**Angular 8 Forms example**

Angular provides two different approaches for managing the user input through the forms.

1. Reactive approach
2. Template-driven approach

Both reactive and template-driven forms share underlying standard building blocks, which are the following.

1. [FormControl](https://angular.io/api/forms/FormControl): It tracks the value and validation status of the individual form control.
2. [FormGroup](https://angular.io/api/forms/FormGroup): It tracks the same values and status for the collection of form controls.
3. [FormArray](https://angular.io/api/forms/FormArray):It tracks the same values and status for the array of the form controls.
4. [ControlValueAccessor](https://angular.io/api/forms/ControlValueAccessor): It creates a bridge between Angular FormControl instances and native DOM elements.

Okay, now, let’s start with Angular forms types.

## Angular 8  Reactive Forms

Reactive forms or Model-driven forms are more robust, scalable, reusable, and testable. If forms are the key part of your application, or you’re already using reactive patterns for building your web application, use reactive forms.

In Reactive Forms, most of the work is done in the component class.

## Angular 8 Template-driven Forms

Template-driven forms are useful for adding the simple form to an app, such as the email list signup form. They’re easy to add to a web app, but they don’t scale as well as the reactive forms.

If you have the requirement of basic form and logic that can be managed solely in the template, use template-driven forms.

In template-driven forms, most of the work is done in the template.

## #Example of Template-driven form in Angular

Okay, now let’s install the fresh Angular application.

Then go to an app.module.ts file and add the following code.

// app.module.ts

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

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms';

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

In the above file, we have imported the FormsModule from the @angular/forms library.

Now, we can use the FormsModule inside our app.component.ts file.

*// app.component.ts*

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

constructor() { }

ngOnInit() {

}

onClickSubmit(formData) {

alert('Your Email is : ' + formData.email);

}

}

In this file, we have defined on the onClickSubmit function, which will trigger when the user clicks the submit button on the form.

The function also accepts one argument called formData, which is the object of the entire form values.

Through that formData object, we can access all the form field values. In our case, I only need the user’s email id.

In the general scenario, we want to pass that object to the [POST request](https://appdividend.com/2018/08/22/express-post-request-example-tutorial/) to the backend server to store the user’s data.

Now, the only remaining thing is to write the HTML of the app component. So, write the following code inside the app.component.html file.

<form #login = "ngForm" (ngSubmit) = "onClickSubmit(login.value)" >

<input type = "text" name = "email" placeholder = "email" ngModel>

<br/>

<input type = "password" name = "pwd" placeholder = "password" ngModel>

<br/>

<input type = "submit" value = "submit">

</form>

We have created the web form with input tags having email, password, and the submit button. We have assigned a type, name, and placeholder attributes to it.

In the template-driven forms, we need to create a model form controls by adding the [ngModel](https://appdividend.com/2018/10/03/angular-ngmodel-directive-example-tutorial/" \t "_blank) directive and the **name** attribute.

Thus, whenever we want Angular to access our data from forms, add ngModel to that tag, as shown above. Now, if we have to read an email and password, we need to add a [ngModel](https://appdividend.com/2018/10/03/angular-ngmodel-directive-example-tutorial/" \t "_blank) across it.

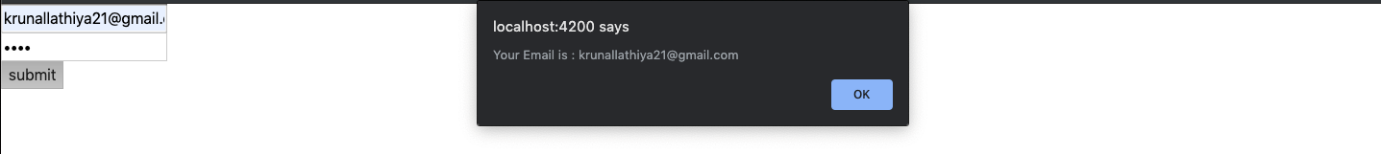
If you closely look at the form, we have also added the ngForm to the **#login**. The **ngForm**directive needs to be added to a form template that we have created.

At last, we have also added the function **onClickSubmit** and assigned **login.value** to it.

Save the file and start the angular development server by the following command.

ng serve -o

You will see the browser screen with the login form. Now fill the form, and you will see something like the following image.

[](https://appdividend.com/wp-content/uploads/2019/06/Angular-8-Forms-Tutorial-Angular-Forms-Example.png)

So, that is it for the Template-driven forms in Angular 8.

## #Model-driven or Reactive Forms in Angular

In the model-driven form, we need to import a ReactiveFormsModule from @angular/forms and use the same in the imports array.

There is a change that goes into **app.module.ts.**See the app.module.ts file.

// app.module.ts

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

import { NgModule } from '@angular/core';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule,

ReactiveFormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

In the **app.component.ts**file, we need to import the few modules for the model-driven form. For example, **import { FormGroup, FormBuilder } from ‘@angular/forms’**.

## #Forms controls and form groups in Angular

Form controls are classes that can hold both the data values and the validation information of any form element, which means to say a form control should bind every form input you have in a reactive form. They are the basic units that make up reactive forms.

Form groups are constructs that wrap a collection of form controls. Just as the control gives you access to the state of an element, the group gives the same access, but to the state of the wrapped controls. Every single form control in the form group is identified by name when initializing.

## #Reactive Forms Validation App Component

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 angForm property.

The angForm is then bound to the form in the app template below using the [formGroup] directive.

You can also add the getter function as a convenience property to make it easier to access form controls from the template. I have not added, but you can add that.

See the following app.component.ts file.

*// app.component.ts*

import { Component, OnInit } from '@angular/core';

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

angForm: FormGroup;

constructor(private fb: FormBuilder) {

this.createForm();

}

createForm() {

this.angForm = this.fb.group({

email: [''],

password: ['']

});

}

onClickSubmit(email, password) {

alert('Your Email is : ' + email);

}

ngOnInit() {

}

}

In the above code, we have used a FormBuilder to create a form and initialize the value of two fields, which is empty. To register the form builder in a component, the first thing to do is import it from Angular forms.

The next step is to inject the form builder service, which is an injectable provider that comes with the reactive forms module. You can then use the form builder after injecting it.

We have also defined one function called onClickSubmit(), which accepts the two-parameter email and password.

Form builders not only help to make your reactive forms’ code efficient, but they are also important for form validation.

So,  when the user pressed the submit button, we got the values here, and then we send these values with the POST request to the backend server.

## #Reactive Forms Validation App Template

The app component template contains all the HTML markup for displaying the example registration form in your browser.

The form element uses the [formGroup] directive to bind to the registerForm FormGroup in the app component above.

We have defined the submit button click event on the button markup.

When the user clicks on the submit button, the onClickSubmit() function will be called from the app component with the value of email and password as parameters to the function.

You can also define the form submit event to the onSubmit() handler in the app component using the Angular event binding (ngSubmit)=”onSubmit()”.

But we are not going for this approach. We are going for the button click event and not form submit event.

Validation messages are displayed only after the user attempts to submit the form for the first time, and this is controlled with the submitted property of the app component.

Finally, write the app.component.html file.

<form [formGroup]="angForm">

<input type = "text"

name = "email"

placeholder = "email"

formControlName="email"

*#email>*

<br/>

<input type = "password"

name = "pwd"

placeholder = "password"

formControlName="password"

*#password>*

<br/>

<input type = "submit"

value = "submit"

(click) = "onClickSubmit(email.value, password.value)" >

</form>

Now, save the above file and go to the Angular app, and you will see the same output as we have got in the template-driven approach.

One of the advantages of the Model-driven approach is that the Validations become very easy.