycle Events   Stack Frame:
.csproj   MyContext.cs   Startup.cs   appsettings.json
ore   Core.Startup   Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<MyContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DatabaseConnection")));

            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
```

- Install **NInject 3.3.4 or latest and NInject.mvc4** in core project (DI)
- Inject Mycontext directly in Controller
- Open “Nuget console Manager” and run
    - Add-Migration <AnyName> (Enable-migrations is obsolete)
    - update-database
    - It will generate database for you.

[Now we will add Repository design as service in DI and use it for getting data and sending data to database.]

- Add generic IRepository and Repository which inherit from that generic interface
- Make sure when you register this service with configurationServices in start.cs follow this



Registering Open Generics in ASPNET Core Dependency Injection

If you have a generic interface and implementation that you want to configure for dependency injection in ASP.NET Core startup, there is a simple way to do so. If you only use the generic methods for adding services, such as:

```
services.AddScoped<IImageService,ImageService>();
```

then you will not find a way to do it. You can’t do this:

```
// does not work
services.AddScoped<IGenericRepository<T>,EFRepository<T>>();
```

Instead, you need to use the non-generic overload of the `Add[Lifetime]` method, and use the `typeof` keyword to specify your open generic interface and implementation. Here’s an example:

```
// this works
services.AddScoped(typeof(IGenericRepository<>), typeof(EFGenericRepository<>));
```

FILED UNDER: UNCATEGORIZED

TAGGED WITH: ASP.NET CORE

- And then inject in controller

-

```
20180905051517_Dat...eWithrepository.cs   HomeController.cs   Startup.cs
CoreRepository                            Core.Controllers.HomeController
10  namespace Core.Controllers
11  {
12      public class HomeController : Controller
13      {
14          private readonly IRepository<Customer> _customerRepository;
15          private readonly IRepository<Product> _productRepository;
16          private readonly IRepository<Sale> _saleRepository;
17
18          public HomeController(
19              IRepository<Customer> CustomerRepository,
20              IRepository<Product> ProductRepository,
21              IRepository<Sale> SaleRepository
22              )
23          {
24              _customerRepository = CustomerRepository;
25              _productRepository = ProductRepository;
26              _saleRepository = SaleRepository;
27
28          }
29
30          public ActionResult Index()
31          {
32              return View();
33          }
34          public ActionResult Customer()
```

If its new project Add-Migration and update-database

Add controller and read me
https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/controller-methods-views?view=aspnetcore-2.1