Angular Forms

.NET

Reactive forms provide a model-driven approach to handling form inputs whose values change over time.

[HTTPS://ANGULAR.IO/GUIDE/REACTIVE-FORMS](https://angular.io/guide/reactive-forms)

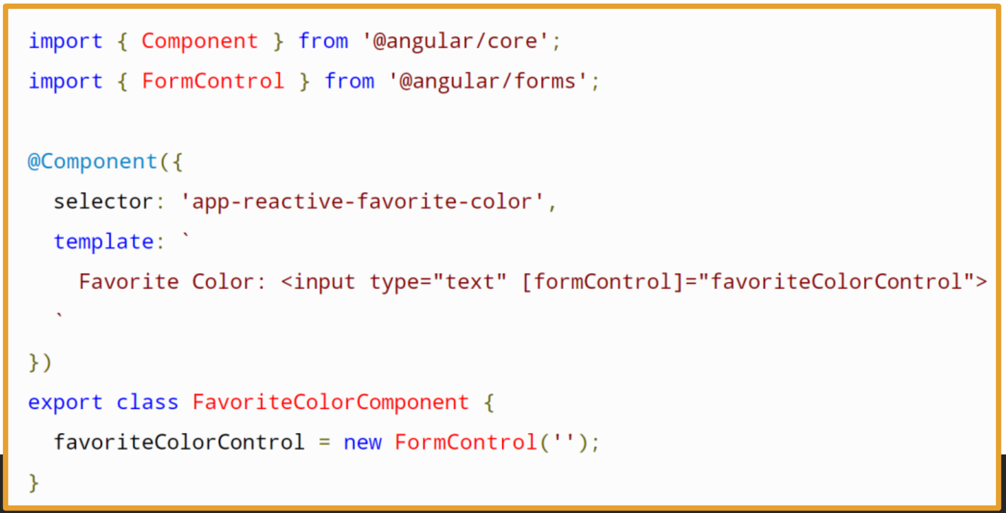
Angular Forms - Overview

https://angular.io/start/start-forms#forms-in-angular
https://angular.io/api/forms/FormBuilder
https://angular.io/guide/forms-overview

Angular provides two different form types: ***reactive*** and ***template-driven***. Both capture user input ***events*** from the view (template), validate the user input, create a ***form model*** and data model to update, and provide a way to track changes.

- Reactive forms are more robust: they're more scalable, reusable, and testable. If forms are a key part of your application, use ***Reactive Forms***.
- Template-driven forms are useful for adding a simple form to an app but don't scale as well as ***Reactive Forms***. If you have very basic form requirements and logic that can be managed solely in the template, use ***Template-Driven Forms***.

Reactive and ***Template-Driven Forms*** both use a ***form model*** to track value changes between Angular forms and form input elements.
The example shows how one of the four ***form model*** types, ***FormControl***, is defined and created.



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

@Component({
  selector: 'app-reactive-favorite-color',
  template: `
    Favorite Color: <input type="text" [formControl]="favoriteColorControl">
  `
})
export class FavoriteColorComponent {
  favoriteColorControl = new FormControl('');
}
```

Form Model Classes

https://angular.io/guide/forms-overview#testing

Reactive and ***template-driven*** forms differ in how form-control instances are created and managed.

Both form types are built using these four base classes:

Class Name	Details
FormControl	tracks the value and validation status of an individual form control.
FormGroup	tracks the values and status for a collection of form controls.
FormArray	tracks the values and status for an array of form controls.
ControlValueAccessor	creates a bridge between Angular FormControl instances and native DOM elements.

Reactive (Model-Driven) Forms

https://angular.io/start/start-forms#forms-in-angular
https://angular.io/api/forms/FormBuilder
https://angular.io/guide/forms-overview

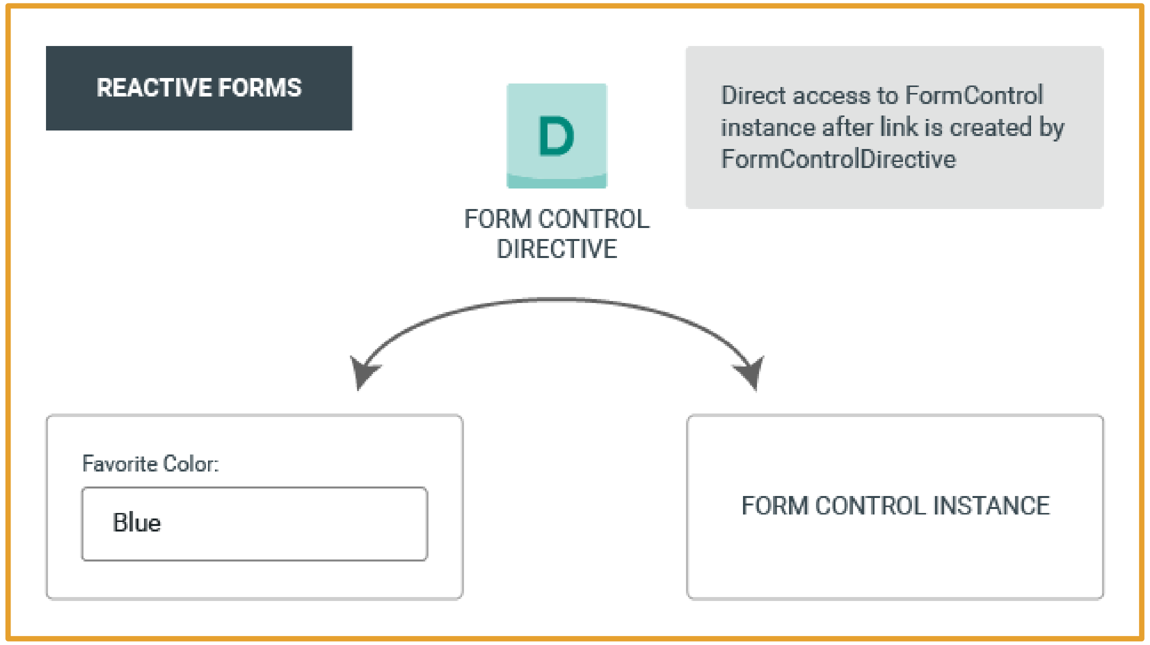
Reactive forms are built around observable streams, where form inputs and values are provided as streams of input values.

There are two parts to an ***Angular Reactive form***:

- The objects in ***Component*** to store and manage the form.
- The visualization of the form in the HTML ***template***.

The ***ReactiveFormsModule*** provides the ***FormBuilder*** service.

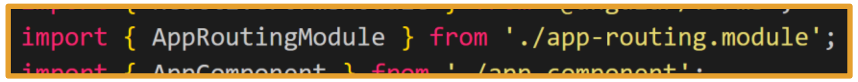
The ***form model*** is the “source of truth” and provides the value and status of the form element at a given point in time.


REACTIVE FORMS
D
FORM CONTROL DIRECTIVE
Direct access to FormControl instance after link is created by FormControlDirective
Favorite Color:
Blue
FORM CONTROL INSTANCE

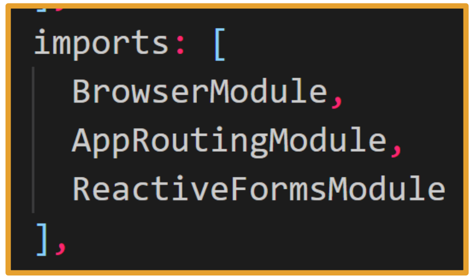
Reactive Form Setup (1/2)

https://angular.io/guide/reactive-forms#adding-a-basic-form-control
https://codecraft.tv/courses/angular/forms/model-driven/

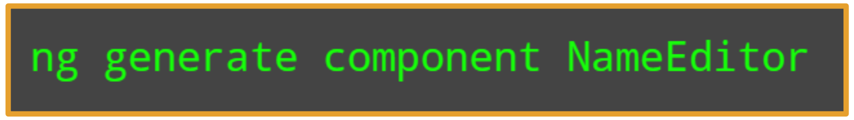
1. Import ***ReactiveFormsModule*** to ***app.module.ts*** and add it to imports[]:
   - import { ReactiveFormsModule } from '@angular/forms';
2. Generate the new component that will have the form:
   - ng generate component [ComponentName].
3. Import ***FormControl*** and ***FormGroup*** into the new component:
   - import { FormControl, FormGroup } from '@angular/forms';.
4. Create an instance of ***FormGroup*** in your component class to represent the form itself.
5. Inside NgOnInit(), set the ***FormGroup*** Instance equal to a new FormGroup() instance.
6. Set the name of each <input> element of the form equal to a new FormControl().



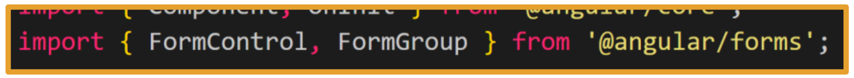
```
import { AppRoutingModule } from './app-routing.module';
```



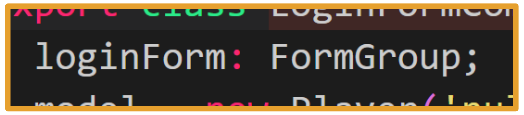
```
imports: [
  BrowserModule,
  AppRoutingModule,
  ReactiveFormsModule
],
```



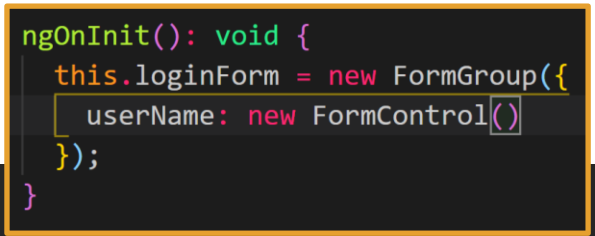
```
ng generate component NameEditor
```



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



```
loginForm: FormGroup;
```



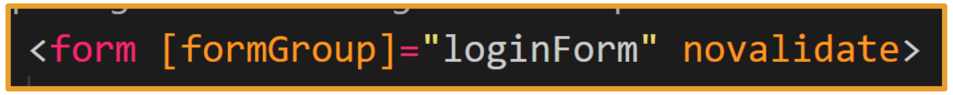
```
ngOnInit(): void {
  this.loginForm = new FormGroup({
    userName: new FormControl()
  });
}
```

Reactive Form Setup (2/2)

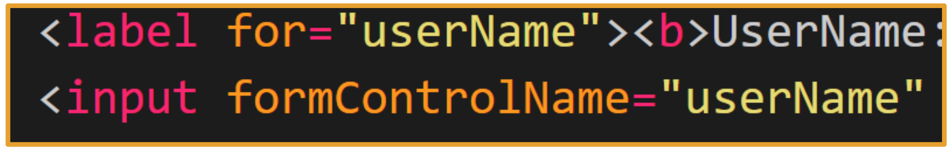
https://angular.io/guide/reactive-forms#adding-a-basic-form-control
https://codecraft.tv/courses/angular/forms/model-driven/

7. In the HTML template, add the *FormControl* to the form with:
   - [formGroup] ="loginForm"
8. Add the ***formControlName*** to each <input> of the form with:
   - [formControlName]="inputName"
9. If needed, add the component selector name to any parent ***Component*** template.
   - <app-name-editor></app-name-editor>

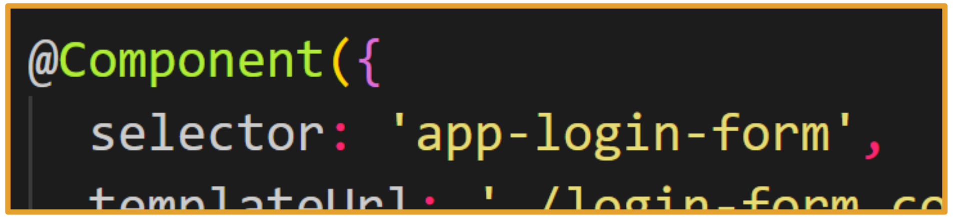
Now, whatever input you place in the input field will be transferred to the ***FormGroup's FormControls*** on the component.



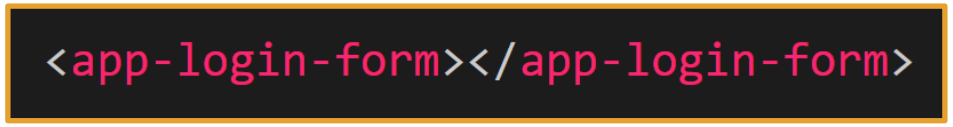
```
<form [formGroup]="loginForm" novalidate>
```



```
<label for="userName"><b>UserName:
<input formControlName="userName"
```



```
@Component({
  selector: 'app-login-form',
  templateUrl: './login-form.co
```



```
<app-login-form></app-login-form>
```

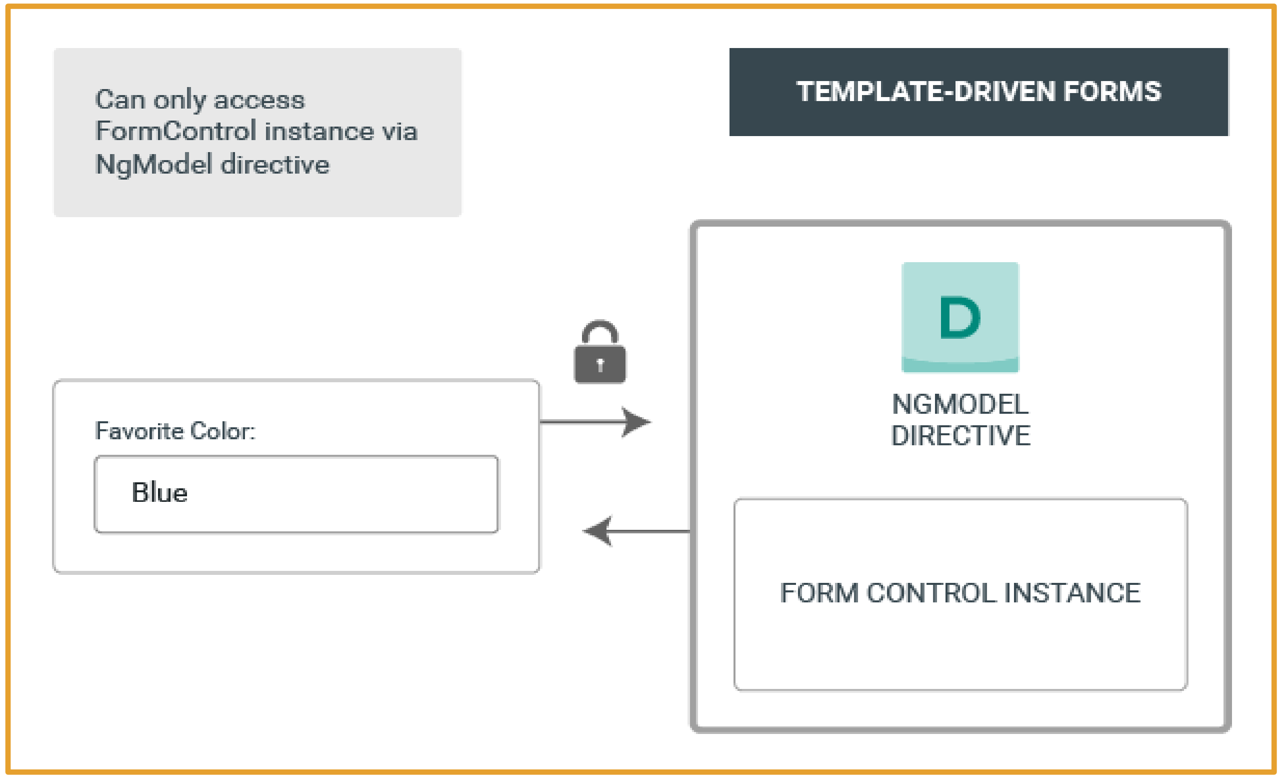
Template Driven Forms

https://angular.io/start/start-forms#forms-in-angular
https://angular.io/api/forms/FormBuilder
https://angular.io/guide/forms-overview

You can build almost any form with an Angular template.

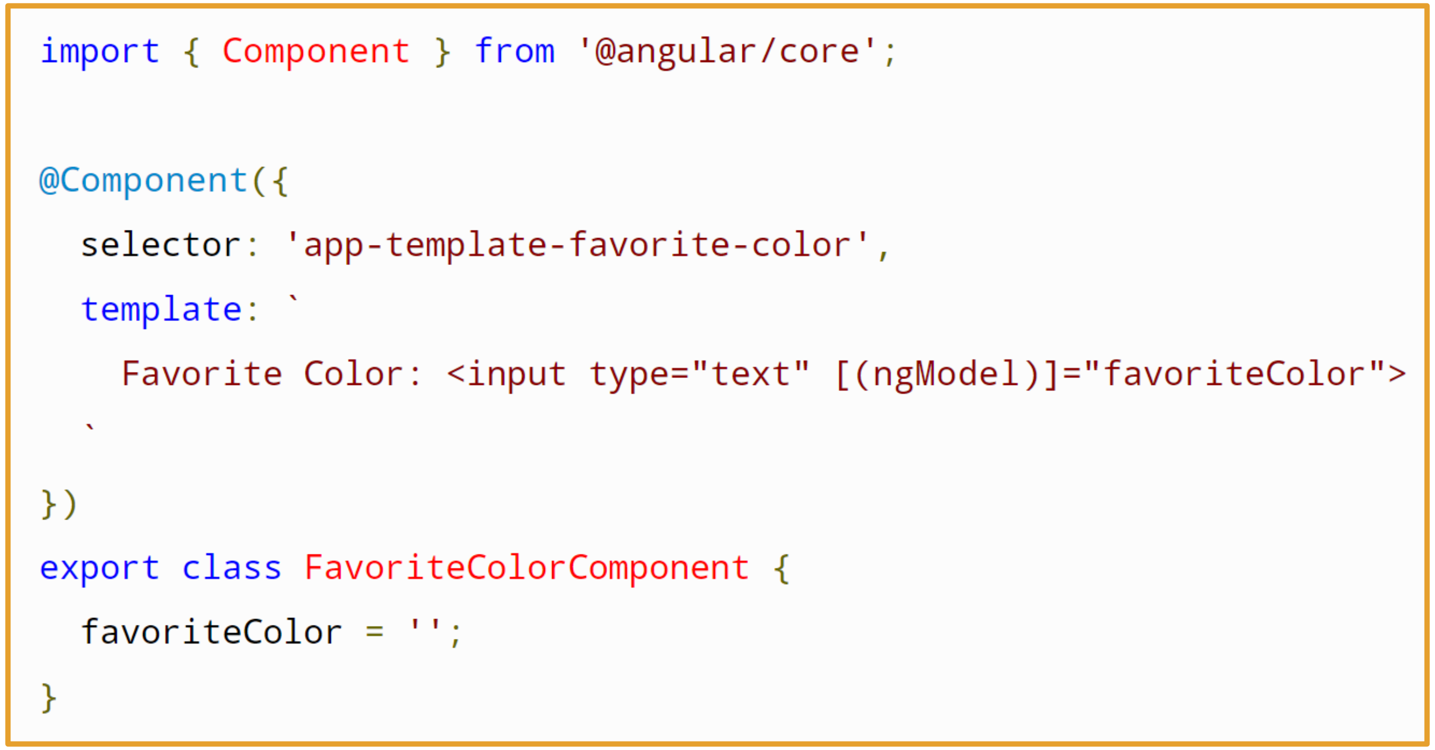
You can lay out the controls creatively, bind them to data, specify validation rules, and display validation errors.

Angular makes the process easy by handling many of the repetitive, boilerplate tasks you'd otherwise code yourself.



Template Driven Forms

https://angular.io/start/start-forms#forms-in-angular
https://angular.io/api/forms/FormBuilder
https://angular.io/guide/forms-overview



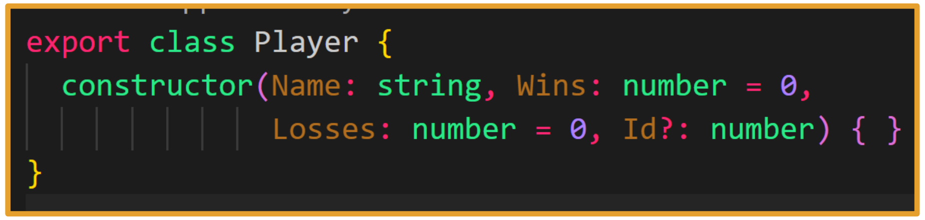
```
import { Component } from '@angular/core';


@Component({
  selector: 'app-template-favorite-color',
  template: `
    Favorite Color: <input type="text" [(ngModel)]="favoriteColor">
  `
})
export class FavoriteColorComponent {
  favoriteColor = '';
}
```

Template Driven Forms Setup (1/2)

https://angular.io/guide/forms#introduction-to-template-driven-forms

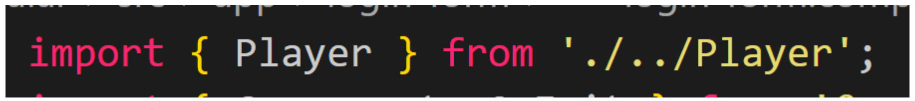
1. Make sure ***NgModule*** has been imported into app.module.ts:
   - import { NgModule } from '@angular/core';
2. Create a new class with all the fields that this form will help populate:
   - ng generate class <className>.
3. Create a new component with:
   - ng generate component <componentName>
4. import the Class (from step 2).
5. Create an instance of the class in the component.

```
ng generate class Hero
```

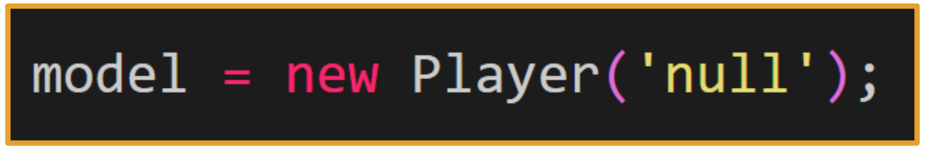


```
export class Player {
  constructor(Name: string, Wins: number = 0,
              Losses: number = 0, Id?: number) { }
}
```

```
ng generate component HeroForm
```



```
import { Player } from './../Player';
```

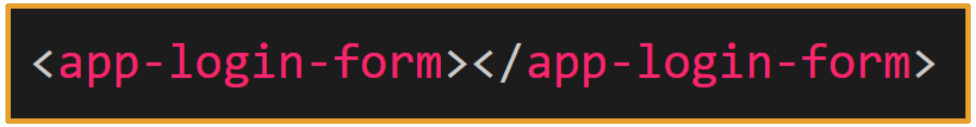


```
model = new Player('null');
```

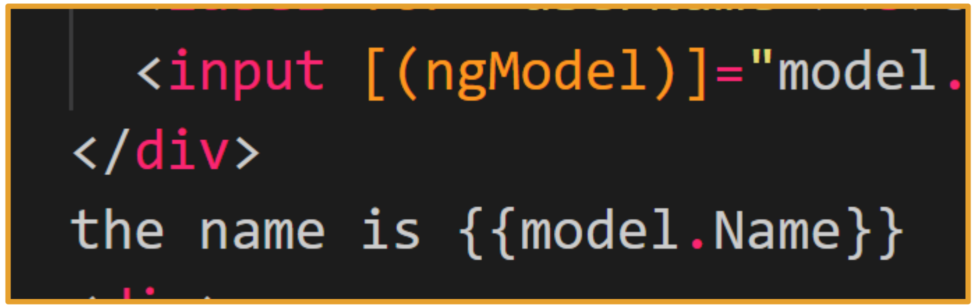
Template Driven Forms Setup (2/2)

https://angular.io/guide/forms#introduction-to-template-driven-forms

6. Add @ngModel to the forms <input> elements that correspond to the fields in the Component model (step 4) with:
   - [(ngModel)]=model.Name"
7. If needed, add the ***component selector*** to a parent view template (.html)
8. Add a check value below your input element text box to see what you are entering live.
9. Enter values into the text box to see the model field value change.



```
<app-login-form></app-login-form>
```



```
    <input [(ngModel)]="model.
</div>
the name is {{model.Name}}
```

Template Reference Variables

https://angular.io/guide/template-reference-variables

The #myName in any element in the .html file.

```
<input #phone placeholder="phone number" />

<!-- phone refers to the input element; pass its `value` to an event handler -->
<button (click)="callPhone(phone.value)">
    Call your muhthah! She misses you!
</button>
```