|  |  |
| --- | --- |
| Template  -Uses HTML (template) with two way binding  -easier to use  -automatically tracks form and input element state. Use the state information to determine if the form is valid | Reactive  -Component handles the code  -more flexible  -since no data binding, immutable data model  -Easier to add input elements dynamically and unit test  -bind to the form model not the data model |
| Value Changed (Pristine/Dirty)  Validity (Valid/errors)  Visited (Touched/untouched) | -These states define whether the user has changed a value in an input element  -If one is dirty, then entire form is dirty  -All input elements must be valid  -key of each array element is the name of the  Validation rule  -Form is touched when any of the input element has been touched |
| Form Control  -tracks the value and state of an individual input elements  --same for Template & Reactive but created differently  Form Model (not a data model)  -data structure that represents HTML form  -retains form state, user’s entries | Form Group  -tracks the value and state of a group of FormControls  -form itself is a FormGroup (what is a form but a group of input elements) |
| Directives | Template-driven (FormsModule)  -ng Form (to access the Form Model), Model (two way binding/access state), ModelGroup  Reactive (ReactiveFormsModule) |
| Template  -Angular generates the FormModel by creating FormGroup and FormControl instances  -HTML validation | Reactive  -Create the FormModel by creating FormGroup and FormControl ourselves in our component class  A screenshot of a computer code  Description automatically generated-Then bind the template to the form model, this means that our form is not directly modifying our data model  -validation in class |
| Template  -requires name attribute to associate the FormControl with FormGroup  - | -template reference variable  #firstNameVar=”ngModel” to access the FormControl instance  -use that TFV to check state properties  [ngClass]=”{‘is-invalid’: firtNameVar.touched}” |
| 7. FormGroup, FormControl | -Can pass validations in FormControl (next module) |
| 8. FormModel | -Don’t confuse this form model (defines the set of FormGroups/Controls that match up with the HTML form and input elements) with our data model (customers = new Customer()) which defines the data passed to and from a back-end server  -AbstractControl is base class for FormControl and FormGroup |
| 9. Form Model Properties |  |
| 10. setValue, patchValue | -use a method to set values    -use patch for only subset |
| 11. FormBuilder | -a class to build reactive forms  -shortens boilerplate code  -provided as a service  Option 1: An Object    Option 2: An Array |
| 12. Validators | -Use Validators class for built in    -use an array to pass multiple validators  -3rd element is asynchronous validators, common use is calling a server-side validation method, to minimize asynchronous calls, asynchronous validators are not executed until all synchronous validators pass validation |
| 13. Runtime Validation | -call setValidators method on FormControl instance then update it |
| 14. Custom Validator | -Takes a FromControl or Group as a parameter |
| 15. Custom Validator with Parameters | -must import ValidatorFn |
| 16. Cross-field Validation | -Define a nested FormGroup for FormControls that are validated together |
| 17. Cross-field Custom Validator | -Requires that we provide an object with a validator key and function as value |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |