**112. Using Services for Cross-Component Communication**

* -: We talked a lot about services in this module and I hope you saw how services can clean our app up make the code a bit leaner, more centralized easier to maintain.
* You don't have to build these complex output, input chains where you pass events and properties to get data from component A to component B.
* It's much leaner now.
* ***Let me show you how much we actually saved.***
* Let's say that if we click on the button here in the account component for some reason we want to output something in the new account component.
* *A) Normally without services, we would have to emit an event in the account component that something was clicked or something happened so that we changed the status, for example.*

*B) We would have to catch the event here in app account as a site or we can of course remove these events here.*

* *They don't, they aren't triggered anymore.*
* *C) So we would have to catch these events here and then we would have to pass the new data down via property binding to the component where we wanted to handle it.*
* Pretty complicated.
* And building these chains of property and event binding is not the most convenient way of writing code.

**How it got easier:**

* It's so much easier with services.
* *Let's say in our account service here we want to provide some event which we can trigger in one component and listen to in another.*
* A) So we could simply add the status updated event which could be a new event emitter which we import from @angular/core.
* So make sure to add this import at the top.

Text

Description automatically generated

* Which will pass on the new event, excuse me, the new status.
* So a string.
* B) And yes, we could trigger it now here in the update status method.
* C) But we can also, since we inject the account service here where we set the new status, call account service, status updated and emit this with the new status.

Text

Description automatically generated

* So now I'm emitting an event I set up in the service.
* And as a side note, later in the observable section, you will learn about another construct you can use to submit or to emit events and subscribe to it instead of using the event emitter.
* But for now, that's absolutely fine.
* So here we are emitting an event.
* Again, the event emitter lives in our service.
* D) And in a new account, I now want to listen to it.
* So here, let's say I want to throw an alert.
* So I will do this in a constructor for now access my account service.
* And now for the status updated event here, I will subscribe to it because event emitter in the end just kind of wraps an observable.
* So here I could then receive the new status, which I know will be a string.
* And then I will simply throw an alert where I say, New status, status, something like that.
* So now I'm not building any chain of property and event binding.
* I do have cross component communication through a service with the event emitter.

Text

Description automatically generated

* And you can see this, if I click a button here, they actually got opened on an hour window.
* But you see this alert was thrown with the new status unknown.
* Same here, inactive.
* Active.
* So now we're communicating between components through a service, which really can save you a lot of time.
* And with that, I hope you saw a lot of reasons why services can be very helpful and how you can use them to your advantages.
* Make sure to use the right amount of instances.
* And if you inject services into services make sure to provide the service on the app module level and to add @injectable to disservice where you want to inject it in.
* Let's move on to some assignments now.