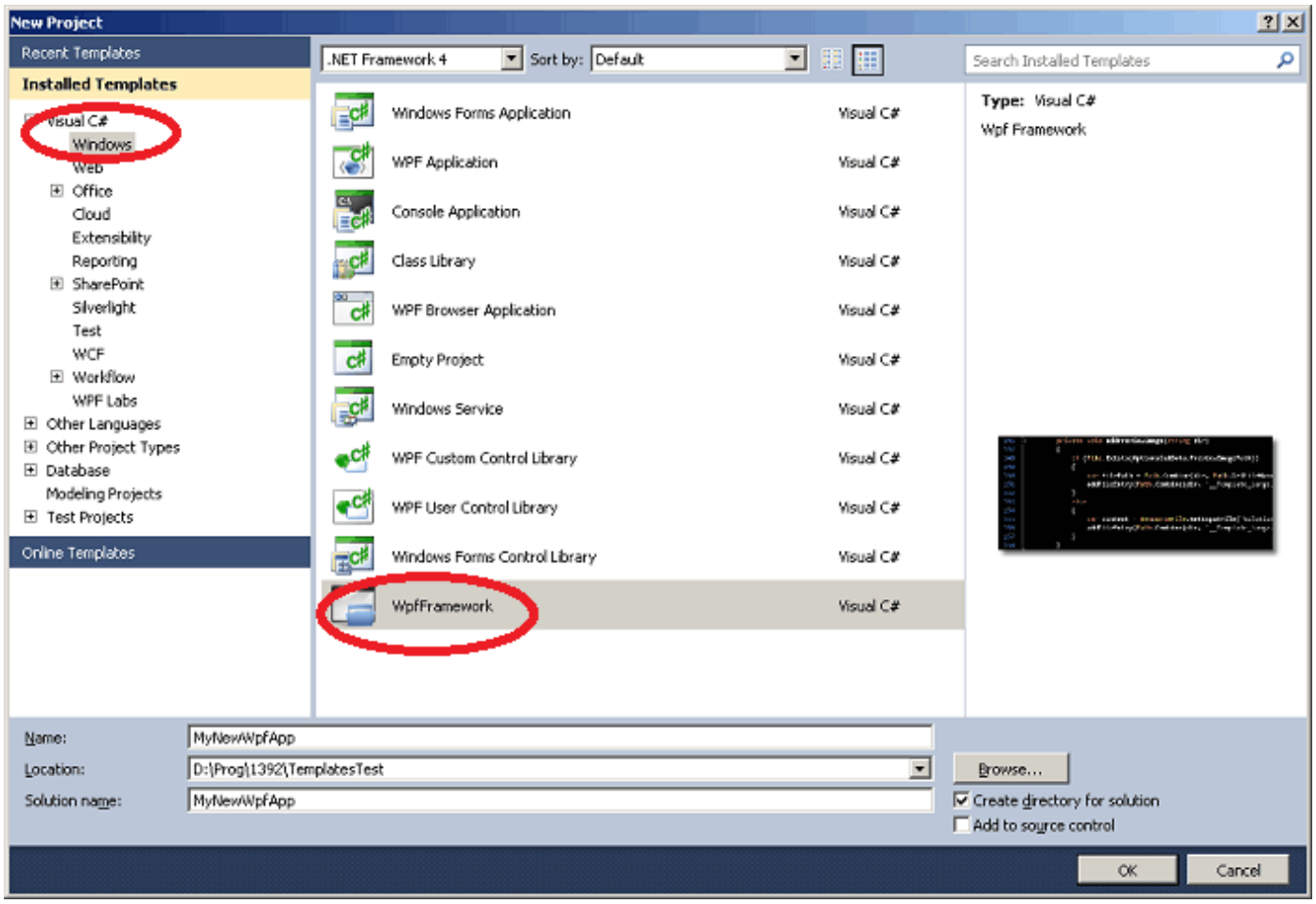
عنوان: **دریافت قالب WpfFramework.vsix و نحوه نصب و راه اندازی آن**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۵ ۱۸:۳۱
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

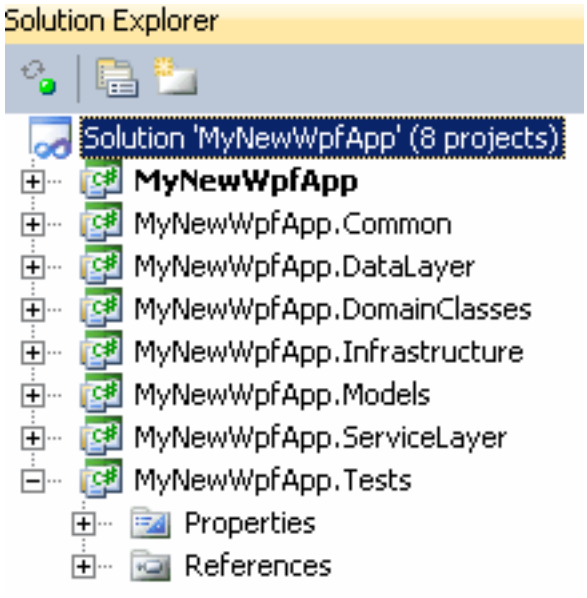
فایل قالب پروژه دوره جاری را از آدرس ذیل می‌توانید دریافت کنید:

WpfFramework.vsix

نصب ابتدایی آن بسیار ساده است و نکته خاصی ندارد.
پس از نصب، یکبار VS.NET را بسته و سپس باز کنید. این قالب جدید، ذیل قسمت پروژه‌های ویندوز مرتبط با ویژوال سی شارپ، ظاهر خواهد شد:

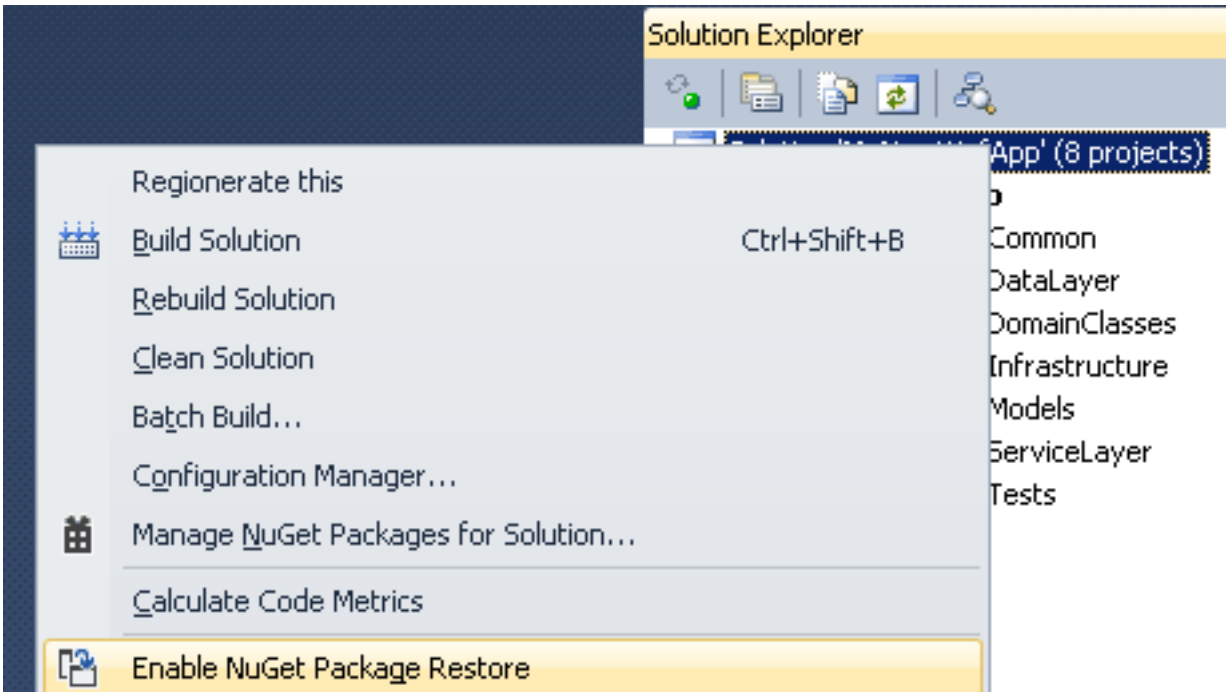


در این حال اگر یک پروژه جدید را آغاز کنید، این قالب تدارک دیده شده، لایه‌ها و قسمت‌های مختلف را به صورت خودکار اضافه خواهد کرد:

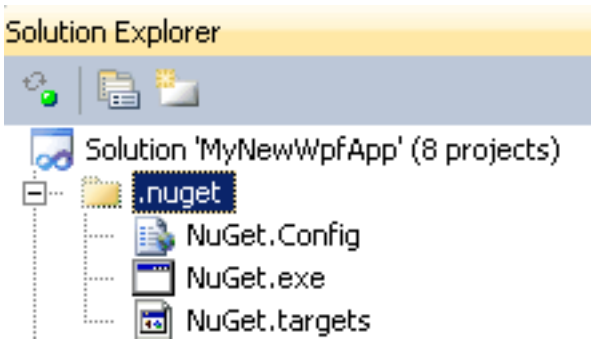


نکته مهم! برنامه کامپایل نمی‌شود!

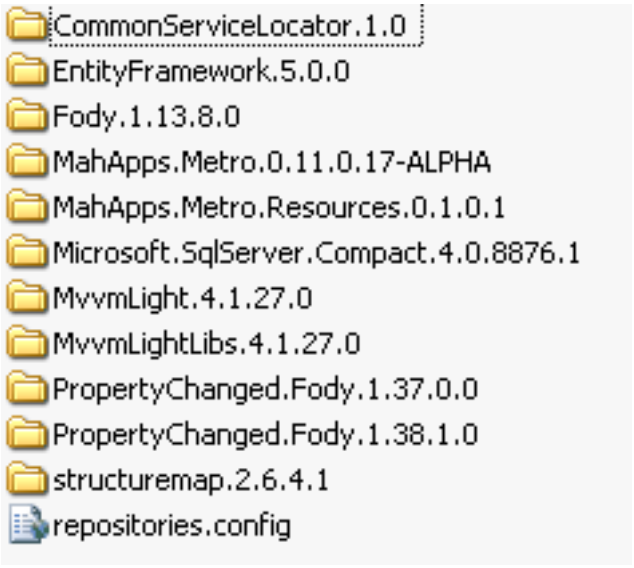
به عمد، جهت کاهش حجم قالب دریافتی فوق، فایل‌های باینری آن پیوست نشده‌اند. وگرنه باید بالای 30 مگابایت را دریافت می‌کردید که واقعا نیازی نیست.
اما اگر به هر پروژه داخل Solution دقت کنید، فایل متنی packages.config آن نیز پیوست شده است. به کمک این فایل‌ها به سادگی می‌توان تمام وابستگی‌ها را از طریق NuGet بازیابی کرد.
برای این منظور ابتدا به اینترنت متصل شده (مهم) و سپس بر روی Solution کلیک راست کرده و گزینه فعال سازی Restore کلیه بسته‌های نیوگت را انتخاب کنید.



پس از اینکار، آخرین نگارش NuGet.exe از اینترنت دریافت و به پروژه اضافه می‌شود:



اکنون اگر Solution را Build کنید، اولین کاری که صورت خواهد گرفت، دریافت کلیه وابستگی‌ها از سایت NuGet است. اندکی تامل کنید تا اینکار تمام شود.
پس از پایان کار دریافت، پوشه packages در کنار فایل sln پروژه ایجاد شده، تشکیل می‌شود.
یکبار وجود آن‌را بررسی کنید.



اکنون اگر برنامه را Build کنید احتمالا پیغام می‌دهد که Fody را نمی‌تواند پیدا کند با اینکه دریافت شده و در پوشه packages موجود است. هر زمان، هر پیغام خطایی در مورد Fody مشاهده کردید، فقط یکبار VS.NET را بسته و باز کنید. مشکل حل می‌شود!

تنها کاری که پس از بازسازی پوشه packages بهتر است صورت گیرد (اختیاری است البته)، صدور دستور ذیل در خط فرمان پاور شل است:

```
PM> Update-Package
```

با این دستور، دریافت کامل مجددی از اینترنت صورت نمی‌گیرد؛ مگر اینکه به روز رسانی جدیدی را یافت کند. در سایر حالات از کش داخل سیستم اطلاعات را دریافت می‌کند.

پس از اینکار، نیاز است یکبار VS.NET را بسته و مجددا باز کنید (مهم) تا تمام وابستگی‌ها به درستی بارگذاری شوند. خصوصا بسته Fody که کار AOP را انجام می‌دهد. در غیر اینصورت موفق به Build پروژه نخواهید شد.

بنابراین به صورت خلاصه:
الف) یکبار گزینه فعال سازی Restore کلیه بسته‌های نیوگت را انتخاب کنید.
ب) پروژه را Build کنید تا وابستگی‌ها را از سایت NuGet دریافت کند.
ج) دستور Update-Package را اجرا نمائید (اختیاری).
ج) VS.NET را پس از سه مرحله فوق، یکبار بسته و باز کنید.

در کل بسته‌های مورد استفاده به این شرح هستند:

```
PM> Install-Package MahApps.Metro -Pre
PM> Install-Package MahApps.Metro.Resources
PM> Install-Package EntityFramework
PM> Install-Package structuremap
PM> Install-Package PropertyChanged.Fody
PM> Install-Package MvvmLight
PM> Install-Package Microsoft.SqlServer.Compact
```

یک نکته جانبی
فید NuGet در VS.NET به Https تنظیم شده است. اگر دسترسی به **s** Http برای شما به کندی صورت می‌گیرد فقط کافی است مسیر فید آن‌را در منوی Tools، گزینه‌ی Options، ذیل قسمت Package manager یافته و به http://nuget.org/api/v2 تغییر دهید؛ یعنی به Http خالی، بجای **s** Http ؛ تا سرعت دریافت بسته‌های NuGet مورد نظر افزایش یابند.

اجرای پروژه و برنامه

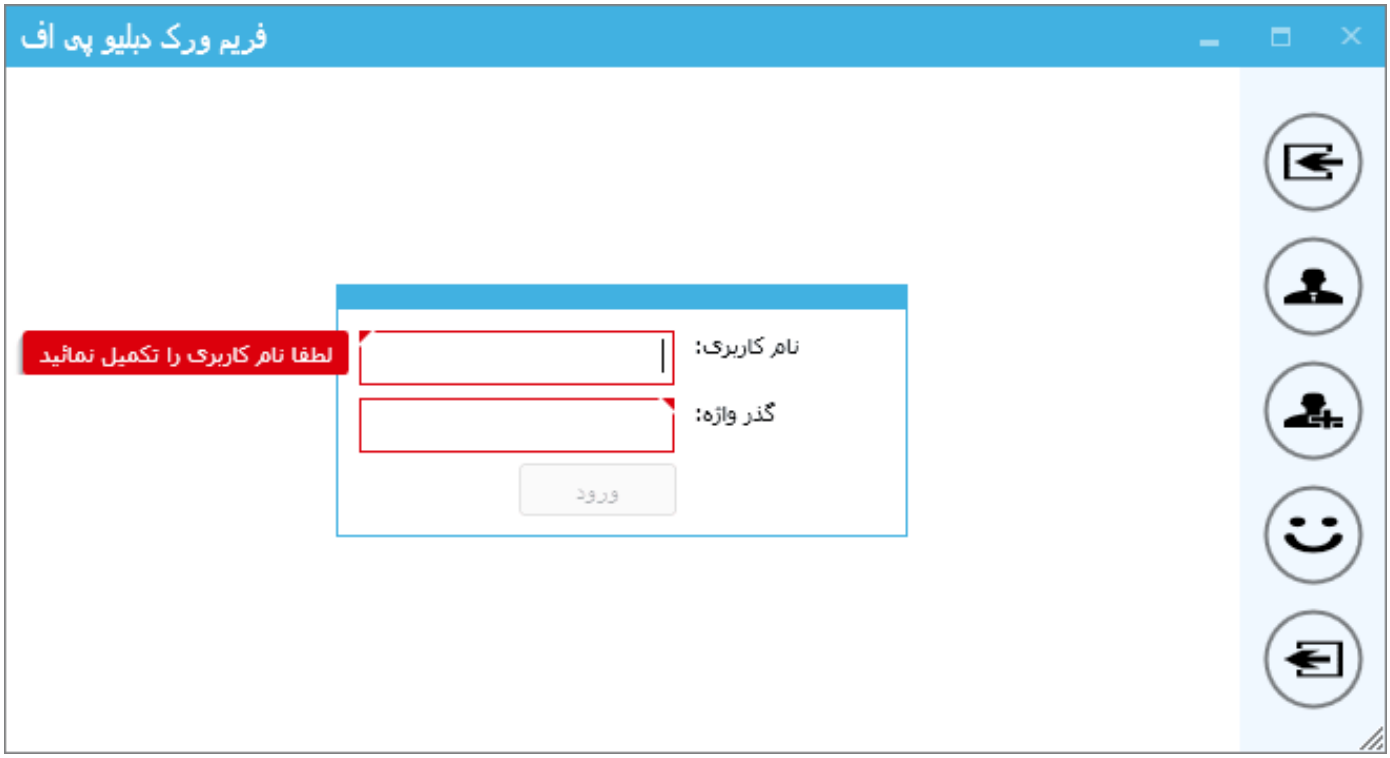
پس از این مراحل و Build موفقیت آمیز پروژه، برنامه را اجرا کنید. در اولین بار اجرای برنامه به صورت خودکار بانک اطلاعاتی به همراه ساختار جداول تشکیل می‌شوند. اما ... با خطای زیر مواجه خواهید شد:

```
The path is not valid. Check the directory for the database. [ Path = D:\...\bin\Debug\Db\db.sdf ]
```

علت این است که در فایل app.config پروژه ریشه برنامه :

```
<connectionStrings>
    <clear/>
    <add name="MyWpfFrameworkContext"
         connectionString="Data Source=|DataDirectory|\Db\db.sdf;Max Buffer Size=30720;File Mode=Read
Write;"
         providerName="System.Data.SqlServerCE.4.0" />
  </connectionStrings>
```

مسیر تشکیل بانک اطلاعاتی به پوشه db کنار فایل اجرایی برنامه تنظیم شده است. این پوشه db را در پوشه bin\debug ایجاد کرده و سپس برنامه را اجرا کنید.



برای ورود به برنامه از نام کاربری Admin و کلمه عبور 123456 استفاده نمائید.
این کاربر پیش فرض در کلاس MyWpfFrameworkMigrations لایه داده‌های برنامه، توسط متد addRolesAndAdmin اضافه شده است.

خوب! تا اینجا با نحوه نصب و راه اندازی این قالب جدید آشنا شدیم. در قسمت‌های بعد، به جزئیات ارتباطات و نحوه استفاده از آن به عنوان پایه یک کار و پروژه جدید، خواهیم پرداخت.

به روز رسانی 1.1

جهت سازگاری با StructureMap 3 ، EF 6 و همچنین VS 2013، پروژه به روز شد: WpfFrmwork_1.1.zip

نظرات خوانندگان

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۰۹ ۲۲:۵۸

با سلام
با تشکر بابت شروع این دوره ، می‌خواستم بدونم این فریمورک قراره ماهیت آموزشی داشته باشه یا اینکه توسعه داده میشه؟ آیا می‌تونیم توی توسعه این فریم ورک همکاری داشته باشیم؟
یک باگ کوچیک توی قسمت GetTableName برای sql server وجود داره، با توجه به نام جدول ترتیب جایگزینی صحیح نیست.

```
//table="[dbo].[Users]"
    return table
                    .Replace("dbo.", string.Empty)
                    .Replace("`", string.Empty)
                    .Replace("[", string.Empty)
                    .Replace("]", string.Empty)
                    .Trim()


     return table
                    .Replace("`", string.Empty)
                    .Replace("[", string.Empty)
                    .Replace("]", string.Empty)
                    .Replace("dbo.", string.Empty)
                    .Trim();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۹ ۲۳:۹

ممنون. بله. اگر مشکلی مشاهده شد یا بهبودی مدنظر شما بود، لطفا همینجا مطرح کنید. یا در قسمت اختصاصی پرسش و پاسخ دوره مطرح کنید. با تشکر.

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۵:۴۵

با سلام.
امکان دارد در مورد نحوه ایجاد قالب برای یک solution با چند پروژه (همانند دوره‌ی جاری) راهنمایی بفرمایید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۷:۳۴

از برنامه solution template generator استفاده کردم.

نویسنده: مهدی ع
تاریخ: ۱۳۹۲/۰۴/۰۴ ۹:۳۶

با عرض سلام و خسته نباشید
چرا در این فریمورک نمی‌توان با Webbrowser پیش فرض Wpf کار کرد؟
محتوای سایت بارگزاری می‌شود اما کاربر نمی‌تواند چیزی را مشاهده کند

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۰۴ ۱۰:۱۰

مشکلی مشاهده نشد: (تصویری است از یک کنترل مرورگر وب که در آن یک صفحه html بارگذاری شده است)

ضمنا، موضوع بحث مطلب جاری «نصب و راه اندازی» است. هر دوره یک قسمت پرسش و پاسخ جداگانه *هم دارد.*

نویسنده: سید اسماعیل
تاریخ: ۱۳۹۲/۰۹/۱۶ ۲۱:۲۸

با سلام
پیشنهاد می‌کنم قبل از نصب فریم ورک NuGet را ارتقا دهید تا جدیدترین ورژن بسته‌ها هم قابل نصب باشند
`NuGet upadate path`

نویسنده: م ش
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۴:۴

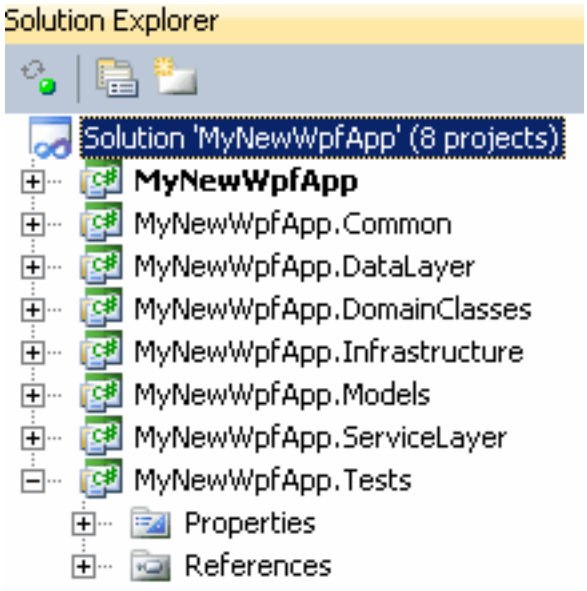
با سلام، بسته‌ی نصبی بر روی ویژوال استودیو 2013 نصب نمی‌شود. لطفاً راهنمایی کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۹:۴۶

نگارش جدید آن‌را از انتهای مطلب دریافت کنید.

عنوان: **بررسی قسمت‌های مختلف قالب پروژه WPF Framework تهیه شده**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۵ ۲۲:۳۴
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

پس از ایجاد یک Solution جدید توسط قالب WPF Framework، هشت پروژه به صورت خودکار اضافه خواهند شد:



1) پروژه ریشه که بسته به نامی که در ابتدای کار انتخاب می‌کنید، تغییر نام خواهد یافت.

برای مثال اگر نام وارد شده در ابتدای کار MyWpfFramework باشد، این پروژه ریشه نیز، MyWpfFramework نام خواهد داشت. از آن صرفا جهت افزودن Viewهای برنامه استفاده می‌کنیم. کلیه Viewها در پوشه View قرار خواهند گرفت و با توجه به ساختار خاصی که در اینجا انتخاب شده، این Viewها باید از نوع Page انتخاب شوند تا با سیستم راهبری فریم ورک هماهنگ کار کنند.

در داخل پوشه Views، هر بخش از برنامه را می‌توان داخل یک زیر پوشه قرار داد. برای مثال قسمت Login سیستم، دارای سه صفحه ورود، نمایش پیام خوش آمد و نمایش صفحه عدم دسترسی است.

متناظر با هر Page اضافه شده، در پروژه MyWpfFramework.Infrastructure یک ViewModel در صورت نیاز اضافه خواهد شد. قرار داد ما در اینجا ترکیب نام View به علاوه کلمه ViewModel است. برای مثال اگر نام View اضافه شده به پروژه ریشه برنامه، LoginPage است، نام ViewModel متناظر با آن باید LoginPageViewModel باشد تا به صورت خودکار توسط برنامه ردیابی و وهله سازی گردد.

این پروژه از کتابخانه MahApps.Metro استفاده می‌کند و اگر به فایل MainWindow.xaml.cs آن مراجعه کنید، ارث بری پنجره اصلی برنامه را از کلاس MetroWindow مشاهده خواهید نمود. این فایل‌ها نیازی به تغییر خاصی نداشته و به همین نحو در این قالب قابل استفاده هستند.

و در پوشه Resources آن یک سری قلم و آیکون را می‌توانید مشاهده نمائید.

2) پروژه MyWpfFramework.Common

در این پروژه کلاس‌هایی قرار می‌گیرند که قابلیت استفاده در انواع و اقسام پروژه‌های WPF را دارند و الزاما وابسته به پروژه جاری نیستند. یک سری کلاس‌های کمکی در این پروژه Common قرار گرفته‌اند و قسمت‌های مختلف سیستم را تغذیه می‌کنند؛ مانند خواندن اطلاعات از فایل کانفیگ، هش کردن کلمه عبور، یک سری متد عمومی برای کار با EF، کلاس‌های عمومی مورد نیاز در حین استفاده از الگوی MVVM، اعتبارسنجی و امثال آن.

در این پروژه از کلاس PageAuthorizationAttribute آن جهت مشخص سازی وضعیت دسترسی به صفحات تعریف شده در پروژه ریشه استفاده خواهد شد.

نمونه‌ای از آن‌را برای مثال با مراجعه به سورس صفحه About.xaml.cs می‌توانید مشاهده کنید که در آن AuthorizationType.AllowAnonymous تنظیم شده و به این ترتیب تمام کاربران اعتبارسنجی نشده می‌توانند این صفحه را مشاهده کنند.

همچنین در این پروژه کلاس BaseViewModel قرار دارد که جهت مشخص سازی کلیه کلاس‌های ViewModel برنامه باید مورد استفاده قرار گیرد. سیستم طراحی شده، به کمک این کلاس پایه است که می‌تواند به صورت خودکار ViewModelهای متناظر با Viewها را یافته و وهله سازی کند (به همراه تمام وابستگی‌های تزریق شده به آن‌ها).

به علاوه کلاس DataErrorInfoBase آن برای یکپارچه سازی اعتبارسنجی با EF طراحی شده است. اگر به کلاس BaseEntity.cs مراجعه کنید که در پروژه MyWpfFramework.DomainClasses قرار دارد، نحوه استفاده آن‌را ملاحظه خواهید نمود. به این ترتیب حجم بالایی از کدهای تکرای، کپسوله شده و قابلیت استفاده مجدد را پیدا می‌کنند.

قسمت‌های دیگر پروژه Common، برای ثبت وقایع برنامه مورد استفاده قرار می‌گیرند. استفاده از آن‌ها را در فایل App.xaml.cs پروژه ریشه برنامه ملاحظه می‌کنید و نیاز به تنظیم خاص دیگری در اینجا وجود ندارد.

3) پروژه MyWpfFramework.DataLayer

کار تنظیمات EF در اینجا انجام می‌شود (و قسمت عمده‌ای از آن انجام شده است). تنها کاری که در آینده برای استفاده از آن نیاز است انجام شود، مراجعه به کلاس MyWpfFrameworkContext.cs و افزودن DbSetهای لازم است. همچنین اگر نیاز به تعریف نگاشت‌های اضافه‌تری وجود داشت، می‌توان از پوشه Mappings آن استفاده کرد.

در این پروژه الگوی واحد کار پیاده سازی شده است و همچنین سعی شده تمام کلاس‌های آن دارای کامنت‌های کافی جهت توضیح قسمت‌های مختلف باشند.

کلاس MyDbContextBase به اندازه کافی غنی سازی شده‌است، تا در وقت شما، در زمینه تنظیم مباحثی مانند اعتبارسنجی و نمایش پیغام‌های لازم به کاربر، یک دست سازی ی و ک ورودی در برنامه و بسیاری از نکات ریز دیگر صرفه جویی شود.

در اینجا از خاصیت ContextHasChanges جهت بررسی وضعیت Context جاری و نمایش پیغامی به کاربر در مورد اینکه آیا مایل هستید تغییرات را ذخیره کنید یا خیر استفاده می‌شود.

در متد auditFields آن یک سری خاصیت کلاس BaseEntity که پایه تمامی کلاس‌های Domain model برنامه خواهد بود به صورت خودکار مقدار دهی می‌شوند. مثلا این رکورد را چه کسی ثبت کرده یا چه کسی ویرایش و در چه زمانی. به این ترتیب دیگر نیازی نیست تا در برنامه نگران تنظیم و مقدار دهی آن‌ها بود.

کلاس MyWpfFrameworkMigrations به حالت AutomaticMigrationsEnabled تنظیم شده است و ... برای یک برنامه دسکتاپ WPF کافی و مطلوب است و ما را از عذاب به روز رسانی دستی ساختار بانک اطلاعاتی برنامه با تغییرات مدل‌ها، رها خواهد ساخت. عموما برنامه‌های دسکتاپ پس از طراحی، آنچنان تغییرات گسترده‌ای ندارند و انتخاب حالت Automatic در اینجا می‌تواند کار توزیع آن‌را نیز بسیار ساده کند. از این جهت که بانک اطلاعاتی انتخابی از نوع SQL Server CE نیز عمدا این هدف را دنبال می‌کند: عدم نیاز به نگهداری و وارد شدن به جزئیات نصب یک بانک اطلاعاتی بسیار پیشرفته مانند نگارش‌های کامل SQL Server. هرچند زمانیکه با EF کار می‌کنیم، سوئیچ به بانک‌های اطلاعاتی صرفا با تغییر رشته اتصالی فایل app.config برنامه اصلی و مشخص سازی پروایدر مناسب قابل انجام خواهد بود.

در فایل MyWpfFrameworkMigrations، توسط متد addRolesAndAdmin کاربر مدیر سیستم در آغاز کار ساخت بانک اطلاعاتی به صورت خودکار افزوده خواهد شد.

4) پروژه MyWpfFramework.DomainClasses

کلیه کلاس‌های متناظر با جداول بانک اطلاعاتی در پروژه MyWpfFramework.DomainClasses قرار خواهند گرفت. نکته مهمی که در اینجا باید رعایت شود، مزین کردن این کلاس‌ها به کلاس پایه BaseEntity می‌باشد که نمونه‌ای از آن‌را در کلاس User پروژه می‌توانید ملاحظه کنید.

BaseEntity چند کار را با هم انجام می‌دهد:

- اعمال خودکار DataErrorInfoBase جهت یکپارچه سازی سیستم اعتبارسنجی EF با WPF (برای مثال به این ترتیب خطاهای ذکر شده در ویژگی‌های خواص کلاس‌ها توسط WPF نیز خوانده خواهند شد)
- اعمال ImplementPropertyChanged به کلاس‌های دومین برنامه. به این ترتیب برنامه کمکی Fody که کار Aspect oriented programming را انجام می‌دهد، اسمبلی برنامه را ویرایش کرده و متدها و تغییرات لازم جهت پیاده سازی INotifyPropertyChanged را اضافه می‌کند. به این ترتیب به کلاس‌های دومین بسیار تمیزی خواهیم رسید با حداقل نیاز به تغییرات و نگهداری ثانویه.

- فراهم آوردن فیلدهای مورد نیاز جهت بازرسی سیستم؛ مانند اینکه چه کسی یک رکورد را ثبت کرده یا ویرایش و در چه زمانی

نقش‌های سیستم در کلاس SystemRole تعریف می‌شوند. به ازای هر نقش جدیدی که نیاز بود، تنها کافی است یک خاصیت bool را در اینجا اضافه کنید. سپس نام این خاصیت در ویژگی PageAuthorizationAttribute به صورت خودکار قابل استفاده خواهد بود. برای مثال به پروژه ریشه مراجعه و به فایل AddNewUser.xaml.cs دقت کنید؛ چنین تعریفی را در بالای کلاس مرتبط مشاهده خواهید کرد:

```
[PageAuthorization(AuthorizationType.ApplyRequiredRoles, "IsAdmin, CanAddNewUser")]
```

در اینجا AuthorizationType سه حالت را می‌تواند داشته باشد:

```
/// <summary>
    /// وضعیت اعتبار سنجی صفحه را مشخص می‌کند
    /// </summary>
    public enum AuthorizationType
    {
        /// <summary>
        /// همه می‌توانند بدون اعتبار سنجی، دسترسی به این صفحات داشته باشند
        /// </summary>
        AllowAnonymous,

        /// <summary>
        /// کاربران وارد شده به سیستم بدون محدودیت به این صفحات دسترسی خواهند داشت
        /// </summary>
        FreeForAuthenticatedUsers,

        /// <summary>
        /// بر اساس نام نقش‌هایی که مشخص می‌شوند تصمیم گیری خواهد شد
        /// </summary>
        ApplyRequiredRoles
    }
```

اگر حالت ApplyRequiredRoles را انتخاب کردید، در پارامتر اختیاری دوم ویژگی PageAuthorization نیاز است نام یک یا چند خاصیت کلاس SystemRole را قید کنید. بدیهی است کاربر متناظر نیز باید دارای این نقش‌ها باشد تا بتواند به این صفحه دسترسی پیدا کند، یا خیر.

5) پروژه MyWpfFramework.Models

در پروژه MyWpfFramework.Models کلیه Modelهای مورد استفاده در UI که الزاما قرار نیست در بانک اطلاعاتی قرارگیرند، تعریف خواهند شد. برای نمونه مدل صفحه لاگین در آن قرار دارد و ذکر دو نکته در آن حائز اهمیت است:

```
[ImplementPropertyChanged] // AOP
public class LoginPageModel : DataErrorInfoBase
```

- ویژگی ImplementPropertyChanged کار پیاده سازی INotifyPropertyChanged را به صورت خودکار سبب خواهد شد.
- کلاس پایه DataErrorInfoBase سبب می‌شود تا مثلا در اینجا اگر از ویژگی Required استفاده کردید، اطلاعات آن توسط برنامه خوانده شود و با WPF یکپارچه گردد.

6) پروژه MyWpfFramework.Infrastructure.csproj

در پروژه MyWpfFramework.Infrastructure.csproj تعاریف ViewModelهای برنامه اضافه خواهند شد.
این پروژه دارای یک سری کلاس پایه است که تنظیمات IoC برنامه را انجام می‌دهد. برای مثال FrameFactory.cs آن یک کنترل Frame جدید را ایجاد کرده است که کار تزریق وابستگی‌ها را به صورت خودکار انجام خواهد داد. فایل IocConfig آن جایی است که کار سیم کشی کلاس‌های لایه سرویس و اینترفیس‌های متناظر با آن‌ها انجام می‌شود. البته پیش فرض‌های آن را اگر رعایت کنید، نیازی به تغییری در آن نخواهید داشت. برای مثال در آن scan.TheCallingAssembly قید شده است. در این حالت اگر نام کلاس لایه سرویس شما Test و نام اینترفیس متناظر با آن ITest باشد، به صورت خودکار به هم متصل خواهند شد.
همانطور که پیشتر نیز عنوان شد، در پوشه ViewModels آن، به ازای هر View یک ViewModel خواهیم داشت که نام آن مطابق

قرار داد، نام View مدنظر به همراه کلمه ViewModel باید درنظر گرفته شود تا توسط برنامه شناخته شده و مورد استفاده قرار گیرد. همچنین هر ViewModel نیز باید دارای کلاس پایه BaseViewModel باشد تا توسط IoC Container برنامه جهت تزریق وابستگی‌های خودکار در سازنده‌های کلاس‌ها شناسایی شده و وهله سازی گردد.

7) پروژه MyWpfFramework.ServiceLayer

کلیه کلاس‌های لایه سرویس که منطق تجاری برنامه را پیاده سازی می‌کنند (خصوصا توسط EF) در این لایه قرار خواهند گرفت. در اینجا دو نمونه سرویس کاربران و سرویس عمومی AppContextService را ملاحظه می‌کنید.

سرویس AppContextService قلب سیستم اعتبارسنجی سیستم است و در IocConfig برنامه به صورت سینگلتون تعریف شده است. چون در برنامه‌های دسکتاپ در هر لحظه فقط یک نفر وارد سیستم می‌شود و نیاز است تا پایان طول عمر برنامه، اطلاعات لاگین و نقش‌های او را در حافظه نگه داری کرد.

8) پروژه MyWpfFramework.Tests

یک پروژه خالی Class library هم در اینجا جهت تعریف آزمون‌های واحد سیستم درنظر گرفته شده است.

نظرات خوانندگان

نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۲/۰۳/۳۱ ۱۵:۵۶

با سلام.
آیا بهتر نیست در پروژه DataLayer به جای استفاده مستقیم از کد زیر ،خطاها را درون کلاسی کپسوله کرده و بازگشت دهیم تا خود لایه UI در مورد نحوه نمایش خطا تصمیم بگیرد؟

```
new SendMsg().ShowMsg(
                    new AlertConfirmBoxModel
                    {
                        ErrorTitle = "خطای اعتبار سنجی",
                        Errors = errors,
                    }, validationException);
```

به جای آن :

```
public class DomainResult
    {
        public bool Succeed { get; set; }
        public IEnumerable<Exception> Errors { get; set; }
        public DomainErrorType ErrorType { get; set; }
    }
    public enum DomainErrorType
    {
        Validation, Concurrency, Update
    }
```

```
public DomainResult ApplyAllChanges(string userName, bool updateAuditFields = true) ...
```

با تشکر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۳۱ ۱۷:۳۳

فقط سه مورد (DbEntityValidationException, DbUpdateException, DbUpdateConcurrencyException) از استثناهای ویژه EF Code first در متد ApplyAllChanges بررسی شدند به همراه خطاهای اعتبارسنجی. مابقی استثناها در صورت رخ دادن به لایه‌های بالاتر منتشر خواهند شد (چون catch نشدند).
همچنین کدهای تکراری نحوه نمایش فقط این سه مورد ویژه و بررسی خطاهای اعتبارسنجی، ضرورتی به تکرار یا بررسی در کل برنامه را ندارد. نیازی نیست در کل برنامه if/else نوشت که بررسی شود آیا خطای اعتبارسنجی هست یا خیر، زمانیکه می‌شود آن‌را به صورت مرکزی و پاکیزه، مدیریت کرد و مدیریت این‌ها هم حالت خاص دیگری ندارد (باید لاگ شوند و باید به اطلاع کاربر رسانده شوند که هر دو مورد در اینجا خودکار است). حداکثر این است که از نحوه نمایش آن راضی نیستید. کار سورس باز است. تغییرش بدید. این روش و این صفحه دیالوگ مطابق سلیقه من طراحی شده.
به علاوه در لایه‌های بالاتر نیز نیازی به بررسی سایر استثناها نیست چون این موارد در فایل App.xaml.cs در بالاترین سطح ممکن دریافت و لاگ می‌شوند؛ همچنین به کاربر هم نمایش داده خواهند شد (در متدهای appDispatcherUnhandledException و currentDomainUnhandledException).

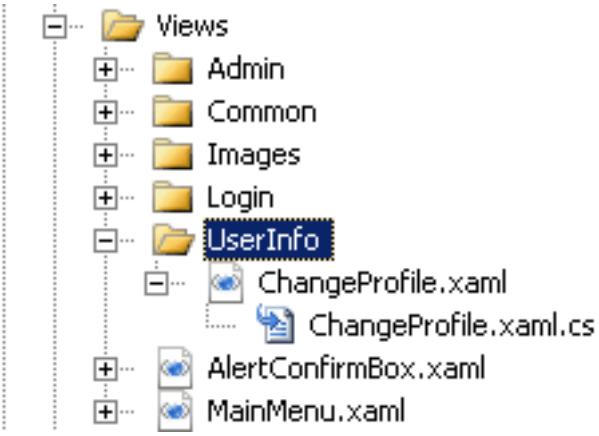
البته این برنامه دسکتاپ است که یک چنین اجازه‌ای رو می‌ده. در برنامه‌های وب این موارد توسط ELMAH لاگ خواهند شد و به کاربر پیغام کلی خطایی رخ داده نمایش داده می‌شود.

عنوان: **نحوه برقراری ارتباطات بین صفحات، سیستم راهبری و ViewModelها در قالب پروژه WPF Framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۶ ۱۰:۷
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

هدف از قالب پروژه WPF Framework ایجاد یک پایه، برای شروع سریع یک برنامه تجاری WPF جدید است. بنابراین فرض کنید که این قالب، هم اکنون در اختیار شما است و قصد دارید یک صفحه جدید، مثلا تغییر مشخصات کاربری را به آن اضافه کنید. کدهای کامل این قابلیت هم اکنون در قالب پروژه موجود است و به این ترتیب توضیح جزئیات روابط آن در اینجا ساده‌تر خواهد بود.

1) ایجاد صفحه تغییر مشخصات کاربر

کلیه Viewهای برنامه، در پروژه ریشه، ذیل پوشه Views اضافه خواهند شد. همچنین چون در آینده تعداد این فایل‌ها افزایش خواهند یافت، بهتر است جهت مدیریت آن‌ها، به ازای هر گروه از قابلیت‌ها، یک پوشه جدید را ذیل پوشه Views اضافه کرد.



همانطور که ملاحظه می‌کنید در اینجا پوشه UserInfo به همراه یک فایل جدید XAML به نام ChangeProfile.xaml، ذیل پوشه Views پروژه ریشه اصلی اضافه شده‌اند.
ChangeProfile.xaml از نوع Page است؛ از این جهت که اگر به فایل MainWindow.xaml که سیستم راهبری برنامه در آن تعبیه شده است مراجعه کنید، یک چنین تعریفی را ملاحظه خواهید نمود:

```
<CustomControls:FrameFactory
                    x:Name="ActiveScreen"
                    HorizontalContentAlignment="Stretch"
                    VerticalContentAlignment="Stretch"
                    NavigationUIVisibility="Hidden"
                    Grid.Column="1"
                    Margin="0" />
```

سورس کامل کنترل سفارشی FrameFactory.cs را در پروژه Infrastructure برنامه می‌توانید مشاهده کنید. FrameFactory در حقیقت یک کنترل Frame استاندارد است که مباحث تزریق وابستگی‌ها و همچنین راهبری خودکار سیستم در آن تعریف شده‌اند.
مرحله بعد، تعریف محتویات فایل ChangeProfile.xaml است. در این فایل اطلاعات انقیاد داده‌ها از ViewModel مرتبط که در ادامه توضیح داده خواهد شد دریافت می‌گردد. مثلا مقدار خاصیت ChangeProfileData.Password، از ViewModel به صورت خودکار تغذیه خواهد شد.
در این فایل یک سری DynamicResource را هم برای تعریف دکمه‌های سبک مترو ملاحظه می‌کنید. کلیدهای متناظر با آن در فایل Icons.xaml که در فایل App.xaml برای کل برنامه ثبت شده است، تامین می‌گردد.

2) تنظیم اعتبارسنجی صفحه اضافه شده

پس از اینکه صفحه جدید اضافه شد، نیاز است وضعیت دسترسی به آن مشخص شود:

```
/// <summary>
/// تغییر مشخصات کاربر جاری
/// </summary>
[PageAuthorization(AuthorizationType.FreeForAuthenticatedUsers)]
public partial class ChangeProfile
```

برای این منظور به فایل code behind این صفحه یعنی ChangeProfile.xaml.cs مراجعه و تنها، ویژگی فوق را به آن اضافه خواهیم کرد. ویژگی PageAuthorization به صورت خودکار توسط فریم ورک تهیه شده خوانده و اعمال خواهد شد. برای نمونه در اینجا کلیه کاربران اعتبارسنجی شده در سیستم می‌توانند مشخصات کاربری خود را تغییر دهند.
در مورد نحوه تعیین نقش‌های متفاوت در صورت نیاز، در قسمت قبل بحث گردید.

3) تغییر منوی برنامه جهت اشاره به صفحه جدید

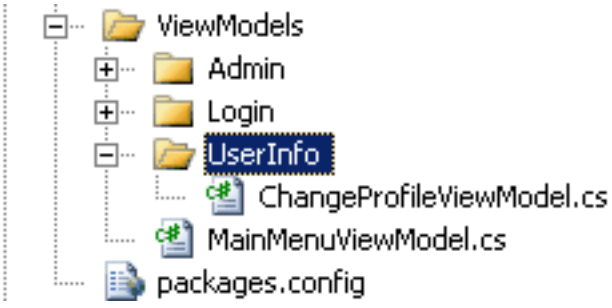
خوب، ما تا اینجا یک صفحه جدید را تهیه کرده‌ایم. در مرحله بعد باید مدخلی را در منوی برنامه جهت اشاره به آن تهیه کنیم.
منوی برنامه در فایل MainMenu.xaml قرار دارد. اطلاعات متناظر با دکمه ورود به صفحه تغییر مشخصات کاربری نیز به شکل ذیل تعریف شده است:

```
<Button Style="{DynamicResource MetroCircleButtonStyle}"
                        Height="55" Width="55"
                        Command="{Binding DoNavigate}"
                        CommandParameter="\Views\UserInfo\ChangeProfile.xaml"
                        Margin="2">
                    <Rectangle Width="28" Height="17.25">
                        <Rectangle.Fill>
                            <VisualBrush Stretch="Fill" Visual="{StaticResource appbar_user_tie}" />
                        </Rectangle.Fill>
                    </Rectangle>
                </Button>
```

به ازای هر صفحه جدیدی که تعریف می‌شود تنها کافی است CommandParameter ایی مساوی مسیر فایل XAML مورد نظر، در فایل منوی برنامه قید شود. منوی اصلی دارای ViewModel ایی است به نام MainMenuViewModel.cs که در پروژه Infrastructure پیشتر تهیه شده است.
در این ViewModel تعاریف DoNavigate و پردازش پارامتر دریافتی به صورت خودکار صورت خواهد گرفت و سورس کامل آن در اختیار شما است. بنابراین تنها کافی است CommandParameter را مشخص کنید، DoNavigate کار هدایت به آن‌را انجام خواهد داد.

4) ایجاد ViewModel متناظر با صفحه

مرحله آخر افزودن یک صفحه، تعیین ViewModel متناظر با آن است. عنوان شد که اطلاعات مورد نیاز جهت عملیات Binding در این فایل قرار می‌گیرند و اگر به فایل ChangeProfileViewModel.cs مراجعه کنید (نام آن مطابق قرارداد، یک کلمه ViewModel را نسبت به نام View متناظر بیشتر دارد)، چنین خاصیت عمومی را در آن خواهید یافت.



مطابق قراردادهای توکار قالب تهیه شده:

- نیاز است ViewModel تعریف شده از کلاس پایه BaseViewModel مشتق شود تا علاوه بر تامین یک سری کدهای تکراری مانند:

```
public abstract class BaseViewModel : DataErrorInfoBase, INotifyPropertyChanged, IViewModel
```

سبب شناسایی این کلاس به عنوان ViewModel و برقرار تزریق وابستگی‌های خودکار در سازنده آن نیز گردد.
- پس از اضافه شدن کلاس پایه BaseViewModel نیاز است تکلیف خاصیت public override bool ViewModelContextHasChanges را نیز مشخص کنید. در اینجا به سیستم راهبری اعلام می‌کنید که آیا در ViewModel جاری تغییرات ذخیره نشده‌ای وجود دارند؟ فقط باید true یا false را بازگردانید. برای مثال خاصیت uow.ContextHasChanges برای این منظور بسیار مناسب است و از طریق پیاده سازی الگوی واحد کار به صورت خودکار چنین اطلاعاتی را در اختیار برنامه قرار می‌دهد.

در ViewModelها هرجایی که نیاز به اطلاعات کاربر وارد شده به سیستم داشتید، از اینترفیس IAppContextService در سازنده کلاس ViewModel جاری استفاده کنید. اینترفیس IUnitOfWork امکانات ذخیره سازی اطلاعات و همچنین مشخص سازی وضعیت Context جاری را در اختیار شما قرار می‌دهد.
کلیه کدهای کار کردن با یک موجودیت باید در کلاس سرویس متناظر با آن قرار گیرند و این کلاس سرویس توسط اینترفیس آن مانند IUsersService در اینجا باید توسط سازنده کلاس در اختیار ViewModel قرار گیرد.
تزریق وابستگی‌ها در اینجا خودکار بوده و تنظیمات آن در فایل IocConfig.cs پروژه Infrastructure قرار دارد. این کلاس آنچنان نیازی به تغییر ندارد؛ اگر پیش فرض‌های نامگذاری آن‌را مانند کلاس‌های **Test** و اینترفیس‌های I **Test** ، در لایه سرویس برنامه رعایت شوند.

عنوان: **مدیریت تغییرات گریدی از اطلاعات به کمک استفاده از الگوی واحد کار مشترک بین ViewModel و لایه سرویس**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۶ ۲۳:۱۸
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

قالب پروژه WPF Framework به همراه چندین صفحه ابتدایی لازم، برای شروع هر برنامه‌ی تجاری دسکتاپی است؛ مثال مانند صفحه لاگین، صفحه تغییرات مشخصات کاربر وارد شده به سیستم و امثال آن. صفحه‌ای را که در این قسمت بررسی خواهیم کرد، صفحه تعریف کاربران جدید و ویرایش اطلاعات کاربران موجود است.

در این صفحه با کلیک بر روی دکمه به علاوه، یک ردیف به ردیف‌های موجود اضافه شده و در اینجا می‌توان اطلاعات کاربر جدیدی به همراه سطح دسترسی او را وارد و ذخیره کرد و یا حتی اطلاعات کاربران موجود را ویرایش نمود. اگر بخواهیم مانند مراحلی که در قسمت قبل در مورد تعریف یک صفحه جدید در برنامه توضیح داده شد، عمل کنیم، به صورت خلاصه به ترتیب ذیل عمل شده است:

1) ایجاد صفحه تغییر مشخصات کاربر

ابتدا صفحه Views\Admin\AddNewUser.xaml به پروژه ریشه که Viewهای برنامه در آن تعریف می‌شوند، اضافه شده است. به همراه دو دکمه و یک ListView که تطابق بهتری با قالب متروی مورد استفاده دارد.

2) تنظیم اعتبارسنجی صفحه اضافه شده

مرحله بعد تعریف هر صفحه‌ای در سیستم، مشخص سازی وضعیت دسترسی به آن است:

```
/// <summary>
/// افزودن و مدیریت کاربران سیستم
/// </summary>
[PageAuthorization(AuthorizationType.ApplyRequiredRoles, "IsAdmin, CanAddNewUser")]
```

ویژگی PageAuthorization به فایل Views\Admin\AddNewUser.xaml.cs اعمال شده است. در اینجا تنها کاربرانی که خاصیت‌های IsAdmin و CanAddNewUser آن‌ها true باشند، مجوز دسترسی به صفحه تعریف کاربران را خواهند یافت.

3) تغییر منوی برنامه جهت اشاره به صفحه جدید

در ادامه در فایل منوی برنامه Views\MainMenu.xaml تعریف دسترسی به صفحه Views\Admin\AddNewUser.xaml قید شده است:

```
<Button Style="{DynamicResource MetroCircleButtonStyle}"
                        Height="55" Width="55"
                        Command="{Binding DoNavigate}"
                        CommandParameter="\Views\Admin\AddNewUser.xaml"
                        Margin="2">
                    <Rectangle Width="28" Height="17.25">
                        <Rectangle.Fill>
                            <VisualBrush Stretch="Fill" Visual="{StaticResource appbar_user_add}" />
                        </Rectangle.Fill>
                    </Rectangle>
                </Button>
```

همانطور که در قسمت قبل نیز توضیح داده شده، تنها کافی است در اینجا CommandParameter را مساوی مسیر فایل AddNewUser.xaml قرار دهیم تا سیستم راهبری به صورت خودکار از آن استفاده کند.

4) ایجاد ViewModel متناظر با صفحه

مرحله نهایی تعریف صفحه AddNewUser، افزودن ViewModel متناظر با آن است که سورس کامل آن‌را در فایل ViewModels\Admin\AddNewUserViewModel.cs پروژه Infrastructure می‌توانید ملاحظه کنید.

نکته مهم این ViewModel، ارائه خاصیت لیست کاربران از نوع ObservableCollection به View و گرید برنامه است:

```
public ObservableCollection<User> UsersList { set; get; }
```

اطلاعات آن از IUsersService تزریق شده در سازنده کلاس ViewModel دریافت می‌شود:

```
/// <summary>
        /// جهت مقاصد انقیاد داده‌ها در دبلیو پی اف طراحی شده است
        /// لیستی از کاربران سیستم را باز می‌گرداند
        /// </summary>
        /// <param name="count">تعداد کاربر مد نظر</param>
        /// <returns>لیستی از کاربران</returns>
        public ObservableCollection<User> GetSyncedUsersList(int count = 1000)
        {
            _users.OrderBy(x => x.FriendlyName).Take(count)
                  .Load();

            // For Databinding with WPF.
            // Before calling this method you need to fill the context by using `Load()` method.
            return _users.Local;
        }
```

این کدها را در فایل UsersService.cs لایه سرویس برنامه می‌توانید مشاهده نمائید.

در اینجا از قابلیت خاصیتی به نام Local که یک ObservableCollection تحت نظر EF را بازگشت می‌دهد، استفاده شده است. برای استفاده از این خاصیت، ابتدا باید کوئری خود را تهیه و سپس متد Load را بر روی آن فراخوانی کرد. سپس خاصیت Local بر اساس اطلاعات کوئری قبلی پر و مقدار دهی خواهد شد.

علت انتخاب نام Synced برای این متد، تحت نظر بودن اطلاعات خاصیت Local است تا زمانیکه Context تعریف شده زنده نگه داشته شود. به همین جهت در برنامه جاری از روش زنده نگه داشتن Context به ازای یک ViewModel استفاده شده است.

به Context، توسط اینترفیس IUnitOfWork تزریق شده در سازنده کلاس ViewModel می‌توان دسترسی یافت. چون در اینجا از تزریق وابستگی‌ها استفاده شده است، وهله‌ای که IUnitOfWork کلاس AddNewUserViewModel را تشکیل می‌دهد، دقیقا همان وهله‌ای است که در کلاس UsersService لایه سرویس استفاده شده است. در نتیجه، در گرید برنامه هر تغییری اعمال شود، تحت نظر IUnitOfWork خواهد بود و صرفا با فراخوانی متد uow.ApplyAllChanges آن، کلیه تغییرات تمام ردیف‌های تحت نظر EF به صورت خودکار در بانک اطلاعاتی درج و یا به روز خواهند شد.

همچنین در مورد ViewModelContextHasChanges نیز در قسمت قبل بحث شد. در اینجا پیاده سازی کننده آن صرفا خاصیت uow.ContextHasChanges است. به این ترتیب اگر کاربر، تغییری را در صفحه داده باشد و بخواهد به صفحه دیگری رجوع کند، با

پیام زیر مواجه خواهد شد:

از همین خاصیت برای فعال و غیرفعال کردن دکمه ذخیره سازی اطلاعات نیز استفاده شده است:

```
/// <summary>
  /// فعال و غیرفعال سازی خودکار دکمه ثبت
  /// کنترل می‌شود این متد به صورت خودکار توسط RelayCommand
  /// </summary>
  private bool canDoSave()
  {
     // آیا در حین نمایش صفحه‌ای دیگر باید به کاربر پیغام داد که اطلاعات ذخیره نشده‌ای وجود دارد؟
     return ViewModelContextHasChanges;
  }
```

این متد توسط RelayCommand ایی به نام DoSave

```
/// <summary>
  /// رخداد ذخیره سازی اطلاعات را دریافت می‌کند
  /// </summary>
  public RelayCommand DoSave { set; get; }
```

که به نحو زیر مقدار دهی شده است، مورد استفاده قرار می‌گیرد:

```
DoSave = new RelayCommand(doSave, canDoSave);
```

به ازای هر تغییری در UI، این RelayCommand به نتیجه canDoSave مراجعه کرده و اگر خروجی آن true باشد، دکمه متناظر را به صورت خودکار فعال می‌کند و یا برعکس.
این بررسی نیز بسیار سبک و سریع است. از این جهت که تغییرات Context در حافظه نگهداری می‌شوند و مراجعه به آن مساوی مراجعه به بانک اطلاعاتی نیست.

نظرات خوانندگان

نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۳/۲۸ ۱۰:۵۵

سلام.وقتتون بخیر و ممنون از مطالب خوبتون.
من از ContextHasChanges شما استفاده میکنم برای ردگیری تغییرات ولی یه مشکل دارم اونم اینه که وقتی Navigation Propertyها تغییر میکنند،تغییر رو شناسایی نمیکنه؟!مثلا من خصوصیت زیر رو دارم توی یکی از کلاسهام:

```
public class Check : DomainEntityBase
  {
      ...
      public virtual Bank Bank {  get; set; }
  }
```

```
public bool HasChanges
        {
            get
            {
                return this.ChangeTracker
                           .Entries()
                           .Any(x => x.State == EntityState.Added ||
                                     x.State == EntityState.Deleted ||
                                     x.State == EntityState.Modified);
            }
        }
```

ولی وقتی مقدار بانک رو تغییر میدهم ، HasChanges همچنان False بر میگردونه!دلیلش چیه؟

باز هم ممنونم.موفق باشید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۲۸ ۱۱:۱۷

این مورد بحث پیشرفته change tracking در EF است. DbContext رابطه‌های مستقل رو Track نمی‌کنه . در این حالت باید به لایه زیرین آن یعنی Object Context سوئیچ کرد:

```
ObjectContext objectContext = ((IObjectContextAdapter)dbContext).ObjectContext;
foreach (ObjectStateEntry entry = objectContext.ObjectStateManager
                                               .GetObjectStateEntries(~EntityState.Detached)
                                               .Where(e => e.IsRelationship))
{
    // Track changes here
}
```

همچنین پیش از اینکار باید متد DetectChanges نیز فراخوانی شود.

نویسنده: م ش
تاریخ: ۱۳۹۳/۰۱/۲۸ ۱۸:۵۷

لطفاً نحوه سوال از کاربر جهت تایید حذف یک ردیف از جدول را هم توضیح دهید. با تشکر

نویسنده: م ش
تاریخ: ۱۳۹۳/۰۱/۲۹ ۷:۳۹

در این فریم‌ورک جهت نمایش پیغام به کاربر کلاس SendMsg تدارک دیده شده است. نحوه استفاده از آن به شکل زیر است:
ابتدا در کلاس AddNewUserViewModel یک فیلد خصوصی از نوع کلاس SendMsg ایجاد کنید

```
private SendMsg _sendMsg = new SendMsg();
```

سپس در متد حذف، تابع ShowMsg آن را فراخوانی کنید

```
private void doDelete()
{
    _sendMsg.ShowMsg(new AlertConfirmBoxModel
    {
        Errors = new List<string> { "آیا کاربر انتخاب شده حذف شود؟" },
        ShowConfirm = Visibility.Visible,
        ShowCancel = Visibility.Visible
    },
    confirmed: input => delete(input));
}

private void delete(AlertConfirmBoxModel input)
{
    UsersList.Remove(SelectedItem);
}
```

عنوان: **بررسی جزئیات تزریق وابستگی‌ها در قالب پروژه WPF Framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۷ ۱۰:۴۱
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

در قالب طراحی شده، نه در کدهای Viewهای اضافه شده و نه در ViewModelها، اثری از کدهای مرتبط با تزریق وابستگی‌ها و یا حتی وهله سازی ViewModel مرتبط با یک View مشاهده نمی‌شود. در ادامه قصد داریم جزئیات پیاده سازی آن‌را مرور کنیم.

مدیریت خودکار وهله سازی ViewModelها

اگر به فایل MVVM\ViewModelFactory.cs قرار گرفته در پروژه Common مراجعه کنید، کدهای کلاسی که کار وهله سازی ViewModelها را انجام می‌دهد، مشاهده خواهید کرد:

```
using System.Windows;
using StructureMap;

namespace WpfFramework1999.Common.MVVM
{
    /// <summary>
    /// Stitches together a view and its view-model
    /// </summary>
    public class ViewModelFactory
    {
        private readonly FrameworkElement _control;

        /// <summary>
        /// سازنده کلاس تزریق وابستگی‌ها به ویوو مدل و وهله سازی آن
        /// </summary>
        /// <param name="control">وهله‌ای از شیءایی که باید کار تزریق وابستگی‌ها در آن انجام شود</param>
        public ViewModelFactory(FrameworkElement control)
        {
            _control = control;
        }

        /// <summary>
        /// وهله متناظر با ویوو مدل
        /// </summary>
        public IViewModel ViewModelInstance { get; private set; }

        /// <summary>
        /// کار تزریق خودکار وابستگی‌ها و وهله سازی ویوو مدل مرتبط انجام خواهد شد
        /// </summary>
        public void WireUp()
        {
            var viewName = _control.GetType().Name;
            var viewModelName = string.Concat(viewName, "ViewModel"); //قرار داد نامگذاری ما است

            if (!_control.IsLoaded)
            {
                _control.Loaded += (s, e) =>
                {
                    setDataContext(viewModelName);
                };
            }
            else
            {
                setDataContext(viewModelName);
            }
        }

        private void setDataContext(string viewModelName)
        {
            //کار تزریق خودکار وابستگی‌ها و وهله سازی ویوو مدل مرتبط انجام خواهد شد
            ViewModelInstance = ObjectFactory.TryGetInstance<IViewModel>(viewModelName);
            if (ViewModelInstance == null) // این صفحه ویوو مدل ندارد
                return;

            _control.DataContext = ViewModelInstance;
        }
    }
}
```

در این کلاس، یک وهله از صفحه‌ای که توسط کاربر درخواست شده‌است، در سازنده کلاس دریافت گردیده و سپس در متد WireUp، بر اساس قرارداد نامگذاری که پیشتر نیز عنوان شد، ViewModel متناظر با نام View از IoC Container استخراج و وهله سازی می‌گردد. سپس این وهله به DataContext صفحه انتساب داده می‌شود.
چند سؤال مهم:
- IoC Container از کجا می‌داند که ViewModelها در کجا قرار دارند؟
- این کلاس ViewModelFactory چگونه به وهله‌ای از یک صفحه درخواستی توسط کاربر دسترسی پیدا می‌کند و در کجا؟

IoC Container از کجا می‌داند که ViewModelها در کجا قرار دارند؟

اگر بحث سری جاری را از ابتدا دنبال کرده باشید، عنوان شد که ViewModelها را در این قالب، باید مشتق شده از کلاس پایه‌ای به نام BaseViewModel تهیه کنیم. برای مثال:

```
/// <summary>
/// ویوو مدل افزودن و مدیریت کاربران
/// </summary>
public class AddNewUserViewModel : BaseViewModel
```

این کلاس پایه که در فایل MVVM\BaseViewModel.cs پروژه Common قرار دارد، به نحو زیر آغاز شده است:

```
/// <summary>
/// کلاس پایه ویوو مدل‌های برنامه که جهت علامتگذاری آن‌ها برای سیم کشی‌های تزریق وابستگی‌های برنامه نیز استفاده می‌شود
/// </summary>
public abstract class BaseViewModel : DataErrorInfoBase, INotifyPropertyChanged, IViewModel
```

اگر دقت کنید در اینجا اینترفیس IViewModel نیز ذکر شده است. این اینترفیس برای علامتگذاری ViewModelها و یافتن خودکار آن‌ها توسط IoC Container مورد استفاده درنظر گرفته شده است. اگر به فایل Core\IocConfig.cs پروژه Infrastructure مراجعه کنید، چنین تنظیمی را در آن مشاهده خواهید نمود:

```
// Add all types that implement IView into the container,
// and name each specific type by the short type name.
scan.AddAllTypesOf<IViewModel>().NameBy(type => type.Name);
```

به این ترتیب StructureMap با اسکن اسمبلی Infrastructure کلیه کلاس‌های پیاده سازی کننده IViewModel را یافته و سپس آن‌ها را بر اساس نام متناظری که دارند، ذخیره می‌کند. با این تنظیم، اکنون در کلاس ViewModelFactory یک چنین کدی کار خواهد کرد:

```
//کار تزریق خودکار وابستگی‌ها و وهله سازی ویوو مدل مرتبط انجام خواهد شد
ViewModelInstance = ObjectFactory.TryGetInstance<IViewModel>(viewModelName);
```

کلاس ViewModelFactory چگونه به وهله‌ای از یک صفحه درخواستی توسط کاربر دسترسی پیدا می‌کند و در کجا؟

در اینجا قسمتی از کدهای فایل Core\FrameFactory.cs قرار گرفته در پروژه Infrastructure را ملاحظه می‌کنید:

```
namespace WpfFramework.Infrastructure.Core
{
    /// <summary>
    /// ایجاد یک کنترل فریم سفارشی که قابلیت تزریق وابستگی‌ها را به صورت خودکار دارد
    /// به همراه اعمال مسایل راهبری برنامه که از منوی اصلی دریافت می‌شوند
    /// </summary>
    public class FrameFactory : Frame
```

```
    {
        /// <summary>
        /// در اینجا می‌شود به وهله‌ای از صفحه‌ای که قرار است اضافه گردد دسترسی یافت
        /// </summary>
        protected override void OnContentChanged(object oldContent, object newContent)
        {
            base.OnContentChanged(oldContent, newContent);

            var newPage = newContent as FrameworkElement;
            if (newPage == null)
                return;

            _currentViewModelFactory = new ViewModelFactory(newPage);
            _currentViewModelFactory.WireUp(); //کار تزریق وابستگی‌ها و وهله سازی ویوو مدل مرتبط انجام
خواهد شد
        }
    }
}
```

در این کلاس، یک Frame سفارشی را طراحی کرده‌ایم؛ از این جهت که بتوان متد OnContentChanged آن‌را تحریف کرد. در این متد، newContent دقیقا وهله‌ای از صفحه جدیدی است که توسط کاربر درخواست شده‌است. خوب ... این وهله را داریم، بنابراین تنها کافی است آن‌را به کلاس ViewModelFactory ارسال کنیم و متد WireUp آن‌را بر روی وهله کلاس صفحه درخواستی فراخوانی نمائیم. به این ترتیب، صفحه‌ای نمایش داده خواهد شد که DataContext آن با وهله‌ای از ViewModel متناظر مقدار دهی شده‌است. از این جهت که این وهله سازی توسط IoC Container صورت می‌گیرد، کلیه وابستگی‌های تعریف شده در سازنده کلاس ViewModel نیز به صورت خودکار وهله سازی و مقدار دهی خواهند شد.

نهایتا فراخوانی متد IocConfig.Init، در فایل App.xaml.cs پروژه ریشه، در آغاز برنامه قرار گرفته است.

نظرات خوانندگان

نویسنده: ایمان محمدی
تاریخ: ۱۳۹۲/۰۳/۱۱ ۱:۳۴

وقتی پیج تغییر می‌کنه ، برای پیج قبلی چه اتفاقی میفته؟
چطور می‌تونیم یک page manager بنویسیم که بین صفحات باز سوییچ کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۱ ۸:۵۰

- از بین میره. تمام منابع مرتبط با اون هم مانند DbContext رها خواهند شد.
- به پروژه Infrastructure مراجعه کنید. یک کلاس Redirect برای هدایت به صفحات مختلف با برنامه نویسی طراحی شده. نمونه‌ای از استفاده از این کلاس رو در ViewModel مرتبط با لاگین به سیستم می‌تونید مشاهده کنید.

نویسنده: علیرضا پایدار
تاریخ: ۱۳۹۲/۰۸/۰۳ ۱۷:۳۱

من توی سازنده کلاس Locator کد زیر را نوشتم

```
SimpleIoc.Default.Register<IUnitOfWork, SampleContext>();
```

و بعد هر کجا نیاز دارم از این کد استفاده می‌کنم:

```
_uow = ServiceLocator.Current.GetInstance<IUnitOfWork>();
```

مشکلی که دارم وقتی SaveChanges را فراخوانی می‌کنم داخلش this.GetValidationErrors را فراخوانی کردم چون تنها یک instance از SampleContext ساخته میشه خطاهای قبلی دوباره وجود داره.
راهکاری هست خطاها را پاک کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۳ ۱۸:۴۷

- بله. باید Context تخریب و بازسازی مجدد شود؛ یا باید از سیستم Tracking آن کوئری بگیرید و سپس مواردی را که تغییر کرده‌اند به حالت اول برگردانید و یا روش سوم استفاده از متد Reload بر روی یک dbContext.Entry است.
+ طول عمر یک Context باید کوتاه باشد یا حداکثر در حد طول عمر یک فرم و نه طول عمر یک برنامه.
+ دو مطلب مرتبط
بایدها و نبایدهای استفاده از IoC Containers
نکته‌ای در مورد مدیریت طول عمر اشیاء در حالت HybridHttpOrThreadLocalScoped در برنامه‌های دسکتاپ

برای اینکه با SimpleIoc هم بتوانید وهله‌های متفاوتی را دریافت کنید و نه الزاما استفاده به صورت سینگلتون، باید به متد GetInstance آن یک کلید مشخص را ارسال کنید (مثلا نام فرم جاری یا آدرس صفحه جاری)؛ اگر کلیدی ارسال نشود، سینگلتون عمل می‌کند. در کل بهتر است از یک IoC Container بهتر استفاده کنید که در مورد طول عمر اشیاء راه‌حل‌های متفاوتی را به همراه دارد؛ مانند StrcutureMap. همچنین استفاده از آن به الگوی Service locator محدود نباشد.

نویسنده: علیرضا پایدار
تاریخ: ۱۳۹۲/۰۸/۰۳ ۱۹:۴۳

با کد زیر مشکلم حل میشه؟

```
ObjectFactory.Configure(cfg =>
            {
 cfg.For<IUnitOfWork>().Use(() => new SampleContext());
});
```

البته با استفاده از StrcutureMap

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۳ ۱۹:۴۸

در مورد سینگلتون نبودن، بله. نیازی به new هم نداره (حالت معمولی ذکر آن کافی است). در حالت پیش فرض، به ازای هربار فراخوانی، یک وهله جدید را ایجاد می‌کند. مگر اینکه در همینجا صریحا ذکر کنید که سینگلتون نیز مدنظر شما است یا مثلا حالت زنده نگه داشتن شیء در طول عمر یک درخواست وب، مهم است.

عنوان: **یکپارچه سازی اعتبارسنجی EF Code first با امکانات WPF و حذف کدهای تکرای INotifyPropertyChanged**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۸ ۲۲:۴۷
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

در لابلای توضیحات قسمت‌های قبل، به نحوه استفاده از کلاس‌های پایه‌ای که اعتبارسنجی یکپارچه‌ای را با WPF و EF Code first در قالب پروژه WPF Framework ارائه می‌دهند، اشاره شد. در این قسمت قصد داریم جزئیات بیشتری از پیاده سازی آن‌ها را بررسی کنیم.

بررسی سطح بالای مکانیزم‌های اعتبارسنجی و AOP بکارگرفته شده

در حین کار با قالب پروژه WPF Framework، هنگام طراحی Modelهای خود (تفاوتی نمی‌کند که Domain model باشند یا صرفا Model متناظر با یک View)، نیاز است دو مورد را رعایت کنید:

```
[ImplementPropertyChanged] // AOP
public class LoginPageModel : DataErrorInfoBase
```

الف) کلاس مدل شما باید مزین به ویژگی ImplementPropertyChanged شود.
ب) از کلاس پایه DataErrorInfoBase مشتق گردد

البته اگر به کلاس‌های Domain model برنامه مراجعه کنید، صرفا مشتق شدن از BaseEntity را ملاحظه می‌کنید:

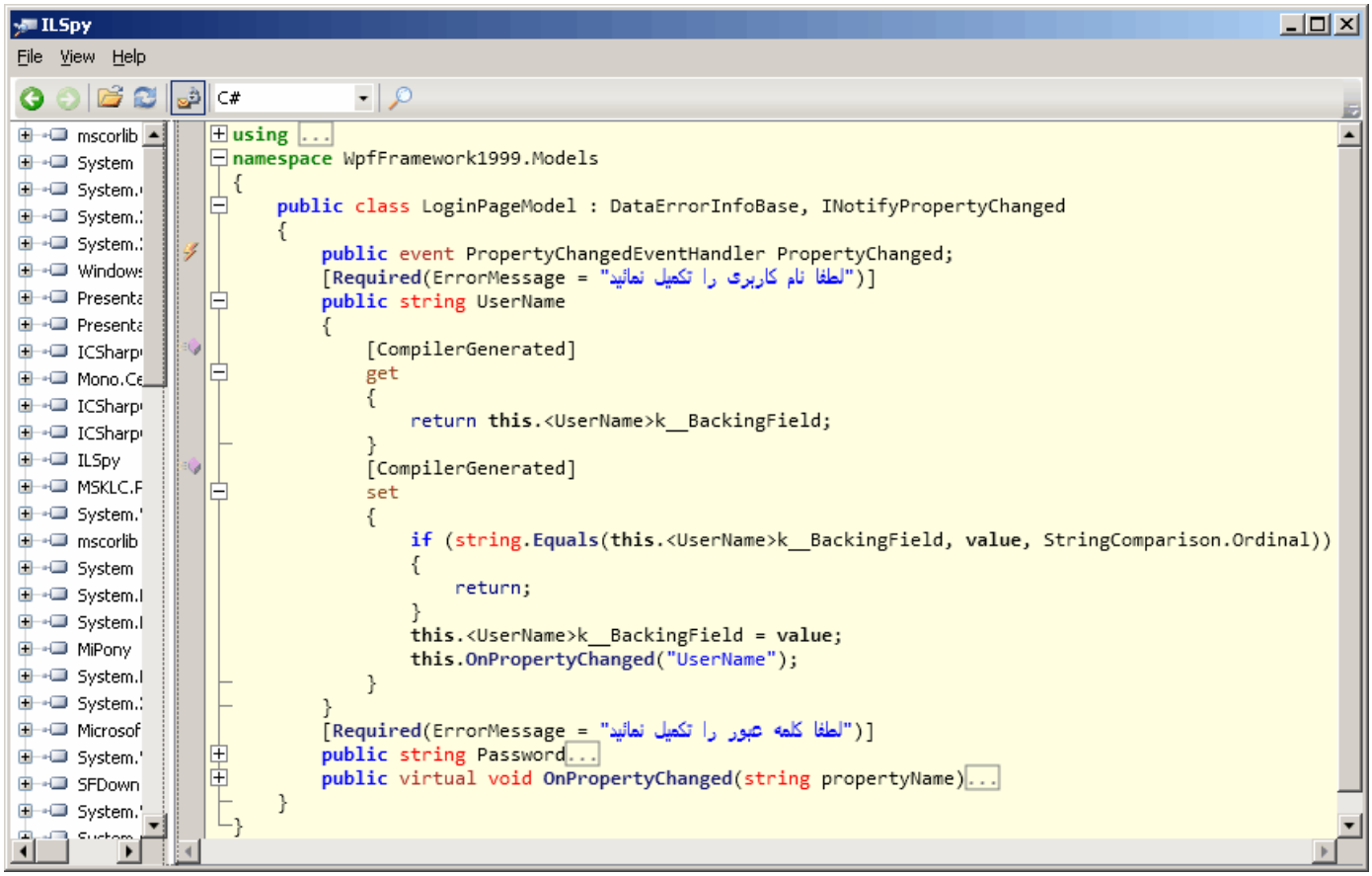
```
public class User : BaseEntity
```

علت این است که دو نکته یاد شده در کلاس پایه BaseEntity پیشتر پیاده سازی شده‌اند:

```
[ImplementPropertyChanged] // AOP
public abstract class BaseEntity : DataErrorInfoBase //پیاده سازی خودکار سیستم اعتبارسنجی یکپارچه
```

بررسی جزئیات مکانیزم AOP بکارگرفته شده

بسیار خوب؛ این‌ها چطور کار می‌کنند؟!
ابتدا نیاز است مطلب « معرفی پروژه NotifyPropertyWeaver » را یکبار مطالعه نمائید. خلاصه‌ای جهت تکرار نکات مهم آن:
ویژگی ImplementPropertyChanged به ابزار Fody اعلام می‌کند که لطفا کدهای تکراری INotifyPropertyChanged را پس از کامپایل اسمبلی جاری، بر اساس تزریق کدهای IL متناظر، به اسمبلی اضافه کن. این روش از لحاظ کارآیی و همچنین تمیز نگه داشتن کدهای نهایی برنامه، فوق العاده است.
برای بررسی کارکرد آن نیاز است اسمبلی مثلا Models را دی‌کامپایل کرد:



همانطور که ملاحظه می‌کنید، کدهای تکراری INotifyPropertyChanged به صورت خودکار به اسمبلی نهایی اضافه شده‌اند.
البته بدیهی است که استفاده از Fody الزامی نیست. اگر علاقمند هستید که این اطلاعات را دستی اضافه کنید، بهتر است از کلاس پایه BaseViewModel قرار گرفته در مسیر MVVM\BaseViewModel.cs پروژه Common استفاده نمائید.
در این کلاس، پیاده سازی‌های NotifyPropertyChanged را بر اساس متدهایی که یک رشته را به عنوان نام خاصیت دریافت می‌کنند و یا متدی که امکان دسترسی strongly typed به نام رشته را میسر ساخته است، ملاحظه می‌کنید.

```
/// <summary>
  /// تغییر مقدار یک خاصیت را اطلاع رسانی خواهد کرد
  /// </summary>
  /// <param name="propertyName">نام خاصیت</param>
  public void NotifyPropertyChanged(string propertyName)

  /// <summary>
  /// تغییر مقدار یک خاصیت را اطلاع رسانی خواهد کرد
  /// </summary>
  /// <param name="expression">نام خاصیت مورد نظر</param>
  public void NotifyPropertyChanged(Expression<Func<object>> expression)
```

برای مثال در اینجا خواهیم داشت:

```
public class AlertConfirmBoxViewModel : BaseViewModel
    {
        AlertConfirmBoxModel _alertConfirmBoxModel;
        public AlertConfirmBoxModel AlertConfirmBoxModel
        {
            set
            {
                _alertConfirmBoxModel = value;
                NotifyPropertyChanged("AlertConfirmBoxModel");
                // ویا ....
                NotifyPropertyChanged(()=>AlertConfirmBoxModel);
            }
            get { return _alertConfirmBoxModel; }
```

```
        }
```

هر دو حالت استفاده از متدهای NotifyPropertyChanged به همراه کلاس پایه BaseViewModel در اینجا ذکر شده‌اند. حالت استفاده از Expression به علت اینکه تحت نظر کامپایلر است، در دراز مدت نگه‌داری برنامه را ساده‌تر خواهد کرد.

بررسی جزئیات اعتبارسنجی‌های تعریف شده

EF دارای یک سری ویژگی مانند Required و امثال آن است. WPF دارای اینترفیسی است به نام IDataErrorInfo. این دو را باید به نحوی به هم مرتبط ساخت که پیاده سازی‌های مرتبط با آن‌ها را در مسیرهای WpfValidation\DataErrorInfoBase.cs و WpfValidation\ValidationHelper.cs پروژه Common می‌توانید ملاحظه نمائید.

```
<TextBox Text="{Binding Path=ChangeProfileData.UserName,
Mode=TwoWay,UpdateSourceTrigger=PropertyChanged,
 NotifyOnValidationError=true, ValidatesOnExceptions=true, ValidatesOnDataErrors=True,
TargetNullValue=''}"  />
```

برای نمونه در اینجا خاصیت Text یک TextBox به خاصیت UserName شیء ChangeProfileData تعریف شده در ViewModel تغییر اطلاعات کاربری برنامه مقید شده است.
همچنین حالت‌های بررسی اعتبارسنجی آن نیز به PropertyChanged تنظیم گردیده است. در این حالت WPF به تعاریف شیء ChangeProfileData مراجعه کرده و برای نمونه اگر این شیء اینترفیس IDataErrorInfo را پیاده سازی کرده بود، نام خاصیت جاری را به آن ارسال و از آن خطاهای اعتبارسنجی متناظر را درخواست می‌کند. در اینجا وقت خواهیم داشت تا بر اساس ویژگی‌ها و Data annotaions اعمالی، کار اعتبارسنجی را انجام داده و نتیجه را بازگشت دهیم.
خلاصه‌ی تمام این اعمال و کلاس‌ها، در کلاس پایه DataErrorInfoBase این قالب پروژه قرار گرفته‌اند. بنابراین تنها کاری که باید صورت گیرد، مشتق کردن کلاس مدل مورد نظر از آن می‌باشد.
همچنین باید دقت داشت که نمایش اطلاعات خطاهای حاصل از اعتبارسنجی در این قالب پروژه بر اساس امکانات قالب متروی MahApps.Metro انجام می‌گیرد (این مورد از Silverlight toolkit به ارث رسیده است) و در حالت کلی خودکار نیست؛ اما در اینجا نیازی به کدنویسی اضافه‌تری ندارد.

به علاوه باید دقت داشت که این مورد ویژه را باید بر اساس آخرین Build کتابخانه MahApps.Metro که به‌روزتر است دریافت و استفاده کرد. در اینجا با پارامتر Pre ذکر شده است.

```
PM> Install-Package MahApps.Metro -Pre
```

عنوان: **نکته‌ای در مورد مدیریت طول عمر اشیاء در حالت HybridHttpOrThreadLocalScoped در برنامه‌های دسکتاپ**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۲۳ ۱۰:۱۲
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

اگر به فایل IocConfig.cs پروژه دقت کرده باشید، مدیریت طول عمر واحد کار به صورت معمولی و متداولی تعریف شده است:

```
cfg.For<IUnitOfWork>().Use(() => new MyWpfFrameworkContext());
```

و در اینجا از حالت HybridHttpOrThreadLocalScoped که در برنامه‌های وب نیز مورد استفاده قرار می‌گیرد، استفاده نشده است. چرا؟

```
ThreadLocal - A single instance will be created for each requesting thread. Caches the instances with ThreadLocalStorage.
Hybrid - Uses HttpContext storage if it exists, otherwise uses ThreadLocal storage
```

تعاریف رسمی مرتبط با Thread local و حالت Hypbrid را در اینجا ملاحظه می‌کنید. در حالت Thread local از یک وهله در طی طول عمر یک ترد استفاده می‌شود. حالت Hybrid مشخص می‌کند که اگر برنامه وب بود، از HttpContext برای مدیریت این طول عمر استفاده کن و اگر برنامه دسکتاپ بود از ThreadLocal storage.
خوب، مشکل کجا است؟ در یک برنامه دسکتاپ اگر ترد جدیدی را ایجاد نکنیم، کل برنامه در طول مدتی که در حال اجرا است در یک ترد اصلی اجرا می‌شود. در نتیجه استفاده از حالت HybridHttpOrThreadLocalScoped در برنامه‌های دسکتاپ، بسیار شبیه به حالت Singleton است. به این ترتیب با بسته شدن یک پنجره یا هدایت به صفحه‌ای دیگر، Context برنامه و EF رها نخواهند شد (چون تضمین شده است که یک وهله از این شیء در طول عمر ترد جاری وجود داشته باشد). نتیجه آن روبرو شدن با خطاهایی است که ردیابی و دیباگ بسیار مشکلی دارند. برای مثال چندی قبل در سایت مطرح شده بود که چرا در یک برنامه WinForms خطای زیر را دریافت می‌کنم:

```
An object with the same key already exists in the ObjectStateManager
```

علت پس از چند سؤال جواب، به مدیریت طول عمر UOW مرتبط شد. چون از حالت HybridHttpOrThreadLocalScoped استفاده شده بوده، در صفحه‌ای، شیءایی به Context اضافه شده. پس از مراجعه به صفحه‌ای دیگر، مجددا سعی در اضافه کردن همین شیء گردیده است (با این تصور که با بسته شدن صفحه قبلی، Context هم از بین رفته است). بلافاصله خطای ذکر شده، توسط EF گزارش گردیده است؛ چون هنوز Context باز است و از بین نرفته است.

در برنامه‌های وب چطور؟

در برنامه‌های وب، به ازای هر درخواست رسیده، یک ترد جدید ایجاد می‌شود. به همین جهت استفاده از حالت HybridHttpOrThreadLocalScoped برای داشتن تنها یک Context در طول یک درخواست ضروری است.

نتیجه گیری

در برنامه‌های دسکتاپ خود اگر از ترد استفاده نمی‌کنید، از حالت HybridHttpOrThreadLocalScoped برای مدیریت طول عمر UOW استفاده نکنید.