Tutorial de Angular 6: aprende Angular 6 en este curso intensivo

Por Gary Simon - 31 de mayo de 2018

Cómo instalar Angular 6

La forma más rápida y fácil de iniciar una aplicación Angular 6 es a través de la CLI angular (interfaz de línea de comandos).

Para instalarlo, necesitará el administrador de paquetes de hilo o el administrador de paquetes de nodos. Para verificar si tiene o no npm, en su consola / línea de comando, escriba:

```
> npm -v
```

Si esto no se reconoce (y no es un número de versión), debe instalar NodeJS .

Una vez que instale NodeJS, vuelva a cargar su consola o línea de comando y tendrá acceso a NPM.

Ahora, podemos usar NPM para instalar la CLI angular. Para instalarlo:

> npm install -g @angular/cli

Si ejecuta

ng -v

después de la instalación, le proporcionará el número de versión.

Una vez que se instala la CLI, ahora podemos usarla para comenzar a instalar un nuevo proyecto de Angular 6:

```
> ng new ng6-proj --style=scss --routing
```

- **ng** : así es como llamamos a la CLI angular.
- **nuevo** : este es uno de los muchos comandos que podemos enviar a la CLI.
- **ng6-proj** es simplemente el nombre que llamamos nuestro proyecto. La CLI creará una carpeta con este nombre, en función de dónde esté ejecutando este comando.

- **Indicadores opcionales** : a continuación hay dos indicadores opcionales que decidí agregar. Primero, le estamos diciendo a la CLI que genere un proyecto que tenga Sass habilitado (de manera predeterminada, se usa CSS si no se especifica), y luego se agrega el *enrutamiento* porque queremos un proyecto que tenga URL de página diferentes y agregue Esta bandera nos ayudará a crear un archivo de enrutamiento.

Para ver todos los comandos y opciones disponibles, ejecute **ng** en la línea de comandos.

Una vez que la CLI ha generado el nuevo proyecto, puede acceder a él:

```
> cd ng6-proj
```

Si usa Visual Studio Code, puede escribir *código.* para iniciar el editor de código. Luego, para servir el proyecto al navegador, ejecuta:

```
> ng serve -o
```

La bandera -o le dice a la CLI que inicie su navegador con el proyecto. Ahora, puede ver su proyecto

Angular 6 mientras codifica y se recargará automáticamente. ¡Increíble!

Estructura del proyecto angular 6

Este es un tutorial para principiantes, por lo que no vamos a profundizar en cada archivo. Todo lo que es importante que comprenda son los conceptos básicos absolutos.

Cuando veas la estructura de carpetas y archivos de tu aplicación

Angular 6, debería tener un aspecto similar a este:

```
 > e2e > node_modules > src > app ...a bunch of files ...a bunch of files
```

Pasará la mayor parte del tiempo trabajando en la carpeta **/ src / app** . Aquí es donde se almacenan los componentes y servicios (los veremos en breve).

En la carpeta **/ src /** , verá un *index.html* (el punto de entrada de las aplicaciones) y un archivo *styles.css* , que es donde se ubican los conjuntos de reglas CSS globales.

La carpeta / **src** / **assets** es donde coloca los activos, como los gráficos.

No está presente en este momento una carpeta / **dist** / , que se genera cuando compila el proyecto, lo que haremos más adelante.

El archivo de módulo angular 6

Antes de abordar los componentes, vale la pena mirar el archivo **/src/app/app.module.ts** . Ah, y por cierto, ¿cuál es esa extensión **.ts** ? Significa TypeScript, y Angular 6 usa TypeScript. En resumen, TypeScript proporciona una fuerte verificación de tipos en JavaScript.

El archivo **app.module.ts** tiene este aspecto:

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

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

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

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

@NgModule({
 declarations: [ AppComponent ],

 imports: [ BrowserModule,
AppRoutingModule ],

 providers: [],

 bootstrap: [AppComponent] })

export class AppModule { }
```

Cada vez que use la CLI para generar componentes y servicios, actualizará automáticamente este archivo para importarlo y agregarlo al decorador ***@NgModule*** .

```
@NgModule({
 declarations: [ AppComponent ],

 imports: [ BrowserModule,
AppRoutingModule ],

 providers: [],

 bootstrap: [AppComponent] })
```

Los componentes se agregan a la matriz de **`declarations`** .

Los servicios se agregan como **`providers.`**

También se encontrará agregando varias importaciones a la matriz de **`imports.`** Por ejemplo, cuando queremos agregar animaciones, las agregaremos aquí.

Si estás un poco confundido en este punto, no te preocupes. Solo comprenda que este es un archivo importante que deberá visitar de forma rutinaria. La CLI se encargará de las cosas en su mayor parte, especialmente al generar componentes, pero al generar servicios y realizar otras tareas, deberá visitar este archivo. Verás como proceder.

Tutorial de 6 componentes angulares

Un componente en Angular 6 le proporciona los componentes básicos de su aplicación Angular. Cuando utilizamos la CLI angular para generar nuestro proyecto, creó un solo componente.

Cuando utiliza la CLI para generar componentes, crea 4 archivos:

```
> src >app >
  app.component.html
  app.component.scss (or .css)
  app.component.spec.ts
  app.component.ts
```

- El archivo HTML es la plantilla HTML asociada con ese componente.
- El SCSS o CSS son los conjuntos de reglas CSS asociados para ese componente (lo que esté definido en el archivo HTML)
- El archivo .*spec.ts* es para fines de prueba.

- El archivo .*ts* es el archivo de componente real, y es donde probablemente pasará la mayor parte de su tiempo. Define una serie de cosas.

Abra el archivo **app.component.ts** :

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


@Component({
  selector: 'app-root',

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

  styleUrls:
['./app.component.scss'] })



 export class AppComponent {
title = 'app'; }
```

En la parte superior, tenemos nuestras importaciones. Importará otros componentes aquí, junto con

los archivos de servicio. Lo haremos un poco más tarde.

El decorador *@Component* es un objeto con pares de propiedad / valor asociados que define cosas importantes asociadas con este componente. Por ejemplo, verá **selector: 'app-root'** : proporciona a este componente un identificador único que se utiliza en otras áreas de la aplicación.

También puede ver *templateUrl* y *styleUrls* , que le dice a Angular dónde se encuentran los archivos HTML y CSS de este componente.

Aquí se pueden agregar otras propiedades, como animaciones, pero veremos eso más adelante.

Finalmente, tenemos la sección lógica del archivo componente. Aquí es donde se definen las

- propiedades (vemos que el *título* fue definido por la CLI),
- inyección de dependencia y
- métodos.

Ahora que comprende los conceptos básicos de un componente, ¡creemos algunos propios!

En la consola, ejecute:

```
 > ng generate component sidebar
CREATE
src/app/sidebar/sidebar.componen
t.html (26 bytes) CREATE
src/app/sidebar/sidebar.componen
t.spec.ts (635 bytes) CREATE
src/app/sidebar/sidebar.componen
t.ts (274 bytes) CREATE
src/app/sidebar/sidebar.componen
t.scss (0 bytes) UPDATE
src/app/app.module.ts (479
bytes)
```

Aquí, le hemos dicho a la CLI que **genere** un **componente** con el nombre de **barra lateral** . Produce los 4 archivos que creó, junto con el archivo del módulo de la aplicación que actualizó.

Generemos algunos componentes más. Ejecute los siguientes comandos para generar 3 componentes más:

```
 > ng gc posts > ng gc users > ng gc details
```

Ahora, debe tener un total de 5 componentes, 4 de los cuales acabamos de crear nosotros mismos. En breve, verá cómo todos estos comienzan a funcionar entre sí y en relación con la aplicación.

Angular 6 Templado

Digamos, por ejemplo, que queremos que nuestra aplicación en particular tenga una barra lateral con algunos íconos. Esta barra lateral siempre estará presente en la aplicación. El componente de la **barra lateral** es algo que ya generamos con la CLI.

Abra el archivo

src / app / app.component.html .

Verá todo el HTML repetitivo que generó la CLI y, en consecuencia, lo que ve en el navegador por el momento. Elimine todo eso y pegue esto (o mejor aún, ¡escríbalo!):

```
 <div id="container">

  <app-sidebar></app-sidebar>

  <div id="content">

<router-outlet></router-outlet>

  </div>

</div>
```

Hemos envuelto todo en una identificación de **contenedor** .

Luego, notará un elemento HTML personalizado llamado **app-sidebar** . ¿Que es eso?

Bueno, cuando la CLI generó el componente de la *barra lateral* , convirtió el valor del *selector* del componente en la *barra lateral de la aplicación* . No me creas Echa un vistazo a **/src/app/sidebar/sidebar.component.ts**

: ¡está justo en el decorador de componentes! Así es como se incrusta un componente dentro de otro componente.

Ahora, cualquier cosa definida en el HTML de ese componente, se mostrará donde se define

<app-sidebar> </app-sidebar> .

Otro elemento muy importante es el **enrutador de salida** . Esto fue agregado por la CLI cuando agregamos el indicador -*enrutamiento* (también generó un archivo de enrutamiento en

/ src / app .

Este elemento define dónde se mostrarán los componentes definidos por sus rutas.

Pasemos al archivo **/src/app/sidebar/sidebar.component.html**

para definir la plantilla de la barra lateral:

```
 <nav> <ul> <li> <a routerLink=""> <i class="material-icons">supervised_user_circle</i> </a> </li> <li> <a routerLink="posts"> <i class="material-icons">message</i> </a> </li> </ul> </nav>
```

Notará *routerLink* = aquí. En lugar de *href* , usamos *routerLink* para dirigir al usuario a diferentes rutas.

En este momento, esto no funcionará, llegaremos a eso durante la sección de enrutamiento.

También vamos a usar iconos de material , por lo que debemos importar eso primero.

Guarde este archivo y abra

/src/index.html

y agregue las siguientes 2 líneas entre las etiquetas *<head>* :

```
<link
href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">

<link
href="https://fonts.googleapis.com/css?family=Montserrat:300,700"
rel="stylesheet">
```

Primero estamos importando íconos de material y luego una fuente web de Google llamada Montserrat.

Angular 6 CSS

Agreguemos algunos conjuntos de reglas CSS para que nuestra aplicación se vea mejor. Primero, abra

/src/styles.css .

Cualquier CSS / Sass definido aquí se aplicará a la plantilla HTML de todos los componentes, mientras que los archivos CSS específicos del componente solo se aplicarán a la plantilla HTML de ese componente.

Agregue los siguientes conjuntos de reglas:

```
/* You can add global styles to this file, and also import other style files */ body { margin: 0; background: #F2F2F2; font-family: 'Montserrat', sans-serif; height: 100vh; } #container { display: grid; grid-template-columns: 70px auto; height: 100%; #content { padding: 30px 50px; ul { list-style-type: none; margin:0;padding:0; li { background: #fff; border-radius: 8px; padding: 20px; margin-bottom: 8px; a { font-size: 1.5em; text-decoration: none; font-weight: bold; color:#00A8FF; } ul { margin-top: 20px; li { padding:0; a { font-size: 1em; font-weight: 300; } } } } } } }
```

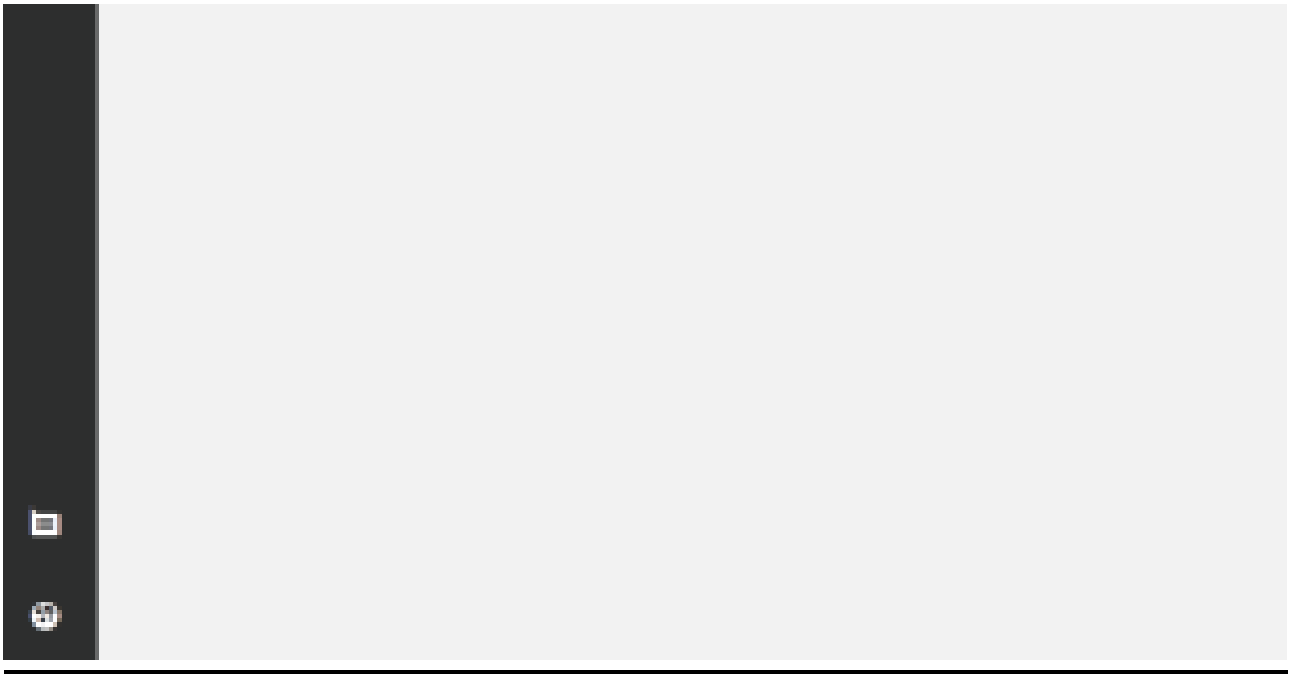
Esto es un poco largo porque no quiero seguir revisando este archivo. Los conjuntos de reglas aquí se aplican a algunos elementos que aún no hemos definido. Sin embargo, aquí no sucede nada demasiado emocionante, solo algunos Sass / CSS estándar.

A continuación, abra el archivo CSS de la barra lateral

/src/app/sidebar/sidebar.component.scss :

```
 nav {
background: #2D2E2E;
height: 100%;
ul { list-style-type: none;
padding: 0; margin: 0;
li { a { color: #fff; padding:
20px; display: block; }
.activated { background-color:
#00a8ff; } } } }
```

Excelente. Vea su navegador y el resultado debería verse así:



Tutorial de enrutamiento de Angular 6

Ahora, hagamos que nuestros 2 iconos funcionen cuando se hace clic en ellos. Para hacer eso, debemos visitar el archivo

/src/app/app-routing.module.ts .

Esto es lo que parece:

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

import { Routes, RouterModule }
from '@angular/router';

const routes: Routes = [];

@NgModule({
  imports:
[RouterModule.forRoot(routes)],

  exports: [RouterModule] })

export class AppRoutingModule {
}
```

Necesitamos importar nuestros componentes en la parte superior y agregarlos a la matriz de *Rutas que se* muestra en la línea 4 anterior.

Para hacer eso, agregue lo siguiente:

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

import { Routes, RouterModule }
from '@angular/router';

 import { UsersComponent } from
'./users/users.component';

import { DetailsComponent } from
'./details/details.component';

import { PostsComponent } from
'./posts/posts.component';
```

```
 const routes: Routes = [ {
path: '',

component: UsersComponent }, {
path: 'details/:id',

component: DetailsComponent }, {
path: 'posts',

component: PostsComponent }, ];
```

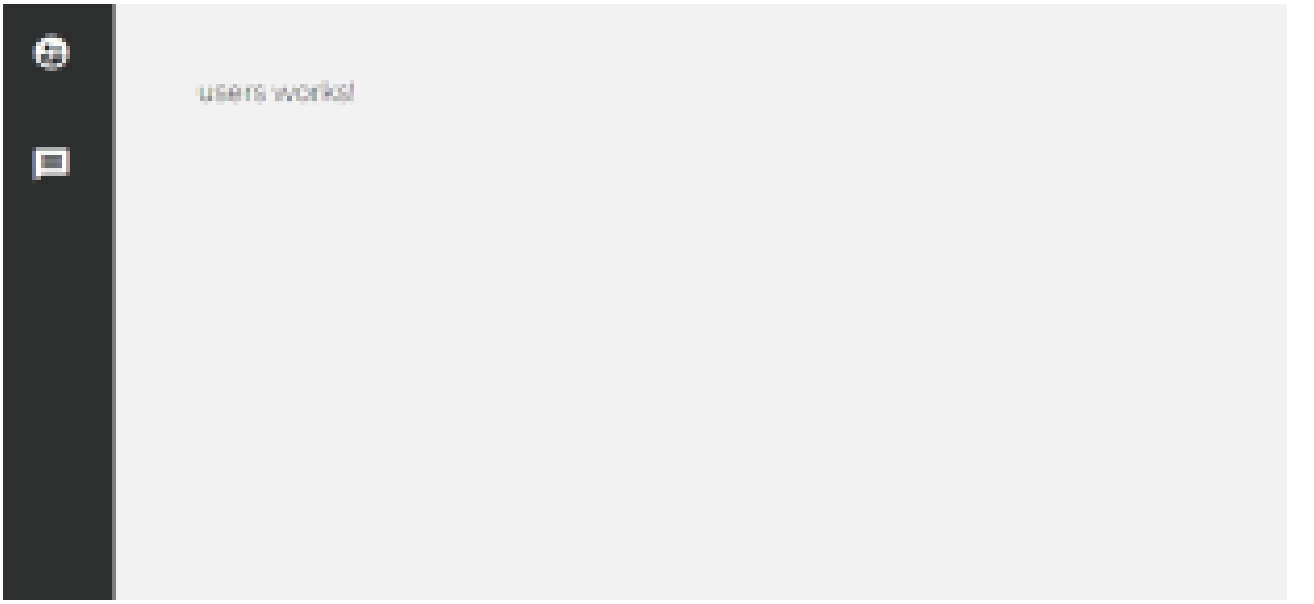
Importamos nuestros 3 componentes y luego definimos tres objetos en la matriz de *Rutas* .

El primer objeto especifica que *UsersComponent* será el componente predeterminado que se *cargará* en la ruta raíz. Dejamos vacío el valor del camino.

La siguiente ruta es para una sección de detalles del usuario. Hemos especificado un comodín llamado *id* . Usaremos esto para obtener ese valor del enrutador para recuperar los detalles correctos del usuario.

Luego, otra ruta para un componente y ruta llamada posts.

Guarde este archivo, y el navegador ahora debería mostrar:



¡Increíble!

Servicios Angulares 6

Para nuestro componente de usuarios, queremos obtener una lista de usuarios de una API pública. Para hacer eso, vamos a usar la CLI angular para generar un servicio para nosotros.

Un servicio Angular 6 es útil para colocar código que sea reutilizable en los diferentes componentes de su aplicación.

En la consola, escriba:

```
> ng generate service data
```

Abra el nuevo archivo de servicio: **/src/app/data.service.ts** :

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

 @Injectable({
providedIn: 'root' })

 export class DataService {
constructor() { } }
```

Se parece a un componente normal, ¿verdad? Defina sus importaciones en la parte superior, y sus métodos y propiedades en la clase que se exporta.

El propósito de nuestro archivo de servicio será comunicarnos con una

API a través del Angular 6 HTTP Client.

Angular 6 HTTP Client

Angular viene con un cliente HTTP incorporado. **Importemos** eso en la parte superior de nuestro archivo **data.service.ts** :

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

import { HttpClient } from
'@angular/common/http';
```

A continuación, para usar el HttpClient, necesitamos crear una instancia a través de la inyección de dependencia dentro del constructor:

```
constructor(private http:
HttpClient) { } getUsers() {
return
this.http.get('https://jsonplace
holder.typicode.com/users') }
```

También definimos un método llamado *getUsers ()* que llamaremos a nuestro componente en breve. Devuelve una lista de usuarios de una API de prueba pública.

Antes de que podamos usar el HTTPClient, necesitamos agregarlo como una importación en el archivo

/src/app/app.module.ts

de nuestra aplicación:

```
// Other imports removed for
brevity

import { HttpClientModule } from
'@angular/common/http';

// <-Add here @NgModule({
declarations: [
// Removed for brevity ],

imports: [ BrowserModule,
AppRoutingModule,
HttpClientModule,
// <-Add here ],
providers: [],
 bootstrap: [AppComponent] })
```

¡Excelente!

A continuación, abramos el archivo

/src/app/users/users.component.ts

e **importemos** nuestro servicio:

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

import { DataService } from
'../data.service';

import { Observable } from
'rxjs';
```

Para mostrar los resultados, vamos a utilizar un Observable, por lo que también lo importaremos aquí.

En la clase, agregue:

```
 export class UsersComponent
implements OnInit { users$:
Object; constructor(private
data: DataService) { }
ngOnInit() {
this.data.getUsers().subscribe(
data => this.users$ = data ); }
}
```

En el constructor, estamos creando una instancia de nuestro servicio. Luego, dentro del *enlace* del ciclo de vida *ngOnInit ()* (esto se ejecuta cuando se carga el componente), llamamos a nuestro método *getUsers ()* y nos suscribimos a él. En el interior, vinculamos a nuestros *usuarios $* object con el resultado devuelto por la API.

A continuación, abra **/src/app/users/users.component.html** :

```
 <h1>Users</h1>
<ul>
<li *ngFor="let user of users$">
 <a
routerLink="/details/{{user.id}}
">{{ user.name }}</a>
       <ul>
<li>{{ user.email }}</li>
<li><a href="http://{{
user.website }}">{{ user.website
}}</a></li>
       </ul>
</li>
</ul>
```

Siempre que desee iterar sobre una matriz o matriz de objetos, utilice la directiva angular * **ngFor** .

¡Luego usamos paréntesis de interpolación para invocar las propiedades del objeto devuelto para mostrarlas en el navegador!

Si guarda esto y actualiza, ahora debería ver una lista de usuarios y su información:

Users

Leanne Graham

Sincere@april.biz

Ervin Howell

Shanna@melissa.tv

Clementine Bauch

Nathan@yesenia.net

Patricia Lebsack

Julianne.OConner@kory.org

Obteniendo más datos de la API

Revisemos el archivo de servicio **/src/app/data.service.ts** y agreguemos los siguientes métodos:

```
  getUser(userId) { return
this.http.get('https://jsonplace
holder.typicode.com/users/'+user
Id) } getPosts() { return
this.http.get('https://jsonplace
holder.typicode.com/posts') }
```

El método *getUser ()* nos proporcionará la información de un solo usuario, que aceptará un *ID de usuario* como parámetro.

getPosts () obtendrá algunas publicaciones ficticias para que podamos obtener más memoria muscular con este proceso de comunicación con los servicios.

Visite **/src/app/details/details.component.ts** :

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

import { DataService } from
'../data.service';

import { Observable } from
'rxjs';

import { ActivatedRoute } from
"@angular/router";

@Component({
selector: 'app-details',
templateUrl:
'./details.component.html',
styleUrls:
['./details.component.scss'] })

export class DetailsComponent
implements OnInit { user$:
Object;
```

```
constructor(
private route: ActivatedRoute,
private data: DataService) {
this.route.params.subscribe(
params => this.user$ = params.id
); } ngOnInit() {
this.data.getUser(this.user$).su
bscribe( data => this.user$ =
data ); } }
```

Esto, como puede ver, es muy similar a nuestro componente de *usuarios* . La única diferencia viene cuando importamos *ActivatedRoute* y lo llamamos dentro del constructor.

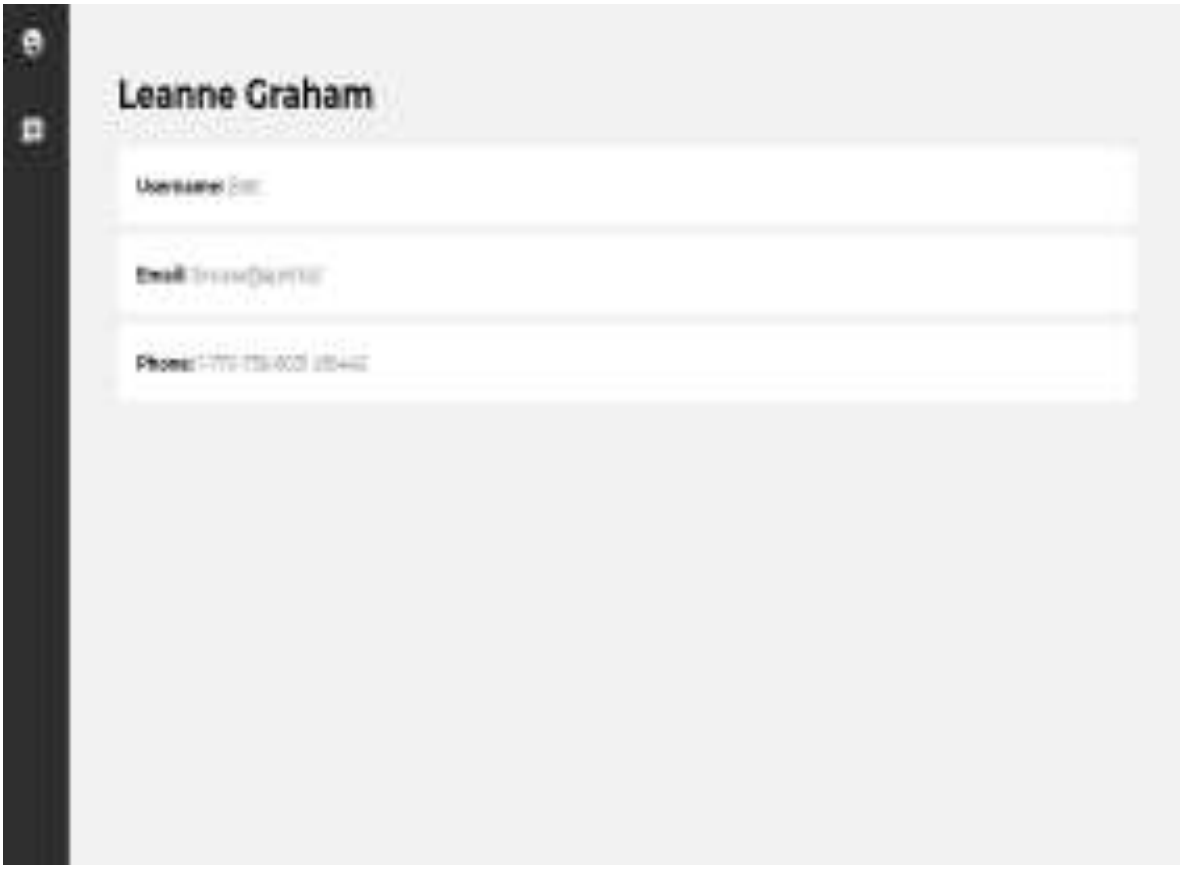
El propósito de este código nos permite obtener el parámetro de enrutador de *identificación* que definimos anteriormente en el

archivo de enrutamiento de la aplicación. Esto nos dará acceso a la ID de usuario y luego la pasará al método *getUser ()* que definimos.

Abra los **detalles.component.html** y especifique:

```
 <h1>{{ user$.name }}</h1>
<ul>
<li><strong>Username:</strong>
{{ user$.username }}</li>

<li><strong>Email:</strong> {{
user$.email }}</li>

<li><strong>Phone:</strong> {{
user$.phone }}</li>
</ul>
```

Guárdelo y haga clic en uno de los nombres de usuario:



¡Increíble!

Para obtener más memoria muscular, repitamos este proceso para el archivo **/src/app/posts/posts.component.ts** :

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

import { DataService } from
'../data.service';

import { Observable } from
'rxjs';

 @Component({ selector: 'app-
posts',

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

styleUrls:
['./posts.component.scss'] })
export class PostsComponent
implements OnInit { posts$:
Object; constructor(private
data: DataService) { }
ngOnInit() {
this.data.getPosts().subscribe(
data => this.posts$ = data ); }
}
```

Y el archivo **posts.component.html** :

```
 <h1>Posts</h1>
<ul>
<li *ngFor="let post of posts$">
<a routerLink="">{{ post.title
}}</a>
<p>{{ post.body }}</p> </li>
</ul>
```

Guárdelo y haga clic en el icono de publicaciones en la barra lateral:

Enlace angular de 6 clases

Sería bueno indicar en qué página se encuentra actualmente un usuario en la barra lateral izquierda, ¿quizás agregando una clase al icono que hará que su fondo sea azul?
¡Seguro!

Visite el archivo **/src/app/sidebar/sidebar.component.ts** y agregue lo siguiente:

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

import { Router, NavigationEnd }
from '@angular/router';

export class SidebarComponent
implements OnInit { currentUrl:
string; constructor(private
router: Router) {
router.events.subscribe((_:
NavigationEnd) =>
this.currentUrl = _.url); }
ngOnInit() {} }
```

Estamos importando Router y NavigationEnd, luego *definimos* una propiedad de cadena *currentUrl* . Luego, creamos una instancia del enrutador para suscribirnos a los *eventos* . Nos proporcionará una

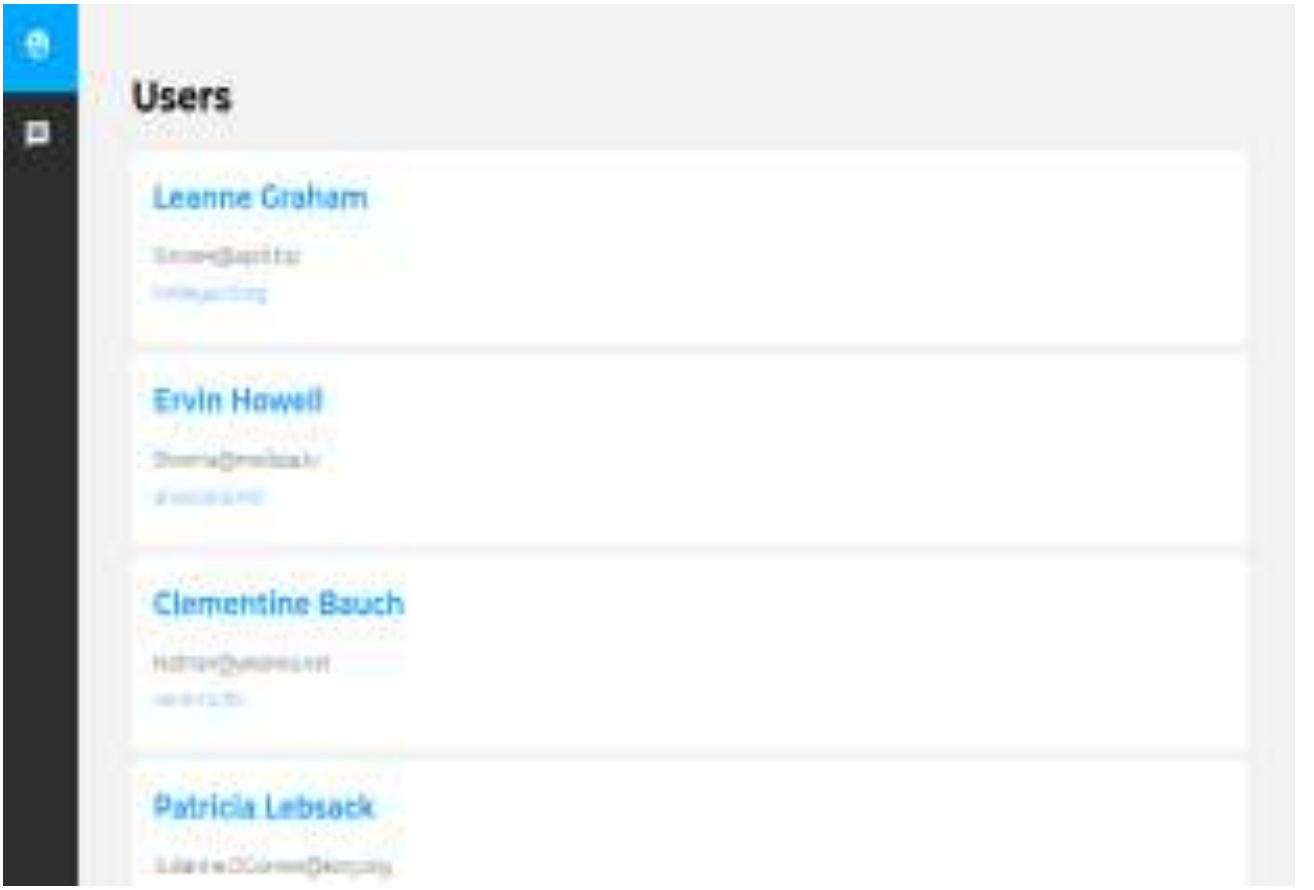
cadena, que es la ruta actual del enrutador.

Abra el archivo **sidebar.component.html** y actualícelo para que coincida con:

```
<nav>

<ul>
      <li>

<a routerLink=""
[class.activated]="currentUrl ==
'/'">
<i class="material-
icons">supervised_user_circle</i
> </a>
      </li>
      <li>
<a routerLink="posts"
[class.activated]="currentUrl ==
'/posts'"> <i class="material-
icons">message</i>
</a>
      </li>
</ul>

</nav>
```

El enlace de clase funciona vinculando *[class.csspropertyname]* a una expresión de plantilla. Solo agregará el conjunto de reglas CSS *activado* (definido en *styles.scss*) si nuestro *currentUrl* es igual a / o */posts* .

Guárdalo El resultado ahora debería verse así:



¡Intenta hacer clic en el otro icono!

Tutorial de animación de Angular 6

Digamos, por ejemplo, que queremos que nuestra lista de usuarios en la página del usuario se desvanezca cuando se carga el componente. Podemos usar la poderosa biblioteca de animación de Angular para ayudarnos a lograr esto.

Para obtener acceso a la biblioteca de animación, primero tenemos que instalarla desde la consola:

```
> npm install
@angular/animations@latest --
save
```

Luego, lo agregamos a las importaciones de **/src/app/app.module.ts** :

```
 // Other imports removed for
brevity

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

 @NgModule({ ... imports: [ //
other modules removed for
brevity BrowserAnimationsModule
], })
```

A continuación, abra **/src/app/users/users.component.ts** y agregue lo siguiente a las principales importaciones:

```
 import {
trigger,style,transition,animate
,keyframes,query,stagger } from
'@angular/animations';
```

Luego, en el decorador de componentes, agregue la siguiente propiedad de animaciones con el código asociado:

```
 @Component({ selector: 'app-
users',
templateUrl:
'./users.component.html',
styleUrls:
['./users.component.scss'], //
Add this: animations: [
```

```
trigger('listStagger', [
transition('* <=> *', [ query(
':enter', [ style({ opacity: 0,
transform: 'translateY(-15px)'
}), stagger( '50ms', animate(
'550ms ease-out', style({
opacity: 1, transform:
'translateY(0px)' }) ) ) ], {
optional: true } ),
query(':leave', animate('50ms',
style({ opacity: 0 })), {
optional: true }) ]) ]) ] })
```

¡Uf, están pasando muchas cosas aquí!

- Comenzamos definiendo una animación dándole un disparador con una lista de nombres *Tagger* .
- A continuación, usamos la *transición* para definir cuándo

tendrán lugar las animaciones, de un estado de animación a otro. Un comodín se usa para decir de cualquier estado a cualquier estado, en este caso.

- Luego, usamos *query* para decir que en *: enter* , aplicamos un estilo inicial que está oculto y se mueve en el eje Y en -15px, y hacemos una animación escalonada para cada elemento secuencial.
- Al final, definimos un opcional *: dejar* animación.

Para que esto funcione, visite el archivo **/src/app/users/users.component.ht**

ml y haga referencia al activador de animación:

```
<ul [@listStagger]="users$">
```

Guárdelo y haga clic en el icono de los usuarios. ¡Notarás que la lista se anima!

La animación angular tiene mucho más, por lo que este es solo un caso de uso potencial.

Conclusión

¡Espero que hayas disfrutado aprendiendo sobre los fundamentos de Angular 6! Te veo pronto.