.NET: Refactoring / Dependency Injection

This section will be a more around architectural and design considerations.

Someone who is affluent with .NET Core will probably have wondered "why didn't we start off like this", while we focused on the integration and left the considerations of other aspects of the application to the reader.

Someone who isn't so used to .NET Core, Dependency Injection (DI) or other of such design concepts - might've gotten quite easily overwhelmed without understanding what problem in the design we'll solve now.

Context

So far, we've considered the application as some sort of webpage and could sneak around into Android via the MvvM messenger.
Which behaves pretty much as a message bus in engineering software.

Now, we could write a single "page" with all functionality and logic, and depending on clicks and events, we could model it towards the single task the user is trying to get done.

But this would violate the "seperation of function" and breaks all the rules of modern "OOP" (Object Oriented Programming).

Consider our example: if we want to trigger a function from the GUI, we need to prepare the server side code for this "page" to receive and send these messages back and forth.

If we create a "new page", we have to do it again ? And what about, when someone comes back to a previous page, does it need to re-initialize everything ? Or, are we still interested in messages when we're not watching ?

.NET Core is - actually - a program rendering a GUI. And interfacing with this GUI over a sort of asynchronous back and forth to read the state, or see if the state changed (and it has to redraw) - while the logical engine is in front of either the Iphone, or ANdroid, or web, ...

BUt then the problem becomes: we could create an instance in the main program to handle everything. but then, how to get it into "a page" ? And how can "a page" communicate with this instance in the main program ?

Services

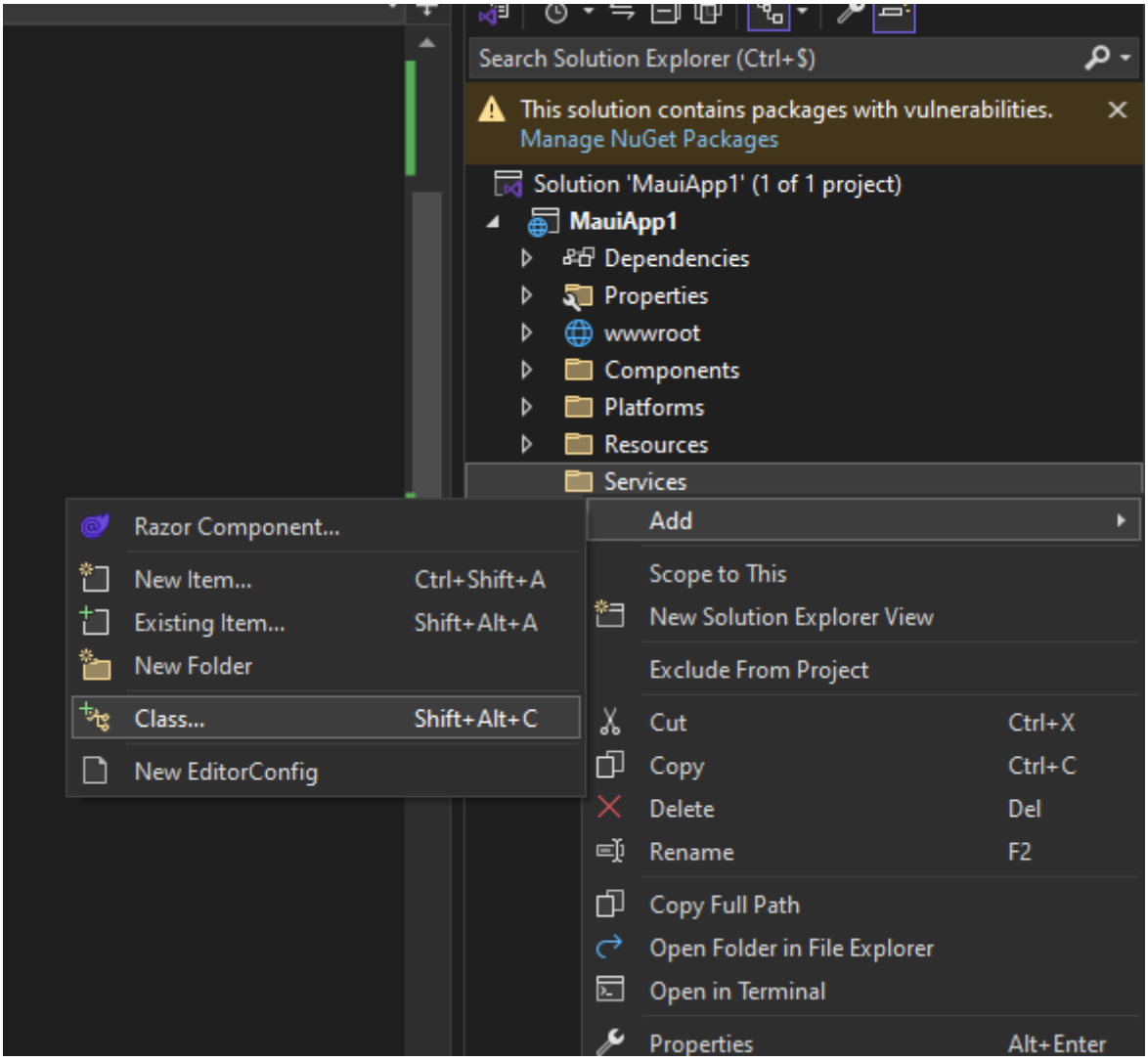
This is what is called "services" in .NET Core: we'll create a class that handles everything around a certain function or context in the program. And, whichever component i nthe program that wants to use this class - we can use the **@inject** keyword, so, when the page is created, the framework will *inject the service into the page* which references the single instance in the main application. So, every "page" will consume the same service.

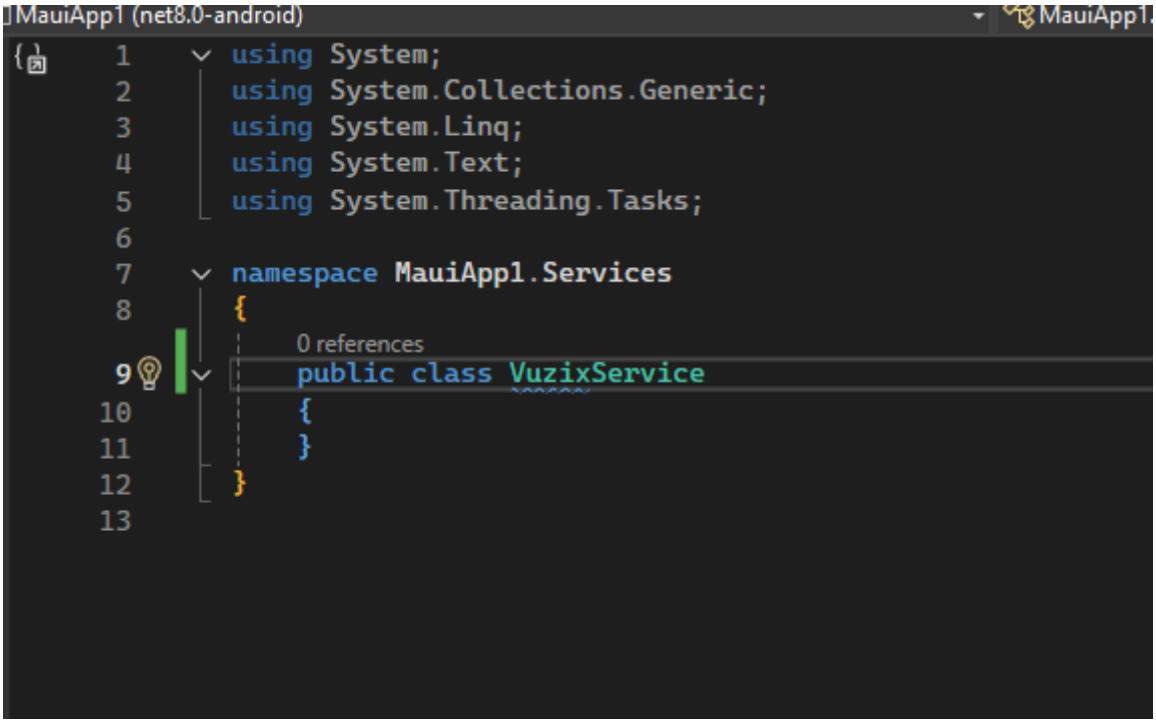
To get "notified of changes", we can add event handlers.

IN this way, we only have one instance of this "service" we'll make. And, the service will know who to notify, based on event handlers that are added to it.

The code

Create a folder services and add a new class "VuzixService"





Now, collect all the functions that were written to interface with the glasses into this class.

Revision #1
Created 12 October 2024 21:02:50 by Tim
Updated 13 October 2024 06:46:46 by Tim