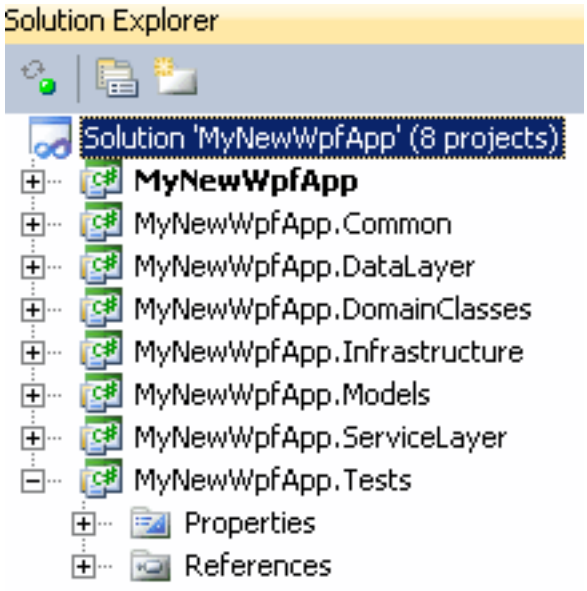
عنوان: **بررسی قسمت‌های مختلف قالب پروژه WPF Framework تهیه شده**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۵ ۲۲:۳۴
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

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.