(https://www.invincix.com)

Onion architecture in ASP.Net MVC

Author : Suraj Sahoo

25 JAN
2015

(H

Introduction

Pre-Requisites

This article is meant for my fellow mates who have knowledge on ASP.NET MVC and wondering for an architecture to choose. I am trying to pen this article to be as simple and easy to understand and implement.

Lets Start..

Onion Architecture, the concept introduced by the *Jeffery Palermo* in 2008 with a aim to make the application loosely coupled and with proper separation between the folders and the different areas of concern in the application. This makes the development easier, the testing of the application easier, the maintainance becomes easier. During the initial stage of development, the SRS(Software Requirement Specifications) is made and proper planning is done regarding what should be the approach, what technologies to be used & after that is done, the most difficult part is choosing the proper architecture so that the maintainance becomes easier. Here, the points that are kept in mind are:-

1. All code depends on layers *closer or the center*
2. Domain Models will be at the *Center or the Core*
3. The Inner layer define the Interfaces where as the outer layer implements these interfaces members
4. Layered Behaviour around the Domain
5. Infrastructure, that would contain Data and the service implementation should be pushed to the edge. Along with the Infrastructure, the UI concerns are also pushed to the edge.

Background

There are a lot of architectures used in web applications, but being decisive and choosing the architecture that would help achieve loose coupling, which is most essential. Loose Coupling, depends on separation of concern, which means each layer would be independent of each other. What is tightly coupled and Loosely coupled?

(http://surajpassion.in/wp-content/uploads/2015/01/thinkzoo.jpg)

Yes, exactly as you are thinking my dear readers. But still let me discuss the difference quickly.

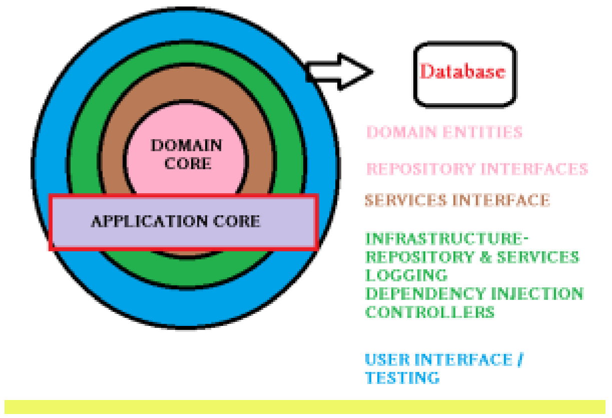
A tightly coupling, means where the one object/entity needs to have knowledge of other objects or we can say they depend largely on the interfaces where the service methods are declared. This can be avoided in small applications, but in large applications, these terms are required to be kept in mind else, it may lead to chaos.

(http://surajpassion.in/wp-content/uploads/2015/01/chaos.jpg)

A loose coupling, yes the opposite, where there is very less dependency amongst the objects and the interfaces for which there is a chance of clean separation of concern and proper flexibility in the applications as it makes the framework more stable and lets proper maintainability. And it makes the developers happy.

(http://surajpassion.in/wp-content/uploads/2014/09/done.jpg)

Onion Architecture at glance



(http://surajpassion.in/wp-content/uploads/2015/01/onionfinal.png)

In the above image as we can see, the Core is the Domain model. This layer contains the POCO entities (http://www.codeproject.com/Articles/615499/Models-POCO-Entity-Framework-and-Data-Patterns).

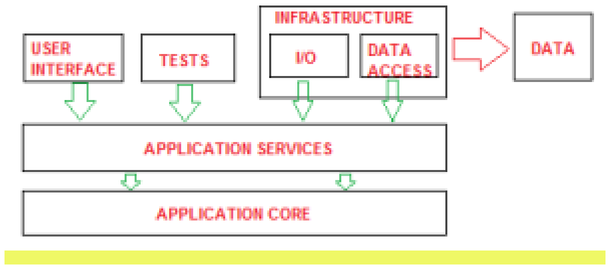
Domain objects are:-

encapsulates application business logic and rules
maintains any state that is required
does not depend on external infrastructure concerns
In this article and demo I have added the Repository interfaces in the Core.

The Layers above in brown has the Service Interfaces. The Layer in green is the implementation layer, or the Infrastructure layer, where the Repositories and the Services methods are implemented. The Error Logging(specially NLog) is used. Also the Dependency Injection is implemented here. To inject the dependencies into the controllers

In the layer in blue or the outer layer, has the testing and the User Interfaces.

Thus, this was a simple descriptions of the Architecture, but the below diagram explains better:-

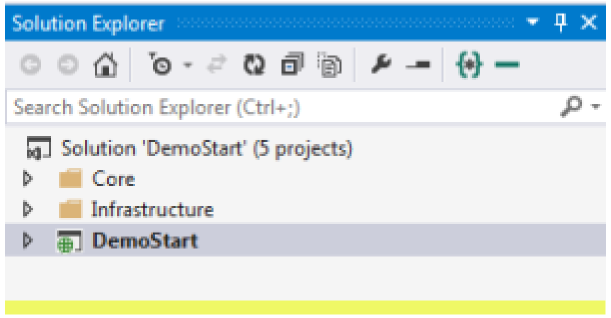


(http://surajpassion.in/wp-content/uploads/2015/01/descrp.png)

Here we can see there is no transitive dependency betweern the Test, UI and the Data Access which seems better for the Unit testing in MVC applications. This is the layered onion arcitecture that proves to make an application loosely coupled.
The application service implementation as we can see is in a separate laye & the dependency finally is on the Core Domain. The green arrows in the diagram represents the dependencies. Now letslook at a sample code/ The demo is provided in the article for download.

Using the code

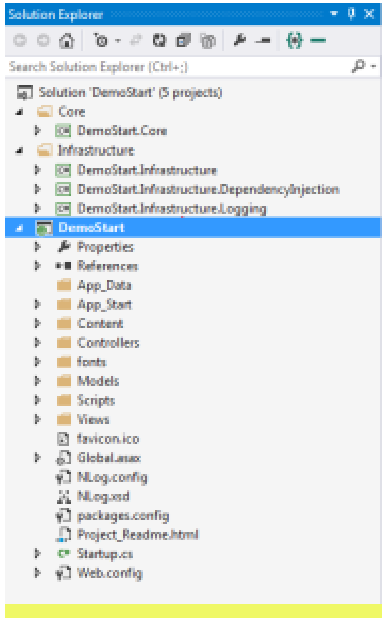
In this demo we will see how simple the folder structure of the solution is. Lets see the structure:-


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DemoStart' (5 projects)
Core
Infrastructure
DemoStart

(http://surajpassion.in/wp-content/uploads/2015/01/archi1.png)

This is how simple the solution of the project would look like if we ollow Onion architecture.

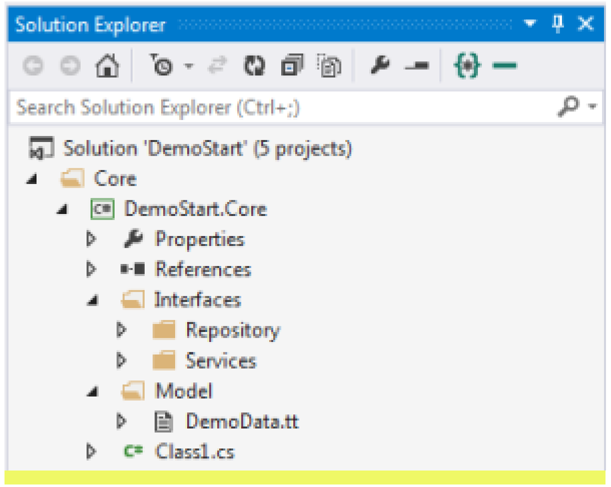
Now lets look at an expanded project:-



(http://surajpassion.in/wp-content/uploads/2015/01/archi2.png)

Lets Discuss on each of the folders one by one

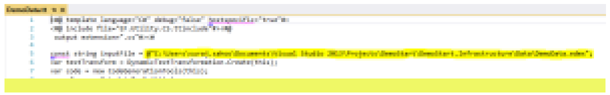
The Core



```
Solution 'DemoStart' (5 projects)
◢ Core
    ◢ DemoStart.Core
        ▷ Properties
        ▷ References
        ◢ Interfaces
            ▷ Repository
            ▷ Services
        ◢ Model
            ▷ DemoData.tt
        ▷ Class1.cs
```

(http://surajpassion.in/wp-content/uploads/2015/01/coredetail.png)

This is how the core looks like. The core folder contains a class library project, that has Interfaces both for Repositories & Services and Model with the .tt(T4 template)file that is autogenerated containing the POCO entities as I have used Database first approach here. An important thing to note here is, the .edmx file that is generated on using the *Database first approach* that contains the .tt file. To move that to the Core folder from the .edmx, *cut the .tt file and paste that in the folder you want to, here Core->Models->. Only doing this doesnot end the topic, we need to specify the physical path of that file as shown in the diagram below:-*



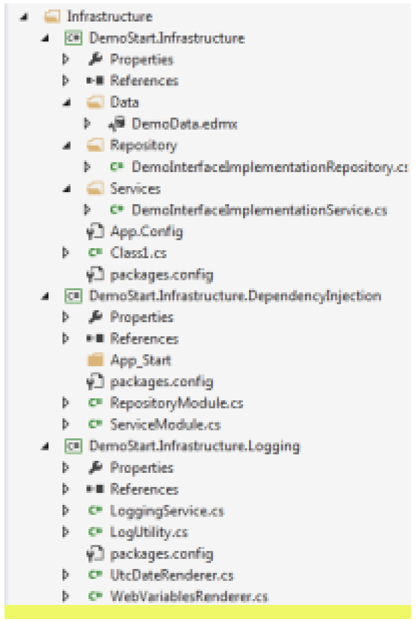
```
<#@ template language="C#" debug="false" hostspecific="true"#>
<#@ include file="EF.Utility.CS.ttinclude"#><#@
 output extension=".cs"#><#

const string inputFile = @"C:\Users\[illegible]\Documents\Visual Studio 2013\Projects\DemoStart\DemoStart.Infrastructure\Data\DemoData.edmx";
var textTransform = DynamicTextTransformation.Create(this);
var code = new CodeGenerationTools(this);
```

(http://surajpassion.in/wp-content/uploads/2015/01/ttfile.png)

As, you can see in the image, there is a folder called Interface, as the name suggests, this contains the Repositories and Service Interfaces.

Infrastructure



```
Infrastructure
  DemoStart.Infrastructure
    Properties
    References
    Data
      DemoData.edmx
    Repository
      DemoInterfaceImplementationRepository.cs
    Services
      DemoInterfaceImplementationService.cs
    App.Config
    Class1.cs
    packages.config
  DemoStart.Infrastructure.DependencyInjection
    Properties
    References
    App_Start
    packages.config
    RepositoryModule.cs
    ServiceModule.cs
  DemoStart.Infrastructure.Logging
    Properties
    References
    LoggingService.cs
    LogUtility.cs
    packages.config
    UtcDateRenderer.cs
    WebVariablesRenderer.cs
```

(http://surajpassion.in/wp-content/uploads/2015/01/infrastrcutredeta.png)

This folder contains more than one project & contains the Integral part of an application. The first project in this project is a class library project, with the first folder Data containing the .edmx file. An *.edmx* file according to MSDN (http://msdn.microsoft.com/en-us/library/vstudio/cc982042(v=vs.100).aspx) is a conceptual model and a storage model and the relationships between them. This also contains the Context classes and the .tt file, but regarding .tt file we have already discussed in the core, as we have moved this file to the core project. The first projectalso containes the Repository & Service implementations as mentioned in the interface.

The main and the best part is the second project in the Infrastructure i.e. the Dependency Injection an integral part when we want the Separation of Concern/Loose coupling in an application. In the above figure, we have two classes one for Repository Module and the Service Module. In the demo project, I have used Ninject. On adding Ninject for MVC from Nuget Package, it adds NinjectWebCommon.cs in the App_Start folder which looks like below snippet:

```
using System.Collections.Generic; using DemoStart.Core.Interfaces; using DemoStart.Infrastructure.DependencyInjection; using DemoStart.Infrastructure.Logging; using DemoStart.Infrastructure.Services; using Ninject.Modules; [assembly: WebActivatorEx.PreApplicationStartMethod(typeof(DemoStart.App_Start.NinjectWebCommon), &quot;Start&quot;)] [assembly: WebActivatorEx.ApplicationShutdownMethodAttribute(typeof(DemoStart.App_Start.NinjectWebCommon), &quot;Stop&quot;)] namespace DemoStart.App_Start { using System; using System.Web; using Microsoft.Web.Infrastructure.DynamicModuleHelper; using Ninject; using Ninject.Web.Common; public static class NinjectWebCommon { private static readonly Bootstrapper bootstrapper = new Bootstrapper(); /// &lt;summary&gt; /// Starts the application /// &lt;/summary&gt; public static void Start() { DynamicModuleUtility.RegisterModule(typeof(OnePerRequestHttpModule)); DynamicModuleUtility.RegisterModule(typeof(NinjectHttpModule)); bootstrapper.Initialize(CreateKernel); } &lt;summary&gt; /// Stops the application. /// &lt;/summary&gt; public static void Stop() { bootstrapper.ShutDown(); } /// &lt;summary&gt; /// Creates the kernel that will manage your application. /// &lt;/summary&gt; /// &lt;returns&gt;The created kernel.&lt;/returns&gt; private static IKernel CreateKernel() { var kernel = new StandardKernel(); try { kernel.Bind&lt;func&lt;ikernel&gt;&gt;().ToMethod(ctx =&gt; () =&gt; new Bootstrapper().Kernel); kernel.Bind&lt;ihttpmodule&gt;().To&lt;httpapplicationinitializationhttpmodule&gt;(); RegisterServices(kernel); return kernel; } catch { kernel.Dispose(); throw; } } &lt;summary&gt; /// Load your modules or register your services here! /// &lt;/summary&gt; ///The kernel. private static void RegisterServices(IKernel kernel) { var modules = new List&lt;ininjectmodule&gt; { new RepositoryModule(), new ServiceModule() }; kernel.Load(modules); kernel.Bind&lt;iloggingservice&gt;().To&lt;loggingservice&gt;(); } } }
```

```
using System.Collections.Generic;
using DemoStart.Core.Interfaces;
using DemoStart.Infrastructure.DependencyInjection;
using DemoStart.Infrastructure.Logging;
using DemoStart.Infrastructure.Services;
using Ninject.Modules;
[assembly: WebActivatorEx.PreApplicationStartMethod(typeof(DemoStart.App_Start.NinjectWebCommon),
&quot;Start&quot;)]
[assembly:
WebActivatorEx.ApplicationShutdownMethodAttribute(typeof(DemoStart.App_Start.NinjectWebCommon),
&quot;Stop&quot;)]

namespace DemoStart.App_Start
{
    using System;
    using System.Web;

    using Microsoft.Web.Infrastructure.DynamicModuleHelper;

    using Ninject;
    using Ninject.Web.Common;

    public static class NinjectWebCommon
    {
        private static readonly Bootstrapper bootstrapper = new Bootstrapper();

        ///
        &lt;summary&gt; /// Starts the application /// &lt;/summary&gt;

        public static void Start()
        {
            DynamicModuleUtility.RegisterModule(typeof(OnePerRequestHttpModule));
            DynamicModuleUtility.RegisterModule(typeof(NinjectHttpModule));
            bootstrapper.Initialize(CreateKernel);
        }

        &lt;summary&gt; /// Stops the application. /// &lt;/summary&gt;

        public static void Stop()
        {
            bootstrapper.ShutDown();
        } ///

        &lt;summary&gt; /// Creates the kernel that will manage your application. /// &lt;/summary&gt;

        /// &lt;returns&gt;The created kernel.&lt;/returns&gt;

        private static IKernel CreateKernel()
        {
            var kernel = new StandardKernel();
            try
            {
                kernel.Bind&lt;func&lt;ikernel&gt;&gt;().ToMethod(ctx =&gt; () =&gt; new Bootstrapper().Kernel);
                kernel.Bind&lt;ihttpmodule&gt;().To&lt;httpapplicationinitializationhttpmodule&gt;();
                RegisterServices(kernel);
                return kernel;
            }
            catch
            {
                kernel.Dispose();
                throw;
            }
        }

        &lt;summary&gt; /// Load your modules or register your services here! /// &lt;/summary&gt;

        ///The kernel.

        private static void RegisterServices(IKernel kernel)
        {
            var modules = new List&lt;ininjectmodule&gt;
            {
                new RepositoryModule(), new ServiceModule()
            };

            kernel.Load(modules);
            kernel.Bind&lt;iloggingservice&gt;().To&lt;loggingservice&gt;();
        }
    }
}
```

Once we have added the Ninject for the MVC in our project, the next big thing to do is to bind the Interfaces of the repositories and services to the Implementations like as below snippet:

For RepositoryModule.cs

public class RepositoryModule :NinjectModule { public override void Load() { // BINDINGS.. Bind&lt;idemointerfacerepository&gt;().To&lt;demointerfaceimplementationrepository&gt;(); } }

```
public class RepositoryModule :NinjectModule
    {
        public override void Load()
        {
            // BINDINGS..
            Bind&lt;idemointerfacerepository&gt;().To&lt;demointerfaceimplementationrepository&gt;();
        }
    }
```

For ServiceModule.cs

public class ServiceModule : NinjectModule { public override void Load() { // BINDINGS.. Bind&lt;idemointerfaceservice&gt;().To&lt;demointerfaceimplementationservice&gt;(); } }

```
public class ServiceModule : NinjectModule
  {
     public override void Load()
     {
         // BINDINGS..
         Bind&lt;idemointerfaceservice&gt;().To&lt;demointerfaceimplementationservice&gt;();

     }
  }
```

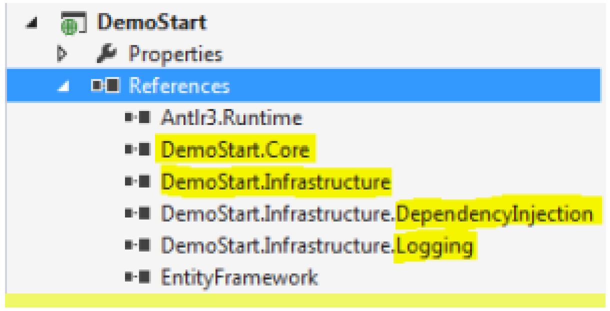
Now, you would be wondering why Repository and Service?? The Service methods are the Bridge/Flyover between the Controller/Business Logic and the Repository implementation, for making it more loosely couple as there would be another layer between the Business logic and the Data access layer.

The next project is the Logging project that is the error logging that helps in logging the errors/exceptions to the database. Here in this demo project, I have used NLog.

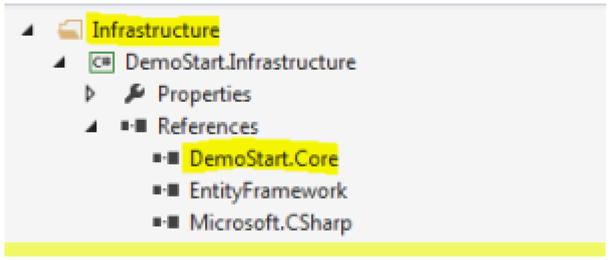
Dependencies

In this architecture, the Web project has its dependencies on the other two projects i.e. the Core & the Infrastructure.

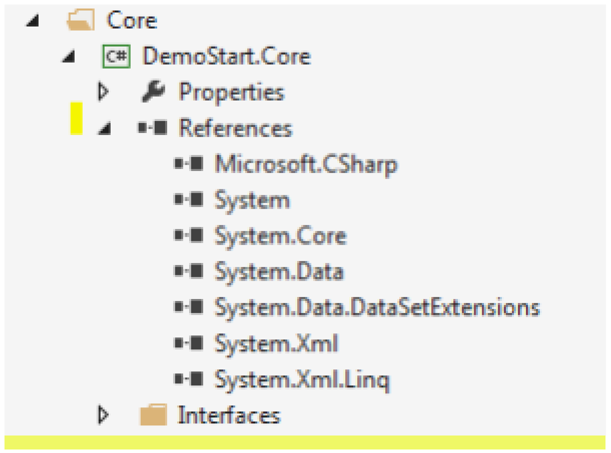
The Infrastructure project depends on the Core & the Core is independent for every other. As per the diagram for the Onion Architecture we have proved that the Core remains in the Core/Center and there is no transitive dependency amongst the UI , the Test with the data Access Layer. Check the images below:-


```
DemoStart
  Properties
  References
    Antlr3.Runtime
    DemoStart.Core
    DemoStart.Infrastructure
    DemoStart.Infrastructure.DependencyInjection
    DemoStart.Infrastructure.Logging
    EntityFramework
```

(http://surajpassion.in/wp-content/uploads/2015/01/webprjdepndency.png)


```
Infrastructure
  DemoStart.Infrastructure
    Properties
    References
      DemoStart.Core
      EntityFramework
      Microsoft.CSharp
```

(http://surajpassion.in/wp-content/uploads/2015/01/infrsdepndency.png)



(http://surajpassion.in/wp-content/uploads/2015/01/coredepndency.png)

You can see check the dependencies among the projects and the Core which is independent.

Points of Interest

Finally, straight from the words of Jeffery Palermo, the father of this concept:-

The application is built around an independent object model.

The layers inside define the Interfaces(the core) and the Outer layers implement

The coupling direction is towards the center as the name suggests ONION

Thus, the Onion Architecture helps decouple the Infrastructure and the Business(Controller) & the User Interface(Views) without getting into the OOPS concept or has no new concepts with regards to the domain-driven approach.

This is thus a very simple and general, but very effective architecture, I hope readers would love this.

References

Jeffery Palmero (http://jeffreypalermo.com/blog/the-onion-architecture-part-1/)

Peel Onion Archi. (https://www.develop.com/onionarchitecture)

Download the source code from dropbox and get set goo...:)

Search

Regd. Office

Shradha Bhawan, Madhupatana,
Rajendra Nagar, Cuttack-753010
Odisha, India

Branch Office

Plot No CP-117, 1st & 2nd Floor,
Niladri Vihar, Bhubaneswar-751021,
Odisha, India

Call Us

+91 674 2972 316
+91 9861 351 470

Write Us

info@invincix.com
business@invincix.com
hr@invincix.com
careers@invincix.com

Follow Us On

We will keep our feet grounded, to ensure, your head is in the cloud

~ INVINCIANS

Terms of Use (/terms-use)

Privacy Policy (/privacy-policy)