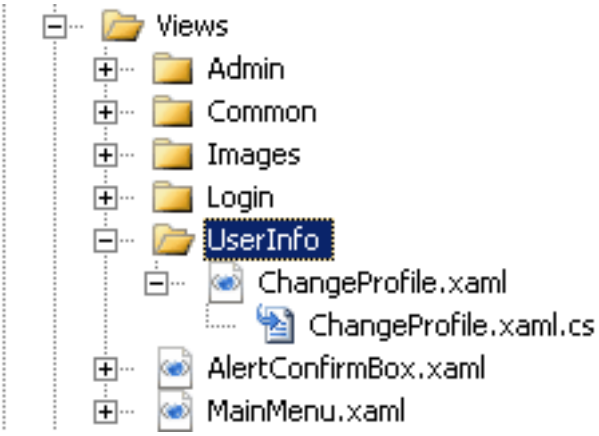
عنوان: **نحوه برقراری ارتباطات بین صفحات، سیستم راهبری و ViewModelها در قالب پروژه WPF Framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۰۶ ۱۰:۷
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, AOP, Dependency Injection, Entity framework, MVVM, WPF

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

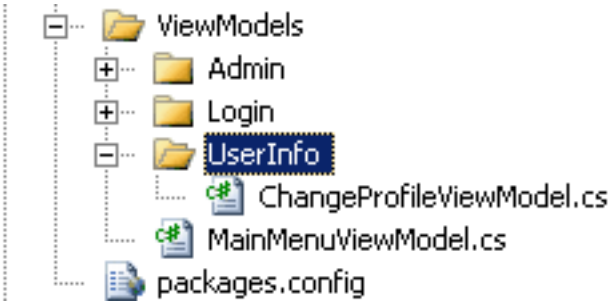
خوب، ما تا اینجا یک صفحه جدید را تهیه کرده‌ایم. در مرحله بعد باید مدخلی را در منوی برنامه جهت اشاره به آن تهیه کنیم.
منوی برنامه در فایل MainMenu.xaml قرار دارد. اطلاعات متناظر با دکمه ورود به صفحه تغییر مشخصات کاربری نیز به شکل ذیل تعریف شده است:

```
<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.