**Everything there is to know about Databinding - Angular**

**String interpolation (Double curly braces - {{ \*string\* }} ):**

There are 2 restrictions for the double curly braces, first one - we can write anything we want but it has to return a string in the end. The second one is that we can't write anything that’s multi-line or block expressions (a ternary expression is an option – an if condition with '?' and ':').



**Property Binding (Square brackets – [\*property name\*]):**

With this feature, we can assign native properties (such as the disabled property for buttons) to actual variables. So whenever the allowAddServerButton variable will change so does the status of the disabled property too.



**DO NOT MIX BETWEEN STRING INTERPOLATION TO PROPERTY BINDING**

**Event Binding:**

With this feature, we can listen to events and have our code do something when those events are happening. For example, if we want to listen to a click event of a button we will use angular syntax to select the event click, and activate our onClick function.



If we want to pass and use data with event binding we can use the reserved word $event (only between the quotation marks of the event name) and it will represent the data emitted with that event. If we want to see what data passed in the event we should console.log(event) in our function





In red we can see a typescript syntax so it will know we will receive an event from an HTML input element.

**The MDN (Mozilla Developer Network) offers nice lists of all properties and events of the element you're interested in. Googling for YOUR\_ELEMENT properties or YOUR\_ELEMENT events should yield nice results.**

**Two-way Databinding:**

With two-way databinding we combine the syntaxes of the event and property bindings. Let’s look at the following example:



**Custom Properties & Events Binding:**

Binding to a custom properties & events is a way to pass data between components.

**Custom Property Binding:**

By default, all properties of components are only accessible inside the component itself and not outside. If we want to expose the property to the world, we need to use decorators.

**@Input –** First we will need to import this decorator from angular/core. The Input decorator gives us the ability to make our propertied bindable from outside. Second, we will add the decorator itself near the property we want to expose like this:



Now, every parent component that uses our component will be able to bind to our element property like this:



And that’s how you can pass data down from a father to his child by property data binding.

**Custom Event Binding:**

**@Output -** To pass data from component to outside (from child to parent component), we need to use events, so let’s create a custom event. First, import “Output” from “angular/core” by using the Output decorator we are making our events listenable. Second, let’s import “**EventEmitter**” (An object in the angular framework which allows to emit our own events) from “angular/core” so that we could create the custom event.

In the export part of the component, we will add for each event we want to create a property with the @Output decorator in the start of the line, then we will assign a new EventEmitter to it from the type of the object we want to pass outside – this is the actual place we decide how our data will come out from this event. Example:



Then we will create functions that will use the properties of the events and assign the values we want to pass up. Here is an example:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

In the parent component we need to create functions that will listen to our new custom events, the functions will receive an event as a parameter and will use this data as need, this is the data we are passing from the child component. Example:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

In this example our event is the serverData and we receive a serverName and serverContent.

Now we need to bind to the relevnent event like this: תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

serverCreated and blueprintCreated is the events we listen to, and onServerAdded and onBlueprintAdded is the functions we made in the father component to react to the event.

\*\*For Input and Output we can assign them an alias, giving them a name inside the parenthesis. Once we gave our property/event an alias we can no longer use their actual name outside their own component.

**One way databinding:**

To create a one way databinding we will place a local reference on the element (can be placed on any HTML element) and is placed by “#\*name\*”, this will set a reference to the whole HTML element and can only be used in the template, if we want to pass it to the TypeScript it will be possible only by sending it to a function. In the picture the 1st mark is for the referencing and the 2nd and 3rd for passing it to typescript functions.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

In the type script functions we will identify the element as what it is (in this example an html input element – see 2nd and 3rd red marks), and assign its value to the serverName, see that the property newServerName is no longer in use (because we switched from 2-way-databinding to reference an html object – see 1st red mark) and we can comment it.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**How to access the Template & DOM:**

Sometimes we want to access the template and the DOM before we are calling a method, so we need to use a decorator, first we will create a reference to an element in the HTML template and then in the ts file create a new property using the **@ViewChild()** decorator.

**@ViewChild() –** ViewChild will create an element reference from the HTML template that we choose. Should be imported from angular/core and we need to pass arguments, the first one is how we want to select the element (could be a string, or a component type to get an access to the first occurrence of that component), and the second parameter is if we are using the element inside ngOnInit or not, if we do we need to add {static: true} else don’t add anything. Then we will add a reference to the element we want to use the decorator then we would use it by calling the field and then .nativeElement will make us treat the element ref like a regular element and let us get his value. Example:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

\*\***We shouldn’t change the elements using this method because that not how we are supposed to change the DOM, Angular offers better ways of accessing the DOM, we better use string interpolation or property binding or use directives and NOT directly mess with any element.**

**@ContentChild –** ContentChild is used to access element reference from the HTML that’s being sent to the component via ng-component decorator.

