Forms

Template Driven: Angular selects the structure of the form.

Reactive Approach: You set up the structure of the form .

Make sure you import forms in you module.ts

ngSubmit will be fired whenever the form is submitted.

Give the element an Id and make it equal to ngForm so that you can get some javascript data of everything submitted in the form. This is the normal use of forms.

Dirty means you changed something about the form.

Touched means you clicked into a field.

You can use viewchild to get the data when you submit or even earlier.

You can add required directive to make sure the user has to enter data for that input. You can also use the email directive to make sure that users are entering email addresses and not just random characters.

You can still use two way binding to update every keystroke and save the data on for you to see.

Radio buttons act like any other input tag.

setValue command will let you set a form value however it will overwrite any data that is currently in the form.

patchValue is the recommended use because it will only overwrite certain data and not all data.

If there is not a value on the html than use the class of the wrapper div to get values. Ex: this.signupForm.value.userData.username vs this.signupForm.value.username

.reset() will reset the form data and the state of anything.

TD approach will work in most use cases.

Reactive you create the form programmatically in typescript.

For reactive forms you need the ReactiveFormsModule from forms

To setup a new form group use new FormGroup({});

To control the data entered in use new FormControl(null, validators, asyncronousValidators) where the first value is the initial state and the validators. If there is more than one validator put them into an array. You can have the method without validators but it is good practice to validate data.

To connect to the formGroup use property binding to connect to the form and the typescript. In each input or form item add formControlName where you will end up connect to the FormControl.

To pass validators the syntax is Validators.whateverValidatorYouWant.

Use the get method to get access to the information you want.

You can group inside of form groups to make data a little more organized. However if you group inside of a group when you call the get command you have to use your group name to get the data. Ex: normal: signupForm.get(‘username’).valid, inside a smaller form group: signupForm.get(‘userData.username’).valid.

Form array is used to hold an array of controls. It is created by using new FormArray([])

To create a custom validator you send the method a control and make it return a type that is a javascript object and whatever variable or type you want, the type is most likely a Boolean because you want to make check whether it passed or not. Ex: forbiddenNames(control: FormControl): {[s:string]: Boolean]} and this will return something that looks like {‘nameIsForbidden’: true}. Always return null not false.

To be able to access the variable you will have to bind this to forbidden names so that you can access the data. Ex: this.forbiddenNames.bind(this)

-1 means true on an array

You can use error to handle or show data to allow your users to see the error message. Ex: signupForm.get(‘userData.username’).errors[‘nameIsForbidden’] would only show if name is forbidden erroring and the same would happen if you put an error for ‘required’.

In asynchronous validation the method should be sent a control and should return an observable or a promise. Ex: forbiddenEmails(control: FormControl): Promise<any> | Observable<any>

You can look at value changes and subscribe to them to see every change that is made.

You can look at status changes to see the status of a form.

You can use setValue to set the values of the form.

You can also use patchValue to just patch values instead of changing all of the values.

You can reset in the same was as the td method.