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);
}
```