People In Space Uno Demo

The demo can be found here <https://github.com/Appmilla/PeopleInSpace_Uno>.

The demo works well on all platforms I tried except for MacOS. I developed the demo running on a Mac with a Windows BootCamp partition running under Parallels.

This gave an overall good development experience; I was able to run using Visual Studio 2019 on Windows and Visual Studio for Mac also.

Overall from what I have seen so far Uno Platform is a great choice, I would say sharing the UI between Windows Desktop and Web would work well.

The MacOS app doesn’t work very well at all, I’m not sure if this is an issue for you.

The Wasm web platform uses the Mono linker to reduce the application size, I found this broke when I introduced the MS DI container, I spent some time trying to include the correct assemblies in the LinkerConfig.xml but ended up turning it off in the project file as I didn’t want to waste time on this. Its worth being aware of this as it can slow development down.

I found a good approach was to implement something new on Windows UWP and then check on the other platforms so if problems are introduced the surface areas is smaller.

For the next stage I would suggest investigating how to handle the form factor differences between the landscape desktop/web and portrait mobile phone. For example, selecting one of the crew members in the list should provide a detail view which would be displayed side by side with the list on web/desktop and would drill into a new page on a mobile phone.

Another issue I found was there was no pull to refresh control – this is ubiquitous in mobile apps and that surprised me initially but then the UI toolkit is a Windows Desktop based Api. I would suggest further investigation as to what other commonly used mobile phone UX controls are missing and how to implement UX interactions such as alerts/pop-ups, action sheets and toast messages across the different platforms.

I encountered some issues on an iPhone with the images appearing and disappearing while scrolling. Also, the ReactiveCommand we usually schedule the output of the command to run on a ThreadPool scheduler and marshal back to the UI Main Thread and the last opportunity. I found this was working in Windows UWP but when I ran on iOS a UIKit thread exception was raised, I would suggest we look into this further.

The Uno docs promote the use of a shared project where code is compiled into each platform app with hash ifdef’s for platform specific code. We have used this approach in our early days with Xamarin back in 2015 and would advise against this. It makes the code hard to read and somewhat brittle, also harder to test. Also suggested is the use of Partial classes to provide platform specific implementations – again I would advise against this. Andy and I used this approach way back for the MS surface table and touch Desktop apps and it’s hard to use.

I left the view xaml declaration in the template created SharedProject and created a new .Net standard class library for the view models queries and so forth. This will enable experimentation in the way we share code across platforms for example a Xamarin Native, Xamarin.Forms or Native.Forms app can also use this code if we experiment in those areas.

It should also help enable unit testing these implementations.

We have found bootstrapping the application start-up a good way of implementing platform specific implementations. This injects both common cross platform dependencies such as queries, view models, services, and controllers and then a section for the platform specifics such as Http handlers, authentication controllers and navigation services.

We have established some good patterns of usage for ReactiveUI now, and this demo builds on those, one of the next steps would be to introduce Akavache <https://github.com/reactiveui/Akavache> to demonstrate caching the data and updating from the Api. This allows for good offline support, helps with rate limiting Api calls and provides data to the screen very quickly.

The demo uses ReactiveUI, we have used this to great effect in Xamarin mobile apps, it is powerful and enables the application code to be simplified. The concept of reactive streams of data/events is particularly applicable to mobile apps and the declarative approach to state changes using WhenAny greatly simplifies ViewModel logic and reduces bugs.

It also allows a mix and match approach and is not an all or nothing framework. We found that it has good testing support and has enabled us to use unit testing to test a class as a unit but also a vertical slice of the app and test the behaviour.

This I found one of the biggest advantages as you can test a screen from the setup of the view model to loading the data and validating the screen and it's behaviour. When used in conjunction with a Http Intercept library you can test the error scenarios easily and verify the user interface behaves well.

A downside is that the ReactiveUI Api can be at times complex and not entirely natural, having established patterns of usages offsets this and enables developers unfamiliar with it to copy/paste their way to success.