1. **Properties and Event Binding:**

**Passing Info From Parent Component(App) to Child Component(Cockpit)**

* In the last lecture, we split up our app into multiple components which is great, that is why we use components but one issue is that we need to pass data between these components.
* Now in the basic section, we learned about property and event binding and there we used it to, for example bind the disabled property of a HTML element.
* So we kind of passed data to that element.
* We passed the information that it should be disabled that this is set to true to this HTML element.
* The same with event binding.
* When we clicked this HTML button, something happened.
* This button emitted an event to which we were able to listen, so it sent us some data.
* The same for the input where we even used that data with dollar sign event, if you remember this.
* There we got data from the input element.
* This is exactly the behavior we need now with our own components.
* We need to be able to send data into a component or receive data, receive an event.
* And Angler, of course gives us great tools to implement this flow.
* We can use property and event binding not only on HTML elements and their native properties and events as we did thus far.
* We can also use it on directives and we also did this with Angie class and Angie style that we use property binding.
* But and that is important, we can also use it on our own components and bind to our own custom properties and custom events.
* We can emit our own events.
* That is what I'll dive into in the next lecture starting with custom property binding.

1. **Binding to Custom Properties:**

* -: So we learned that we can use property binding to bind to our own properties, properties of our components.
* Now this is what I'll start with.
* So to temporarily prevent my app from crashing, I will simply comment out the code in the cockpit so that this doesn't give us any errors anymore.
* For now and we won't be able to use it, but it will not crash.
* Now in the server component, server element component, if we have a look at its TypeScript at its HTML file, excuse me, you see that we try to access the element the single element, the single server element, and therefore we should create a property in this type TypeScript file element which represents our server.
* We can even assign a type by adding a colon and then simply defining the type which will be a JavaScript object, hence to quality braces.
* And just because it's important, this is not the value, this is TypeScripts syntax for defining the type to make sure that element may only have this type.
* And we then know that a element will have, well, what we're trying to access here.
* A type, a name and a content.
* So we'll have a type and type on its own will be of type string, we will then have a name which will be a string and will have a content which will be a string.
* So that is all now our type definition for a property we want to use in our HTML code we use here in all these places.

Graphical user interface, text

Description automatically generated

* Now still this element, this property is part of this component only of the server element component.
* We can't access it from outside.
* Now it would be great if in our app component where we manage our server or our array of server elements I should say, if we could somehow access this property, because let's say just so that we can see something, we start with one server here, which is of type server has a name of TestServer and a content of just a test.
* And just to be super clear, these colons here are not type definitions, we're on the right side of the equal sign, so here we are assigning a value and the value here simply is a JavaScript object, and in JavaScript objects we simply assign key value pairs or we create key value pairs by having the key then the content and then the value.
* So normal JavaScript syntax here.

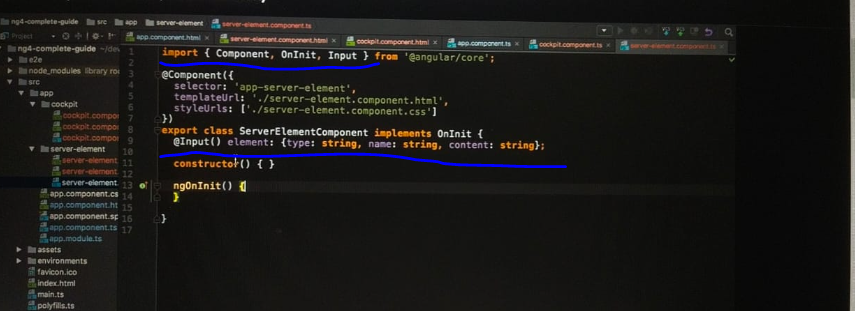
Graphical user interface, text

Description automatically generated

* So now we got our server elements array, we're looping through this array here on the app server element, and in the app server element we have our element property.
* So a property in that component.
* Now we want to access this component from outside.
* So it would be great if we could bind to it, just like we were able to bind to disabled.
* So maybe add squared brackets, enter the name of the property in between, so the name of the property in the component here, this name and then assign the server element.
* So simply the element we store for each iteration in this variable here.
* What be great if we could do this? And if we try this and our app reloads, we see that we get an error that we can't bind to element because it isn't a known property of app server element.
* Now you might say that is not true, it is a property of server element, here we defined it as a property, it's even public, right? We didn't add private in front of it or anything like that.
* So why can't we access it? Simple answer.
* *Because by default all properties of components are only accessible inside these components, not from outside and that generally is a good thing, you don't want to make all your properties bindable from outside.*
* So you have to be explicit about which properties you want to expose to the world so to say.
* So if you want to allow parent components to be able to bind to this property, you need to add something to that element property.
* You need to add a decorator.
* Remember, I said that decorators are not only available for classes, here we're going to add one to this property.

**@Input Decorator for exposing the property:**

* The decorator we need to add is @Input, and here important, you need to execute it, it's like a function in the end, so add parentheses and input now needs to be imported from @angular core.
* With this in place, now we are successfully exposing this property to the world.



* So now any parent component, any component hosting our server element component, so implementing it through its selector like we do here, is now able to bind to element.
* And I can demonstrate this if we save this, you now see just the test here you see our component because now we replicate this component for, well, each element in the array and we successfully can pass the element, the property down to that component and bind to the element property in that component.
* And this is how you can set up custom property binding on your own events, exposing your own properties to the components where you implement these components through the selector.