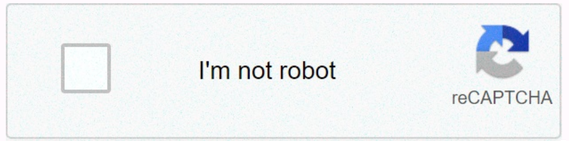

Continue

Template reference variable angular 6 example

A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an angular component or instruction or a web component (more information Angular.io). That means you can easily access the variable anywhere in the template. You declare a reference variable using the nosh symbol (#). the #firstNameInput announceds a firstNameInput variable on an <input> element. After that, you can access the variable anywhere within the template. For example, I pass the variable as a parameter in an event. Remember that lastNameInput belongs to the HTMLInput Spoo type. Usually the reference variable is only accessible within the template. However, you can use ViewChild decorator to reference it inside your component. After that, you can use this.nameInputRef anywhere inside your component.Working with <ng-template>In the case of ng-template is a little different because each template has its own set of input variables. For example: we use the default let-up - to declare the input variable fullNamethis fullNamethis variable is visible within the ng-template, not the outsideIn command to access the variable within the ng-template, we must declare contextYou can simplify my angular test 2 + variable reference pattern in Stackblitz. 30. December 18 at 1:42 pm 60392 View this article is all about the template reference variable or local reference variable, whatever you can say. I think you can get some idea with your name template reference which says we look forward to putting a reference to each template. Let me explain this with an example. Let's say that we have some basic DOM elements in our template display. Can anything like <div><h2>I am h2</h2>>>p>I am paragraphs</p>>>/div> and we want to access the content of the h2 tag element within our template but not in the component class. After accessing it, we also want to be displayed under the parent div. How can we do that? The answer is the template reference variable. Yes, by placing a pattern or local reference variable on that element. Angular provides us, a way to get a reference to any particular tail element, component or instructions in order to use it somewhere in our format itself. We just need to mark that particular element with the name that follows the hashtag symbol(#). When the element is specified, it is treated as a local reference variable for that mold and can be used to access its properties inside the mold view only. You can't access it within the component class or logic typing codes. <div><h2 #h2elm=>I am h2</h2>>>p>I am paragraphs</p>></div><hr> {{h2Elm.textContent}} We can do the same here for too much component and instructions {{helloRef.name}} // Template Ref Note Variable: The name of the identifier used for the template reference variable should be unique and should not conflict with any other template reference variable <hello name=Template Ref Variable #helloref=>>>/hello> Well! I told you you can't access that template reference by </ng-template>Inside the component class or logic typing but still, you can use that variable by passing it through a method that is called by the event listener. <div>>label for=Name>Name</label>>input type=text #nameinput=>Save <button(click)=onSaveName(nameInput)>name</button>>lt;/div> pass through that local variable by deleting # symbol. import { Component, } from '@angular/core'; @Component({ selector: 'my-app', templateUrl: './app.component.html', styleUrls: ['./app.component.css'] }) export class AppComponent { constructor() {} onSaveName(name: HTMLInput Espoo) { console.log (((lt; htmlinputelement=>name).value); } } Here, if you see, we are waiting for a name parameter that will be htmlInput Spoo type as well as inside the method body, we have the name variable wrapped with the same type. This is because we are accessing an example of the element that is <input> type. You can see in the image above that all the general properties of the input element are wrapped under HTMLInput Espoo. This is one good workout to explicitly define the type of parameter you're about to get in the method to avoid further confusion with the variable types in the body of the procedure. If you do this, Intellisense Editor will show you all the methods and properties in the incoming parameter property. All right, now get back to it. However, there is another way to access the template reference variable using the @ViewChild() decorator, using this decorator, we can apply that variable within our component without passing through the method as a reference parameter, or we can say, if we need to access it before the onSaveName() method is implemented. import { Component, OnInit, ViewChild, ElementRef, AfterViewChecked } from '@angular/core'; @Component({ selector: 'my-app', templateUrl: './app.component.html', styleUrls: ['./app.component.css'] }) export class AppComponent implements OnInit, AfterViewChecked { @ViewChild('nameInput') inputNameRef: ElementRef; constructor() {} ngOnInit() {} onSaveName(name: HTMLInput Espoo) { {console.log((lt; htmlinputelement=> name).value); console.log (this.inputNameRef.native Espoo .value); } ngAfterViewChecked() { console.log(After view got Checked); console.log(this.inputNameRef.native Espoo.value); } } Here, output 3 times on the control, twice for the procedure and one after the life cycle hook AfterViewChecked will be signed in. Don't forget that @ViewChild() decorator, our reference to variable will be ElementRef type since we are trying to access a reference to an element from our template perspective. ElementRef has a sub-property called native Vienna that wraps all the underlying properties of that particular reference element. We can also get the input value within the same format by placing a reference variable on it. <input type=text(change)=true #groupref=> {{groupRef.value}} Here, whenever an event changes, changes will be detected by the angular change detection system and we will receive its value Through threaded interpolation. Check mySQL code above. Or we can only connect access values via local reference with ngModel. This will generate the same output as the example above. <input type=text [ngmodel]=groupRef.value #groupref=> {{groupRef.value}} Property the value of the input element is bound to ngModel. Although, you can also customize your string to [ngModel] like [ngModel]='abc' //you will see abc in the input box but box, we just tried to assign the value property of input element which will hold whatever you type and then it will be passed to [ngModel] rendered in the input box. Here, we're at the end of this article but I haven't discussed anything important yet. Note that if you use <ng-template > or a structural instruction such as *ngIf or *ngFor creates a new template range and Wait template reference variables, I can fix the above lines with the following example. Let's say we have a groupList object defined in the groupList app.component.ts = [{ Name: ''}, { Name: ''}] and we have done some more than this object on the div element. <div *ngfor=let group of groupList>>input type=text(blur)=group.name=groupRef.value #groupref=> {{groupRef.value}} </div><hr><pre>>lt;/pre> it looks like the following thing thanks for reading this article. Please let me know your sugestions in the comment box. Photo By John Tyson in Reference Variables UnsplashTemplate is a little gem that allows to get a lot of good things done angularly. I usually call that feature a hashtag syntax because, well, it relies on a simple hashtag to create a reference to an element in a template:What the above syntax does is fairly simple: it creates a reference to the input element that can be used later in my format. Note that the domain for this variable is the entire HTML format in which the reference is defined. Here's how I can use that reference to get input value, for example:Note that the phone refers to the HTML object sample between for input. As a result, the phone has all the properties and methods of each HTML between (ID, name, innerHTML, value, etc.) above is a good way to avoid using ngModel or some other form of data connection in a simple form that doesn't require much validation. Does it work with parts as well? The answer is yes! Assuming we follow HelloWorldComponent:We can now get a reference to that component as follows, using the hashtag syntax:and the best part of it is that we are getting a reference to the actual component sample, HelloWorldComponent, so We can access any method or properties of that component (even if they have been declared as private or protected, which is surprising): obviously we can use that syntax not only to read data from a component, but also to change it. Does it work with instructions as well? Of course, but there's something more to know about it. Most instructions are used as features, which are </ng-template>We can't really apply hashtag syntax there, unless we use the same syntax with a twist:In the example above, myForm is a reference to the ngForm instructions applied to that form. Now if you're looking closely at the above HTML element, you'll probably think: Wait a minute, there are no ngForm instructions out there! I don't see any features called ngForm! And you have the right to think that the answer lies in the source code of the ngForm instruction:See the selector of that instruction? This applies to any form element that does not have the ngNoForm feature nor the formGroup. Because, my form element automatically gets the ngForm instructions applied. The second interesting thing in that property code is the exportAs in the decorator. This tells Angular: Hey, if anyone wants to refer this instruction with a template reference variable, the name for it is ngForm. now we know how all this works . We can create custom instructions and expose them by any kind of name with exportAs.My name is Alyn Chautard. I'm an angular Google developer expert, as well as an advisor and mentor in angular training where I help web development teams learn and become fluent with angles. Check @AngularTraining! If you enjoyed this article, please clap for it or share it. Your help is always commendable. Appreciate.