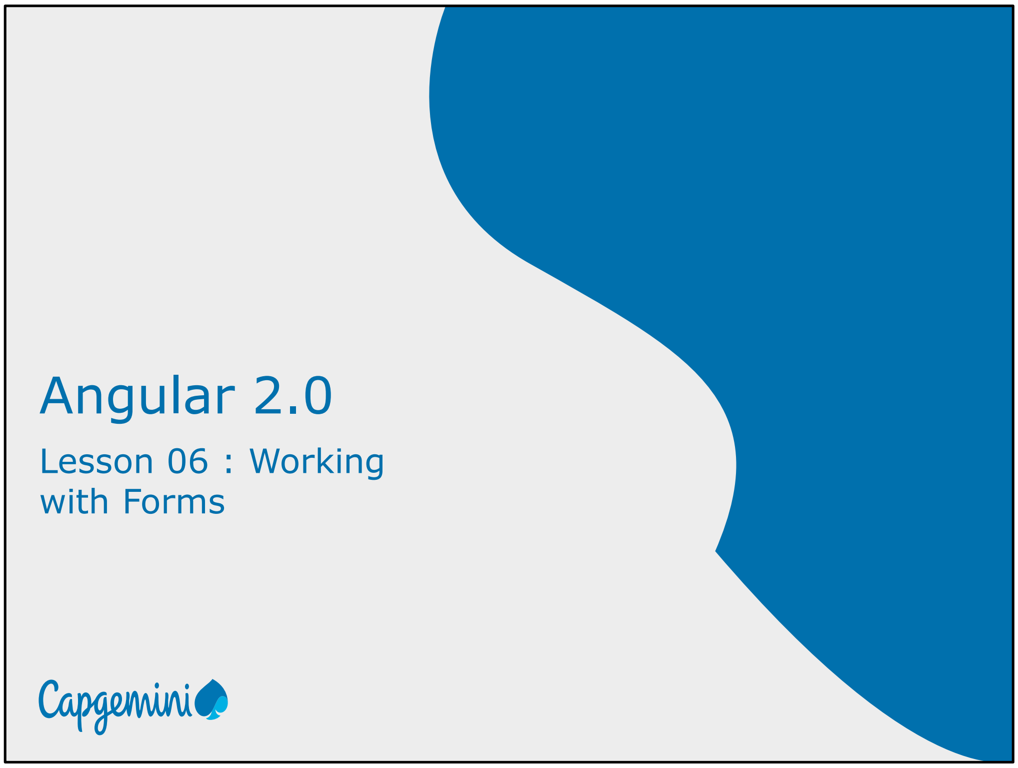
Instructor Notes:

Add instructor notes here.



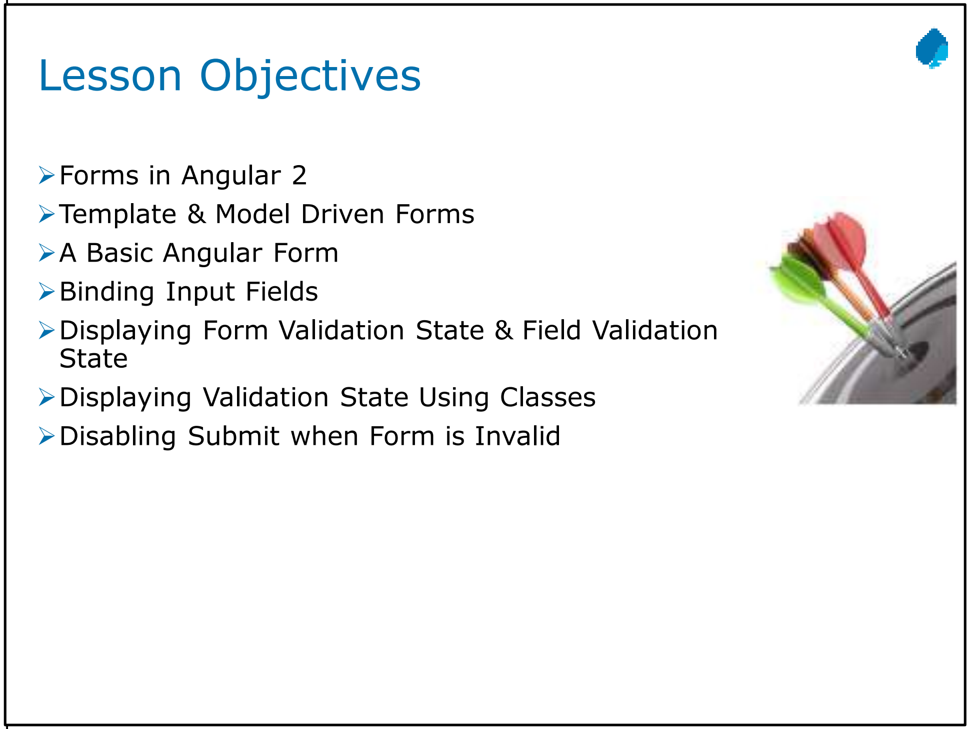
Angular 2.0

Lesson 06 : Working with Forms

Capgemini

Instructor Notes:

Add instructor notes here.

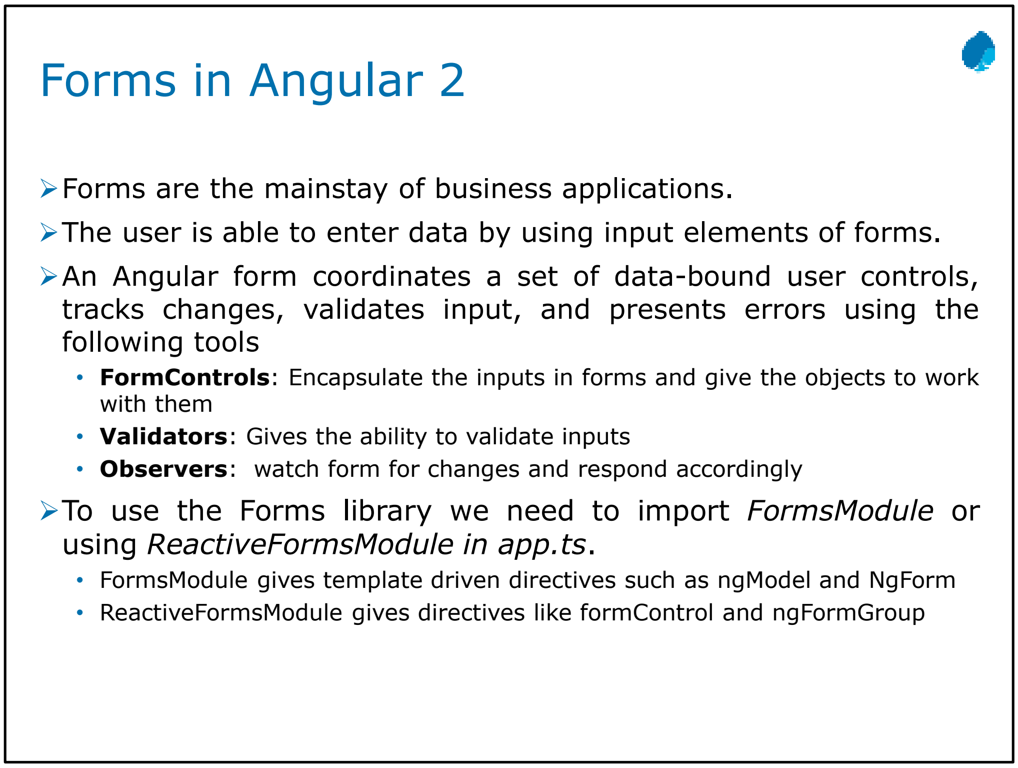


Lesson Objectives

- Forms in Angular 2
- Template & Model Driven Forms
- A Basic Angular Form
- Binding Input Fields
- Displaying Form Validation State & Field Validation State
- Displaying Validation State Using Classes
- Disabling Submit when Form is Invalid

Instructor Notes:

Add instructor notes here.

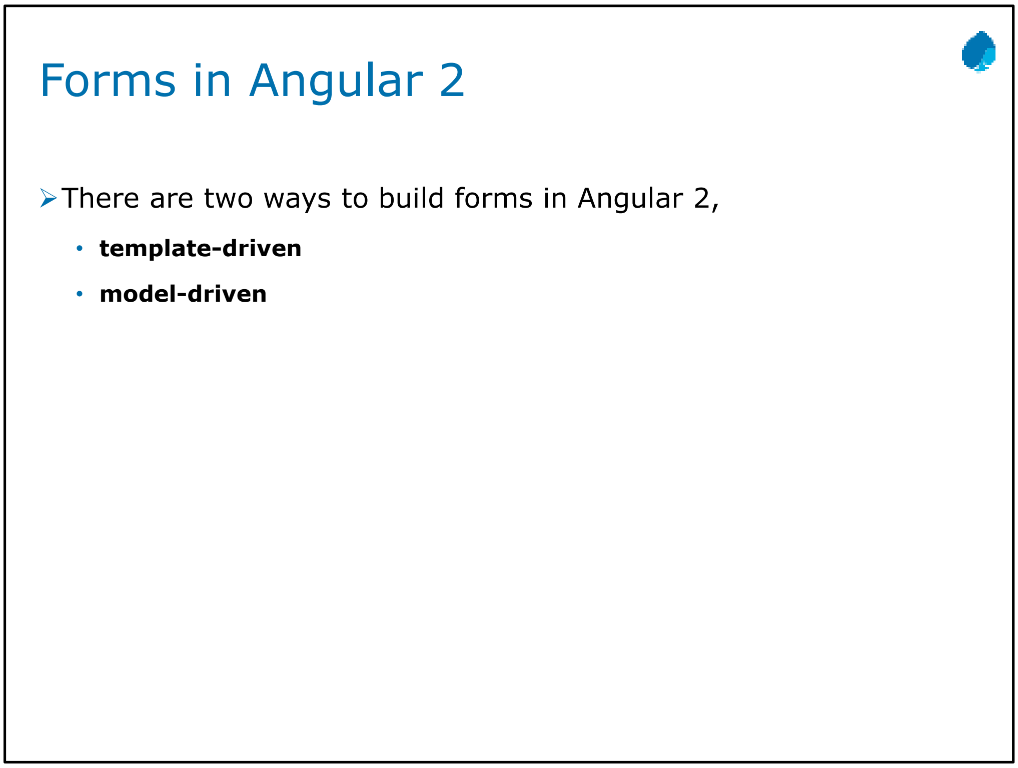


Forms in Angular 2

- Forms are the mainstay of business applications.
- The user is able to enter data by using input elements of forms.
- An Angular form coordinates a set of data-bound user controls, tracks changes, validates input, and presents errors using the following tools
  - **FormControls**: Encapsulate the inputs in forms and give the objects to work with them
  - **Validators**: Gives the ability to validate inputs
  - **Observers**: watch form for changes and respond accordingly
- To use the Forms library we need to import *FormsModule* or using *ReactiveFormsModule in app.ts*.
  - FormsModule gives template driven directives such as ngModel and NgForm
  - ReactiveFormsModule gives directives like formControl and ngFormGroup

Forms are the cornerstone of any real app. In Angular 2, forms have changed quite a bit from their v1 counterpart.
Where we used to use ngModel and map to our internal data model, in Angular 2 we more explicitly build forms and form controls.
Import FormsModule
To be able to use Angular 2 forms the FormsModule needs to be imported in our application. To make it available application-wide we’re importing in *app.module.ts*:
import { FormsModule } from '@angular/forms';

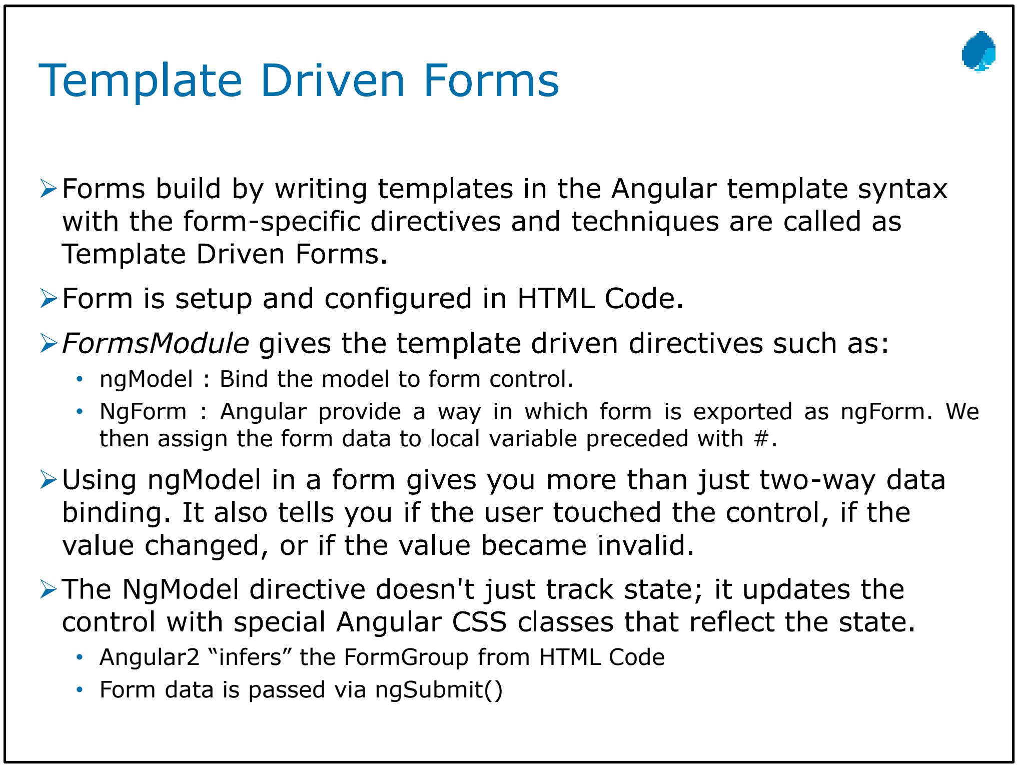
Instructor Notes:



Forms in Angular 2

- There are two ways to build forms in Angular 2,
  - **template-driven**
  - **model-driven**

Instructor Notes:



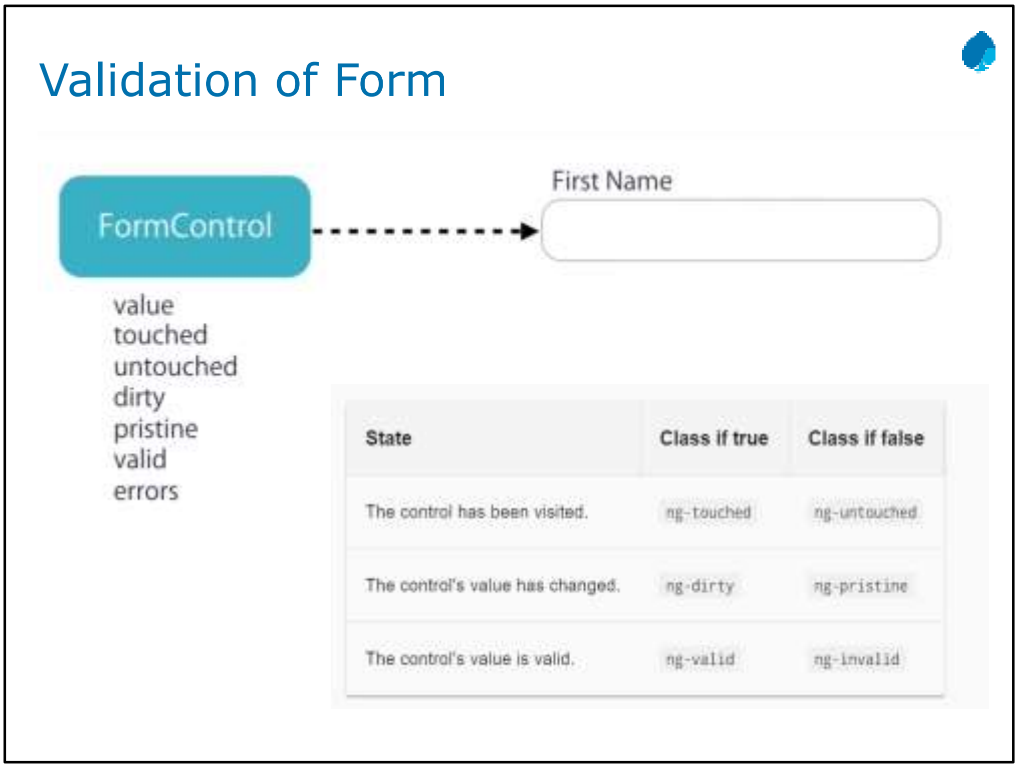
Template Driven Forms

- Forms build by writing templates in the Angular template syntax with the form-specific directives and techniques are called as Template Driven Forms.
- Form is setup and configured in HTML Code.
- *FormsModule* gives the template driven directives such as:
  - ngModel : Bind the model to form control.
  - NgForm : Angular provide a way in which form is exported as ngForm. We then assign the form data to local variable preceded with #.
- 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()

The user should be able to submit this form after filling it in. The *Submit* button at the bottom of the form does nothing on its own, but it will trigger a form submit because of its type (type="submit").
A "form submit" is useless at the moment. To make it useful, bind the form's ngSubmit event property to the hero form component's onSubmit() method:
src/app/hero-form/hero-form.component.html (ngSubmit)content_copy<form (ngSubmit)="onSubmit()" #heroForm="ngForm">

The <input> element carries the HTML validation attributes: required and minlength. It also carries a custom validator directive, forbiddenName. For more information, see Custom validatorssection.
#name="ngModel" exports NgModel into a local variable called name. NgModel mirrors many of the properties of its underlying FormControl instance, so you can use this in the template to check for control states such as valid and dirty. For a full list of control properties, see the AbstractControl API reference.
The *ngIf on the <div> element reveals a set of nested message divs but only if the nameis invalid and the control is either dirty or touched.
Each nested <div> can present a custom message for one of the possible validation errors. There are messages for required, minlength, and forbiddenName.

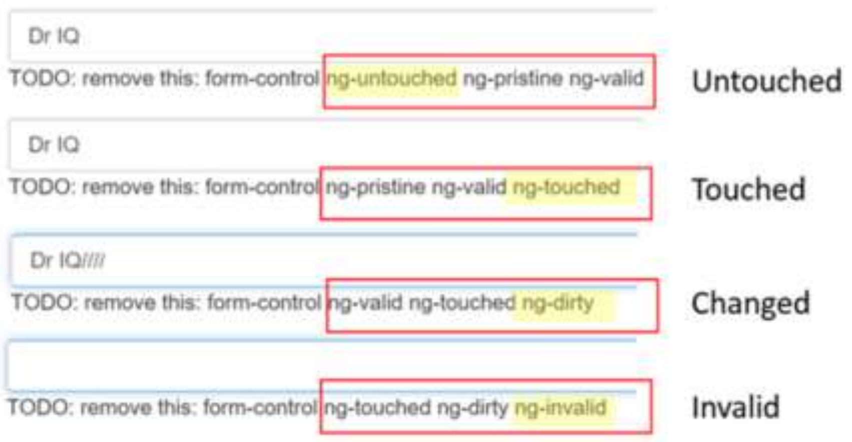
Instructor Notes:



Validation of Form

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

The ng-valid/ng-invalid pair is the most interesting, because you want to send a strong visual signal when the values are invalid.
hide the message when the control is valid or pristine; "pristine" means the user hasn't changed the value since it was displayed in this form.
This user experience is the developer's choice. Some developers want the message to display at all times. If you ignore the pristine state, you would hide the message only when the value is valid. ISome developers want the message to display only when the user makes an invalid change. Hiding the message while the control is "pristine" achieves that goal. You'll see the significance of this choice when you add a new hero to the form.
Required - The form field is mandatory
Maxlength - Maximum number of characters in the field.
Minlength - Minimum number of characters in the field.
Pattern - Regular expression to match the input values and validate.
Custom Validator - Writing custom validation function to match password and confirm password fields.



Dr IQ
TODO: remove this: form-control ng-untouched ng-pristine ng-valid
Untouched

Dr IQ
TODO: remove this: form-control ng-pristine ng-valid ng-touched
Touched

Dr IQ////
TODO: remove this: form-control ng-valid ng-touched ng-dirty
Changed

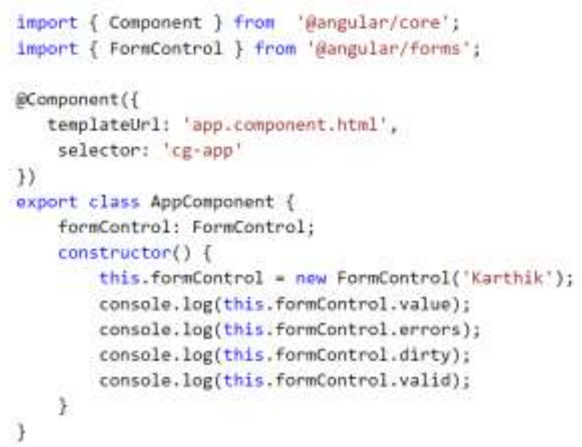
TODO: remove this: form-control ng-touched ng-dirty ng-invalid
Invalid

Instructor Notes:

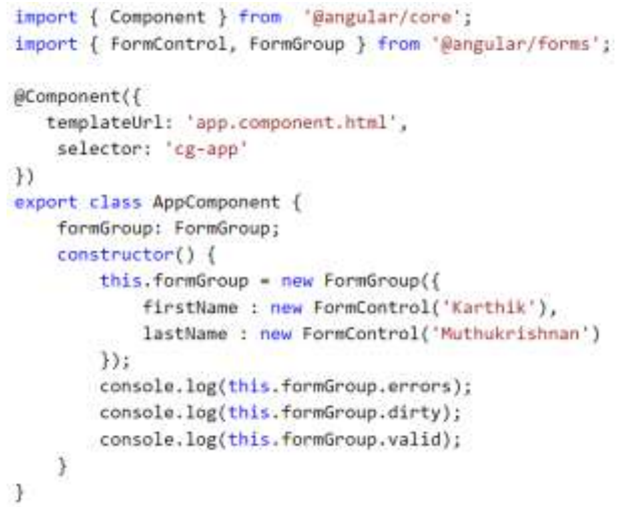
Add instructor notes here.

Validation of Form

- Form Validation is used to improve overall data quality by validating user input for accuracy and completeness.
- Form element carries the HTML validation attributes like required, minlength, maxlength and pattern. Angular interprets those and adds validator functions to the control model.
- Angular exposes information about the state of the controls including whether the user has "touched" the control or made changes and if the control values are valid.
- We can add multiple validators in a single field using Validators.compose
- To look up a specific validation failure use the hasError method

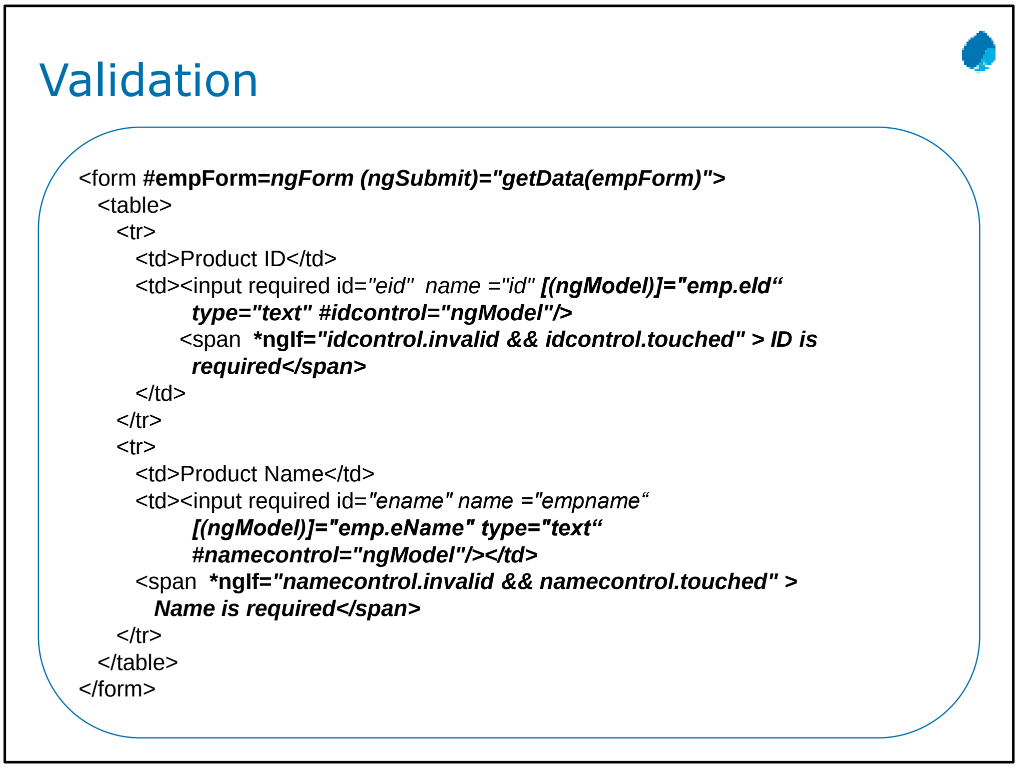


```
import { Component } from  '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
   templateUrl: 'app.component.html',
    selector: 'cg-app'
})
export class AppComponent {
    formControl: FormControl;
    constructor() {
        this.formControl = new FormControl('Karthik');
        console.log(this.formControl.value);
        console.log(this.formControl.errors);
        console.log(this.formControl.dirty);
        console.log(this.formControl.valid);
    }
}
```



```
import { Component } from  '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({
   templateUrl: 'app.component.html',
    selector: 'cg-app'
})
export class AppComponent {
    formGroup: FormGroup;
    constructor() {
        this.formGroup = new FormGroup({
            firstName : new FormControl('Karthik'),
            lastName : new FormControl('Muthukrishnan')
        });
        console.log(this.formGroup.errors);
        console.log(this.formGroup.dirty);
        console.log(this.formGroup.valid);
    }
}
```

Instructor Notes:

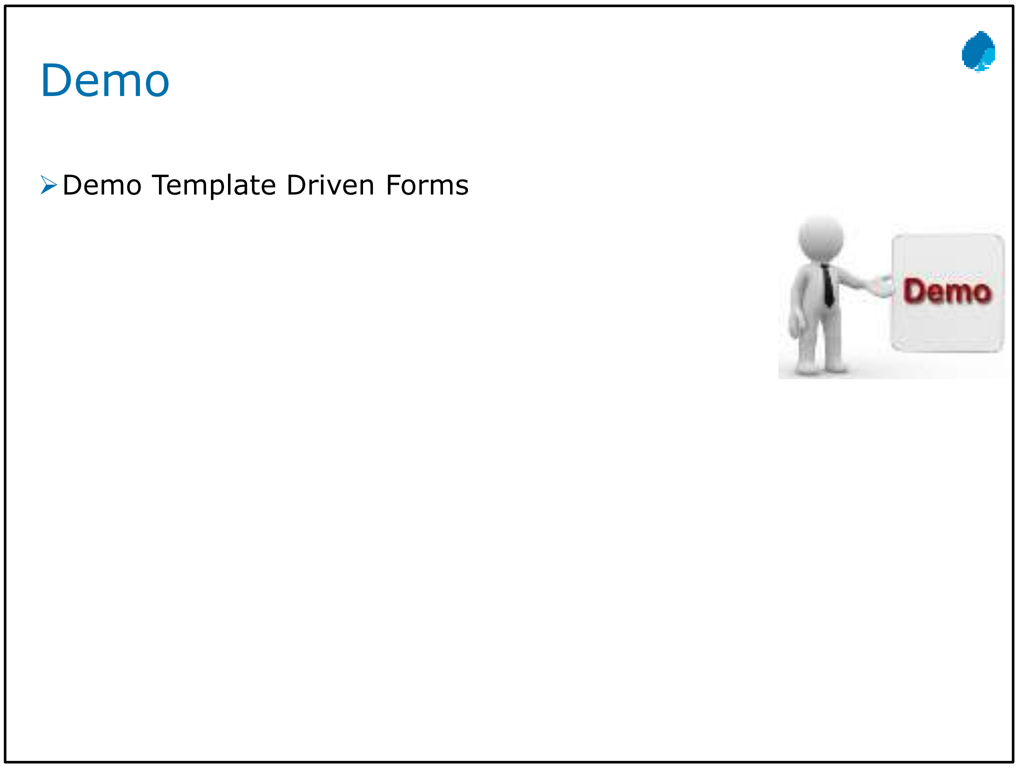


Validation

```
<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>
```

Instructor Notes:

Add instructor notes here.



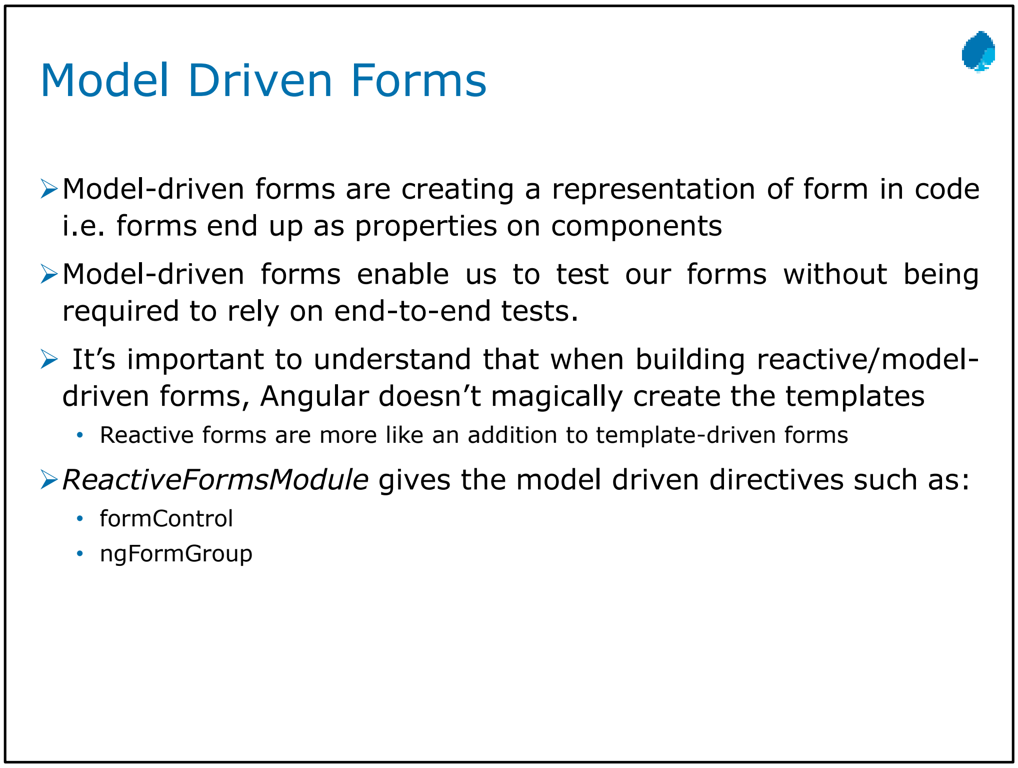
Demo

- Demo Template Driven Forms

Add the notes here.

Instructor Notes:

Add instructor notes here.

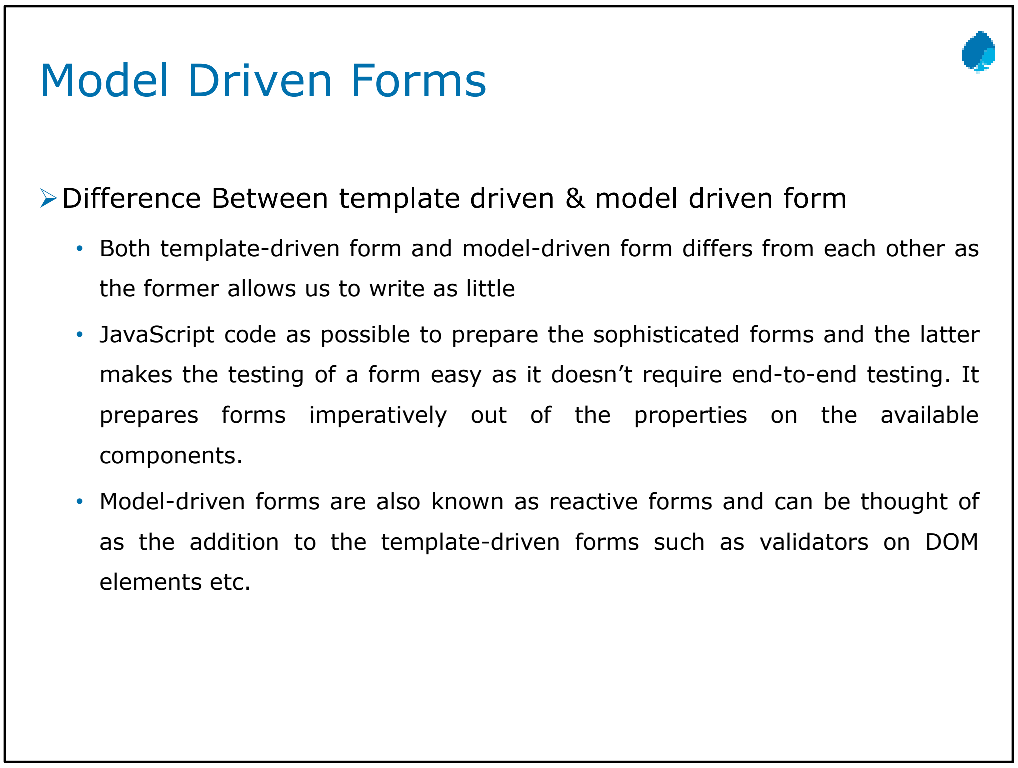


Model Driven Forms

- Model-driven forms are creating a representation of form in code i.e. forms end up as properties on components
- Model-driven forms enable us to test our forms without being required to rely on end-to-end tests.
- It's important to understand that when building reactive/model-driven forms, Angular doesn't magically create the templates
  - Reactive forms are more like an addition to template-driven forms
- *ReactiveFormsModule* gives the model driven directives such as:
  - formControl
  - ngFormGroup

Instructor Notes:

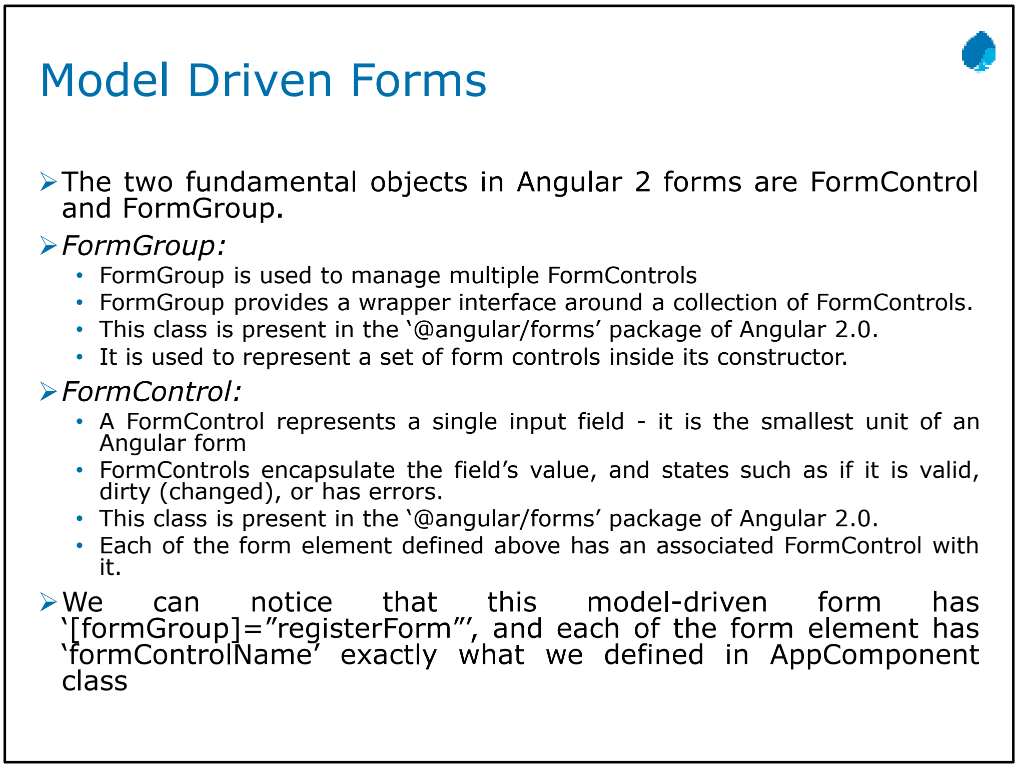
Add instructor notes here.



Model Driven Forms

- Difference Between template driven & model driven form
  - Both template-driven form and model-driven form differs from each other as the former allows us to write as little
  - JavaScript code as possible to prepare the sophisticated forms and the latter makes the testing of a form easy as it doesn’t require end-to-end testing. It prepares forms imperatively out of the properties on the available components.
  - Model-driven forms are also known as reactive forms and can be thought of as the addition to the template-driven forms such as validators on DOM elements etc.

Instructor Notes:

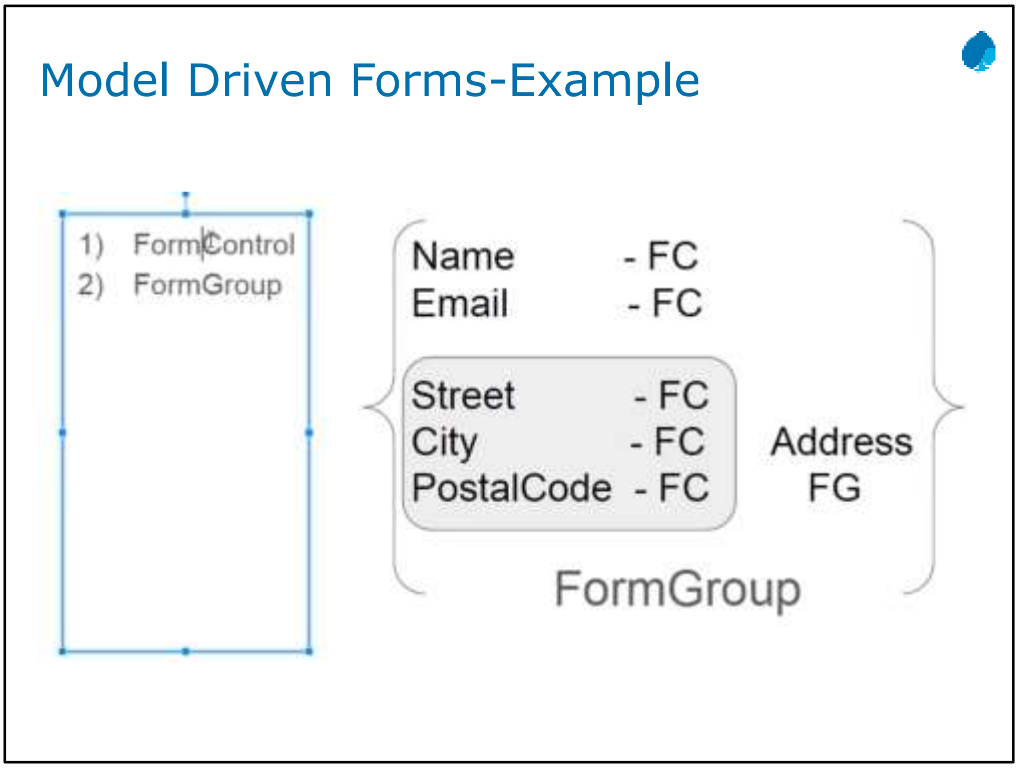


Model Driven Forms

- The two fundamental objects in Angular 2 forms are FormControl and FormGroup.
- *FormGroup:*
  - FormGroup is used to manage multiple FormControls
  - FormGroup provides a wrapper interface around a collection of FormControls.
  - This class is present in the '@angular/forms' package of Angular 2.0.
  - It is used to represent a set of form controls inside its constructor.
- *FormControl:*
  - A FormControl represents a single input field - it is the smallest unit of an Angular form
  - FormControls encapsulate the field's value, and states such as if it is valid, dirty (changed), or has errors.
  - This class is present in the '@angular/forms' package of Angular 2.0.
  - Each of the form element defined above has an associated FormControl with it.
- We can notice that this model-driven form has '[formGroup]="registerForm"', and each of the form element has 'formControlName' exactly what we defined in AppComponent class

myForm will be our model driven form. It implements FormGroup interface.
FormBuilder is not a mandatory to building model driven form, but it simplify the syntax, we'll cover this later.
A form is a type of **FormGroup**. A **FormGroup** can contain one **FormGroup** or **FormControl**

Instructor Notes:

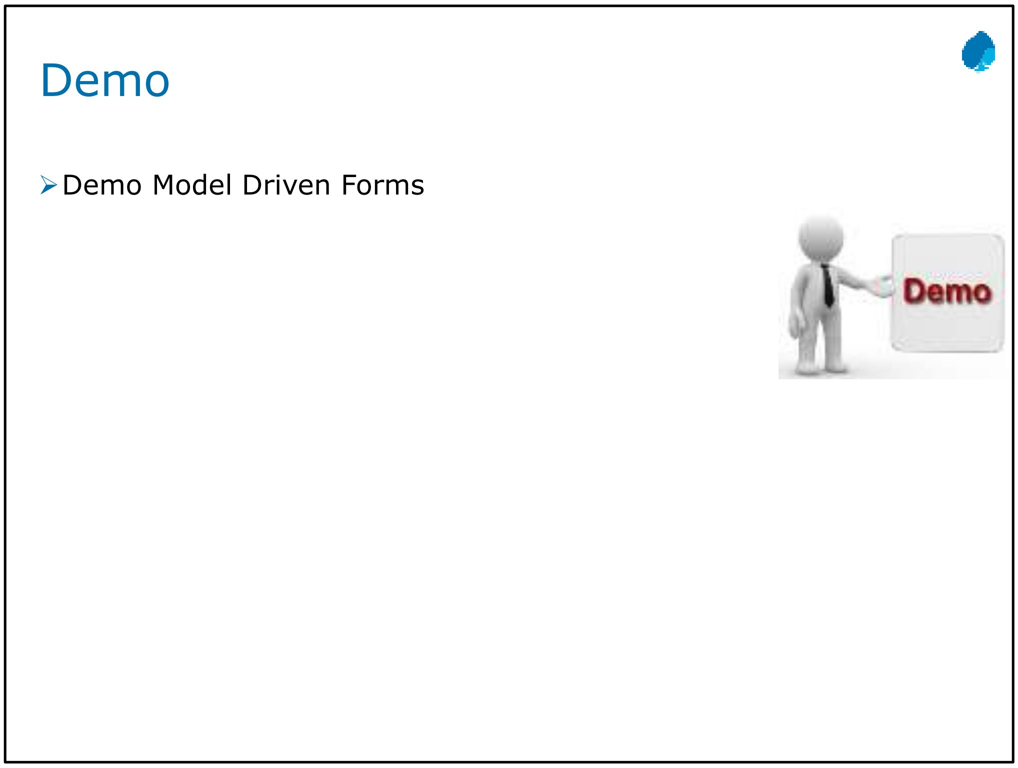


Model Driven Forms-Example

It has the selector ‘my-modal-form-app’ which is used on the index.html to load the complete app on the browser. It has the template URL as ‘resource/app-component.html’ and styleUrls as ‘assets/styles.css’ as shown below. In the controller class ‘AppComponent’, we are implementing OnInit interface for the method ‘ngOnInit ()’, where we are using the FormGroup and FormControl classes to create a model-driven form in Angular 2.0. Both FormGroup and FormControl are low level APIs where the FormGroup always represents a set of FormControls. They collectively form the model-driven form in Angular 2.0.
FormGroup: This class is present in the ‘@angular/forms’ package of Angular 2.0. It is used to represent a set of form controls inside its constructor as shown above.
FormControl: This class is present in the ‘@angular/forms’ package of Angular 2.0. Each of the form element defined above has an associated FormControl with it.
t is the HTML code template for ‘app-component-ts’. Here, we can notice that this model-driven form has ‘[formGroup]=”registerForm”’, and each of the form element has ‘formControlName’ exactly what we defined in AppComponent class. Also, for the nested FormGroup ‘address’, we have used the ‘fieldset’ tag with the ‘formGroupName’ directive

Instructor Notes:

Add instructor notes here.



Demo

- Demo Model Driven Forms

Add the notes here.

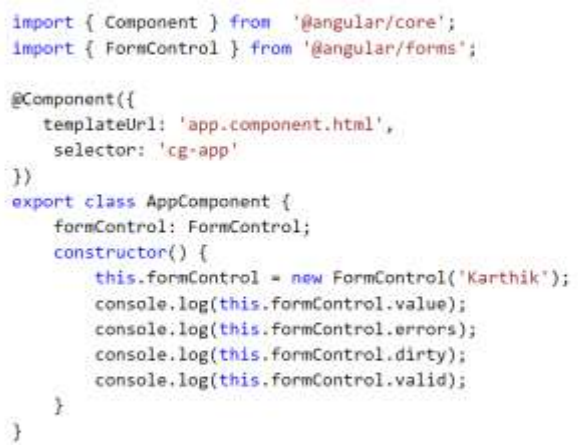
Instructor Notes:

Add instructor notes here.

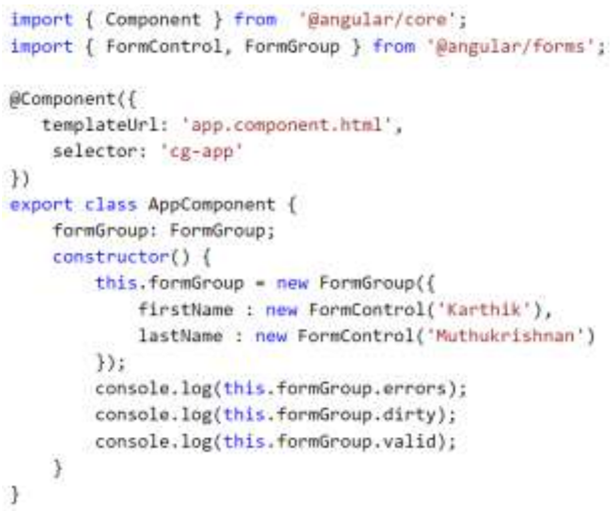
Custom Validations

- Angular allows to create custom validators as well.
- A validator: - Takes a FormControl as its input and - Returns a StringMap<string, boolean> where the key is “error code” and the value is true if it fails

```
function pinCodeValidator(control: FormControl): { [s: string]:
boolean } {
        if (!control.value.match(/^\d{6}$/)) {
                return {invalidPinCode: true};
        }
}
```



```
import { Component } from  '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
   templateUrl: 'app.component.html',
    selector: 'cg-app'
})
export class AppComponent {
    formControl: FormControl;
    constructor() {
        this.formControl = new FormControl('Karthik');
        console.log(this.formControl.value);
        console.log(this.formControl.errors);
        console.log(this.formControl.dirty);
        console.log(this.formControl.valid);
    }
}
```



```
import { Component } from  '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({
   templateUrl: 'app.component.html',
    selector: 'cg-app'
})
export class AppComponent {
    formGroup: FormGroup;
    constructor() {
        this.formGroup = new FormGroup({
            firstName : new FormControl('Karthik'),
            lastName : new FormControl('Muthukrishnan')
        });
        console.log(this.formGroup.errors);
        console.log(this.formGroup.dirty);
        console.log(this.formGroup.valid);
    }
}
```

Instructor Notes:

Add instructor notes here.

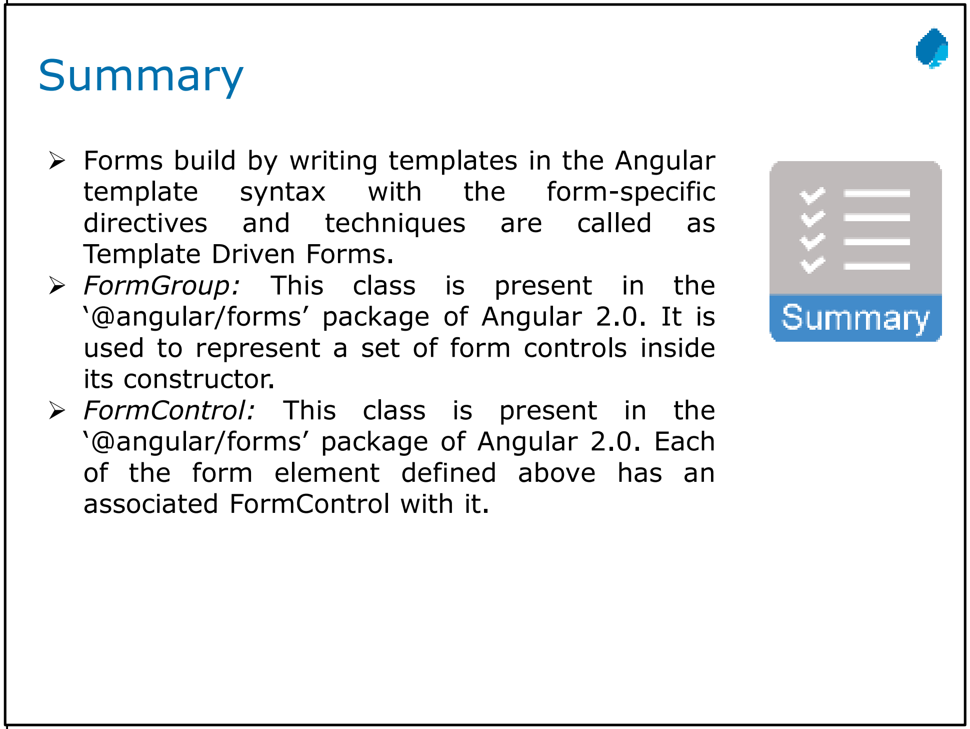
Lab

- Lab 3

Add the notes here.

Instructor Notes:

Add instructor notes here.



Summary

- Forms build by writing templates in the Angular template syntax with the form-specific directives and techniques are called as Template Driven Forms.
- *FormGroup:* This class is present in the '@angular/forms' package of Angular 2.0. It is used to represent a set of form controls inside its constructor.
- *FormControl:* This class is present in the '@angular/forms' package of Angular 2.0. Each of the form element defined above has an associated FormControl with it.

Add the notes here.