**Angular template driven form**

Angular template-driven forms are easy to build and the logic of the forms is controlled inside the component template. Angular Template form, where the form controls are defined in the template of the component, and all its validation errors are managed primarily through the template.

Template forms are simple, quick to get started, perfect for simple forms. But it has disadvantages that HTML and business rules are coupled, no unit testing can perform on it.

In template-driven form, we can use only directives in a component’s templates to create, access, and validate form elements. The directives below the list, are the common building blocks of Angular template form. All these directives are part of FormsModule.

1. NgModel
2. NgModelGroup
3. NgForm

An Angular provides a way to use HTML native form elements like select, input, radio, etc, and add some power to them. Angular allows us to add additional control to the form through these directives like ngForm, ngModel, and ngModelGroup features.

* ngForm directive : Represent entire form, here we have bind ngForm object to local reference variable #userForm.
* ngModel: Allow two way data binding, we can attach individual form element to reference using ngModel. In our case we bind input name element to #nameCtrl and we can use this reference to access input element value.**nameCtrl.value**
* Using userForm local reference with value, we can display all input element value

These **ngForm, ngModel, and ngModelGroup**directives are the main ingredients for making template form, without them we can’t work on template form. All of that directives are part of the FormsModule, so we need to import and register this module in the app.module.ts file.

**Directive used in Angular template form**

In our above example, we have used a few template form directives to create a template form. Angular templates use the following directives.

|  |  |
| --- | --- |
| **Name** | **Description** |
| ngModel | Allow two-way data binding to form element and allow angular to create form control on an element. |
| name | Used in templates in form elements to specify its name in the form model. |
| ngForm | Used in component templates to bind the local variable to the form template element. In the above example, #loginForm add a local reference to the form. |
| NgForm | This is used inside the component typescript to represent the entire form. |
| ngSubmit | We have used this directive in the form element and it intercepts the HTML form submits event. |
| ngModelGroup | To create a sub-group of a form element and we used this directive in the template. |

**ngModel:**  
Angular will not automatically detect this input in this form and we may also need to control the form element. In input, we have to add ngModel directive from the two-way data binding. This will tell angular that input is actually control of the form. The ngModel is a directive made available in the formModule.

**Name attribute:**  
By using the name attribute we can specify the name of the control in a form. To recognize the input as a control in a form, we need to give angular the name of control by adding the name attribute on the input. The name is the default attribute you can add to any HTML control.

**The ngForm Directiv**e  
The ngForm directive stores state information for the form, including the following:

* Values for all the controls element inside the form
* form groups fields in the form
* Validators
* Fields in the form

**ngModelGroup Directive**  
The ngModelGroup, we can use this directive if we like to create a sub-group within a form. This can come in handy if you want to validate a sub-group of your form separately from the rest of your form, or if some values in your domain model make more sense to consume together in a nested object.

## Angular template driven forms validation

In Angular template-driven form validation, we have two approaches to adding validation.

1. Using built in form validation.
2. Creating our own custom form validation.

The Angular Forms API, have built in form validation from Validators class, with static functions such as required() , minLength() , maxLength() , pattern() , email() , and others.  
These built-in validators can be used in template form by specifying the directive’s name in the input element.

* required
* minLength
* maxLength
* pattern
* email

The pattern validator enables us to specify a regular expression for Angular form validation on the input element.

## Angular validator CSS

HTML already provides some built-in form validations that can be put onto form elements, such as required or minlength . Angular works with these attributes and automatically will validate inputs based on them.

When we use ngModel directive on a form, it allows angular detection is form control input and tracks the state of the form fields. Angular adds a couple of classes on form elements to indicate form element validation status. Angular automatically mirrors many control properties onto the form control element as CSS classes.

We can use these classes to style form to show warnings or style the form elements according to the state of the form element. The following classes are currently supported.

|  |  |
| --- | --- |
| **CSS class name** | **Angular form element status** |
| ng-pristine | Data entry has not been touched |
| ng-dirty | The input field has been interacted with |
| ng-touched | The input field has received focus |
| ng-valid | Data entry has passed validation |
| ng-invalid | Data entry has not passed validation |
| ng-untouched | Set this CSS class if a user has not tabbed out from the input control. |