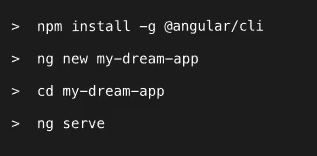
Angular

## Getting started

**Angular CLI** is really important to manage and write Angular applications. The Angular CLI is a toolset, for creating, building and managing Angular applications. It quickly creates Angular projects. 

We write our own applications mostly in the *app* folder.

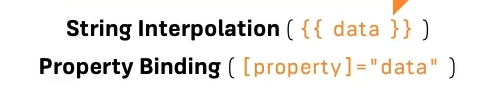
What is TypeScript? It’s a superset to JS, it offers more features than vanilla JS (like classes, interfaces and **types**). TS doesn’t run in the browser. It is always compiled to JS. This compilation is also done by the CLI. Angular is meant to be used together with the TypeScript.

To add BootStrap to our Angular application. We can install it with npm: **npm install –save bootstrap**  
then go to the angular-cli.json file and at the **styles** array you can define it’s path ((../)node\_modules/bootstrap/dist/css/bootstrap.min.css)

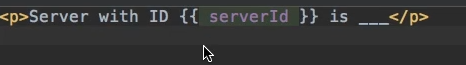
By using the selector in the TypeScript file we can call the different components. We just have to assign a name to the component for example **app-root**, which will be called in the html file like  **<app-root></app-root>.**

In Angular you build your whole application by composing from a couple of components, which you create on your own. Obviously, you can nest components into one another.

**Creating components** – Using CLI you can create components. You just have to type: **ng g(enerate) c(omponent) *name***

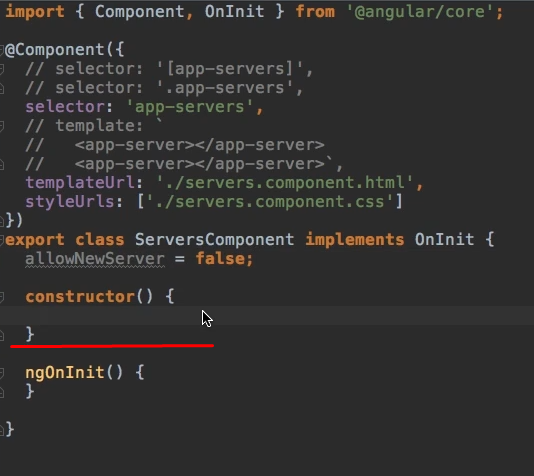
**Databinding -**  It’s a communication between your TypeScript Code (business logic) and the template code (HTML). There are different communications. For example we want to output data from our TS code in the template. We can use **String Interpolation** or **Property Binding.**There is another direction of the communication, when we want to record something (some event) from our template, we want to react to user events (for example a click event etc.). It is done with **Event Binding (event)=’’expression’’.**And then there is the third mode of databinding – the **two-way-binding** which is done with **[(ngModel)] = “data”.**

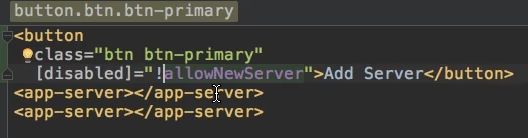
## String Interpolation





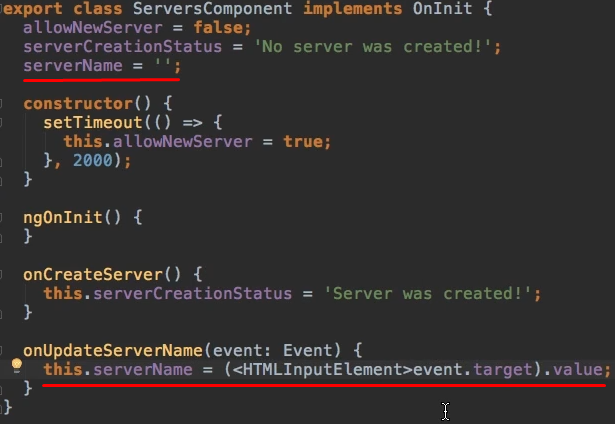
## Property Binding



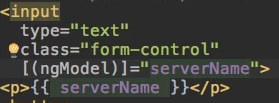
We are in a component’s TS file. The *allowNewServer* is a property, which has a boolean value. The constructor is simply a method executed at the point of time this component is created by Angular.  
  
Square brackets indicate, that we are using property binding, that we want to dynamically bind some property.

## Event Binding

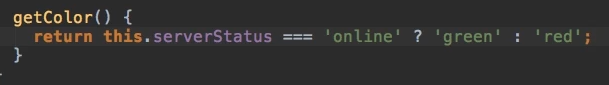
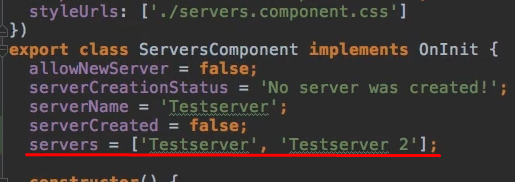
  
When the users clicks this button, the *onCreateServer* method will be executed, which is defined in the TS file.

  
This is how we can pass data to a method by using Event Binding.   
  
And this is how we can catch and use this value.

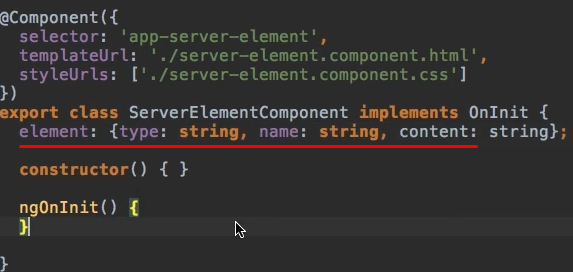
## Two-way-Binding

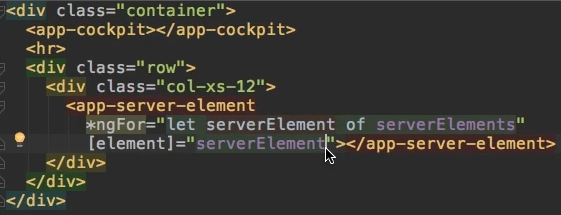
It is an even easier way to bind to some data. With two-way-bindig we can bind property and event binding. We use it by combining their syntaxes (square brackets and parantheses within) and by using a special directive **ngModel**.  
   
It will trigger on the input event and update the value of serverName in our component automatically. On the other hand, since it is two way binding it will also update the value of the input element if we change serverName somewhere else.

## What are Directives?

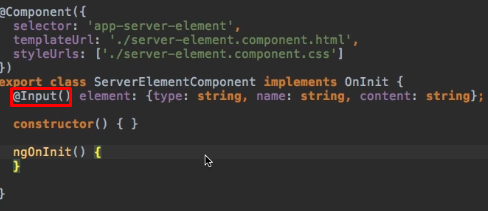
Directives are Instructions in the DOM.   
Screenshot_14.png  
ngIf is a frequently used directive in Angular. In this case we used the ngIf directive to output data conditionally, we just have to pass a boolean value to it.   
  
This is the case when we enhance the ngIf with an else condition.  
We can style elements dynamically with ngStyle. Screenshot_16.png  
  
We can output list by using the ngFor directive.  
Screenshot_18.png  


## Binding to Custom Properties



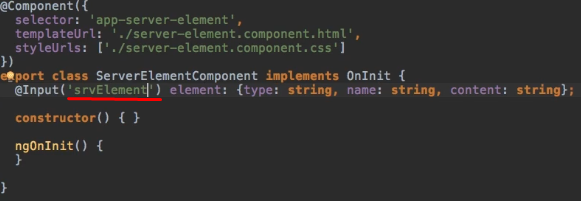
This is how we create our own property. This is also how you define types in TS. This property is part of this component only (of the server-element component). We can’t access it outside. By default every property is accessible only inside it’s component, not from outside. 

This is how you can call the **element** property in another component’s html file. Though this is not enough, you need to do one more thing to succeed with the custom property binding. All you need is a **Decorator**.

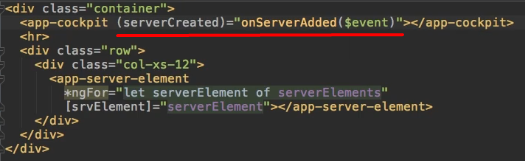
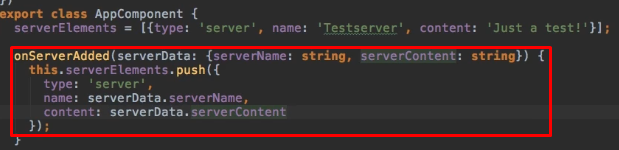
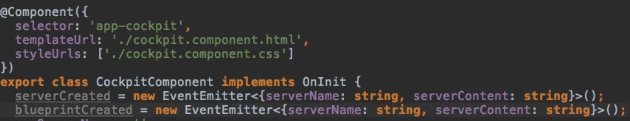
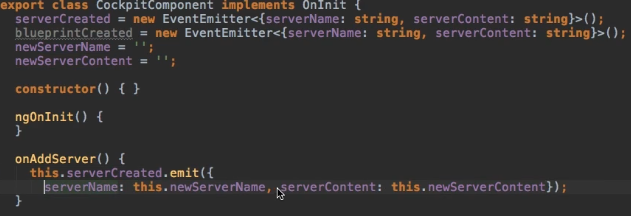


Now we can reach this property from outside.. from another component. Learning how Custom Property Binding works is crucial is Angular.. it’s super-important.

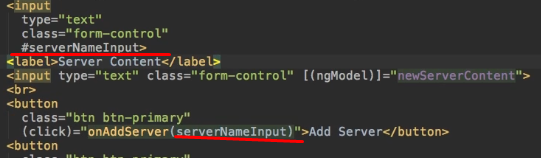
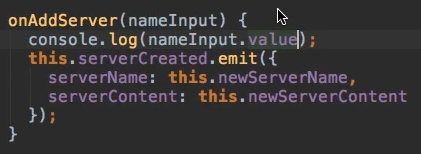
## Assigning an Alias to Custom Properties

Sometimes you don’t want to use the same property name outside of the component as you use inside of it.  


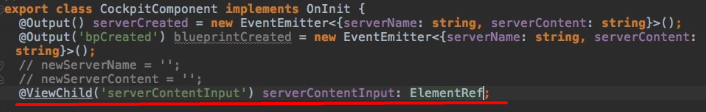
## Binding to Custom Events

What if we have a component and something changes in there, and we want to inform our parent component (so the component, which implements the other component).   
  
We create our custom Event **serverCreated** and we assign a method to it. So when this event happens, the **onServerAdded()** method will be executed which is defined in the parent component’s TS file. And we even expect to get some data with **$event** – this is how you could catch data, that could be some object, which gives us the required information about server (like the name or the content).  
Screenshot_10.png  
This would be the type of object we expect to get in this method.  
  
And this is how you create Custom Events.   
  
And this is how you can emit the event.  
  
Now there is one more thing left we have to manage. To make the event listenable from outside. It is done with another **Decorator** – yes yes yes, with the **@Output()** decorator. To assign an alias to the event, you have to write the assigned name into the paranthesis like **@Output(‘srvCreated’).**

## Using local references in templates

A local reference can be placed on any HTML element and you add it with hashtag like **#serverNameInput.** The reference will hold a reference to the given element. To the whole HTML element, with all it’s properties. You can use the references everywhere in your template, but (important) ONLY there. Local reference is a very nice feature to get access to some elements in your template and then use it even directly in the template.  
  


## Getting access to the template and DOM with @ViewChild

There is an alternative way to access templates.   
  
Here the **serverContentInput** is the local reference name itself, which is of course implemented in the template.

