ANGULAR 6

Lesson 06

TEMPLATE-DRIVEN AND REACTIVE FORMS

Objectives

Template-Driven and Reactive Forms

- Template-Driven vs Reactive Approach
- Understanding Form State
- Built-in Validators & Using HTML5 Validation
- Grouping Form Controls
- FormGroup, FormControl, FormBuilder, FormArray
- Forms with Reactive Approach
- Predefined Validators & Custom Validators
- Async Validators
- Showing validation errors

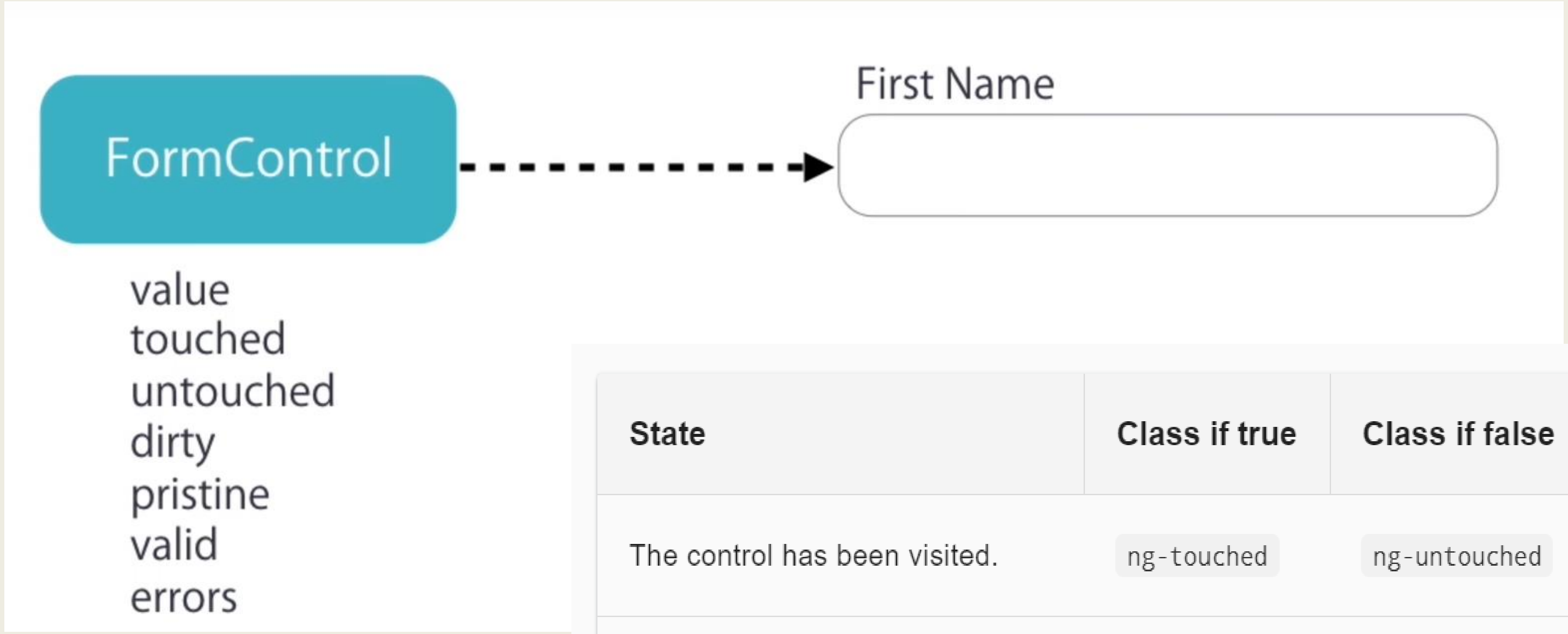
Template-Driven vs Reactive Approach

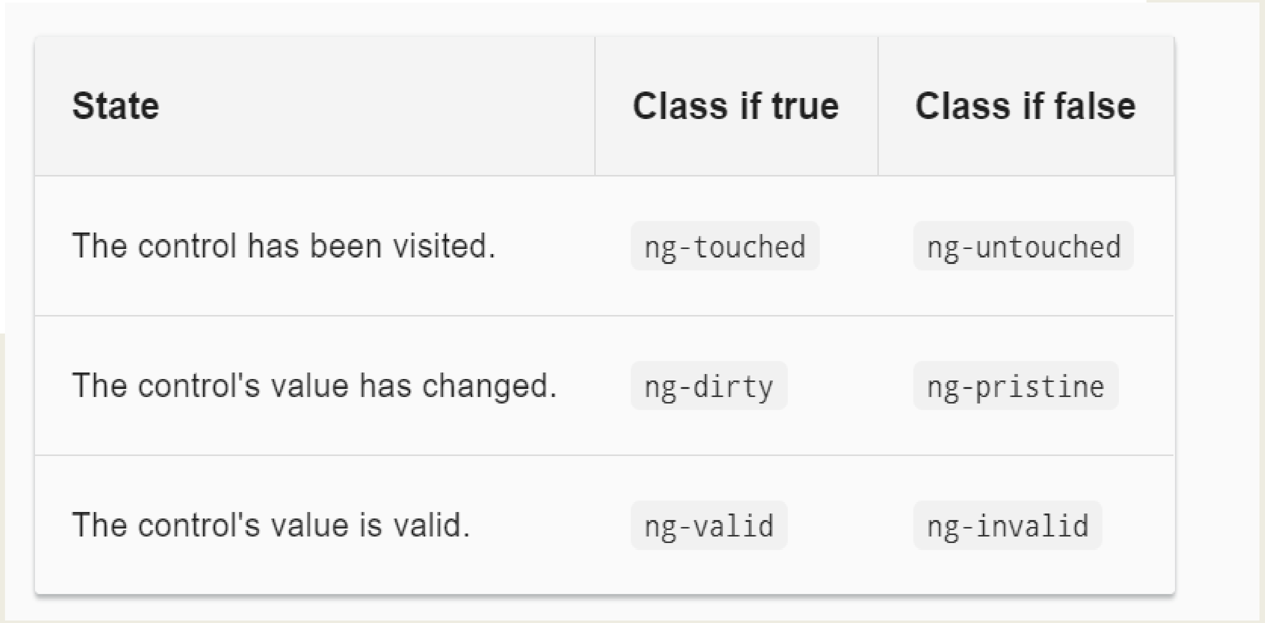
TEMPLATE-DRIVEN FORMS

Template Driven Forms

- Forms build by writing templates in the Angular template syntax with the form-specific directives and techniques are called as Template Driven Forms.
- The app component doesn't need to do much since the form fields and validators are defined in the template when using Angular template-driven forms
- The component defines a model object which is bound to the form fields in the template in order to give you access to the data entered into the form from the app component
- Using ngModel in a form gives you more than just two-way data binding. It also tells you if the user touched the control, if the value changed, or if the value became invalid.
- The NgModel directive doesn't just track state; it updates the control with special Angular CSS classes that reflect the state.
  - *Angular2 “infers” the FormGroup from HTML Code*
  - *Form data is passed via ngSubmit()*

Understanding Form State





State	Class if true	Class if false
The control has been visited.	`ng-touched`	`ng-untouched`
The control's value has changed.	`ng-dirty`	`ng-pristine`
The control's value is valid.	`ng-valid`	`ng-invalid`

Validation

```
<form #empForm=ngForm (ngSubmit)="getData(empForm)">
   <table>
      <tr>
         <td>Product ID</td>
         <td><input required id="eid"  name ="id" [(ngModel)]="emp.eId“
                  type="text" #idcontrol="ngModel"/>
                <span  *ngIf="idcontrol.invalid && idcontrol.touched" > ID is
                  required</span>
         </td>
      </tr>
      <tr>
         <td>Product Name</td>
         <td><input required id="ename" name ="empname“
                  [(ngModel)]="emp.eName" type="text“
                  #namecontrol="ngModel"/></td>
         <span  *ngIf="namecontrol.invalid && namecontrol.touched" >
            Name is required</span>
      </tr>
   </table>
</form>
```

Built-in Validators & Using HTML5 Validation

Grouping Form Controls

FormGroup, FormControl, FormBuilder, FormArray

Demo

- Demo Template Driven Forms

REACTIVE FORMS

Forms with Reactive Approach

- The app component defines the form fields and validators for our registration form using an Angular FormBuilder to create an instance of a FormGroup that is stored in the registerForm property.
- The registerForm is then bound to the form in the template below using the [formGroup] directive.
- Also need to be added a getter 'f' as a convenience property to make it easier to access form controls from the template. So for example you can access the email field in the template using f.email instead of registerForm.controls.email.

Forms with Reactive Approach

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
 @Component({
        selector: 'app',
        templateUrl: 'app.component.html'
})
 export class AppComponent implements OnInit {
        registerForm: FormGroup;
        submitted = false;
         constructor(private formBuilder: FormBuilder) { }
         ngOnInit() {
                this.registerForm = this.formBuilder.group({
                        firstName: ['', Validators.required],
                        lastName: ['', Validators.required],
                        email: ['', [Validators.required, Validators.email]],
                        password: ['', [Validators.required, Validators.minLength(6)]]
                });
        }
         // convenience getter for easy access to form fields
        get f() { return this.registerForm.controls; }
         onSubmit() {
                this.submitted = true;
                 // stop here if form is invalid
                if (this.registerForm.invalid) {
                        return;
                }
                 alert('SUCCESS!! :-)')
        }
}
```

Forms with Reactive Approach

```
 <form [formGroup]="registerForm" (ngSubmit)="onSubmit()">
            <div class="form-group">                <label>First Name</label>
      <input type="text" formControlName="firstName" class="form-control" [ngClass]="{ 'is-invalid': submitted
&& f.firstName.errors }" />
      <div *ngIf="submitted && f.firstName.errors" class="invalid-feedback">
      <div *ngIf="f.firstName.errors.required">First Name is required</div>   </div>   </div>
            <div class="form-group">                <label>Last Name</label>
         <input type="text" formControlName="lastName" class="form-control" [ngClass]="{ 'is-invalid':
submitted && f.lastName.errors }" />
         <div *ngIf="submitted && f.lastName.errors" class="invalid-feedback">
         <div *ngIf="f.lastName.errors.required">Last Name is required</div>          </div>    </div>
   <div class="form-group">             <label>Email</label>
      <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted &&
f.email.errors }" />
      <div *ngIf="submitted && f.email.errors" class="invalid-feedback">
            <div *ngIf="f.email.errors.required">Email is required</div>
            <div *ngIf="f.email.errors.email">Email must be a valid email address</div>  </div>    </div>
            <div class="form-group">                <label>Password</label>
      <input type="password" formControlName="password" class="form-control" [ngClass]="{ 'is-invalid':
submitted && f.password.errors }" />
      <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
            <div *ngIf="f.password.errors.required">Password is required</div>
            <div *ngIf="f.password.errors.minlength">Password must be at least 6 characters</div></div> </div>
   <div class="form-group"> <button [disabled]="loading" class="btn btn-primary">Register</button>
            </div>   </form>
```

Predefined Validators & Custom Validators

Async Validators

Showing validation errors