# Reactive Forms

Angular reactive forms facilitate a reactive style of programming that favors explicit management of the data flowing between a non-UI data model (typically retrieved from a server) and a UI-oriented form model that retains the states and values of the HTML controls on screen. Reactive forms offer the ease of using reactive patterns, testing, and validation.

With reactive forms, you create a tree of Angular form control objects in the component class and bind them to native form control elements in the component template.

You create and manipulate form control objects directly in the component class. As the component class has immediate access to both the data model and the form control structure, you can push data model values into the form controls and pull user-changed values back out. The component can observe changes in form control state and react to those changes.

# Which is better, reactive or template-driven?

Neither is "better". They're two different architectural paradigms, with their own strengths and weaknesses. Choose the approach that works best for you. You may decide to use both in the same application.

**Steps to create Reactive Form:**

1. Classes used FormBuilder , FormGroup
2. For validation add Validators

**Inspect FormControl Properties**

At the moment, you're dumping the entire form model onto the page. Sometimes you're interested only in the state of one particular FormControl.

You can inspect an individual FormControl within a form by extracting it with the .get() method. You can do this within the component class or display it on the page by adding the following to the template, immediately after the {{form.value | json}} interpolation as follows:

heroForm.get('name').value

|  |  |
| --- | --- |
| **Property** | **Description** |
| myControl.value | the value of a [FormControl](https://angular.io/api/forms/FormControl). |
| myControl.status | the validity of a [FormControl](https://angular.io/api/forms/FormControl). Possible values: VALID, INVALID, PENDING, or DISABLED. |
| myControl.pristine | true if the user has *not* changed the value in the UI. Its opposite is myControl.dirty. |
| myControl.untouched | true if the control user has not yet entered the HTML control and triggered its blur event. Its opposite is myControl.touched. |

# reset the form flags

You should reset the form when the hero changes so that control values from the previous hero are cleared and status flags are restored to the pristine state. You could call reset at the top of ngOnChanges like this.

ngOnChanges() {

this.heroForm.reset({

name: this.hero.name,

address: this.hero.addresses[0] || new Address()

});

}

# Use FormArray to present an array of FormGroups

To work with a FormArray you do the following:

1. Define the items (FormControls or FormGroups) in the array.
2. Initialize the array with items created from data in the data model.
3. Add and remove items as the user requires.

<div formArrayName="formName" class="well well-lg">

<div \*ngFor="let address of formName.controls; let i=index" [formGroupName]="i" >

<!-- The repeated address template -->

</div>

</div>

addNewAddress () {

this.secretLairs.push(this.fb.group(new Address()));

}