Below is an example of an Angular model-driven (reactive) form to create a login form with a username and password. In this example, we'll use Angular's Reactive Forms module.

Step 1

If you haven't already set up an Angular project, you can create a new one by following these steps:

1. Open your terminal and run:

bash

Copy code

ng new angular-reactive-forms

1. Go into the project folder:

bash

Copy code

cd angular-reactive-forms

1. Add the ReactiveFormsModule to your project by installing Angular Forms module:

bash

Copy code

ng add @angular/forms

Step 2: appcomponent.ts

First, import the ReactiveFormsModule in your app module to make reactive forms available across your app.

typescript

Copy code

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

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

import { ReactiveFormsModule } from '@angular/forms'; // Import ReactiveFormsModule

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

@NgModule({

declarations: [AppComponent],

imports: [BrowserModule, ReactiveFormsModule], // Add ReactiveFormsModule here

providers: [],

bootstrap: [AppComponent]

})

export class AppModule {}

**app.component.ts**

Next, define the form group in your component's TypeScript file.

typescript

Copy code

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

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

loginForm: FormGroup;

ngOnInit() {

// Initialize the form group with username and password fields

this.loginForm = new FormGroup({

username: new FormControl('', [Validators.required, Validators.minLength(4)]),

password: new FormControl('', [Validators.required, Validators.minLength(6)])

});

}

onSubmit() {

if (this.loginForm.valid) {

console.log('Form Submitted!', this.loginForm.value);

// Here you can handle the form submission like sending it to a backend

} else {

console.log('Form is not valid');

}

}

}

**app.component.html**

In the template, bind the form controls to the form inputs and handle form submission.

html

Copy code

<div class="login-container">

<form [formGroup]="loginForm" (ngSubmit)="onSubmit()">

<div class="form-group">

<label for="username">Username</label>

<input id="username" formControlName="username" type="text" class="form-control">

<div \*ngIf="loginForm.get('username')?.invalid && loginForm.get('username')?.touched" class="error">

Username is required and must be at least 4 characters long.

</div>

</div>

<div class="form-group">

<label for="password">Password</label>

<input id="password" formControlName="password" type="password" class="form-control">

<div \*ngIf="loginForm.get('password')?.invalid && loginForm.get('password')?.touched" class="error">

Password is required and must be at least 6 characters long.

</div>

</div>

<button type="submit" [disabled]="loginForm.invalid" class="btn btn-primary">Login</button>

</form>

</div>

**app.component.css**

You can style the form with some simple CSS to make it look good:

css

Copy code

.login-container {

width: 300px;

margin: 100px auto;

padding: 20px;

border: 1px solid #ccc;

border-radius: 8px;

background-color: #f9f9f9;

}

.form-group {

margin-bottom: 15px;

}

label {

font-weight: bold;

}

input.form-control {

width: 100%;

padding: 8px;

font-size: 14px;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

width: 100%;

padding: 10px;

font-size: 16px;

border: none;

border-radius: 4px;

}

button:disabled {

background-color: #ccc;

}

.error {

color: red;

font-size: 12px;

}

**Step 3: Run the Application**

Now, if you run the Angular application:

bash

Copy code

ng serve

You should be able to see a simple login form with validation. The form will only submit if both fields (username and password) are valid. The error messages will show up if the form fields are touched but not valid.

**Explanation**

* **FormControl**: This represents individual form fields like username and password.
* **Validators**: We used Validators.required to ensure the fields are not empty, and Validators.minLength to ensure they meet minimum character requirements.
* **Reactive Form Binding**: The [formGroup] directive binds the form model to the HTML template. The formControlName directive binds individual form controls to the corresponding input fields.
* **Form Validation**: Error messages are shown when the form controls are invalid and have been touched.

**Conclusion**

This setup gives you a simple, functional Angular model-driven form with validation. You can extend it further by adding more complex validation rules or connecting it to a backend for user authentication.

Top of Form

Bottom of Form