

ANGULAR 2

Angular v2.2.1

[Home](#)

[Why Angular2?](#)

[Components](#)

[Inputs](#)

[Outputs](#)

[Lifecycle](#)

[Templates](#)

[Events](#)

[Forms](#)

[ViewChild](#)

ES6/TypeScript

FORMS

[Suggest improvements](#)

[Tweet](#)

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.

While it feels like more code to write, in practice it's easier to reason about than with v1, easier to unit test, and we no longer have to deal with frustrating ngModel and scope data problems.

SIMPLE FORM

Let's start with a simple login form in HTML with Angular 2.

APP.MODULE.TS

```
import { BrowserModule } from '@angular/platform-browser';
```

[Wat?](#)

[Variables](#)

[Classes](#)

[Template Strings](#)

[Arrow Functions](#)

[Promises](#)

```
import { NgModule } from '@angular/core';
// We need to import the ReactiveFormsModule and HttpModule
import { ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

LOGIN-PAGE.HTML

```
<form [formGroup]="loginForm" (ngSubmit)="doLogin($event)"
```

```
<input formControlName="email" type="email" placeholder="Email" />
<input formControlName="password" type="password" placeholder="Password" />
<button type="submit">Log in</button>
</form>
```

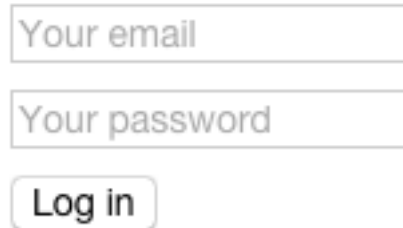
APP.COMPONENT.TS

```
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';

@Component({
  selector: 'login-page',
  templateUrl: 'login-page.html'
})
export class LoginPage {
  public loginForm = this.fb.group({
    email: ['', Validators.required],
    password: ['', Validators.required]
  });
  constructor(public fb: FormBuilder) {}
  doLogin(event) {
    console.log(event);
    console.log(this.loginForm.value);
  }
}
```

```
}
```

When we run this, we are shown a simple login form with email and password:



The image shows a simple login form. It consists of two text input fields stacked vertically. The first field is labeled "Your email" and the second field is labeled "Your password". Below these fields is a button labeled "Log in".

FORMBUILDER

The FormBuilder from the example above makes it easy for us to specify form controls and the various validators we might want to apply to certain controls.

In the example above, we are creating two inputs, an `email` and `password` field:

```
public loginForm = this.fb.group({  
  email: ['', Validators.required],
```

```
password: ['', Validators.required],
});
```

CONTROLGROUP

The `FormBuilder` creates instances of `FormGroup`, which we refer to as a `form`.

Instead of using the `FormBuilder`, we could also construct the `FormGroup` manually:

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

...

public loginForm = new FormGroup({
  email: new FormControl("email", Validators.required),
  password: new FormControl("password", Validators.required),
});
```

In practice though, the `FormBuilder` is what we will use to quickly create forms.

FORM DIRECTIVES

You'll notice the lack of `ngModel` anywhere in our form. Instead, we have the `formControlName` directives that map certain inputs to our control objects:

```
<input formControlName="email" type="email" placeholder=
```

This “binds” the email input to the instance of our `email` control.

CUSTOM VALIDATORS

We can build custom form validators as a simple function:

```
function containsMagicWord(c: FormControl) {  
  if(c.value.indexOf('magic') >= 0) {  
    console.log('not valid')  
    return {  
      noMagic: true  
    }  
  }  
}
```

```
// Null means valid, believe it or not
console.log('valid')
return null
}

this.loginForm = fb.group({
  email: ['', containsMagicWord]
  password: ['', Validators.required],
});
```

HANDLING FORM VALUES

We can easily get the simple Typescript object value of our form, or the value of an individual control:

```
doLogin(event) {
  // Show the value of the form
  let formData = this.loginForm.value;
  // { email: 'blah@blah.net', password: 'imnottelling1' }

  // Or, grab the value of one control:
  let email = this.loginForm.controls.email.value
```

```
}
```

Built by the [Ionic](#) Team. Licensed under Apache 2.