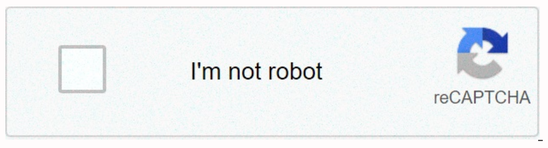

Continue

Angular 2 template form validation

One framework. Cellular & Desktop. In this post, we'll look at how the Form Corner API works and how it can be used to build complex forms. We'll cover the following topics: What form angles are all about Form Driven Templates, or 1 way Reactive Form Angles, or how to ReactiveLy Update new Functional Form Values, How to Reset Form Profits and Disadvantages of both types of Forms Can and should both be used together? Which one to choose by default? Angular Shape - what's it all about? Large categories of frontend applications are very solid form, especially in the case of corporate development. Many of these applications are basically just large forms, covering multiple tabs and dialogs and with non-trivial business validation logic. Each form intensive application should provide an answer to the following problem: how to track the status of the global form know which parts of the form are valid and which are still invalid displaying error messages correctly to the user so that the user knows what to do to create a valid form All these are similar trivial tasks throughout the application, and therefore can benefit from the framework. The Angle framework gives us some alternative strategies for handling forms: Let's start with the option closest to Angular 1. The Angle Driven Angular Shape 1 template handles shapes through the direction of well-known ng-models (read more about it in this post). The instantaneous two-way data binding of the ng model in Angular 1 is truly a lifesaver as it allows to transparently keep the form synchronized with the display model. Forms built with these directives can only be tested in end to end tests as this requires the presence of a DOM, but still, this mechanism is very useful and easy to understand. Angular now provides an identical mechanism named also ngModel, which allows us to build what is now called a Template-Driven form. Note that NgModel includes all the functions of its Angular 1 partner. Enable Form Driven Templates Unlike the case of AngularJs, ngModel and other form related referrals are not available by default, we need to explicitly import them in our application module: We can see here that we have enabled Template Driven Forms by adding FormsModule to our application, and dynamically bootstrapped the application. This is OK for development mode, but you may want to take a look at this post @NgModule for alternative bootstrap strategies for production. With the initial configuration in place, now let's build our first Corner Form. Our First Driven Form Template Let's look at forms that are built according to the way that the template is driven: There's actually quite a lot going on in this simple. What we have done here is declare a simple form with two controls: first name and password, both of which are required fields (indicated by the required attributes). The form will trigger the controller method onSubmitTemplateBased at the time of submission, but but only enabled if both required fields are populated. But that's only a fraction of what's going on here. NgModel Validation Function Note the use of [(ngModel)], this notation emphasizes that the two form controls are bound both ways to the display model variable, named as simple users. The syntax [(ngModel)] is known as the 'Banana Box' syntax:-) More than that, when the user clicks the required field, the field is displayed in red until the user types something. The angle actually tracks the three form field statuses for us and applies the following CSS classes to their forms and controls: touching or untouched valid or invalid pure or dirty CSS country class is very useful for form style form error status. The angle actually tracks the validity status of the entire form as well, using it to enable/disable the send button. This function is actually common for template-driven and reactive shapes. The logic for all this has to be on the controller, right? Let's take a look at the controllers associated with this view to see how all the logic of this form is applied: There's not much to see here! We only have declarations for users of display model objects, and event handlers used by ngSubmit. All very useful functionality of the error tracking form and registration validator is taken care of for us without any special configuration! How did Angular do this? The way it works, is that there is an implicitly defined set of form directives that are being applied to the view. The angle will automatically apply the form-level directive to the form transparently, create a FormGroup and link it to the form. If for some reason you don't want this, you can always disable this functionality by adding ngNoForm as a form attribute. In addition, each input will also be applied a directive that will register itself with the control group, and a registered validator if such an element is required or maxlength is applied to the input. The presence [(ngModel)] will also create a two-way binding between the form and the user model, so in the end there is not much more to do at the controller level. This is why this is called form-based templates, because validation and binding are all arranged in a declarative way at the template level. What if we don't have to tie it both ways, but just initialize the field? Sometimes we just want to create a form and initialize it, but not have to do a two-way binding. We can let the user fill out the form and press submit, and only then get the latest value. We can do this using the plain [ngModel] syntax: This will allow us to initialize the form by filling out the user object field: What if we don't need to field, can we still get validation? For example, a form creation doesn't require an initial value, the form just needs validation. If we just want to get ngModel validation functionality without without without value or two-way binding, we can do it with the following syntax: Advantages and Disadvantages of Form Driven Templates In this simple example we can't really see it, but keeping the template as the source of all the truth of form validation is something that can be quite difficult to read rather quickly. As we add more and more validator tags to the field or when we start adding complex cross-field validation the readability of the form is reduced, to the point where it will be more difficult to submit it to the web designer. The reverse part of the way this form is handled is its simplicity, and perhaps more than enough to build different shapes. On the downside, form validation logic cannot be unit tested. The only way to test this logic is to run end to end tests with a browser, for example using a headless browser like PhantomJs. Template Driven Forms from a functional programming point of view There is nothing wrong with form driven templates, but from a programming point of view programming techniques bi-directional binding is a solution that promotes mutability. Each form has circumstances that can be updated by many different interactions and it is up to the app developer to manage that state and prevent it from being damaged. This can be difficult to do for a very large form and can introduce a category of potential bugs. Again it is important to realize that this only happens in a very large/complex form. Angles do provide different alternatives to managing forms, so let's get through it. Forms Reactive (or Model Driven) Reactive shapes visible on the surface are quite similar to template-driven shapes. But to be able to create this type of form, we need to first import different modules into our application: Note that here we import ReactiveFormsModule instead of FormsModule. This will contain reactive form referrals instead of template-driven referrals. If we find ourselves in a situation where we will happen to need both, then we should import both modules, more on this later. Our First Reactive Form Let's take our previous example and rewrite it but this time in reactive style: There are some differences here. First, there is a formGroup directive that is applied to the entire form, tying it to a controller variable named form. Also note that required validator attributes are not applied to form controls. This means the validation logic must be somewhere in the controller, where it can be tested units. What's the controller like? There is a little more going on in the Reactive Form controller, let's look at the controller for the form above: We can see that the form is really just a FormControl, which tracks the state of global validity. Control it can be individually disinformed or defined using simplified array notation using the form builder. In array notation, the first element of the array is the initial value of the control, and the remaining element is control validator. In this case both controls are created mandatory through the validators.required built-in validator. But what happened to ngModel? Note that ngModel can still be used with reactive forms. It's just that form values will be available in two different places: the display model and FormGroup, which has the potential to cause confusion. For this reason mixing ngModel with reactive forms is best avoided. The advantages and disadvantages of The Reactive Form you may wonder what we gained here. On the surface there is already a big advantage: We can now test the logic of form validation. We can do this simply by instantiating the class, assigning some values in the control form and making a statement against the form of global validity and the validity status of each control. But this is only one possibility. The FormGroup and FormControl classes provide APIs that allow us to build UIs using a completely different programming style known as Functional Reactive Programming. Functional Reactive Programming in This Corner deserves its own blog post, but the main point is that the form controls and the form itself now provides an Observable based API. You can think of observable values only as a set of values over time. This means that the control and the whole form itself can be seen as a continuous flow of value, which can be subscribed and processed using commonly used functional primitius. For example, it is possible to subscribe to a flow of value forms using a Observable API like this: What we do here is retrieve the form value stream (which changes every time the user types in the input field), and then apply to some commonly used functional programming operators: maps and filters. In fact, form flow provides a whole range of functional operators available in Arrays and more. In this case, we convert the first name to uppercase using a map and only retrieve valid form values using filters. This creates a new flow of valid values only that we subscribe to, by providing callbacks that determine how the UI should react to new valid values. The advantages of building UIs using Functional Reactive Programming (FRP) We are not obliged to use FRP techniques with The Reactive Form of Angle. Simply using it to make the template cleaner and allowing component unit testing is already a big plus. But as usual in the case of Observable-based APIs, FRP techniques can help easily implement many use cases that would otherwise be rather difficult to implement such as: pre-save the form in the background at any valid, or even invalid circumstances (e.g. storing invalid values in cookies for use Typical desktop features like Undo/Repeat See this video Introduction to Functional Reactive Programming – Using Async Pipes – Traps to Avoid, part of The Corner Service and HTTP courses for more on Functional Reactive Programming in Angular. Updating the Form Value we now have available to update the entire form, or just a few fields. For example, let's create some new buttons on the reactive form above: We can see here that there are two buttons for updating form values, one for partial updates and the other for a full update. Here's how the component method fits: We can see that FormGroup provides two API methods for updating form values: we have a patchValue() that partially updates the form. This method does not need to accept values for all form fields, which allows partial updates there is also setValue(), where we pass all form values. In the case of this method, the value for all form fields needs to be provided, otherwise we will get an error message saying that the value of a particular field is missing We may think that we can use this same API to rearrange the form by passing the blank value to all fields. It will not work as intended because the pure and untouched status of the form and its fields will not get reset accordingly. But Angular Final provides an API for this use case. How to Reset Form Notifications in the list of buttons above that there is a cancel button that calls the reset() method, which resets everything back to pure and untouched: Let's see if it is possible to mix both types of forms and if that is recommended. Reactive vs. Template Driven: can they be mixed? Reactive and Template-Driven under the hood are implemented in the same way: there is a FormGroup for the entire form, and one FormControl instance per each individual control. If for some reason we need to, we can mix and match two ways of building forms, for example: we can use ngModelto read data and use FormBuilder for validation. we do not have to subscribe to the form or use RxJ if we do not want to. We can declare the control in the controller, and then refer to it in the template to obtain its validity status But in general, it is better to choose one of two ways of performing the form, and use it consistently throughout the application. Which type of form should be selected, and why? The Driven Forms template may be for a slightly less verbose simple form, but the difference is insignificant. The Reactive form is actually much stronger and has almost equivalent readability. Most likely in large-scale applications, we will eventually need reactive form-driven functionality to implement more advanced use cases such as auto-save. Everything can be done in both types of forms, but some things are simpler using reactive forms It is not that there is functionality that cannot be implemented with template-driven forms. there are many functions especially features of more modern forms such as for example auto-save that can be implemented in just a few lines RxJ. Which type of form to choose? Did you migrate the Angular 1 app to Angular? It is an ideal scenario to use Template Driven Forms. Or do you build a build application from the beginning? Reactive forms are a good default option because more complex validation logic is actually easier to implement using them. For example, imagine validations that require checking two fields and comparing them: for example password fields and password confirmation fields must be identical. With a reactive form, we just have to write the function and connect it to formcontrol. With a form driven by a template, there's more to it: we need to define a referral and somehow pass it the value of two fields. The reactive form seems good for example for enterprise applications with a lot of complex inter-field business validation logic. As mentioned earlier, we wanted to avoid situations where we used both types of forms together, as it can be a bit confusing. But it's still possible to use both forms together if for some reason we really need to. Angle provides a way to create forms: Template Driven and Reactive. The Template Driven approach is very familiar to Angular 1 developers and is ideal for easy migration of Angular 1 applications to Angular. The Reactive approach removes validation logic from the template, keeping the template clean from validation logic. But it also allows a completely different way to build UIs that we can use optionally. This is not an exclusive option but for consistency issues, it is better to choose one of two approaches and use it everywhere in our application. If you want to know more about Angular Forms, Victor Savkin's podcast at Angular Air explains in detail about the two types of forms and ngModel. This blog post gives a high level of overview of how Angles would be better off enabling Functional Reactive Programming techniques. If you are interested in learning about how to build components in Angles, also check out the Basics of Angle components. If you want to learn the Corner Form in detail, check out this Youtube Playlist with samples from our course: Finding more Corner Learning Resources? Check out our list of top 10 Corner Resources: Top 10 Corner Tutorials, Blogs, and Other Post Podcasts on Angular If you enjoyed this post, these are some other popular posts on my blog: blog: