MVVM

By: Mosh Hamedani

Exercise

Implement MVVM architectural style in the **ContactBook** app from the last section.

You can download the before and after source code from this lecture.

Tips

Persistence

In order to make a class testable, it should be decoupled from external libraries and frameworks. We use MVVM to decouple our code from a presentation framework like Xamarin (or WPF).

In our ContactBook app, we have a dependency to SQLite, which is a persistence framework. Again, we need to decouple our code from SQLite. For this, you can use the same technique I showed you in the lecture **"Dealing with Dependencies"**. There, I declared an interface (**IPageService**) and passed it as a parameter to the constructor of our view model. With this, our view model doesn't know what is behind this interface. Later, when unit testing this view model, we can pass a fake implementation of **IPageService.**

Similarly, to decouple your view model from SQLite, create an interface called **IContactStore** in **ViewModels** folder. This interface should have operations for loading and saving contacts (eg add, update, delete). Pass **IContactStore** in the constructor of your view model. The view model doesn't know what is behind this implementation. Today we provide an implementation that uses SQLite, tomorrow, we may replace this implementation with a different one that uses Azure cloud storage. The rest of the application will be unaffected as long as that implementation works.

Implement this interface in a class called **SQLiteContactStore** in **Persistence** folder.

Delete MenuItem

You may encounter a problem when using a command with delete MenuItem. The reason for this is that the binding context for each menu item will be a **ContactViewModel** object. Remember, the source of our list view is an **ObservableCollection** of **ContactViewModel**. So, each item in the list is a **ContactViewModel**.

It's likely that you've defined **DeleteContactCommand** in **ContactsListViewModel**, not in **ContactViewModel**. So, the following binding does not work:

```
<MenuItem Command="{Binding DeleteContactCommand}" />
```

The workaround this is to get a reference to the page first:

```
<Page x:Name="page" … />
```

And then use **page** as the source of data binding:

```
<MenuItem Command="{Binding
               Source={x:Reference page},
               Path=ViewModel.DeleteContactCommand}" />
```

Note that here are I'm referencing **ViewModel.DeleteContactCommand** property. This means I've defined a property in the code-behind called **ViewModel**, which is of type **ContactsViewModel**.