Architektura

provider to już jest konkretna implementacja: z konkretnego źródła. Abstrakcją jest interfejs. Dlatego klient (przez managera) wiąże się z abstrakcją - nie wie, skąd pochodzą dane. Czyli tak np. StarDataProvider to tak naprawdę StarDBDataProvider.

DB

plik z bazą danych trzymam w DAL, za EF, ale też dla łatwości testów. Do platform add -> existing -> add as link

UNITY

Chciałem użyć takiego jakiegoś ContainerBootstrapper jak polecają na MSDN, ale prism ma już swoje register types, to wystarczyło tam dodać.

PRISM

set

{

SetProperty(ref \_selectedStar, value, onChanged: () =>

// This is a more standard approach. Another is to use the ObservesCanExecute extension

// in the command init, but unfortunately it requires a property like IsSelected to observe,

// (ObservesCanExecute(() => IsSelected)),

// not a Func (ObservesCanExecute(() => SelectedStar != null))

ShowStarDetailsCommand.RaiseCanExecuteChanged());

}

RaiseCanExecuteChanged może być wywołany gdziekolwiek, nie tylko w setterze.

Zamiast tego RaiseCanExecuteChanged() w setterze, można użyć prism extension ObservesProperty:

ShowStarDetailsCommand = new DelegateCommand(ShowStarDetails, () => SelectedStar != null)

.ObservesProperty(() => SelectedStar);

.ObservesCanExecute(() => SelectedStar!= null); rzuca ArgumentException:

{System.ArgumentException: The expression is not a member access expression.

Parameter name: expression

at Prism.Mvvm.PropertySupport.ExtractPropertyNameFromLambda (System.Linq.Expressions.LambdaExpression expression) [0x00021] in <c7f0f832ef3946009ad46bccbd859f64>:0

at Prism.Mvvm.PropertySupport.ExtractPropertyName[T] (System.Linq.Expressions.Expression`1[TDelegate] propertyExpression) [0x0000e] in <c7f0f832ef3946009ad46bccbd859f64>:0

at Prism.Commands.DelegateCommandBase.ObservesPropertyInternal[T] (System.Linq.Expressions.Expression`1[TDelegate] propertyExpression) [0x00000] in <c7f0f832ef3946009ad46bccbd859f64>:0

at Prism.Commands.DelegateCommand.ObservesCanExecute (System.Linq.Expressions.Expression`1[TDelegate] canExecuteExpression) [0x0000c] in <c7f0f832ef3946009ad46bccbd859f64>:0

Do dialogu: IConfirmNavigation

Myślałem, że to źle, że prism zawsze tworzy nowy VM i muszę w onNavigated wznawiać propertisy, oni mówią, że tak jest OK:

<https://github.com/PrismLibrary/Prism/issues/968>

Thoughts on PubSubEvents:

/\*

\* OK, so the options are two:

\* 1. Don't worry about unsubscribing, just use the filter upon subscribing or do a check in the handler (remember to lock!)

\* - Subscription occurs in the constructor, so no additional check is required

\* - new instance is created upon direct prism navigation, so not when hardware back button is used,

\* navigation page back button (probably, not checked), or when being on the main page and tapping MainPage in the MasterView.

\* - References to the disposed subscribers remain until the publisher gets disposed (MasterDetailVM in this case, which is long)

\* - if keepSubscriberReferenceAlive is true, referenes stay even after that.

\* 2. Subscribing upon OnNavigatEDTo (not INGTo, because hardwareback button doesn't trigger it), unscubscribing with OnNavigatingFrom

\* - keeps the subscribers in check, but I think it's the opposite to what prism's EventAggregator was made for in such scenario.

\*

\* I'm going with option 1 for now. But I think since the difference in publiser's and subscriber's lifespan, option 2 is better in this case.

\* Because now the references are kept untill MasterDetailVM gets disposed, which is almost NEVER.

\*

\* OPTION 2:

protected override async Task Restore()

{

// Need to check if the event is already subscribed, otherwise I'm resubscribing

await Call(() =>

{

// If there are many events, it would be nice to have a generic collection (dictionary extension),

// that would allow to store <TEventType, Action<TPayload>> where TEventType : PubSubEvent<TPayload>

// and just iterate through that. Since it's only one, ore few more, there's no need. It's rather not event ment that way,

// since to be perfectly thorough I would need two such collections, for events with and without the payload.

var conEvent = \_eventAggregator.GetEvent<ConstellationSelectedEvent>();

// When the VM gets newed up by the Prism navigation, the action that this Contains checks differs from the previous one,

// so that the result is false. That's why in this scenario I unsubscribe manually.

//

// It's the desired behavior. If not specified otherwise (keepSubscriberReferenceAlive),

// references to the subscriber (this) from the event are weak, which means, they get GC'd, along with this instance (eventually).

if (!conEvent.Contains(HandleConstellationRequest))

conEvent.Subscribe(HandleConstellationRequest);

});

}

protected override async Task CleanUp()

{

await Call(() =>

{

\_eventAggregator.GetEvent<ConstellationSelectedEvent>().Unsubscribe(HandleConstellationRequest);

});

}

\*

\*/

ASYNC/AWAIT

msdn.microsoft.com/en-us/library/mt674882.aspx - informacja o typach zwracanych, i wyraźnie jest napisanem ze void dla async event handler.

+ super obrazek z flow

- the caller of a void-returning mthod can't catch any exceptions that the method throws - no jasne, ale to można je łapać w tej metodzie.

Dlaczego? Bo jak zwraca task, to Task ma te błędy. I przy awaitowaniu robi rethrow. (msdn.microsoft.com/en-us/library/mt674893.aspx) I dlatego catch działa.

Czyli async void, którego nie można awaitować, jest OK, jeśli Caller nie używa nic z tej metody.

To TaskWasCancelled -> OperationCancelledException

An approach I like to take is to minimize the code in my asynchronous event handler—for example, have it await an async Task method that contains the actual logic.

FORMS

absolute layout: bounds - anchorX, anchorY, width, height

Najwyraźniej nie można z androida w Formsach usunąć tab bar z góry. Próbowałem kilku sposobów, żaden nie działa:

- Xamarin.Forms.Forms.SetTitleBarVisibility(Xamarin.Forms.AndroidTitleBarVisibility.Never);

- musi być wywołane po Xamarin.Forms.Forms.Init(this, bundle);

- Theme = "@style/Theme.NoTitleBar" w atrybucie do Activity

- NavigationPage.SetHasNavigationBar(this, false);

- w Page.onAppearing

- zmiany stylów w styles.xml i tab... .axml

- Window.RequestFeature(WindowFeatures.NoTitle); w MainActivity

<https://www.youtube.com/watch?v=HwWg3nnHztc>

- 6.20 mówi gość wyraźnie, że nie ma desigera dla formsów

- mówi też że xaml previewer w VS to nie ma co jeszcze.

MessagingCenter w formsach albo pubsubevent w prism - PO CO? - pozwala VMsom komunikować się bez wiedzy o sobie (kolejny loose coupling)

-\_- kolumny i wiersze Grida są indeksowane od 0...

Robiąc SliderCell chciałem użyć tych wszystkich bajerow z książki (p. 612), ale okazuje się, że nie trzeba ustawiać ani OnPropertyChanged dla BindableProperty:

public static readonly BindableProperty SelectedValueProperty =

BindableProperty.Create(propertyName: "SelectedValue",

returnType: typeof(double),

declaringType: typeof(SliderCell),

defaultValue: default(double),

defaultBindingMode: BindingMode.TwoWay,

propertyChanged: (sender, oldValue, newValue) =>

{

SliderCell pickerCell = (SliderCell)sender;

Debug.WriteLine($"Changed from SelectedValueProperty: {newValue}");

//pickerCell.slider.

//if (String.IsNullOrEmpty(newValue))

//{

// pickerCell.picker.SelectedIndex = -1;

//}

//else

//{

// pickerCell.picker.SelectedIndex = pickerCell.Items.IndexOf(newValue);

//}

});

A jednak nie, propertychanged trzeba dodać, bo default jest null, i wtedy się nie updateuje w Vm. To tylko się zastanawiam, czy jest jakaś różnica, i jeśli tak, to jaka, między Bindowaniem Value do SelectedValue, a robieniu tego osamego używająvc event handlera \/.

Ani samemu sliderowi nie trzeba ustawiać OnValueChanged, bo wystarczyło zrobić Binding slidera Value do mojej Selected Value, i już VM widzi te zmiany.