VIDEO: https://www.youtube.com/watch?v=5Qga2pniN78&list=PLdo4fOcmZ0oUBAdL2NwBpDs32zwGqb9DY&index=5

|  |  |
| --- | --- |
| **ENGLISH** | **INDONESIA** |
| Today, what we're going to do is continue our journey to build native, cross-platform desktop and mobile apps with .Net MAUI.  Now previously we started to build our user interface with XAML and XML-based markup language.  Now we're going to do is we're going to extend that user interface and make it responsive and reactive using MVVM or Model View View Model.  This architecture pattern is very popular when you're developing applications with XAML because it enables data binding, a way for your user interface to respond to your code behind and vice versa.    It's a way of managing your control and flow of data.  Your view just knows how to display data.  There's a button, there's a label, and there's an entry.  Your view model, you can think of it as code behind but completely decoupled, and what it is representing is what to display.  It may have a list of objects or strings and may know what to do when a button is clicked and it may know what to display in a label, and the binding system inside of .Net MAUI is what brings it all together, enabling your UI to automatically update your code behind and vice versa.  It handles not only properties back and forth, but also events like button clicks or swipes to deletes.  What we're going to do now is head over to my machine and we're going to update that application and integrate MVVM directly into  this application so we can be more productive and abstract our code in a very nice, testable way.  Let's head over there now.  So far we've created our user interface for our my task application.  Here it's running on both Windows and Android.  Now so far we're able to see the items and that's just mock data.  We have no way of entering or deleting the task yet.  Let's go ahead and implement this and we're going to implement this with MVVM.  Let's go ahead and open up the source code.  The first thing that we can see that we're doing is that we don't actually have any data bindings to the entry or any way of interacting  with this button yet either.  Same thing down here with the swipe items, we would want to probably add something to say when I tap on the swipe item, do something. Here's what we're going to do.  We're going to come in and we're going to create a new folder by saying add a new folder.  I'm going to call this View Model and then I'm going to go ahead and add another class.  This one I'm going to call it Main View Model.  Now I like to create my view models very similarly named to my pages,  so main view model, main page.  Now, this is going to be some default code here, so we're just going to delete all of our used things, we don't need those, and we're going to make this a public class.  Now, there are some traditional ways of implementing MVVM into our application, that implementing INotify Property Changed.  Now INotify Property Changed gives us this system component model. When we implement this here, we can see that we get an event of the property change.  Now, this is an event that .Net MAUI will subscribe to automatically,  which means that we can notify .Net MAUI when we wanted to update the user interface.  We will probably create a little method for ourselves called on property changed passing a string, which would be the actual variable name here.    Then what we could do to simply come in as a property changed event and we're going to go ahead and invoke it.  We can pass this as the sender and then give it a new property change args of a name.  Now, if this looks a little bit scary, well, don't worry.  I will walk through exactly what it's doing in an example here.  Let's say we had a string of text and that's what we wanted to data bind our entry to; we're entering text here.  What we could do is create a public string of text.  It's really important that it's public.  That is how the data binding happens.  Here we'll say "Get" and we'll return the text.  There we go. And now when we do set, we can also just set the value.  But the other thing that we would want to do over here is not only set the value but also come in and say "On property changed" and give it the name of the text.  There we go. This means if we set the tax property, it will set the code behind the text value and then raise this property change notification, which means that the entry will automatically be updated and vice versa.  The entry is automatically two-way data bound, which means if we enter text here will automatically set the value right here in our code behind.  Now this does seem like a lot of code, so what's amazing is that there are amazing libraries from the community and from Microsoft to help simplify this, so what we're going to do now is right-click on the project, we're going to say "Manage NuGet packages" and we're going to go ahead and search for the CommunityToolkit.Mvvm.  Specifically, we want version 8.0. Now it may be in preview when you're watching this or it may be in for release.  If you don't see 8.0, just check that includes the pre-release.  Now here, we're just going to go ahead and install it.  What I love about the .Net Community Toolkit is that this works with any .Net application and it gives us source generators  to generate awesome code for us.  Here's what we're going to do. We're going to delete all this code that we just wrote beside the String Text.  Now what we're going to do is we're going to make this a partial class and we're going to go ahead and inherit from observable object and bring in the Community Toolkit MVVM Component Model.  Now what this does is it automatically gives us a bunch of those implementations so set property on property change it implements,  INotify Property Changed for us automatically.  What's also really cool is that it implements a bunch of source generators, so here I can say "Observable property".  When I do that, what's great about this private string is that the Observable property will generate source code for us.  In fact, it's telling us right here, as I hover over what it's going to do, I can go into my dependencies and into any of these analyzers and I can actually see the source generators in action.  Here's the observable property, and specifically here, is that text property.  Now, this is generated code. It's not the prettiest code but it's highly optimized.  Here we can see, our text and it'll automatically call the property change for us, which is really cool.  It does a bunch of data caching as well.  It's super B optimized.  Cool, now we just have one line of code which is a lot prettier.  Now what this means is that we have our observable property up and running.  Now if I come back over to my XAML,  I can now set the text property here binding to text.  Now we can note here that this is something that there's no binding context for the text, so we need to also clean that up too.  What I'm going to do is I'm going to add a new namespace.  Just like we were using a namespace in code,  you can have a namespace in XAML.  Here I can just type in "View model" and it'll find the namespace for us automatically in IntelliSense.  What I'm then going to do is do something called X colon data type.  What this will do is it will automatically associate this content page with the view model and we saw the little squiggles now go away, which is pretty great.  Now the other thing that we need to do inside our view model creates an interaction.  Basically what we want to do is we want to be able to tap this button and call a method.  What we're now going to do is we're going to create something called a command.  Now, commands are actually built into .Net MAUI, and you can use one there, but you can also use a command that is built into the MVVM Toolkit for .Net.  What we're going to do is we're just going to go ahead and create a method called Add, there we go.  What we want to do here is we want to add our item and then we're going to set the text property to a String Empty.  So we just clear it out.  Now, all we need to do is add another attribute called ICommand to the top of it, and ICommand is going to live inside of the Community Toolkit MVVM input.  Just like we saw earlier, if I pull over my source generators there's ICommand and we can see that this generated a bunch of code for us as well.  At the end of the day, this is going to give us an eye relay command and it's going to create a new command for us, and what it's going to do is call the add event automatically.  This is really nice because if this was asynchronous, it would automatically handle that for us back and forth, and it's just really nice tight code.  The other thing we need to do is create another property here.  This is going to be an Observable Collection of type strain and we're going to call it Items.  Now, let's go ahead and bring that into the system collections object model.  Again, we're also going to call this an Observable Property.  Perfect. Now, the final thing I'm going to do is I'm going to create a new one here.  I'm going to say New Main View Model here, and I'm going to say items.  >> [MUSIC]  >> Equals new. Perfect. Now that we have that new up and we can data bind to it, what we'll be able to do is add that item.  I'm going to say Items Add and I'm going to set the text property here.  Perfect. Now, I might want to also do something like say if string.IsNullOrWhitespace(text).  Let's go ahead and return. There we go.  Also, just make sure we always use the capital text there, just to be sure. There we go.  We add it and then we empty it out and that'll be added to our list.  Now if we go back to our main page we want to get rid of this item source.  We're going to go ahead and delete that here.  What we can do now is some data binding.  I'm going to say item source and reset that equal to binding to items.  Notice that IntelliSense automatically picks everything up for me.  Now I need to do one more thing, just like we actually set a data type right here for the view model we also need to set it onto our data template because this data template isn't bound to this main view model, it's bound to a string.  That is what is inside here. I'm going to say x: DataType. I'm going to say x: Type, x: String. Well, what is going to do is associate the data template to be a string.  Now here, most likely we would have some model-like-to-do items inside.  Now the last thing I need to do here before it is ready to go is actually need to come in and I need to say Command.  I need to set what the button is bound to. I'm going to give it binding to the AddCommand.  Now, this isn't a clicked event, but it is a command, which means that when it is clicked or interacted with, it'll call the AddCommand that will execute Add.  Notice that we're binding to the command. That is the one that was generated right here for us automatically.  We can see it's called AddCommand, whereas the method it's going to be calling is Add.  The last thing we need to do is actually create the binding context for this page.  I could come in and say binding context, which is all associated with this page to a new main view model.  That's one option that we can do here.  If we bring this in that is automatically created in our code behind.  But we can actually use the built-in dependency injection system to cascade these down.  That's great when we add additional dependencies to the main view model, which we'll see in the future.  I'm just going to say VM and I'm going to pass in the VM.  Now, the last thing we need to do is to register this system with the dependency service.  I'm going to go into the Maui program before we build it up I'm going to say builder.services.AddSingleton, and there are two different types of things that you can add to a service.  A singleton, which means it's like a global static it creates or a transient which will create it every single time.  We'll use that in the future. We're going to go ahead and add a singleton here and we're going to add a main page first.  We're going to register the page. We're also going to add another singleton here.  Now we're going to call it the main view model. Now, when we do this, these will now be automatically registered with the dependency service, which means that when I create this up and if I put a breakpoint on to the main page view model, we'll see that automatically be created for us.  This is really cool and one of my favorite features I've done in Maui in the show.  There it is. It's automatically created for us and we can see that we have zero items here.  Let's go ahead and continue on. Let's open up our application.  Now what I'm able to do is say bananas and hit "Add". Now our bananas are there. Well, amazing. I can also add a breakpoint here so that if I hit "Add", we can see that that ad is being called and boom, just like that.  The view comes to the text and it says that it's empty. There we go.  I come in and say Apple's hit "Add" and sure enough, now we have Apple's right here.  You can see it in full glory. Now, the last thing we wanted to do is actually add that swipe to delete.  What we're going to do now is we're going to go ahead and we are going to come in, stop debugging one more time, because we're going to add a little bit of code over here.  Now what I'm going to do is I'm going to say ICommand and I'm going to say void Delete.  I'm going to pass it in a string of s. Here, what I'm going to do is I'm going to say if Items.Contains(s).  Then let's go ahead and remove it, so Items.Remove(s).  There we go. That's all we're really going to do.  See if it contains it and then remove it.  That should remove it automatically from our list.  You notice that when I added it, it also automatically added it to the user interface.  That is because, with the observable collection, it automatically will trigger different notifications when things are added or removed, so that's pretty great. There we go. Now we have our command. Now, all we need to do is go to our main page and specifically add a command to the swipe item.  Here's what we're going to do. We're going to come in and say Command.  Now, remember, though, that we're currently bound to this string, so it doesn't actually know anything about the view model.  We need to give it something called ancestry binding, which basically means relative binding.  It says, Hey, don't use this data type or this binding.  Use the one above it. Here's what we're going to do, we're going to say binding source equals and it's going to be a relative source.  Here what we're going to do is give it an ancestry type to say, go up the stack and find something for me.  We're going to say x:type in viewmodel MainViewModel.  That does seem a little bit complex, but bear with me here, because the last thing we need to do is a path and then we're going to say,  please find the delete command.  If we walk through this, it's going to say, don't use the source, use a relative source.  It's an ancestor of the type view model so walk up, look for the view model, and specifically called this command.  Now we do need to pass it one more thing, though. We do need to pass it as a command parameter.  In that case, we are going to set it to bind just like this.  This is going to pass the string to the command and delete it automatically.  Let's go ahead and debug this now and add a breakpoint into our Delete.  What we'll see is that will have generated a command that automatically takes in a parameter.  That's pretty cool. Here's our application. I'm going to say Apple "Add", we had a breakpoint. I'll say Bananas "Add" continue on.  Now we get our swipe. I'm going to hit "Delete" sure enough what's passed in here is our apples automatically.  We're now going to remove them and the animations are all completely built-in. We've now gone from start to finish using MVVM and data binding to create a full application.  Well, there's a lot more that we can do. There you have it. We've just integrated an MVVM architecture pattern with data binding into our .Net  MAUI application.  Now, next up is something very common that you're going to do in any of your applications, which is navigation. Navigating from page A to  page B and back and forth.  Stay tuned to the .Net MAUI Beginner Series. | Hari ini, apa yang akan kita lakukan adalah melanjutkan perjalanan kita untuk membangun aplikasi desktop dan mobile lintas platform dengan .Net MAUI.  Sebelumnya kita sudah mulai membangun antarmuka pengguna dengan XAML dan bahasa markup berbasis XML.  Sekarang yang akan kita lakukan adalah kita akan memperluas antarmuka pengguna tersebut dan membuatnya menjadi responsif dan reaktif menggunakan MVVM atau Model View View Model.  Pola arsitektur ini sangat populer ketika Anda mengembangkan aplikasi dengan XAML karena memungkinkan pengikatan data, sebuah cara agar antarmuka pengguna Anda merespons kode di belakangnya dan sebaliknya.    Ini adalah cara untuk mengelola kontrol dan aliran data.  Tampilan Anda hanya tahu cara menampilkan data.  Ada tombol, ada label, dan ada entri.  Model tampilan Anda, Anda dapat menganggapnya sebagai kode di belakang tetapi benar-benar terpisah, dan apa yang diwakilinya adalah apa yang akan ditampilkan.  Model ini mungkin memiliki daftar objek atau string dan mungkin tahu apa yang harus dilakukan ketika sebuah tombol diklik dan mungkin tahu apa yang harus ditampilkan dalam label, dan sistem pengikatan di dalam .Net MAUI adalah apa yang menyatukan semuanya, memungkinkan UI Anda untuk secara otomatis memperbarui kode di belakang dan sebaliknya.  Ini tidak hanya menangani properti bolak-balik, tetapi juga peristiwa seperti klik tombol atau gesekan untuk menghapus.  Apa yang akan kita lakukan sekarang adalah pergi ke mesin saya dan kita akan memperbarui aplikasi itu dan mengintegrasikan MVVM secara langsung ke  aplikasi ini sehingga kita dapat menjadi lebih produktif dan mengabstraksikan kode kita dengan cara yang sangat bagus dan dapat diuji.  Mari kita pergi ke sana sekarang.  Sejauh ini kita telah membuat antarmuka pengguna untuk aplikasi tugas saya.  Di sini aplikasi ini berjalan di Windows dan Android.  Sejauh ini kita bisa melihat item-itemnya dan itu hanya data tiruan.  Kita belum memiliki cara untuk memasukkan atau menghapus tugas.  Mari kita lanjutkan dan terapkan ini dan kita akan mengimplementasikannya dengan MVVM.  Mari kita buka kode sumbernya.  Hal pertama yang dapat kita lihat yang sedang kita lakukan adalah bahwa kita sebenarnya tidak memiliki binding data ke entri atau cara berinteraksi  dengan tombol ini juga.  Hal yang sama di sini dengan item gesek, kita mungkin ingin menambahkan sesuatu yang mengatakan ketika saya mengetuk item gesek, lakukan sesuatu. Inilah yang akan kita lakukan.  Kita akan masuk dan kita akan membuat folder baru dengan mengatakan tambahkan folder baru.  Saya akan menamainya View Model dan kemudian saya akan menambahkan kelas lain.  Yang satu ini akan saya namakan Main View Model.  Sekarang saya suka membuat model tampilan dengan nama yang sangat mirip dengan halaman saya,  jadi model tampilan utama, halaman utama.  Sekarang, ini akan menjadi kode default di sini, jadi kita hanya akan menghapus semua hal yang kita gunakan, kita tidak membutuhkannya, dan kita akan menjadikannya kelas publik.  Sekarang, ada beberapa cara tradisional untuk mengimplementasikan MVVM ke dalam aplikasi kita, yaitu dengan mengimplementasikan INotify Property Changed.  Sekarang INotify Property Changed memberi kita model komponen sistem ini. Ketika kita mengimplementasikannya di sini, kita dapat melihat bahwa kita mendapatkan sebuah event dari perubahan properti.  Sekarang, ini adalah event yang akan berlangganan ke .Net MAUI secara otomatis,  yang berarti bahwa kita dapat memberi tahu .Net MAUI ketika kita ingin memperbarui antarmuka pengguna.  Kita mungkin akan membuat metode kecil untuk diri kita sendiri yang disebut pada properti berubah melewati string, yang akan menjadi nama variabel yang sebenarnya di sini.    Kemudian apa yang bisa kita lakukan untuk hanya masuk sebagai peristiwa perubahan properti dan kita akan melanjutkan dan memanggilnya.  Kita dapat mengoper ini sebagai pengirim dan kemudian memberinya argumen perubahan properti baru berupa nama.  Sekarang, jika ini terlihat sedikit menakutkan, jangan khawatir.  Saya akan menjelaskan apa yang dilakukannya dalam sebuah contoh di sini.  Katakanlah kita memiliki serangkaian teks dan itulah yang ingin kita ikat dengan data entri kita; kita memasukkan teks di sini.  Yang bisa kita lakukan adalah membuat string teks publik.  Sangat penting untuk menjadikannya publik.  Begitulah cara pengikatan data terjadi.  Di sini kita akan mengatakan "Get" dan kita akan mengembalikan teks.  Itu dia. Dan sekarang ketika kita melakukan set, kita juga bisa menetapkan nilainya.  Tetapi hal lain yang ingin kita lakukan di sini adalah tidak hanya menetapkan nilai tetapi juga masuk dan mengatakan "On property changed" dan memberinya nama teks.  Itu dia. Ini berarti jika kita mengatur properti pajak, itu akan mengatur kode di belakang nilai teks dan kemudian memunculkan pemberitahuan perubahan properti ini, yang berarti entri secara otomatis akan diperbarui dan sebaliknya.  Entri ini secara otomatis terikat data dua arah, yang berarti jika kita memasukkan teks di sini akan secara otomatis mengatur nilai di sini di dalam kode di belakangnya.  Sekarang ini memang tampak seperti banyak kode, jadi yang luar biasa adalah bahwa ada pustaka yang luar biasa dari komunitas dan dari Microsoft untuk membantu menyederhanakan ini, jadi apa yang akan kita lakukan sekarang adalah klik kanan pada proyek, kita akan mengatakan "Kelola paket NuGet" dan kita akan terus mencari CommunityToolkit.Mvvm.  Secara khusus, kita menginginkan versi 8.0. Sekarang mungkin masih dalam pratinjau saat Anda melihat ini atau mungkin sudah siap untuk dirilis.  Jika Anda tidak melihat 8.0, periksa saja apakah itu termasuk pra-rilis.  Sekarang, kita akan langsung saja menginstalnya.  Yang saya sukai dari .Net Community Toolkit adalah bahwa ini dapat digunakan dengan aplikasi .Net apa pun dan memberikan kita source generator  untuk menghasilkan kode yang luar biasa untuk kita.  Inilah yang akan kita lakukan. Kita akan menghapus semua kode yang baru saja kita tulis di samping String Text.  Sekarang apa yang akan kita lakukan adalah kita akan membuat ini menjadi kelas parsial dan kita akan melanjutkan dan mewarisi dari objek yang dapat diamati dan membawa Model Komponen MVVM Community Toolkit.  Sekarang apa yang dilakukannya adalah secara otomatis memberi kita banyak implementasi tersebut jadi set properti pada perubahan properti yang diimplementasikan,  INotify Property Changed untuk kita secara otomatis.  Yang juga sangat keren adalah ia mengimplementasikan banyak generator sumber, jadi di sini saya dapat mengatakan "Properti yang dapat diamati".  Ketika saya melakukan itu, hal yang hebat dari string privat ini adalah properti Observable akan menghasilkan kode sumber untuk kita.  Faktanya, ini memberi tahu kita di sini, ketika saya mengarahkan kursor ke apa yang akan dilakukannya, saya dapat masuk ke dependensi saya dan ke salah satu penganalisis ini dan saya benar-benar dapat melihat generator sumber beraksi.  Inilah properti yang dapat diamati, dan secara khusus di sini, adalah properti teks.  Sekarang, ini adalah kode yang dihasilkan. Ini bukan kode yang paling bagus tapi sangat dioptimalkan.  Di sini kita bisa melihat, teks kita dan secara otomatis akan memanggil perubahan properti untuk kita, yang sangat keren.  Kode ini juga melakukan banyak penyimpanan data.  Ini dioptimalkan dengan super B.  Keren, sekarang kita hanya memiliki satu baris kode yang jauh lebih cantik.  Artinya, properti yang dapat diamati sudah aktif dan berjalan.  Sekarang jika saya kembali ke XAML saya,  Saya sekarang dapat mengatur properti teks di sini untuk mengikat teks.  Sekarang kita dapat mencatat di sini bahwa ini adalah sesuatu yang tidak ada konteks pengikatan untuk teks, jadi kita juga perlu membersihkannya.  Yang akan saya lakukan adalah menambahkan namespace baru.  Sama seperti kita menggunakan namespace dalam kode,  Anda dapat memiliki namespace di XAML.  Di sini saya tinggal mengetikkan "View model" dan akan menemukan namespace untuk kita secara otomatis di IntelliSense.  Apa yang akan saya lakukan selanjutnya adalah melakukan sesuatu yang disebut tipe data titik dua X.  Apa yang akan dilakukannya adalah secara otomatis akan mengaitkan halaman konten ini dengan model tampilan dan kita melihat coretan-coretan kecil sekarang hilang, dan ini cukup bagus.  Sekarang hal lain yang perlu kita lakukan di dalam model tampilan kita adalah membuat interaksi.  Pada dasarnya apa yang ingin kita lakukan adalah kita ingin dapat mengetuk tombol ini dan memanggil sebuah metode.  Apa yang akan kita lakukan sekarang adalah kita akan membuat sesuatu yang disebut perintah.  Sekarang, perintah sebenarnya sudah ada di dalam .Net MAUI, dan Anda bisa menggunakannya di sana, tetapi Anda juga bisa menggunakan perintah yang sudah ada di dalam MVVM Toolkit untuk .Net.  Apa yang akan kita lakukan adalah kita akan langsung saja membuat metode bernama Add, dan itu dia.  Apa yang ingin kita lakukan di sini adalah kita ingin menambahkan item kita dan kemudian kita akan mengatur properti teks menjadi String Kosong.  Jadi kita hanya mengosongkannya saja.  Sekarang, yang perlu kita lakukan adalah menambahkan atribut lain yang disebut ICommand di atasnya, dan ICommand akan berada di dalam input MVVM Community Toolkit.  Seperti yang kita lihat sebelumnya, jika saya menarik generator sumber saya, ada ICommand dan kita dapat melihat bahwa ini menghasilkan banyak kode untuk kita juga.  Pada akhirnya, ini akan memberi kita perintah eye relay dan akan membuat perintah baru untuk kita, dan apa yang akan dilakukannya adalah memanggil event add secara otomatis.  Ini sangat bagus karena jika ini asinkron, maka secara otomatis akan menanganinya untuk kita bolak-balik, dan ini adalah kode yang sangat bagus.  Hal lain yang perlu kita lakukan adalah membuat properti lain di sini.  Ini akan menjadi sebuah Koleksi yang dapat diamati dari tipe strain dan kita akan menyebutnya Item.  Sekarang, mari kita lanjutkan dan membawanya ke dalam model objek koleksi sistem.  Sekali lagi, kita juga akan menyebutnya sebagai Properti yang dapat diamati.  Sempurna. Sekarang, hal terakhir yang akan saya lakukan adalah saya akan membuat yang baru di sini.  Saya akan mengatakan Model Tampilan Utama Baru di sini, dan saya akan mengatakan item.  >> [MUSIK]  >> Sama dengan baru. Sempurna. Sekarang kita memiliki yang baru dan kita dapat mengikat data ke sana, apa yang akan kita lakukan adalah menambahkan item tersebut.  Saya akan mengatakan Item Add dan saya akan mengatur properti teks di sini.  Sempurna. Sekarang, saya mungkin juga ingin melakukan sesuatu seperti if string.IsNullOrWhitespace(text).  Mari kita lanjutkan dan kembali. Ini dia.  Juga, pastikan kita selalu menggunakan teks kapital di sana, hanya untuk memastikan. Ini dia.  Kita tambahkan dan kemudian kita kosongkan dan itu akan ditambahkan ke daftar kita.  Sekarang jika kita kembali ke halaman utama, kita ingin menyingkirkan sumber item ini.  Kita akan menghapusnya di sini.  Apa yang dapat kita lakukan sekarang adalah beberapa pengikatan data.  Saya akan mengatakan sumber item dan mengatur ulang itu sama dengan mengikat item.  Perhatikan bahwa IntelliSense secara otomatis mengambil semuanya untuk saya.  Sekarang saya perlu melakukan satu hal lagi, sama seperti kita benar-benar menetapkan tipe data di sini untuk model tampilan, kita juga perlu menetapkannya ke template data kita karena template data ini tidak terikat pada model tampilan utama ini, ini terikat pada string.  Itulah yang ada di dalam sini. Saya akan mengatakan x: DataType. Saya akan mengatakan x: Tipe, x: String. Nah, yang akan dilakukan adalah mengasosiasikan template data menjadi sebuah string.  Sekarang di sini, kemungkinan besar kita akan memiliki beberapa item yang mirip model di dalamnya.  Sekarang hal terakhir yang perlu saya lakukan di sini sebelum siap digunakan adalah benar-benar perlu masuk dan saya perlu mengatakan Command.  Saya perlu mengatur ke mana tombol itu terikat. Saya akan memberikan ikatan ke AddCommand.  Sekarang, ini bukan event yang diklik, tetapi ini adalah perintah, yang berarti bahwa ketika diklik atau berinteraksi dengan tombol tersebut, tombol tersebut akan memanggil AddCommand yang akan menjalankan Add.  Perhatikan bahwa kita mengikat perintah tersebut. Perintah tersebut adalah perintah yang dibuat secara otomatis untuk kita.  Kita dapat melihat bahwa perintah tersebut dinamakan AddCommand, sedangkan metode yang akan dipanggil adalah Add.  Hal terakhir yang perlu kita lakukan adalah membuat konteks pengikatan untuk halaman ini.  Saya bisa masuk dan mengatakan binding context, yang semuanya terkait dengan halaman ini ke model tampilan utama yang baru.  Itu adalah salah satu opsi yang bisa kita lakukan di sini.  Jika kita memasukkan ini, maka secara otomatis akan dibuat di dalam kode di belakangnya.  Tetapi kita sebenarnya dapat menggunakan sistem injeksi ketergantungan bawaan untuk menurunkannya.  Ini sangat bagus ketika kita menambahkan dependensi tambahan ke model tampilan utama, yang akan kita lihat di masa depan.  Saya hanya akan mengatakan VM dan saya akan meneruskannya ke VM.  Sekarang, hal terakhir yang perlu kita lakukan adalah mendaftarkan sistem ini dengan layanan dependensi.  Saya akan masuk ke program Maui sebelum kita membuatnya, saya akan mengatakan builder.services.AddSingleton, dan ada dua jenis hal berbeda yang dapat Anda tambahkan ke layanan.  Singleton, yang berarti seperti statis global yang dibuatnya atau transien yang akan membuatnya setiap saat.  Kita akan menggunakannya di masa depan. Kita akan melanjutkan dan menambahkan singleton di sini dan kita akan menambahkan halaman utama terlebih dahulu.  Kita akan mendaftarkan halaman tersebut. Kita juga akan menambahkan halaman tunggal lainnya di sini.  Sekarang kita akan menyebutnya model tampilan utama. Sekarang, ketika kita melakukan ini, ini sekarang akan secara otomatis terdaftar dengan layanan ketergantungan, yang berarti bahwa ketika saya membuat ini dan jika saya meletakkan breakpoint ke model tampilan halaman utama, kita akan melihat bahwa secara otomatis dibuat untuk kita.  Ini sangat keren dan salah satu fitur favorit saya yang pernah saya lakukan di Maui dalam acara ini.  Itu dia. Secara otomatis dibuat untuk kita dan kita dapat melihat bahwa kita tidak memiliki item di sini.  Mari kita lanjutkan. Mari kita buka aplikasi kita.  Sekarang yang bisa saya lakukan adalah mengatakan pisang dan menekan "Tambah". Sekarang pisang kita ada di sana. Luar biasa. Saya juga dapat menambahkan breakpoint di sini sehingga jika saya menekan "Add", kita dapat melihat bahwa iklan tersebut sedang dipanggil dan bam, begitu saja.  Tampilannya sampai pada teks dan dikatakan kosong. Itu dia.  Saya masuk dan mengatakan Apple menekan "Tambah" dan tentu saja, sekarang kita memiliki Apple di sini.  Anda bisa melihatnya dengan penuh kemuliaan. Sekarang, hal terakhir yang ingin kita lakukan adalah menambahkan gesekan untuk menghapus.  Apa yang akan kita lakukan sekarang adalah kita akan melanjutkan dan kita akan masuk, berhenti melakukan debug sekali lagi, karena kita akan menambahkan sedikit kode di sini.  Sekarang yang akan saya lakukan adalah saya akan mengatakan ICommand dan saya akan mengatakan batal Hapus.  Saya akan meneruskannya dalam string s. Di sini, apa yang akan saya lakukan adalah saya akan mengatakan jika Items.Contains(s).  Kemudian mari kita lanjutkan dan hapus, jadi Items.Remove(s).  Itu dia. Hanya itu yang akan kita lakukan.  Lihat apakah item tersebut berisi dan kemudian hapus.  Itu akan menghapusnya secara otomatis dari daftar kita.  Anda perhatikan bahwa ketika saya menambahkannya, itu juga secara otomatis menambahkannya ke antarmuka pengguna.  Itu karena, dengan koleksi yang dapat diamati, secara otomatis akan memicu pemberitahuan yang berbeda ketika sesuatu ditambahkan atau dihapus, jadi itu cukup bagus. Itu dia. Sekarang kita memiliki perintah kita. Sekarang, yang perlu kita lakukan adalah pergi ke halaman utama dan secara khusus menambahkan perintah ke item gesek.  Inilah yang akan kita lakukan. Kita akan masuk dan mengatakan Command.  Sekarang, ingatlah bahwa kita saat ini terikat pada string ini, jadi string ini tidak mengetahui apa pun tentang model tampilan.  Kita perlu memberinya sesuatu yang disebut pengikatan leluhur, yang pada dasarnya berarti pengikatan relatif.  Ia mengatakan, Hei, jangan gunakan tipe data ini atau pengikatan ini.  Gunakan yang di atasnya. Inilah yang akan kita lakukan, kita akan mengatakan sumber pengikatan sama dengan dan itu akan menjadi sumber relatif.  Di sini yang akan kita lakukan adalah memberinya tipe ancestry untuk mengatakan, naik ke atas tumpukan dan temukan sesuatu untuk saya.  Kita akan mengatakan x:type di viewmodel MainViewModel.  Itu memang tampak sedikit rumit, tapi bersabarlah di sini, karena hal terakhir yang perlu kita lakukan adalah path dan kemudian kita akan mengatakan, silahkan cari perintah hapus.  Jika kita berjalan melalui ini, ia akan mengatakan, jangan gunakan sumber, gunakan sumber relatif.  Ini adalah nenek moyang dari tipe view model, jadi berjalanlah ke atas, cari view model, dan secara khusus memanggil perintah ini.  Sekarang kita perlu memberikan satu hal lagi. Kita perlu mengopernya sebagai parameter perintah.  Dalam hal ini, kita akan mengaturnya untuk mengikat seperti ini.  Ini akan meneruskan string ke perintah dan menghapusnya secara otomatis.  Mari kita lanjutkan dan lakukan debug sekarang dan tambahkan breakpoint ke dalam perintah Delete.  Apa yang akan kita lihat adalah bahwa kita akan menghasilkan perintah yang secara otomatis mengambil sebuah parameter.  Itu sangat keren. Inilah aplikasi kita. Saya akan mengatakan Apple "Add", kita memiliki breakpoint. Saya akan mengatakan Pisang "Tambah" lanjutkan.  Sekarang kita bisa menggeseknya. Saya akan menekan "Hapus", tentu saja yang masuk ke sini adalah apel-apel kita secara otomatis.  Sekarang kita akan menghapusnya dan semua animasi sudah terpasang. Kita sekarang telah menggunakan MVVM dan pengikatan data dari awal hingga akhir untuk membuat aplikasi yang lengkap.  Masih banyak lagi yang bisa kita lakukan. Itu dia. Kita baru saja mengintegrasikan pola arsitektur MVVM dengan pengikatan data ke dalam aplikasi .Net  Net dan aplikasi MAUI kita.  Sekarang, selanjutnya adalah sesuatu yang sangat umum yang akan Anda lakukan di aplikasi Anda, yaitu navigasi. Menavigasi dari halaman A ke  halaman B dan bolak-balik.  Ikuti terus Seri Pemula .Net MAUI. |
|  |  |