

TEMA 3

Maquetación: Angular Material

Desarrollo front-end avanzado

Máster Universitario en Desarrollo de sitios y aplicaciones web

UOC

Universitat Oberta
de Catalunya

Contenido

- Introducción
- Creación del proyecto base
- Ejemplo aplicación **Angular Material**
- Jugando con el componente **card**
- Jugando con animaciones



Introducción

Hoy en día existen diversos **frameworks** para realizar la maquetación de una aplicación web. El principal exponente de **frameworks** es el clásico **Bootstrap**, el cual ha servido de base para una gran cantidad de prototipos (y de apps en producción). El otro gran **framework** de diseño es **Material**, el cual sigue la guía de estilos de Google denominada **Material Design**.

La elección natural de **framework** de diseño en el mundo **Angular** es **Material**. Una de las principales ventajas de **Angular Material** es que su **look&feel** permite adaptarse perfectamente a aplicaciones móviles. De hecho, cuando convirtamos nuestras aplicaciones **Angular** en una aplicación móvil haciendo uso de la técnica de desarrollo **PWA (Progressive Web App)**, no tendremos que rediseñar la aplicación Web. Aparte de que el equipo de desarrollo de **Angular** está más unido a la tecnología de **Google**.

En este temario vamos a aplicarle estilos a nuestra aplicación y, para ello, vamos a utilizar **Angular Material** el cual es un paquete específico desarrollado para utilizar **Material** en **Angular**. El sistema de grid (i.e. rejilla/cuadrícula) de este **framework** está basado en **Flex Layout** y en lugar de utilizar las reglas **CSS** nativamente se fundamenta en el paquete **@angular/flex-layout** desarrollado por el propio **core** de **Angular**.

Creación del proyecto base

Lo primero que haremos será crearnos un proyecto nuevo e implementar un formulario reactivo con algunos inputs básicos. Una vez implementado esto, configuraremos **Angular Material** a nuestro proyecto y aplicaremos algunos de sus componentes a nuestro proyecto para darle un aspecto visual más profesional. ¡Vamos a ello!

A continuación, enumeramos una propuesta de paso a paso a seguir:

- **Paso 1:**

Creamos un nuevo proyecto ejecutando el siguiente comando des de la consola:

```
H:\Projectes>ng new angular-material-test
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

ng new << nombre_proyecto >>

Podemos indicar que si queremos que nos añada el **routing** de **Angular** y por ejemplo que utilizaremos el tipo de estilos **CSS**.

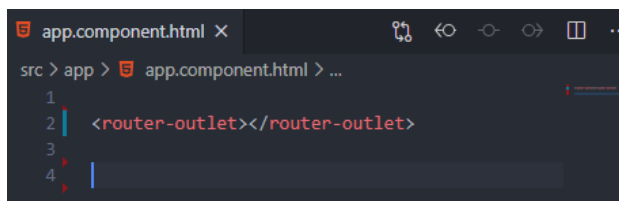
Ejecutamos el proyecto con la instrucción:

```
H:\Projectes\angular-material-test>ng serve --open
```

ng serve --open

Y validamos que vemos la pantalla inicial del proyecto en el navegador.

Veremos la pantalla inicial con mucha información de **Angular** que en principio no nos interesa, así que iremos al fichero **app.component.html** y eliminaremos todo su contenido a excepción de la línea:



```
app.component.html x
src > app > app.component.html > ...
1 <router-outlet></router-outlet>
2
3
4
```

• Paso 2:

Para empezar a trabajar con los formularios reactivos, necesitamos incluir **ReactiveFormsModule** en el módulo principal de la aplicación (**app.module.ts**):

```
app.module.ts X
src > app > app.module.ts > ...
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { ReactiveFormsModule } from '@angular/forms';
7
8 @NgModule({
9   declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule,
15     ReactiveFormsModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

• Paso 3:

Vamos a suponer que queremos implementar un **login** con email y contraseña. Por lo tanto, podemos crearnos una carpeta **Models** y dentro la clase **Credentials**. Dicha clase podría contener las siguientes propiedades:

```
credentials.ts X
src > app > Models > credentials.ts > ...
1 export class Credentials {
2   email: string;
3   password: string;
4 }
5
```

• Paso 4:

Vamos a crear un componente donde implementaremos el formulario para tratar la clase anterior. Nos crearemos una carpeta a la misma altura que la carpeta anterior **Models** y la nombraremos **Components**. Acto seguido, desde la consola podemos ejecutar el siguiente comando:

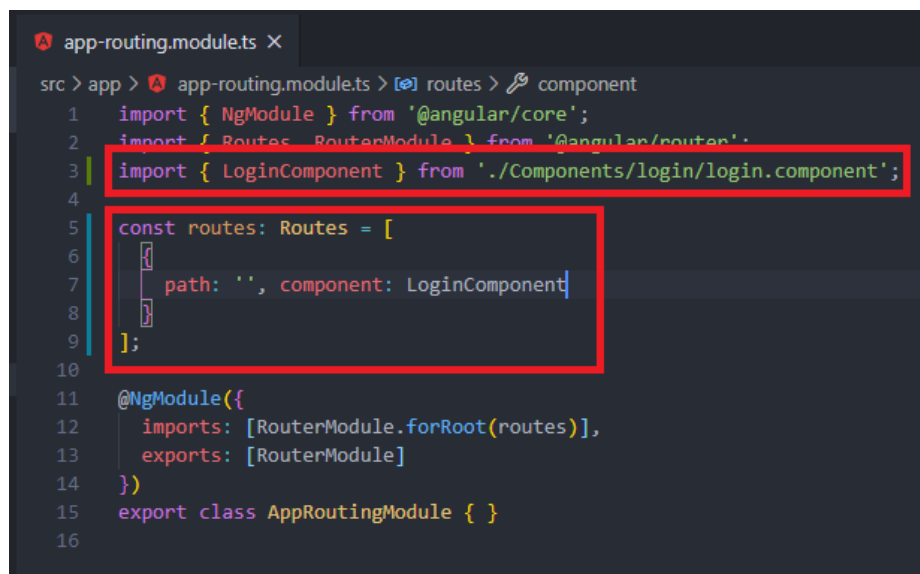
```
H:\Projectes\angular-material-test>ng g c Components/login
CREATE src/app/Components/login/login.component.html (20 bytes)
CREATE src/app/Components/login/login.component.spec.ts (619 bytes)
CREATE src/app/Components/login/login.component.ts (271 bytes)
CREATE src/app/Components/login/login.component.css (0 bytes)
UPDATE src/app/app.module.ts (1164 bytes)
```

ng g c Components/login

Al ejecutar esta instrucción podemos observar que nos ha incluido él mismo el componente **LoginComponent** dentro del apartado **declarations** del fichero **app.module.ts**. Debemos asegurarnos de que este nuevo componente esté declarado en el **app.module.ts** para que lo podamos utilizar.

- **Paso 5:**

Validemos que el nuevo componente funciona. Vamos al fichero **app-routing.module.ts** y definimos que para la ruta de entrada " " redirija al componente del login **LoginComponent**:

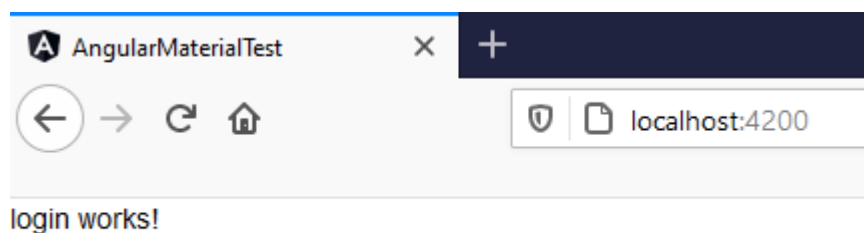


```

src > app > app-routing.module.ts > routes > component
1  import { NgModule } from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3  import { LoginComponent } from '../Components/login/login.component';
4
5  const routes: Routes = [
6    {
7      path: '', component: LoginComponent
8    }
9  ];
10
11 @NgModule({
12   imports: [RouterModule.forRoot(routes)],
13   exports: [RouterModule]
14 })
15 export class AppRoutingModule { }
16

```

Con esto, podremos ver al refrescarse nuestra aplicación algo así:



• Paso 6:

Primera versión de la implementación del controlador **login.component.ts**:

```
login.component.ts X
src > app > Components > login > login.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { FormBuilder, FormControl, FormGroup } from '@angular/forms';
3 import { Credentials } from 'src/app/Models/credentials';
4
5 @Component({
6   selector: 'app-login',
7   templateUrl: './login.component.html',
8   styleUrls: ['./login.component.css'],
9 })
10 export class LoginComponent implements OnInit {
11   public credentials: Credentials = new Credentials();
12
13   public email: FormControl;
14   public password: FormControl;
15
16   public loginForm: FormGroup;
17
18   constructor(private formBuilder: FormBuilder) {}
19
20   ngOnInit(): void {
21     this.email = new FormControl('');
22     this.password = new FormControl('');
23
24     this.loginForm = this.formBuilder.group({
25       email: this.email,
26       password: this.password,
27     });
28   }
29
30   public checkLogin(): void {
31     console.log('Email: ' + this.email.value + ' Password: ' + this.password.value);
32   }
33 }
34
```

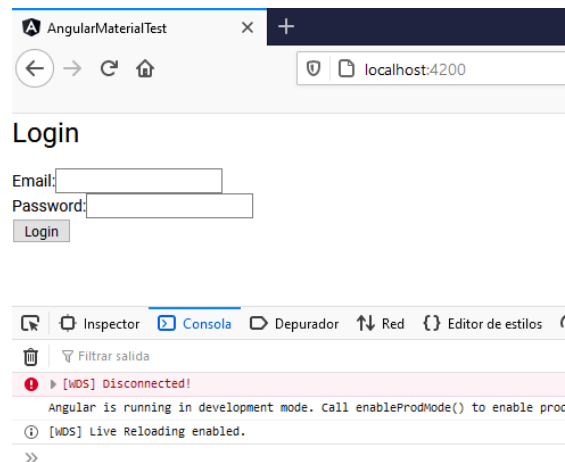
• Paso 7:

Primera versión de la implementación de la vista del componente **login.component.html**:

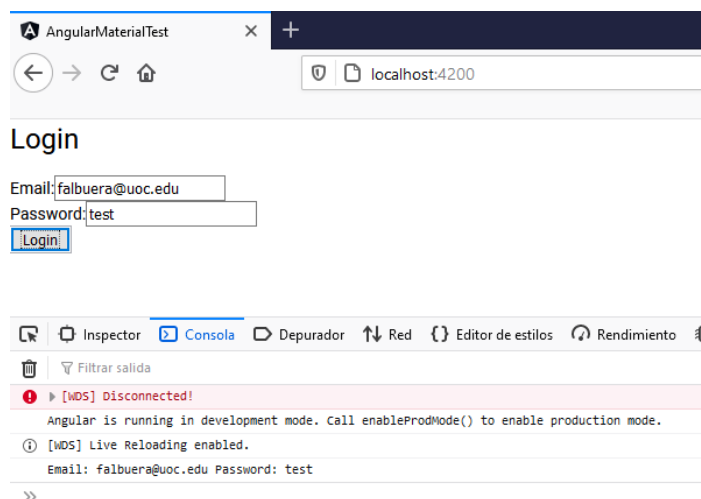
```
login.component.html X
src > app > Components > login > login.component.html > ...
1 <div>
2   <h1>Login</h1>
3   <form [formGroup]="loginForm" (ngSubmit)="checkLogin()">
4
5     <div>
6       <label for="email">Email:</label>
7       <input type="text" [formControl]="email" />
8     </div>
9
10    <div>
11      <label for="description">Password:</label>
12      <input type="text" [formControl]="password" />
13    </div>
14
15    <button type="submit" [disabled]="!loginForm.valid">Login</button>
16  </form>
17 </div>
18
```

Tal y como tenemos la implementación hasta este momento, podemos ver la ejecución en el navegador. Podemos observar que, si introducimos valores en los campos y pulsamos el botón de **Login**, se mostraran por consola los valores insertados.

Inicialmente la aplicación se carga de la siguiente manera:



Rellenamos los datos y pulsamos **Login**:



Podemos ver que mostramos los dos campos por consola correctamente.

Si quisiéramos dar algo de información al **placeholder** del campo email, podríamos hacer por ejemplo:

```
this.email = new FormControl('Ex. pat@example.com');
```

Al refrescarse el navegador veríamos el campo email inicializado con este texto, pero tendríamos que seleccionarlo, eliminarlo y escribir lo que realmente quisiéramos escribir en este campo. Posteriormente veremos el manejo del **placeholder** con **Angular Material**.

- Paso 8:

Añadiendo validaciones básicas.

Para poder trabajar con las validaciones necesitamos importar **Validators** en nuestro controlador:

```
login.component.ts X
src > app > Components > login > login.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';
3 import { Credentials } from 'src/app/Models/credentials';
4
5 @Component({
6   selector: 'app-login',
7   templateUrl: './login.component.html',
8   styleUrls: ['./login.component.css'],
9 })
10 export class LoginComponent implements OnInit {
11   public credentials: Credentials = new Credentials();
12
13   public email: FormControl;
14   public password: FormControl;
15
16   public loginForm: FormGroup;
17
18   constructor(private formBuilder: FormBuilder) {}
19
20   ngOnInit(): void {
21     this.email = new FormControl('', [Validators.required, Validators.email]);
22     this.password = new FormControl('', Validators.required);
23
24     this.loginForm = this.formBuilder.group({
25       email: this.email,
26       password: this.password,
27     });
28   }
29
30   public checkLogin(): void {
31     console.log('Email: ' + this.email.value + ' Password: ' + this.password.value);
32   }
33 }
34
```

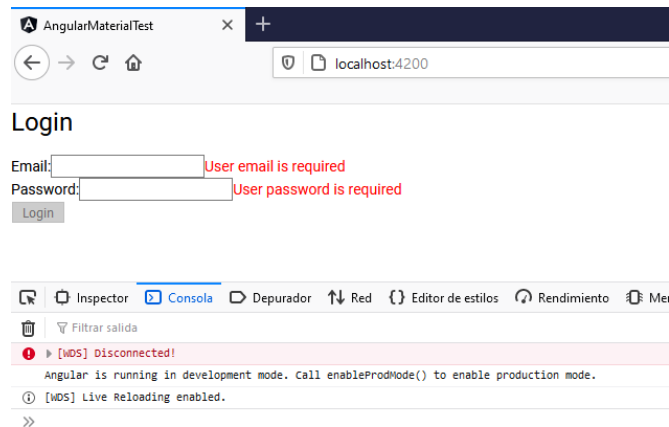
Y vamos a añadir que los campos del formulario sean obligatorios y además que el campo email tenga el formato correcto. Si ejecutamos ahora la aplicación, podremos ver que de entrada tenemos el botón de **Login** deshabilitado, y en el momento en que haya información en los campos éste se habilitará y si se pulsa mostrará la información correcta por la consola del navegador.

Ahora, vamos a darle un poco de **feedback** al usuario para que sepa porque no puede hacer clic al botón de **Login** cuando lo tenga deshabilitado.

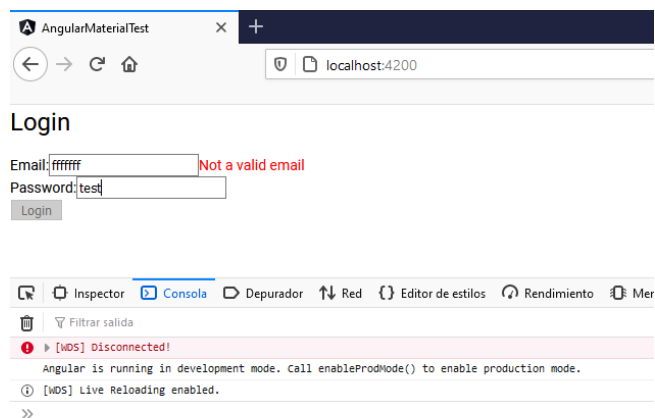
```
login.component.html X
src > app > Components > login > login.component.html > ...
1 <div>
2   <h1>Login</h1>
3   <form [formGroup]="loginForm" (ngSubmit)="checkLogin()">
4
5     <div>
6       <label for="email">Email:</label>
7       <input type="text" [formControl]="email" />
8       <span style="color: red" *ngIf="loginForm.get('email').errors?.required">User email is required</span>
9       <span style="color: red" *ngIf="loginForm.get('email').errors?.email">Not a valid email</span>
10    </div>
11
12    <div>
13      <label for="description">Password:</label>
14      <input type="text" [formControl]="password" />
15      <span style="color: red" *ngIf="loginForm.get('password').errors?.required">User password is required</span>
16    </div>
17
18    <button type="submit" [disabled]="!loginForm.valid">Login</button>
19  </form>
20 </div>
21
```


Añadimos estas líneas a la vista para que si el campo no está informado muestre un mensaje de error en rojo. También mostraremos un mensaje de error si el formato del email no es correcto.

Ahora si ejecutamos podemos ver:

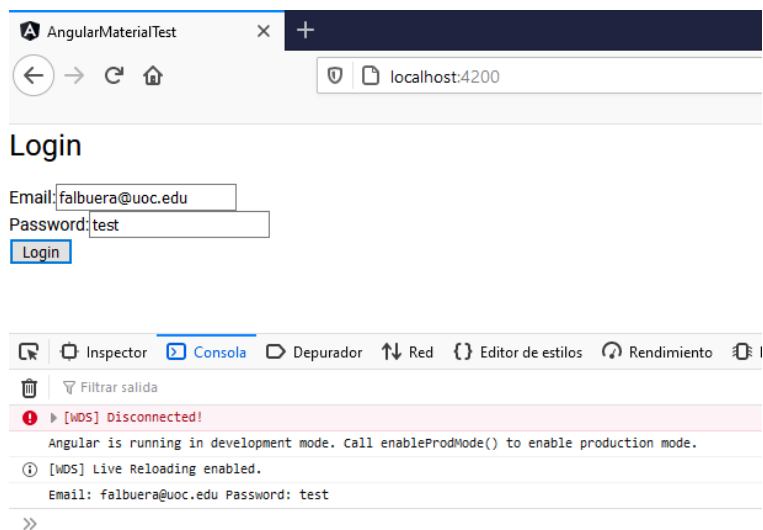


Inicialmente nos indica los mensajes de error, que en este caso sería correcto. Si escribimos algo en un campo o en otro desaparecerá el mensaje de error y en el momento en que haya información en los dos campos se habilitará el botón de **Login**, si el campo email tiene un formato correcto, de lo contrario también mostraremos un mensaje de error.



Aquí podemos ver el ejemplo de un email informado, pero con formato incorrecto.

Insertamos algunos datos correctos y pulsamos **Login** podemos observar la salida de la consola:



Hasta aquí nada nuevo, esto lo estudiamos en el primer tema de esta asignatura.

Vamos ahora a configurar **Angular Material** a nuestro proyecto y a aplicar varias cosas en esta aplicación tan sencilla que tenemos ahora mismo.

Ejemplo aplicación Angular Material

Vamos a aplicar **Angular Material** al formulario actual y a revisar alguna de las muchas posibilidades que nos ofrece este **framework**.

- **Paso 1:**

Configuramos **Angular Material** a nuestro proyecto:

Podemos ir a la **url** de la página oficial para consultar toda la información:

<https://material.angular.io/>

Para instalar **Angular Material** básicamente tenemos que ejecutar la siguiente instrucción:

```
H:\Projectes\angular-material-test>ng add @angular/material
Installing packages for tooling via npm.
Installed packages for tooling via npm.
? Choose a prebuilt theme name, or "custom" for a custom theme: Indigo/Pink [ Preview: https://material.angular.io/theme-indigo-pink ]
? Set up global Angular Material typography styles? Yes
? Set up browser animations for Angular Material? Yes
UPDATE package.json (1330 bytes)
- Installing packages...
```

ng add @angular/material

- Cuando nos pregunte por el tema, podemos escoger uno cualquiera.
- La tipografía de estilo podemos decirle que sí, que la configure a nivel global.
- También podemos decir que si a la configuración de las animaciones.

Esto una vez ejecutado nos instalara **Angular Material**, el **CDK** y el paquete de animaciones.

Si vamos a la web de **Angular Material**, podemos ver dos puntos de menú diferenciados, tenemos los componentes y el **CDK**.

Entrando al apartado de componentes podremos ver muchos tipos diferentes de componentes. Por cada uno, podemos acceder a su detalle, donde normalmente encontraremos una descripción y el código de ejemplo para entender su utilización.

Por otra parte, en el apartado del **CDK**, podemos ver un listado de componentes también, pero esta vez se trata de componentes más complejos, por ejemplo, un **stepper**.

Finalmente, en el apartado de guías hay varios apartados, de entre los cuales destaca el apartado dedicado a los **temas** de **Angular Material**. Nosotros/as al instalar el paquete a nuestro proyecto hemos escogido un tema por defecto, pero también podríamos haber escogido la opción **custom** y personalizar nuestro propio tema.

Esta es la visión global, vamos a poner algún ejemplo de utilización.

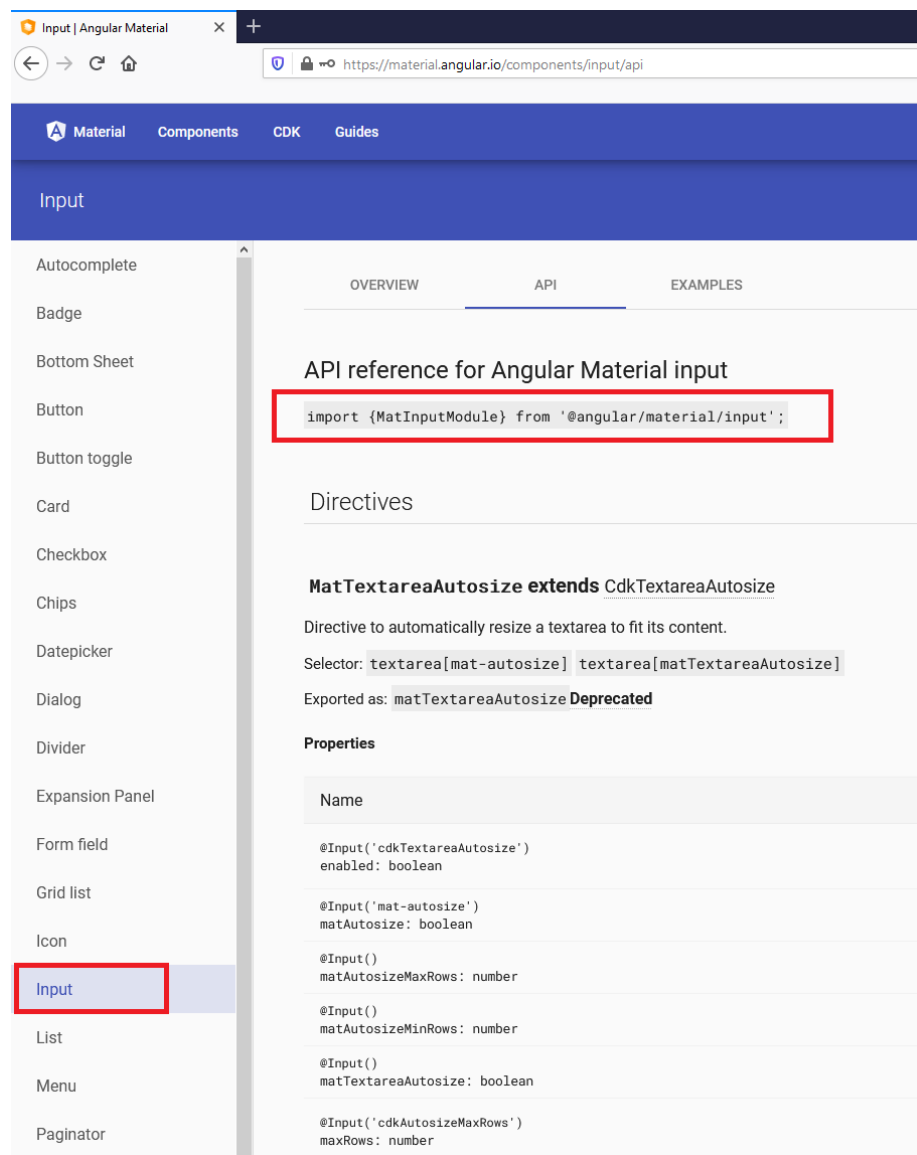
Vamos a traducir nuestro formulario a componentes de **Angular Material**.

Si arrancamos la aplicación de nuevo veremos que la tipografía nos ha cambiado un poco. Vamos a traducir el formulario a **Angular Material**:

• Paso 2:

Lo que debemos tener en cuenta para trabajar con **Angular Material**, al menos al principio, es hacer lo siguiente.

Es decir, por ejemplo, ahora queremos pintar en la vista con **Material** un input de tipo email, vale, pues nos iremos a la web de **Angular Material**, apartado componentes, y buscaremos en el listado el componente que nos implica.



The screenshot shows the Angular Material website in a browser. The URL is <https://material.angular.io/components/input/api>. The page title is "Input | Angular Material". The navigation bar includes "Material", "Components", "CDK", and "Guides". The "Input" component is selected in the left sidebar. The main content area has tabs for "OVERVIEW", "API", and "EXAMPLES". The "API" tab is active, showing the "API reference for Angular Material input". A red box highlights the import statement: `import {MatInputModule} from '@angular/material/input';`. Below this, the "Directives" section lists **MatTextareaAutosize** extending `CdkTextareaAutosize`. The "Properties" section lists various input properties like `enabled`, `matAutosize`, `matAutosizeMaxRows`, `matAutosizeMinRows`, and `matTextareaAutosize`.

En este caso dentro del componente **Input** en la pestaña **Overview** podremos ver todos los tipos de input que podemos tratar, entre ellos está el tipo email. Vale, es importante consultar la pestaña **API**, porque es ahí donde nos indicarán que módulo debemos importar para poder utilizar dicho componente. También es interesante consultar la pestaña **examples** ya que en ella encontraremos diferentes ejemplos con su código.

```

app.modules.ts X
src > app > app.modules > ...
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { ReactiveFormsModule } from '@angular/forms';
7 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
8
9 import { MatFormFieldModule } from '@angular/material/form-field';
10 import { MatInputModule } from '@angular/material/input';
11 import { MatButtonModule } from '@angular/material/button';
12 import { LoginComponent } from './Components/login/login.component';
13
14 @NgModule({
15   declarations: [AppComponent, LoginComponent],
16   imports: [
17     BrowserModule,
18     AppRoutingModule,
19     ReactiveFormsModule,
20     BrowserAnimationsModule,
21     MatFormFieldModule,
22     MatInputModule,
23     MatButtonModule,
24   ],
25   providers: [],
26   bootstrap: [AppComponent],
27 })
28 export class AppModule {}
29

```

En nuestro caso, para el formulario de **login**, necesitaríamos importar estos tres módulos.

• Paso 3:

Transformamos la vista en componentes de **Angular Material**:

```

login.component.html X
src > app > Components > login > login.component.html > div.container-uoc > form > section > button
1 <!--
2 <div>
3   <h1>Login</h1>
4   <form [formGroup]="loginForm" (ngSubmit)="checkLogin()">
5
6     <div>
7       <label for="email">Email:</label>
8       <input type="text" [formControl]="email" />
9       <span style="color: red;" *ngIf="loginForm.get('email').errors?.required">User email is required</span>
10      <span style="color: red;" *ngIf="loginForm.get('email').errors?.email">Not a valid email</span>
11    </div>
12
13    <div>
14      <label for="password">Password:</label>
15      <input type="text" [formControl]="password" />
16      <span style="color: red;" *ngIf="loginForm.get('password').errors?.required">User password is required</span>
17    </div>
18
19    <button type="submit" [disabled]="!loginForm.valid">Login</button>
20  </form>
21 </div>
22 -->
23
24 <div class="container-uoc"> (A)
25   <form [formGroup]="loginForm" (ngSubmit)="checkLogin()">
26
27     <section>
28       <mat-form-field appearance="fill">
29         <mat-label>Enter your email</mat-label>
30         <input matInput type="email" [formControl]="email" required placeholder="Ex. pat@example.com"/>
31         <mat-error *ngIf="email.hasError('email') && !email.hasError('required')">
32           Please enter a valid email address
33         </mat-error>
34         <mat-error *ngIf="email.hasError('required')">
35           Email is required</mat-error>
36       </mat-form-field>
37     </section>
38
39     <section>
40       <mat-form-field appearance="fill">
41         <mat-label>Enter your password</mat-label>
42         <input matInput type="password" [formControl]="password" required />
43         <mat-error *ngIf="password.invalid">{{ getErrorMessage() }}</mat-error>
44       </mat-form-field>
45     </section>
46
47     <section style="text-align: center;">
48       <button mat-raised-button color="primary" type="submit" [disabled]="!loginForm.valid">Login</button>
49     </section>
50  </form>
51 </div>
52

```

Apartado A)

Simplemente nos personalizamos una clase para darle un poco de formato y que esté centrado nuestro formulario de **login**, la clase sería así:

```

styles.css x
src > styles.css > ...
1  /* You can add global styles to this file, and also import other
2
3  html,
4  body {
5      height: 100%;
6  }
7  body {
8      margin: 0;
9      font-family: Roboto, "Helvetica Neue", sans-serif;
10 }
11
12 .container-uoc {
13     display: flex;
14     justify-content: center;
15     align-items: center;
16     flex-direction: column;
17     padding: 30px;
18 }
19

```

Apartado B)

En esta sección vemos la ‘traducción’ a **Material** del input email (módulo **MatInputModule**). Podemos ver que utilizamos las etiquetas **mat-form-field** (módulo **MatFormFieldModule**), definimos una etiqueta, un **placeholder** (nótese el efecto cuando cliquemos al input que el **placeholder** se desplaza hacia arriba, es característico de **Angular Material**), en la parte inferior nos definimos un par de secciones para mostrar los mensajes de error, en este caso los errores de campo requerido y formato de email no válido.

Apartado C)

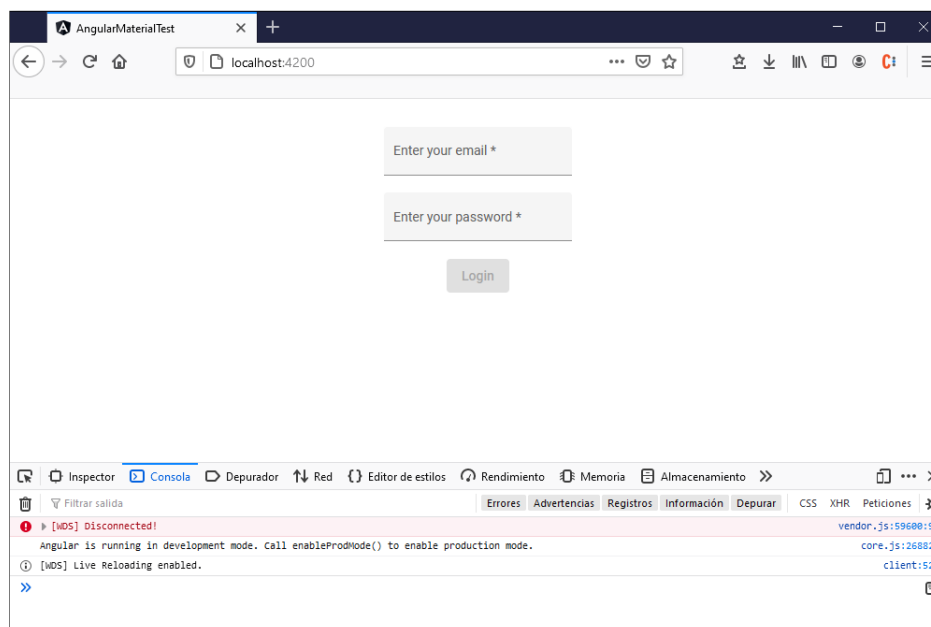
De manera muy parecida a la anterior, implementamos el input para el campo **password**. Aquí la diferencia es que el mensaje de error nos devuelve la función **getErrorMessage**. Simplemente para que nos demos cuenta de que podríamos parametrizar mensajes de error comunes a diferentes inputs de esta manera.

La implementación del **getErrorMessage** sería la siguiente:

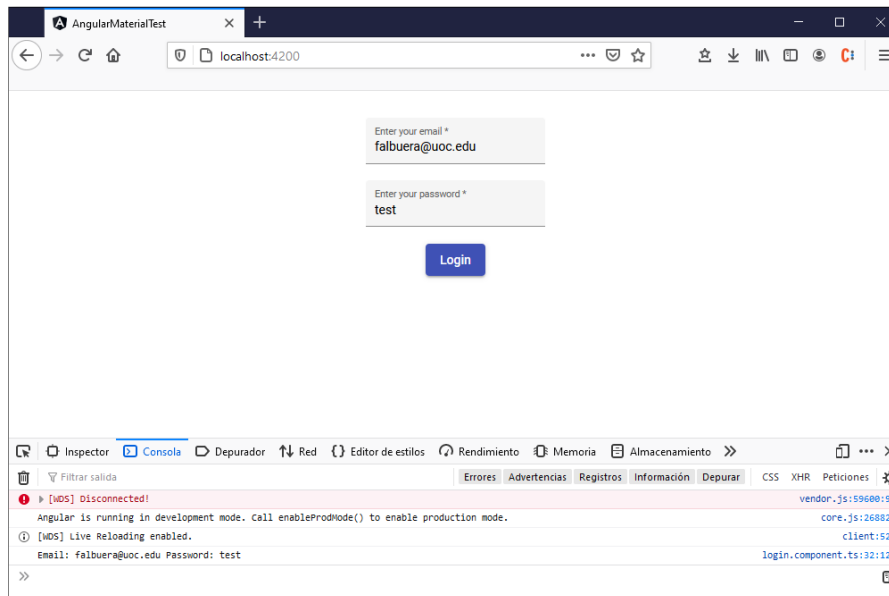
```
login.component.ts X
src > app > Components > login > login.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';
3 import { Credentials } from 'src/app/Models/credentials';
4
5 @Component({
6   selector: 'app-login',
7   templateUrl: './login.component.html',
8   styleUrls: ['./login.component.css'],
9 })
10 export class LoginComponent implements OnInit {
11
12   public credentials: Credentials = new Credentials();
13
14   public email: FormControl;
15   public password: FormControl;
16
17   public loginForm: FormGroup;
18
19   constructor(private formBuilder: FormBuilder) {}
20
21   ngOnInit(): void {
22     this.email = new FormControl('', [Validators.required, Validators.email]);
23     this.password = new FormControl('', Validators.required);
24
25     this.loginForm = this.formBuilder.group({
26       email: this.email,
27       password: this.password,
28     });
29   }
30
31   public checkLogin(): void {
32     console.log('Email: ' + this.email.value + ' Password: ' + this.password.value);
33   }
34
35   public getErrorMessage(): string {
36     if (this.password.hasError('required')) {
37       return 'You must enter a password';
38     }
39   }
40 }
41
```

Al final del formulario tendremos el botón de **login** (modulo **MatButtonModule**)

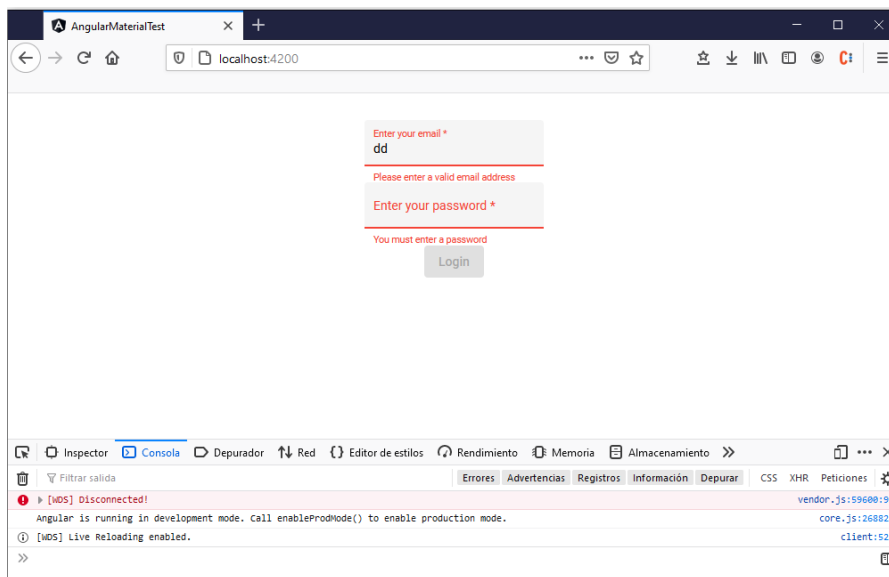
Vemos el resultado:



Funcionando bien:



Forzando algún error:



El aspecto visual de **Material** es bastante elegante y las dinámicas de los **placeholders** que se desplazan hacia arriba van bastante bien.

Ahora hemos trabajado sólo con dos campos, pero si echáis un vistazo a la web oficial, tenemos muchos tipos de componentes diferentes.

Actividad complementaria



*“Consulta en la web de **Angular Material** para ver cómo podríamos añadir el típico botón de mostrar o no lo que escribimos en el campo **password**, de manera que si no queremos mostrar el contenido se muestre en puntos.”*

Jugando con el componente card

Vamos a suponer que queremos mostrar un listado de **cards**, posteriormente estas **cards** podrían ser nuestras actividades turísticas, por ahora, lo que haremos será consumir una api pública que devuelve un listado de imágenes aleatorias con algo de información, que para nuestro caso de ejemplo ya nos valdría.

La **url** que consumiremos será:

<https://picsum.photos/v2/list>

• Paso 1:

Implementemos el servicio:

Nos creamos en la raíz de la carpeta **app** la carpeta **Services** y lanzamos la siguiente instrucción:

```
H:\Projectes\angular-material-test>ng g s Services/images
CREATE src/app/Services/images.service.spec.ts (357 bytes)
CREATE src/app/Services/images.service.ts (135 bytes)
```

Antes de implementar el servicio, como haremos peticiones **http**, necesitaremos importar el módulo:

```
app.modules.ts x
src > app > app.modules.ts > ...
...
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { ReactiveFormsModule } from '@angular/forms';
7 import { BrowserModuleAnimationsModule } from '@angular/platform-browser/animations';
8
9 import { MatFormFieldModule } from '@angular/material/form-field';
10 import { MatInputModule } from '@angular/material/input';
11 import { MatButtonModule } from '@angular/material/button';
12 import { LoginComponent } from './Components/login/login.component';
13
14 import { HttpClientModule } from '@angular/common/http';
15
16 ...
17 @NgModule({
18   declarations: [AppComponent, LoginComponent],
19   imports: [
20     BrowserModule,
21     AppRoutingModule,
22     ReactiveFormsModule,
23     BrowserModuleAnimationsModule,
24     MatFormFieldModule,
25     MatInputModule,
26     MatButtonModule,
27     HttpClientModule
28   ],
29   providers: [],
30   bootstrap: [AppComponent],
31 })
32 export class AppModule {}
```

En el servicio vamos a implementar la petición para recuperar de la **url** anterior el listado de imágenes:

```

images.service.ts X
src > app > Services > images.service.ts > ...
1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { Observable } from 'rxjs';
4
5  @Injectable({
6    providedIn: 'root',
7  })
8  export class ImagesService {
9    constructor(private http: HttpClient) {}
10
11    getAllImages(): Observable<any> {
12      return this.http.get('https://picsum.photos/v2/list');
13    }
14  }
15

```

Normalmente mapearíamos la respuesta en una clase o interfaz, de manera que al **Observable** le pasaríamos un array de la clase **Image** por ejemplo, que tendría las propiedades que nos devolverá la llamada que pasamos por la **url**, pero de momento le pasamos **any** directamente y ya está, posteriormente veremos que para mostrar la información en la vista terminaremos creando la clase **Image** con las propiedades que nos vienen de la api.

• Paso 2:

Vamos a ver que nos devuelve este servicio, llamémoslo desde un componente.

Creémonos un componentes **cards** por ejemplo:

```

H:\Projectes\angular-material-test>ng g c Components/cards
CREATE src/app/Components/cards/cards.component.html (20 bytes)
CREATE src/app/Components/cards/cards.component.spec.ts (619 bytes)
CREATE src/app/Components/cards/cards.component.ts (271 bytes)
CREATE src/app/Components/cards/cards.component.css (0 bytes)
UPDATE src/app/app.module.ts (1074 bytes)

```

Llamamos al servicio:

```
cards.component.ts X
src > app > Components > cards > cards.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { ImagesService } from 'src/app/Services/images.service';
3
4  @Component({
5    selector: 'app-cards',
6    templateUrl: './cards.component.html',
7    styleUrls: ['./cards.component.css'],
8  })
9  export class CardsComponent implements OnInit {
10
11    constructor(private imagesService: ImagesService) {}
12
13    ngOnInit(): void {
14      this.imagesService.getAllImages().subscribe((cards) => console.log(cards));
15    }
16  }
17
```

• Paso 3:

Vamos a hacer que cuando pulsemos el botón de **login** nos redireccione a este componente para así poder revisar en la consola si nos llega el listado de imágenes de esta api pública.

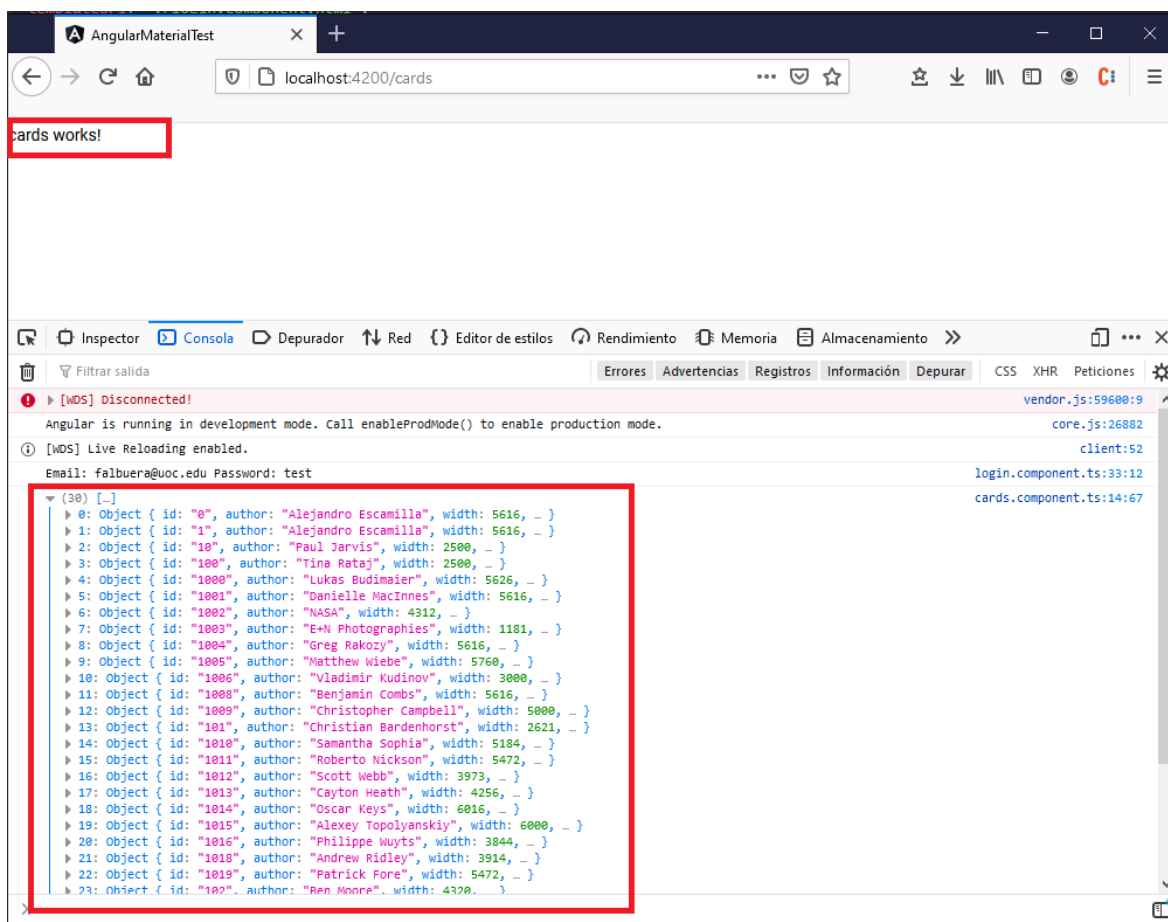
Declaramos la nueva ruta:

```
app-routing.module.ts X
src > app > app-routing.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3  import { CardsComponent } from '../Components/cards/cards.component';
4  import { LoginComponent } from '../Components/login/login.component';
5
6  const routes: Routes = [
7    {
8      path: '', component: LoginComponent
9    },
10   {
11     path: 'cards', component: CardsComponent
12   }
13 ];
14
15 @NgModule({
16   imports: [RouterModule.forRoot(routes)],
17   exports: [RouterModule]
18 })
19 export class AppRoutingModule { }
20
```

Le decimos que vaya al componente de **cards** al pulsar el botón de **login**:

```
login.component.ts X
src > app > Components > login > login.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';
3  import { Router } from '@angular/router';
4  import { Credentials } from 'src/app/Models/credentials';
5
6  @Component({
7    selector: 'app-login',
8    templateUrl: './login.component.html',
9    styleUrls: ['./login.component.css'],
10 })
11 export class LoginComponent implements OnInit {
12
13   public credentials: Credentials = new Credentials();
14
15   public email: FormControl;
16   public password: FormControl;
17
18   public loginForm: FormGroup;
19
20   constructor(private formBuilder: FormBuilder, private route: Router) {}
21
22   ngOnInit(): void {
23     this.email = new FormControl('', [Validators.required, Validators.email]);
24     this.password = new FormControl('', Validators.required);
25
26     this.loginForm = this.formBuilder.group({
27       email: this.email,
28       password: this.password,
29     });
30   }
31
32   public checkLogin(): void {
33     console.log('Email: ' + this.email.value + ' Password: ' + this.password.value);
34     this.route.navigate(['cards']);
35   }
36
37   public getErrorMessage(): string {
38     if (this.password.hasError('required')) {
39       return 'You must enter a password';
40     }
41   }
42 }
```

Ejecutamos la aplicación, insertamos email y contraseña, pulsamos al botón de **login** y veremos:



Podemos ver por consola que nos da un listado de 30 imágenes con su respectiva información. Vale, vamos a mostrarlo en la vista del componente **card**, pero en vez de mostrar la típica lista, vamos a utilizar las **cards** de **Angular Material**.

Como comentamos anteriormente, iríamos a la web de **Angular Material**, iríamos al apartado de las **cards** y revisaríamos qué módulo tenemos que importar y algunos ejemplos para familiarizarnos con este componente y adaptarlo a nuestras necesidades.

• Paso 4:

Primero importamos el módulo:

```

app.module.ts x
src > app > app.module.ts > ...
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { ReactiveFormsModule } from '@angular/forms';
7  import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
8
9  import { MatFormFieldModule } from '@angular/material/form-field';
10 import { MatInputModule } from '@angular/material/input';
11 import { MatButtonModule } from '@angular/material/button';
12 import { LoginComponent } from './Components/login/login.component';
13
14 import { MatCardModule } from '@angular/material/card';
15
16 import { HttpClientModule } from '@angular/common/http';
17 import { CardsComponent } from './Components/cards/cards.component';
18
19 @NgModule({
20   declarations: [AppComponent, LoginComponent, CardsComponent],
21   imports: [
22     BrowserModule,
23     AppRoutingModule,
24     ReactiveFormsModule,
25     BrowserAnimationsModule,
26     MatFormFieldModule,
27     MatInputModule,
28     MatButtonModule,
29     HttpClientModule,
30     MatCardModule
31   ],
32   providers: [],
33   bootstrap: [AppComponent],
34 })
35 export class AppModule {}
36

```

• Paso 5:

El sistema **grid** (rejilla/cuadrícula) de **Angular Material** se basa en **Flex Layout** y en lugar de utilizar las reglas CSS nativamente se fundamenta en el paquete **@angular/flex-layout** desarrollado por el propio **core** de **Angular**.

Vamos a instalar este paquete para poder manejar todas las **cards** que queremos mostrar.

```
npm install -s @angular/flex-layout
```

```
H:\Projectes\angular-material-test>npm install -s @angular/flex-layout
```

Importamos el módulo:

```
app.module.ts X
src > app > app.module.ts > ...
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { ReactiveFormsModule } from '@angular/forms';
7  import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
8
9  import { MatFormFieldModule } from '@angular/material/form-field';
10 import { MatInputModule } from '@angular/material/input';
11 import { MatButtonModule } from '@angular/material/button';
12 import { LoginComponent } from './Components/login/login.component';
13
14 import { MatCardModule } from '@angular/material/card';
15
16 import { HttpClientModule } from '@angular/common/http';
17 import { CardsComponent } from './Components/cards/cards.component';
18 import { FlexLayoutModule } from '@angular/flex-layout';
19
20 @NgModule({
21   declarations: [AppComponent, LoginComponent, CardsComponent],
22   imports: [
23     BrowserModule,
24     AppRoutingModule,
25     ReactiveFormsModule,
26     BrowserAnimationsModule,
27     MatFormFieldModule,
28     MatInputModule,
29     MatButtonModule,
30     HttpClientModule,
31     MatCardModule,
32     FlexLayoutModule
33   ],
34   providers: [],
35   bootstrap: [AppComponent],
36 })
37 export class AppModule {}
38
```

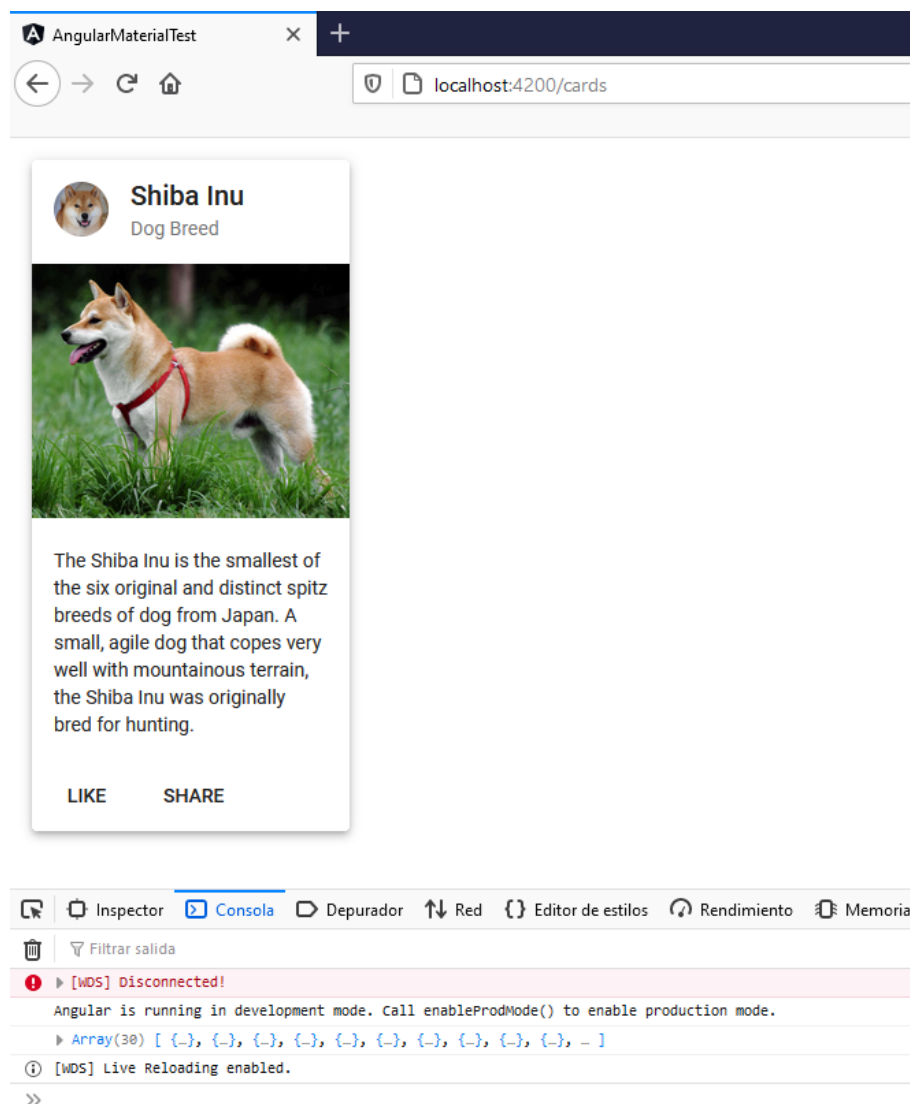
Maquetamos la vista para mostrar la **card** de ejemplo de la propia página de angular material:

```
cards.component.html X
src > app > Components > cards > cards.component.html > ...
1  <div class="content">
2    <mat-card class="mat-elevation-z4">
3      <mat-card-header>
4        <div mat-card-avatar class="example-header-image"></div>
5        <mat-card-title>Shiba Inu</mat-card-title>
6        <mat-card-subtitle>Dog Breed</mat-card-subtitle>
7      </mat-card-header>
8      
9      <mat-card-content>
10       <p>
11         The Shiba Inu is the smallest of the six original and distinct spitz
12         breeds of dog from Japan. A small, agile dog that copes very well with
13         mountainous terrain, the Shiba Inu was originally bred for hunting.
14       </p>
15     </mat-card-content>
16     <mat-card-actions>
17       <button mat-button>LIKE</button>
18       <button mat-button>SHARE</button>
19     </mat-card-actions>
20   </mat-card>
21 </div>
22
```


Los estilos:

```
cards.component.css X
src > app > Components > cards > cards.component.css > ...
1  .content {
2    padding: 16px;
3  }
4
5  .content > mat-card {
6    width: 200px;
7  }
8
9  .example-header-image {
10   background-image: url("https://material.angular.io/assets/img/examples/shiba1.jpg");
11   background-size: cover;
12 }
13
```

Resultado:



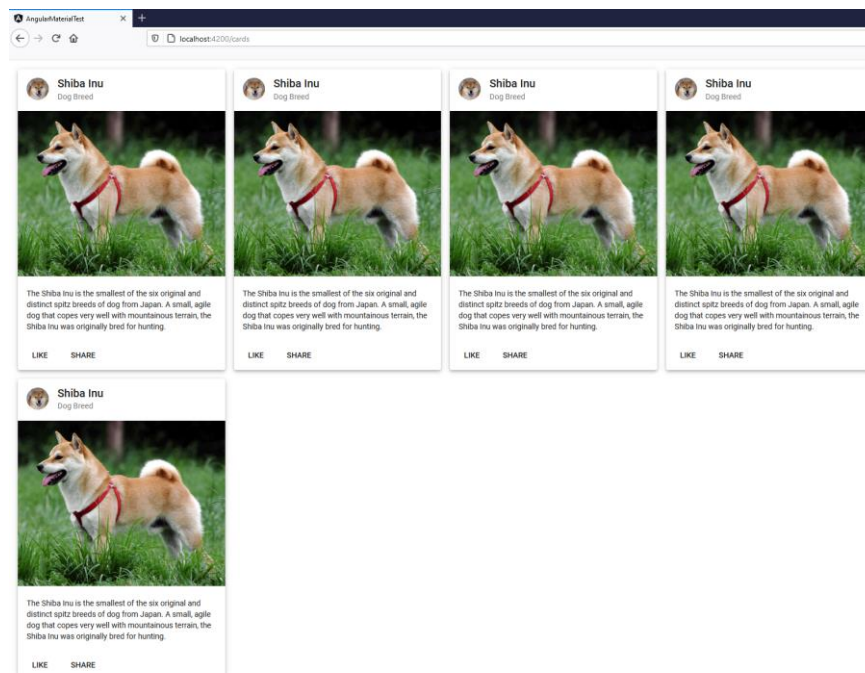
• Paso 6:

Vamos a mostrar ahora un listado de estas **cards**, por el momento las repetiremos, pero veremos como si luego hacemos pequeño el navegador estas **cards** se irán adaptando perfectamente.

Vista:

```
cards.component.html X
src > app > Components > cards > cards.component.html > ...
1 <div class="content">
2 <div fxLayout="row wrap" fxLayoutGap="16px grid">
3 <div fxFlex="25%" fxFlex.xs="100%" fxFlex.sm="33%" *ngFor="let num of [1, 2, 3, 4, 5]">
4 <mat-card class="mat-elevation-z4">
5 <mat-card-header>
6 <div mat-card-avatar class="example-header-image"></div>
7 <mat-card-title>Shiba Inu</mat-card-title>
8 <mat-card-subtitle>Dog Breed</mat-card-subtitle>
9 </mat-card-header>
10 
11 <mat-card-content>
12 <p>
13 The Shiba Inu is the smallest of the six original and distinct spitz
14 breeds of dog from Japan. A small, agile dog that copes very well
15 with mountainous terrain, the Shiba Inu was originally bred for
16 hunting.
17 </p>
18 </mat-card-content>
19 <mat-card-actions>
20 <button mat-button>LIKE</button>
21 <button mat-button>SHARE</button>
22 </mat-card-actions>
23 </mat-card>
24 </div>
25 </div>
26 </div>
27
```

Resultado:

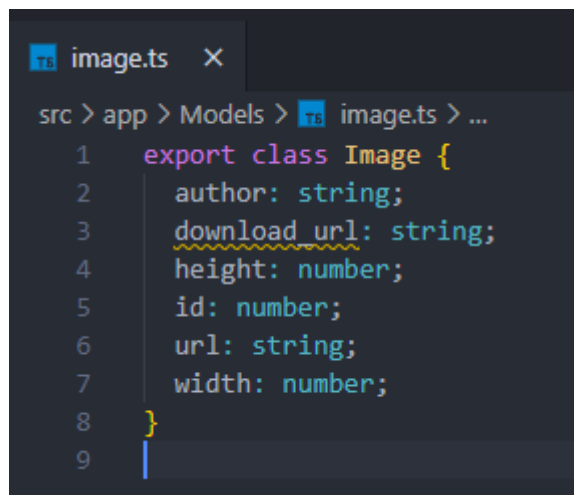


Aquí podemos ver como si hacemos el navegador más pequeño, las **cards** se irán adaptando.

Vale, ahora que tenemos la estructura montada, vamos a hacer que se muestre la información de las 30 imágenes que recibimos como respuesta de la api.

• Paso 7:

Creémonos mejor la clase **Image** para mapear la respuesta de la api, ahora que por consola hemos podido revisar todos los campos que nos vienen:



```

src > app > Models > image.ts > ...
1  export class Image {
2      author: string;
3      download url: string;
4      height: number;
5      id: number;
6      url: string;
7      width: number;
8  }
9

```

Adaptamos el servicio anterior:



```

images.service.ts X
src > app > Services > images.service.ts > ...
1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { Observable } from 'rxjs';
4  import { Image } from 'src/app/Models/image';
5
6  @Injectable({
7      providedIn: 'root',
8  })
9  export class ImagesService {
10     constructor(private http: HttpClient) {}
11
12     getAllImages(): Observable<Image[]> {
13         return this.http.get<Image[]>('https://picsum.photos/v2/list');
14     }
15 }
16

```

Adaptamos la llamada al controlador a la nueva clase:

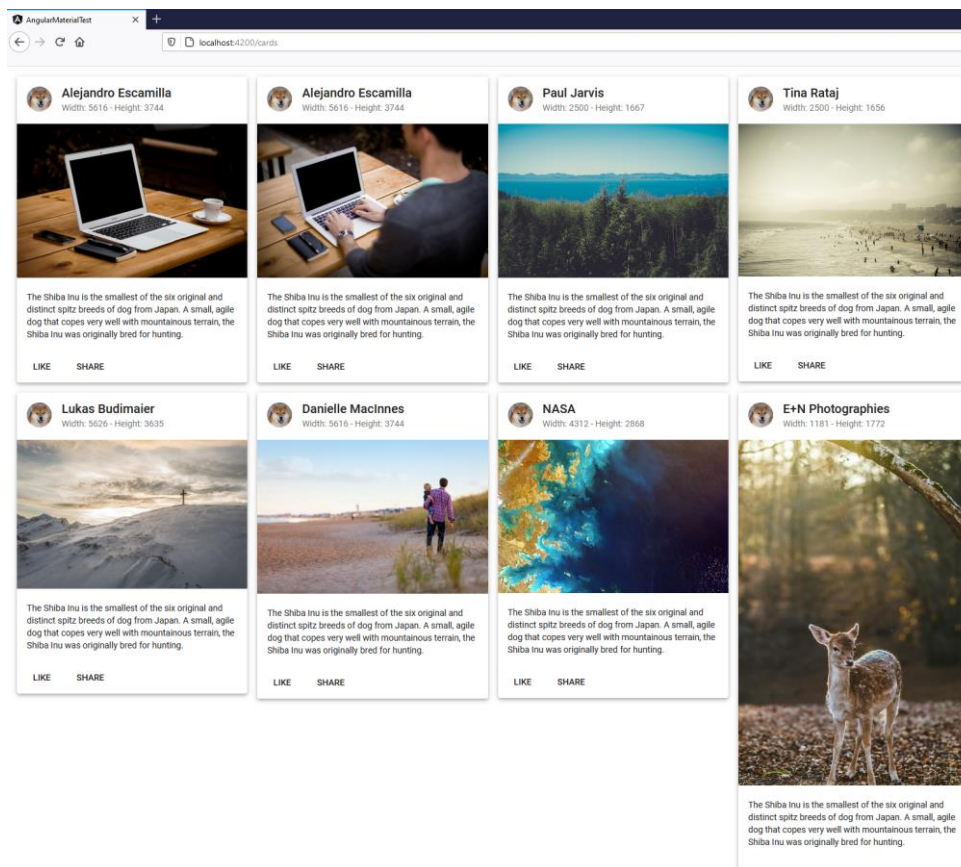
```
cards.component.ts X
src > app > Components > cards > cards.component.ts > ...
2 import { Image } from 'src/app/Models/image';
3 import { ImagesService } from 'src/app/Services/images.service';
4
5 @Component({
6   selector: 'app-cards',
7   templateUrl: './cards.component.html',
8   styleUrls: ['./cards.component.css'],
9 })
10 export class CardsComponent implements OnInit {
11   public cards: Image[];
12
13   constructor(private imagesService: ImagesService) {}
14
15   ngOnInit(): void {
16     this.imagesService
17       .getAllImages()
18       .subscribe((cards) => (this.cards = cards));
19   }
20 }
21
```

Adaptamos la vista:

```
cards.component.html X
src > app > Components > cards > cards.component.html > ...
1 <div class="content">
2   <div fxLayout="row wrap" fxLayoutGap="16px grid">
3     <div fxFlex="25%" fxFlex.xs="100%" fxFlex.sm="33%" *ngFor="let card of cards">
4       <mat-card class="mat-elevation-z4">
5         <mat-card-header>
6           <div mat-card-avatar class="example-header-image"></div>
7           <mat-card-title>{{card.author}}</mat-card-title>
8           <mat-card-subtitle>Width: {{card.width}} - Height: {{card.height}}</mat-card-subtitle>
9         </mat-card-header>
10        
11        <mat-card-content>
12          <p>
13            The Shiba Inu is the smallest of the six original and distinct spitz
14            breeds of dog from Japan. A small, agile dog that copes very well
15            with mountainous terrain, the Shiba Inu was originally bred for
16            hunting.
17          </p>
18        </mat-card-content>
19        <mat-card-actions>
20          <button mat-button>LIKE</button>
21          <button mat-button>SHARE</button>
22        </mat-card-actions>
23      </mat-card>
24    </div>
25  </div>
26 </div>
27
```

Fijémonos que hacemos un bucle de todos los registros de la respuesta de la api y de cada uno pintaremos una **card**. Utilizando la imagen que nos viene y mostrando algún campo más como el autor y las medidas altura y anchura de la imagen. Simplemente es para mostrar alguna cosa en la propia **card**.

Resultado:



Tenemos un resultado bastante profesional, con las **cards** totalmente **responsives**.

Jugando con animaciones

Vamos a utilizar las animaciones para hacer que el formulario de **login** en lugar de que aparezca sin más, lo haga de una manera más fluida.

Pensemos que el paquete de animaciones ya lo tenemos instalado en nuestro proyecto ya que cuando lo hemos creado al principio de este documento, le hemos dicho que sí que nos incluya las animaciones.

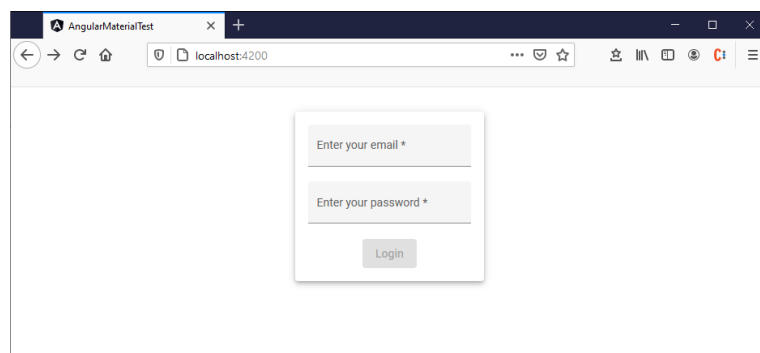
Si vamos al controlador **login.components.ts** podemos añadir la siguiente animación:

```
login.components.ts X
src > app > Components > login > login.components.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';
3 import { Router } from '@angular/router';
4 import { Credentials } from 'src/app/Models/credentials';
5 import { animate, state, style, transition, trigger } from '@angular/animations';
6
7 @Component({
8   selector: 'app-login',
9   templateUrl: './login.component.html',
10  styleUrls: ['./login.component.css'],
11  animations: [
12    trigger('fadeInOut', [
13      state('void', style({
14        opacity: 0.2
15      })),
16      transition('void <=> *', animate(1500)),
17    ])
18  ]
19 })
20 export class LoginComponent implements OnInit {
21
22   public credentials: Credentials = new Credentials();
23
24   public email: FormControl;
25   public password: FormControl;
26
27   public loginForm: FormGroup;
28
29   constructor(private formBuilder: FormBuilder, private route: Router) {}
30
31   ngOnInit(): void {
32     this.email = new FormControl('', [Validators.required, Validators.email]);
33     this.password = new FormControl('', Validators.required);
34
35     this.loginForm = this.formBuilder.group({
36       email: this.email,
37       password: this.password,
38     });
39   }
40
41   public checkLogin(): void {
42     console.log('Email: ' + this.email.value + ' Password: ' + this.password.value);
43     this.route.navigate(['cards']);
44   }
45
46   public getErrorMessage(): string {
47     if (this.password.hasError('required')) {
48       return 'You must enter a password';
49     }
50   }
51 }
52
```

Y, por otro lado, en la vista, le añadimos la directiva **[@fadeInOut]** para que haga la animación. Fijémonos también que hemos puesto el formulario dentro de una **card** simplemente para que resalte un poco más y podamos ver mejor la animación:

```
login.component.html X
src > app > Components > login > login.component.html > ...
1 <!--
2 <div>
3   <h1>Login</h1>
4   <form [formGroup]="loginForm" (ngSubmit)="checkLogin()">
5
6     <div>
7       <label for="email">Email:</label>
8       <input type="text" [formControl]="email" />
9       <span style="color: red" *ngIf="loginForm.get('email').errors?.required">User email is required</span>
10      <span style="color: red" *ngIf="loginForm.get('email').errors?.email">Not a valid email</span>
11    </div>
12
13    <div>
14      <label for="password">Password:</label>
15      <input type="password" [formControl]="password" />
16      <span style="color: red" *ngIf="loginForm.get('password').errors?.required">User password is required</span>
17    </div>
18
19    <button type="submit" [disabled]="!loginForm.valid">Login</button>
20  </form>
21 </div>
22 -->
23
24 <div class="container-uoc" @fadeInOut>
25   <mat-card class="mat-elevation-z4">
26     <form [formGroup]="loginForm" (ngSubmit)="checkLogin()">
27
28       <section>
29         <mat-form-field appearance="fill">
30           <mat-label>Enter your email</mat-label>
31           <input matInput type="email" [formControl]="email" required placeholder="Ex. pat@example.com"/>
32           <mat-error *ngIf="email.hasError('email') && !email.hasError('required')">
33             Please enter a valid email address
34           </mat-error>
35           <mat-error *ngIf="email.hasError('required')">
36             Email is <strong>required</strong>
37           </mat-error>
38         </mat-form-field>
39       </section>
40
41       <section>
42         <mat-form-field appearance="fill">
43           <mat-label>Enter your password</mat-label>
44           <input matInput type="password" [formControl]="password" required />
45           <mat-error *ngIf="password.invalid">{{ getErrorMessage() }}</mat-error>
46         </mat-form-field>
47       </section>
48
49       <section style="text-align: center;">
50         <button mat-raised-button color="primary" type="submit" [disabled]="!loginForm.valid">Login</button>
51       </section>
52     </form>
53   </mat-card>
54 </div>
55
```

Resultado:



Aunque aquí vemos una imagen estática, en la ejecución se hace una carga del formulario muy fluida donde el componente va apareciendo poco a poco.