

Creando una Aplicación con Angular

Daniel Frejo Ballesteros

Index

Parte 1: Creando la Aplicación Base

- Renderizado del Servidor y Cliente
- Instalando Angular CLI
- Generando una Nueva Aplicación
- Ajustando ajustes predeterminados para Angular CLI
- Configurando Estilos Globales
- Creando Módulos UI y Componentes

Index

Parte 2: Creando el Módulo Aplicación y los Componentes

- Tipos de Componente
- Generando Lazy Load y PostsModule
- Creando Componentes Contenedores
- Creando un Servicio para Recopilar Datos
- Creando los Componentes Presentacionales
- Creando Resolutores para Recopilar Datos usando el Router

Index

Parte 3: Renderizado del Servidor

- Generando la Aplicación Servidor
- Añadiendo Dependencias para la Aplicación Servidor
- Implementando un Servidor Web
- Añadiendo Metadatos Dinámicos

Index

Parte 4: Operadores de Servicio

- ¿Que es un Operador de Servicio?
- ¿Que es una Aplicación Web Progresiva?
- Instalando Dependencias
- Activando el Operador de Servicio
- Configurando el Operador de Servicio

Parte 1: Creando la Aplicación Base

- Renderizado del Servidor y Cliente
- Instalando Angular CLI
- Generando una Nueva Aplicación
- Ajustando ajustes predeterminados para Angular CLI
- Configurando Estilos Globales
- Creando Módulos UI y Componentes

1.2 Renderizado del Servidor y Cliente

Cuando hablamos de renderizado de Servidor para sitios web, generalmente nos referimos a una aplicación o sitio web que utiliza un lenguaje de programación que se ejecuta en un servidor. En dicho servidor las páginas web se crean(renderizan) y la salida de este renderizado(HTML) se envía al navegador donde se observa directamente.

El renderizado de Cliente generalmente se refiere a una aplicación o sitio web que utiliza JavaScript ejecutándose en el navegador para renderizar las páginas.

1.2 Instalando Angular CLI

En este curso nos centraremos en construir una aplicación en Angular que funcione como un sitio web público, utilizando para ello datos de una REST API.

Para ello además de angular necesitaremos instalar una API específica de desarrollo, la cual instalaremos ejecutando la siguiente secuencia de comandos en un terminal:

```
$ git clone <repo_url>  
$ cd packt-angular-seo-api  
$ npm install  
$ npm start
```


1.2 Instalando Angular CLI

Una vez tenemos instalada la API, procederemos a la instalación de Angular, para lo cual ejecutaremos la siguiente secuencia de instrucciones en una terminal. Si la instalación se realiza con éxito, la terminal nos ofrecerá una salida similar a la vista en la imagen de la derecha

```
npm install -g @angular/cli@latest  
ng --version
```

A terminal window with a black background and white text. The first command is 'npm install -g @angular/cli@latest', which outputs the installation path for 'ng' and the number of packages added/updated. The second command is 'ng --version', which outputs the Angular CLI version (1.6.3), Node version (8.9.1), OS (darwin x64), and Angular version (...). The 'Angular CLI' text is displayed in a large, stylized red font.

```
> npm install -g @angular/cli@latest  
/usr/local/bin/ng -> /usr/local/lib/node_modules/@angular/cli/bin/ng  
+ @angular/cli@1.6.3  
added 19 packages and updated 28 packages in 40.485s  
  
~ 41s  
> ng --version  
  
Angular CLI  
  
Angular CLI: 1.6.3  
Node: 8.9.1  
OS: darwin x64  
Angular:  
...  
>
```

1.3 Generando una Nueva Aplicación

Para crear una nueva aplicación simplemente abriremos una terminal y nos dirigiremos a la ruta donde queramos guardar los archivos de la misma. Una vez situados en la ruta deseada simplemente ejecutaremos la siguiente orden:

```
ng new angular-social --routing
```

1.3 Generando una Nueva Aplicación

Si la orden se ejecutó correctamente deberíamos ver una salida similar a la siguiente:

```
~/dev
$ ng new angular-social --routing
create angular-social/README.md (1029 bytes)
create angular-social/.angular-cli.json (1249 bytes)
create angular-social/.editorconfig (245 bytes)
create angular-social/.gitignore (516 bytes)
create angular-social/src/assets/.gitkeep (0 bytes)
create angular-social/src/environments/environment.prod.ts (51 bytes)
create angular-social/src/environments/environment.ts (387 bytes)
create angular-social/src/favicon.ico (5430 bytes)
create angular-social/src/index.html (300 bytes)
create angular-social/src/main.ts (370 bytes)
create angular-social/src/polyfills.ts (2405 bytes)
create angular-social/src/styles.css (80 bytes)
create angular-social/src/test.ts (1085 bytes)
create angular-social/src/tsconfig.app.json (211 bytes)
create angular-social/src/tsconfig.spec.json (304 bytes)
create angular-social/src/typings.d.ts (104 bytes)
create angular-social/e2e/app.e2e-spec.ts (296 bytes)
create angular-social/e2e/app.po.ts (208 bytes)
create angular-social/e2e/tsconfig.e2e.json (235 bytes)
create angular-social/karma.conf.js (923 bytes)
create angular-social/package.json (1326 bytes)
create angular-social/protractor.conf.js (722 bytes)
create angular-social/tsconfig.json (363 bytes)
create angular-social/tslint.json (3040 bytes)
create angular-social/src/app/app-routing.module.ts (245 bytes)
create angular-social/src/app/app.module.ts (395 bytes)
create angular-social/src/app/app.component.css (0 bytes)
create angular-social/src/app/app.component.html (1173 bytes)
create angular-social/src/app/app.component.spec.ts (1103 bytes)
create angular-social/src/app/app.component.ts (207 bytes)
Installing packages for tooling via npm.
Successfully initialized git.
Project 'angular-social' successfully created.
```

```
~/dev 43s
$
```

- src: En esta carpeta se almacenan los archivos fuente de nuestra aplicación.
- src/app/: Esta carpeta contiene los archivos de la aplicación.
- src/assets/: Esta carpeta contiene los archivos estáticos que podemos utilizar (como imágenes).
- src/environments/: Esta carpeta contiene la definición de los entornos predeterminados de nuestra aplicación.

1.3 Generando una Nueva Aplicación

Una vez hemos generado nuestra aplicación, tendremos que ejecutarla, para servirla simplemente tendremos que ir a la ruta donde hayamos instalado todos los archivos y ejecutar el comando `ng serve` cuya salida se muestra en la siguiente imagen.

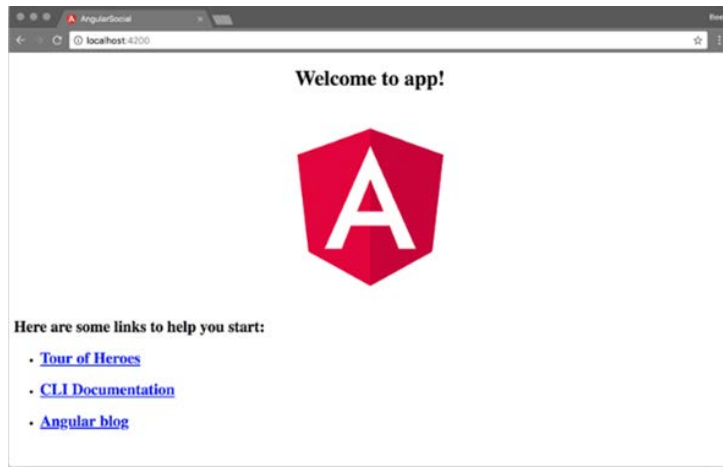
```
~/dev
> cd angular-social

~/dev/angular-social master
> ng serve
++ NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ ++
Date: 2018-01-11T11:48:20.737Z
Hash: b951f9869710a5db5a7f
Time: 6283ms
chunk {inline} inline.bundle.js (inline) 5.79 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 22.4 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 552 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 33.8 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 8.43 MB [initial] [rendered]

webpack: Compiled successfully.
```

1.3 Generando una Nueva Aplicación

Por último, para visualizar nuestra aplicación web simplemente abriremos nuestro navegador y nos dirigiremos a: `http://localhost:4200/` donde podremos ver la siguiente página:



1.4 Ajustando ajustes predeterminados para Angular CLI

Los ajustes predeterminados que Angular nos proporciona nos otorgan una muy buena base para empezar a desarrollar nuestra aplicación sin embargo a modo de ejemplo nosotros cambiaremos algunos de estos ajustes. Esta tarea será tan sencilla como ejecutar en un terminal las siguientes instrucciones:

```
ng set defaults.component.inlineStyle true
ng set defaults.component.inlineTemplate true
git diff
```

1.5 Configurando Estilos Globales

A nuestra aplicación podemos además aplicarle diferentes hojas de estilo. Por defecto encontramos el archivo `src/styles.css` donde usaremos el comando `@import` para enlazar a Bootstrap y Font Awesome, para ello abriremos este archivo(`styles.css`) con un editor de texto y añadiremos las siguientes líneas al final del archivo:

```
@import url('https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/  
bootstrap.min.css');  
@import url('https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/  
font-awesome.min.css');
```

Y refrescaremos la aplicación en nuestro navegador.

1.6 Creando Módulos UI y Componentes

Para generar nuestro módulo UI utilizaremos el comando `ng` e importaremos el módulo UI en el módulo aplicación.

Para generar el módulo UI abriremos una terminal donde ejecutaremos la siguiente orden:

```
$ ng generate module ui  
  create src/app/ui/ui.module.ts (186 bytes)
```


1.6 Creando Módulos UI y Componentes

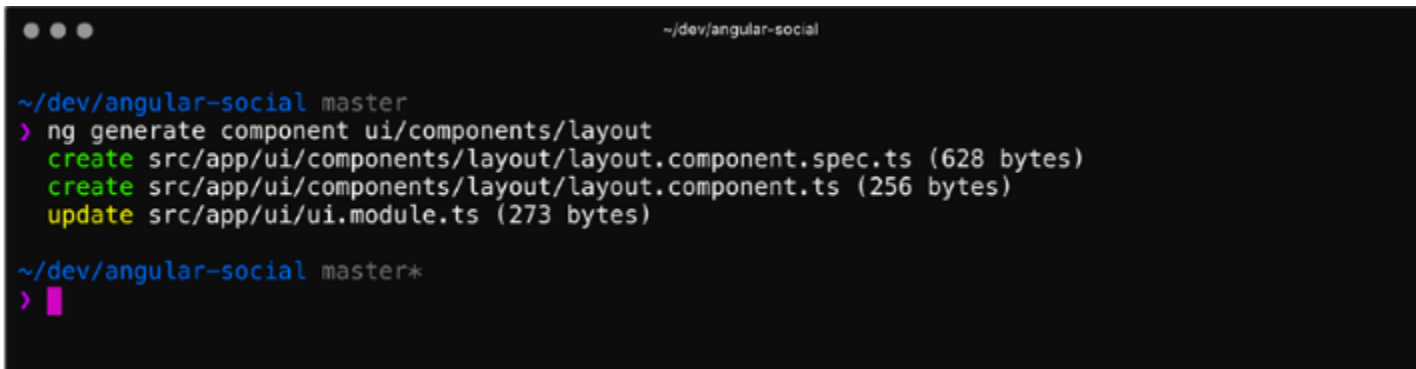
Una vez hemos creado nuestro Módulo UI, deberemos importarlo desde el Módulo Aplicación, para ello abriremos el archivo `src/app/app.module.ts` con un editor de texto y añadiremos la siguiente sentencia “import” al inicio del archivo además de una referencia al módulo en el array “imports” de NgModule.

```
import { UiModule } from './ui/ui.module'
```

```
@NgModule({  
  ...  
  imports: [  
    // other imports  
    UiModule  
  ],  
  ...  
})
```

1.6 Creando Módulos UI y Componentes

Ahora crearemos un `LayoutComponent` dentro de nuestro Módulo UI, para ello utilizaremos nuevamente el comando `ng generate`. Para ello simplemente nos dirigiremos de nuevo a una terminal y ejecutaremos la instrucción `ng generate component ui/components/layout` que nos mostrará una salida similar a la mostrada en la imagen.

A terminal window with a dark background and light-colored text. The title bar at the top shows three window control buttons on the left and the path `~/dev/angular-social` on the right. The terminal content shows a prompt `~/dev/angular-social master` followed by the command `> ng generate component ui/components/layout`. The output consists of three lines: `create src/app/ui/components/layout/layout.component.spec.ts (628 bytes)`, `create src/app/ui/components/layout/layout.component.ts (256 bytes)`, and `update src/app/ui/ui.module.ts (273 bytes)`. Below the output, the prompt changes to `~/dev/angular-social master*` and a new prompt `>` is shown with a cursor.

```

~/dev/angular-social master
> ng generate component ui/components/layout
create src/app/ui/components/layout/layout.component.spec.ts (628 bytes)
create src/app/ui/components/layout/layout.component.ts (256 bytes)
update src/app/ui/ui.module.ts (273 bytes)

~/dev/angular-social master*
> █

```

1.6 Creando Módulos UI y Componentes

El LayoutComponent previamente creado será el que utilizemos como base para construir nuestra aplicación. Para ello deberemos modificar las rutas predeterminadas impuestas por Angular. Para ello editaremos `src/app/app-routing.module.ts` de la

siguiente forma:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LayoutComponent } from '../ui/components/layout/layout.component'

const routes: Routes = [
  { path: '', component: LayoutComponent, children: [] },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

1.6 Creando Módulos UI y Componentes

A continuación empezaremos a construir nuestro layout, para ello abriremos el archivo `src/app/ui/layout/layout.component.ts` y en la sección “template” añadiremos el siguiente contenido:

```
app-header placeholder
<div class="container my-5">
  <router-outlet></router-outlet>
</div>
app-footer placeholder
```

1.6 Creando Módulos UI y Componentes

Para que Angular tenga consciencia sobre cómo renderizar el 'router-outlet' necesitaremos importar el Módulo Router, para ello abriremos el archivo `src/app/ui/ui.module.ts`

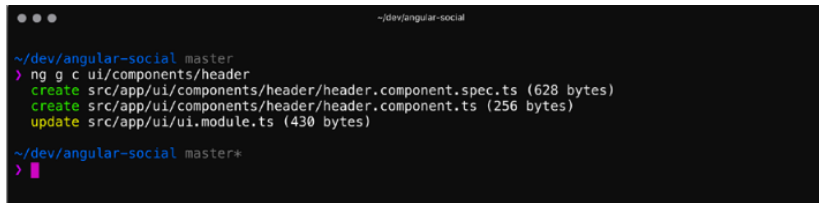
e importaremos el módulo al inicio del fichero y, de manera similar a la vista previamente, añadiremos una referencia a este módulo en el array 'imports'

```
import { RouterModule } from '@angular/router';
```

1.6 Creando Módulos UI y Componentes

Para continuar creando los componentes necesarios, ahora crearemos el componente que actuará como cabecera, para ello utilizaremos nuevamente la instrucción “ng generate” por lo que nos dirigiremos desde una terminal al directorio del proyecto y ejecutaremos la siguiente orden:

```
ng g c ui/components/header
```

A terminal window with a dark background and light-colored text. The window title is '~dev/angular-social'. The prompt is '~dev/angular-social master'. The command 'ng g c ui/components/header' has been entered. The output shows three lines: 'create src/app/ui/components/header/header.component.spec.ts (628 bytes)' in green, 'create src/app/ui/components/header/header.component.ts (256 bytes)' in green, and 'update src/app/ui/ui.module.ts (430 bytes)' in yellow. The prompt is now '~dev/angular-social master*' and the cursor is on a new line.

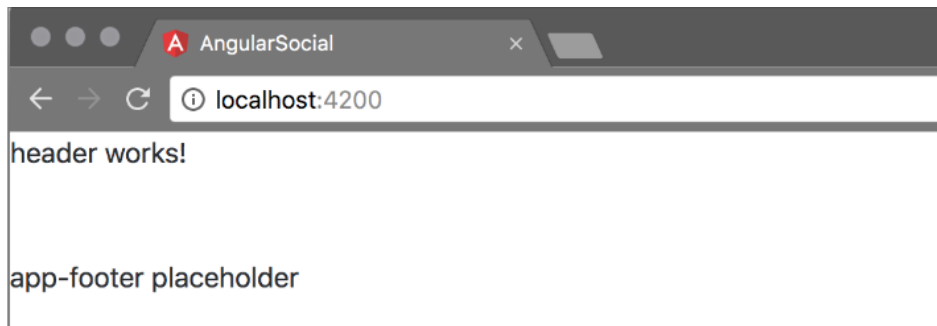
```
~dev/angular-social master
> ng g c ui/components/header
create src/app/ui/components/header/header.component.spec.ts (628 bytes)
create src/app/ui/components/header/header.component.ts (256 bytes)
update src/app/ui/ui.module.ts (430 bytes)
~/dev/angular-social master*
>
```

1.6 Creando Módulos UI y Componentes

Una vez creado nuestro HeaderComponent deberemos referenciarlo desde nuestro LayoutComponent, para ello abriremos el archivo `src/app/ui/components/layout/layout.component.ts` y

buscaremos la etiqueta `'app-header'` y la cambiaremos por `<app-header></app-header>` para

que se renderice correctamente. Al recargar nuestro proyecto en el navegador nos deberá mostrar por pantalla lo siguiente:



1.6 Creando Módulos UI y Componentes

Ahora que ya tenemos la cabecera referenciada procederemos a la creación de nuestra cabecera. Para ello definiremos 3 propiedades, una de tipo string para el logo y el título de la aplicación y un array de objetos representando los links que mostraremos en nuestra cabecera. para ello lo primero que haremos será descargar el archivo `https://angular.io/assets/images/logos/angular/angular.svg` `src/assets/logo.svg`.

y guardarlo en

`src/app/ui/components/header/header.component.ts`

Ahora deberemos abrir el archivo

donde definiremos las tres propiedades previamente citadas añadiendo las siguientes

líneas:

```
public logo = 'assets/logo.svg';  
public title = 'Angular Social';  
public items = [{ label: 'Posts', url: '/posts' }];
```


1.6 Creando Módulos UI y Componentes

Ahora que tenemos las propiedades definidas deberemos reemplazar el contenido de la propiedad 'template' por lo siguiente:

```
<nav class="navbar navbar-expand navbar-dark bg-dark">
  <a class="navbar-brand" routerLink="/">
    <img [src]="logo" width="30" height="30" alt="">
  </a>
  <div class="collapse navbar-collapse">
    <ul class="navbar-nav">
      <li class="nav-item" *ngFor="let item of items"
routerLinkActive="active">
        <a class="nav-link" [routerLink]="item.url">{{item.label}}</a>
      </li>
    </ul>
  </div>
</nav>
```

1.6 Creando Módulos UI y Componentes

Por último en esta sección crearemos el FooterModule que será el tercer componente principal de nuestra aplicación. Para su creación utilizaremos nuevamente la instrucción “ng generate” para lo cual nos dirigiremos mediante la terminal al directorio de nuestro proyecto y ejecutaremos `$ ng g c ui/components/footer`

Una vez creado, deberemos referenciarlo desde nuestro `src/app/ui/components/layout/layout.component.ts` con el HeaderComponent. Para ello editaremos `<app-footer></app-footer>` donde cambiaremos la etiqueta ‘app-footer’ por

1.6 Creando Módulos UI y Componentes

Finalmente crearemos nuestro Footer personalizado, para ello nos dirigiremos al archivo `src/app/ui/components/footer/footer.component.ts` donde definiremos, por ejemplo, dos variables con nuestro nombre y el año de creación de nuestro

```
public developer = 'YOUR_NAME_PLACEHOLDER';  
public year = 'YEAR_PLACEHOLDER';
```

Deberemos modificar también la sección 'template' con el siguiente contenido:

```
<nav class="navbar fixed-bottom navbar-expand navbar-dark bg-dark">  
  <div class="navbar-text m-auto">  
    {{developer}} <i class="fa fa-copyright"></i> {{year}}  
  </div>  
</nav>
```

Parte 2: Creando el Módulo Aplicación y los Componentes

- Tipos de Componente
- Generando Lazy Load y PostsModule
- Creando Componentes Contenedores
- Creando un Servicio para Recopilar Datos
- Creando los Componentes Presentacionales
- Creando Resolutores para Recopilar Datos usando el Router

2.1 Tipos de Componente

Distinguimos dos tipos de componente: Contenedor(o smart) y Presentacional(o dumb).

A grandes rasgos podríamos decir que los contenedores se encargan de “cómo funcionan las cosas” mientras que los componentes presentacionales se encargan de la apariencia.

2.1 Tipos de Componente.Presentacionales

Algunas de las características de estos componentes son:

- Obtienen los datos utilizando el decorador @Input()
- Cualquier operación se puede realizar con el decorador @Output()
- Dirigen el estilo de la aplicación
- Mayormente contienen únicamente otros componentes presentacionales
- No tienen conocimiento de rutas o servicios de la aplicación

Para su correcta distinción, en nuestro proyecto guardaremos los componentes presentacionales en la ruta `src/<module>/components`

2.1 Tipos de Componente.Contenedores

Algunas de las características de estos componentes son:

- Obtienen datos de un servicio o resolutor
- Manejan las operaciones que reciben de los componentes presentacionales
- Tienen muy poco margen y estilo
- Normalmente contendrán tanto componentes contenedores como presentacionales

Para su correcta distinción, en nuestro proyecto guardaremos los componentes contenedores en la ruta

```
src/<module>/containers
```

2.2 Generando Lazy Load y PostsModule

Para generar estos módulos utilizaremos el ya conocido comando “ng generate” para lo que nos dirigiremos mediante una terminal al directorio de nuestro proyecto donde ejecutaremos la orden `ng g m posts --routing` ahora deberemos añadir las rutas pertinentes a nuestro nuevo módulo, para ello a diferencia de lo realizado con el módulo UI, añadiremos 2 rutas, una con una ruta por defecto que nos redirigirá a ‘/posts’ y una segunda ruta que ejecutara el ‘lazy load’ de nuestro PostsModule.

2.2 Generando Lazy Load y PostsModule

Para añadir las rutas, abriremos el archivo `src/app/app-routing.module.ts`

y buscaremos la única ruta existente, definida bajo la propiedad 'routes' donde crearemos 2 nuevas rutas de la siguiente manera:

```
const routes: Routes = [  
  { path: '', component: LayoutComponent, children: [  
    { path: '', redirectTo: '/posts', pathMatch: 'full'},  
    { path: 'posts', loadChildren: './posts/posts.module#PostsModule' },  
  ] },  
];
```

2.3 Creando Componentes Contenedores

Nuevamente utilizaremos el comando “ng generate” para la creación de nuestras nuevas componentes, para ello nos dirigiremos desde una terminal a nuestro directorio y ejecutaremos la instrucción `ng g c posts/containers/posts`

y modificaremos la ruta editar `src/app/posts/posts-routing.module.ts`

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

```
{ path: '', component: PostsComponent },
```

y

añadiremos al array “routes” la línea

2.3 Creando Componentes Contenedores

De manera similar a la creación del PostsComponent, crearemos el ProfileComponent que será el encargado de mostrar el perfil que realiza el post. cada perfil se identificara con un 'ID' y las rutas de estos componentes se almacenará en `posts/<id>`. De igual manera que antes,

ejecutaremos en el directorio `ng g c posts/containers/profile` la siguiente acción

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

y editaremos el archivo

donde importaremos el

ProfileComponent que acabamos de crear:

```
import { ProfileComponent } from '../containers/profile/profile.component';

path: ':profileId', component: ProfileComponent },
```

Y añadiremos al array 'routes' la siguiente ruta:

2.3 Creando Componentes Contenedores

Para tener una intuición acerca del funcionamiento de nuestra aplicación, añadiremos datos ficticios para tener algo con que trabajar, para ello añadiremos un servicio para extraer datos de nuestra API.

Con este fin, deberemos abrir el archivo `src/app/posts/containers/posts/posts.component.ts`

donde crearemos una nueva propiedad `public posts = []` 'posts' con la estructura e

`import { ActivatedRoute } from '@angular/router';` siguiente:

Para crear datos ficticios, crearemos 10 perfiles, para lo cual localizaremos el metodo 'ngOnInit()' y añadiremos el siguiente segmento de código:

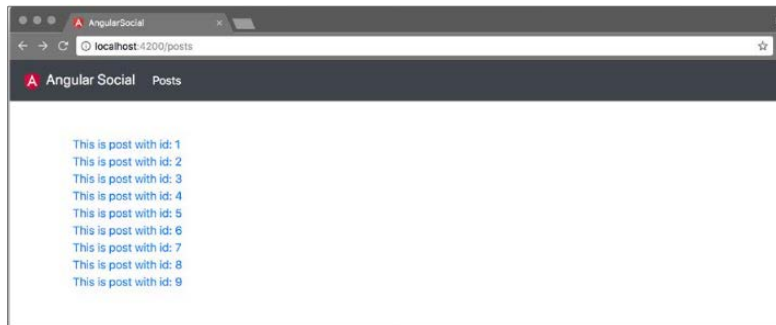
```
for(let i = 1; i < 10; i++) {  
  this.posts.push({ id: i, text: 'This is post with id: ' + i })  
}
```

2.3 Creando Componentes Contenedores

Finalmente deberemos actualizar el contenido de la propiedad 'template' el cual reemplazaremos por:

```
<div *ngFor="let post of posts">
  <a [routerLink]="post.id">
    {{post.text}}
  </a>
</div>
```

Al refrescar nuestra aplicación en el navegador deberíamos observar lo siguiente:



2.3 Creando Componentes Contenedores

Finalmente, para añadir nuestros datos ficticios al ProfileComponent abriremos el archivo

`src/app/posts/containers/profile/profile.component.ts`

donde crearemos una nueva `public profile = { id: null };` definida de la forma :

```
import { ActivatedRoute } from '@angular/router';
```

archivo:

Una vez tenemos importado el módulo router y definida la variable 'profile' deberemos localizar el método

```
'constructor' {  
  this.route.params.subscribe(res => this.profile.id = 'profileId = ' + res['profileId'])  
}
```

Finalmente

2.3 Creando Componentes Contenedores

Por último reemplazaremos el contenido de la propiedad 'template' por la siguiente etiqueta:

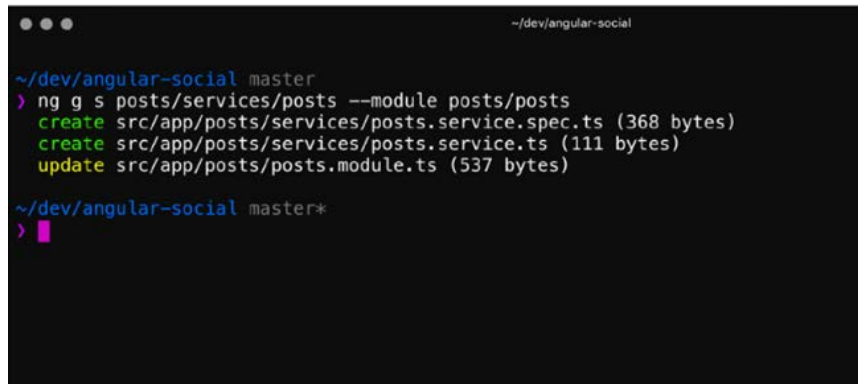
```
<p>
  {{profile.id}}
</p>
```

Para asegurarnos de que funciona correctamente, nos dirigiremos a la dirección `http://localhost:4200/posts/5`, que debería mostrarnos una página similar a la mostrada en la imagen.



2.4 Creando un Servicio para Recopilar Datos

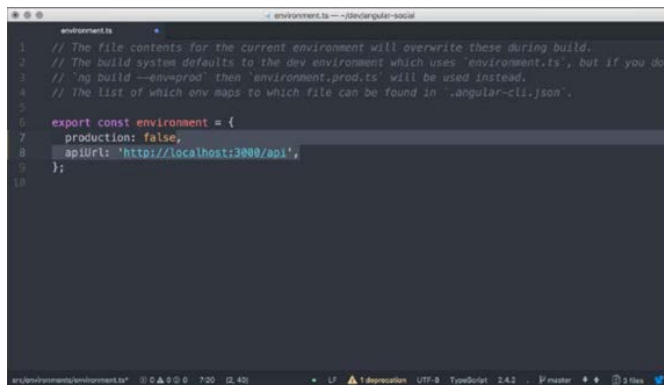
En esta sección crearemos un servicio para interactuar con nuestra API, para ello utilizaremos el conocido comando “ng generate” para crear un servicio, para ello nos dirigiremos con una terminal al directorio de nuestra aplicación y ejecutaremos la instrucción: `ng g s posts/services/posts --module posts/posts` que nos mostrará una salida como la que se muestra en la imagen.

A terminal window with a dark background and light-colored text. The title bar at the top shows three window control buttons and the text '~ /dev/angular-social'. The terminal content shows the prompt '~ /dev/angular-social master' followed by the command 'ng g s posts/services/posts --module posts/posts'. The output consists of three lines: 'create src/app/posts/services/posts.service.spec.ts (368 bytes)' in green, 'create src/app/posts/services/posts.service.ts (111 bytes)' in green, and 'update src/app/posts/posts.module.ts (537 bytes)' in yellow. The prompt returns to '~ /dev/angular-social master*' with a cursor on the next line.

```
~ /dev/angular-social master
> ng g s posts/services/posts --module posts/posts
create src/app/posts/services/posts.service.spec.ts (368 bytes)
create src/app/posts/services/posts.service.ts (111 bytes)
update src/app/posts/posts.module.ts (537 bytes)
~/dev/angular-social master*
> █
```


2.4 Creando un Servicio para Recopilar Datos

Ahora utilizaremos el entorno de Angular CLI para guardar la URL de nuestra API. Por defecto la aplicación generada por Angular CLI contiene 2 entornos predeterminados que se encuentran definidos en el archivo `“.angular-cli.json”` en la raíz del proyecto. Para añadir nuestra URL, abriremos el fichero `src/environments/environment.ts` y lo modificaremos de forma similar a la que se muestra en la imagen.



```
environment.ts
1 // The file contents for the current environment will overwrite these during build.
2 // The build system defaults to the dev environment which uses 'environment.ts', but if you do
3 // 'ng build --env=prod' then 'environment.prod.ts' will be used instead.
4 // The list of which env maps to which file can be found in '.angular-cli.json'.
5
6 export const environment = {
7   production: false,
8   apiUrl: 'http://localhost:3000/api',
9 };
10
```

2.4 Creando un Servicio para Recopilar Datos

A continuación editaremos el archivo `src/environments/environment.prod.ts` de la manera que se muestra en la siguiente imagen:

A screenshot of a code editor window titled 'environment.prod.ts' with a file icon on the left. The editor shows the following TypeScript code:

```
1 export const environment = {  
2   production: true,  
3   apiUrl: 'https://packt-angular-social.now.sh/api',  
4 };  
5
```

The code is syntax-highlighted, with 'export const environment' in blue, 'production: true' in green, and the URL in red. Line numbers 1 through 5 are visible on the left margin. The bottom status bar shows the file path 'src/environments/environment.prod.ts', icons for file operations, and details like 'UTF-8', 'TypeScript', '2.4.2', 'master', and '3 files'.

2.4 Creando un Servicio para Recopilar Datos

El siguiente paso a seguir será referenciar nuestro PostsService en nuestros componentes Contenedores, para ello haremos lo mismo para el PostsComponent y el ProfileComponent.

En primer lugar abriremos el archivo `src/app/posts/containers/posts/posts.component.ts`

donde importaremos el PostsService `import { PostsService } from '../../../services/posts.service';`

al inicio del archivo, a continuación deberemos actualizar el constructor añadiendo la línea “private postsService: PostsService” como parámetro. Finalmente reemplazaremos el contenido de `ngOnInit()` de la siguiente manera:

```
ngOnInit() {  
  this.postsService.getPosts()  
    .map(res => res['items'])  
    .subscribe((result: any) => this.posts = result)  
}
```

2.4 Creando un Servicio para Recopilar Datos

De forma análoga a la anterior, editaremos el archivo `src/app/posts/containers/posts/profile.component.ts`

donde importaremos el `PostsService` al inicio del archivo, y añadiremos al

método `constructor()` como parámetros las sentencias *“private route: ActivatedRoute, private postsService: PostsService”*.

Por último reemplazaremos el contenido del método `ngOnInit` por lo siguiente:

```
this.postsService.getProfile(this.route.snapshot.params['profileId'])  
  .subscribe((result: any) => this.profile = result)
```

2.4 Creando un Servicio para Recopilar Datos

El siguiente paso será definir los métodos públicos en nuestro PostsService para cercionarnos de que podremos acceder a los datos que necesitamos de nuestra API, para ello modificaremos el archivo

`src/app/posts/services/posts.service.ts`

donde importaremos una referencia

al entorno previamente creado(donde definimos la URL a nuestra API) y el cliente http:

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

Posteriormente actualizaremos los parámetros del método 'constructor()' que deberá quedar de la siguiente forma:

```
constructor(private http: HttpClient) { }
```

2.4 Creando un Servicio para Recopilar Datos

A continuación crearemos un método al que llamaremos 'getPosts()' de la siguiente forma:

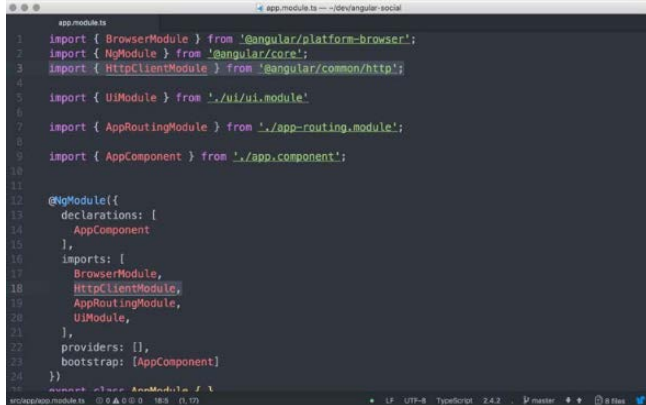
```
getPosts() {  
  const url = `${environment.apiUrl}/posts/timeline?filter[where]  
[type]=text`  
  return this.http.get(url)  
}
```

Por último crearemos el método 'getProfile()' como se muestra abajo:

```
getProfile(profileId) {  
  const url = `${environment.apiUrl}/profiles/${profileId}?filter[inclu  
de]=posts`  
  return this.http.get(url)  
}
```

2.4 Creando un Servicio para Recopilar Datos

Cabe la posibilidad de que al importar el módulo `http` se produzca un error en angular, para solucionar este error abriremos el archivo `src/app/app.module.ts` cuyo contenido se muestra en la imagen.



```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { HttpClientModule } from '@angular/common/http';
4
5 import { UIModule } from './ui/ui.module';
6
7 import { AppRoutingModule } from './app-routing.module';
8
9 import { AppComponent } from './app.component';
10
11
12 @NgModule({
13   declarations: [
14     AppComponent
15   ],
16   imports: [
17     BrowserModule,
18     HttpClientModule,
19     AppRoutingModule,
20     UIModule,
21   ],
22   providers: [],
23   bootstrap: [AppComponent]
24 })
25 export class AppModule {}
```

2.5 Creando los Componentes Presentacionales

En esta sección crearemos los componentes presentacionales de nuestra aplicación, para ello en primer lugar crearemos el componente 'PostListComponent' el cual será llamado desde nuestro 'PostsComponent'.

Para crear este componente deberemos actualizar el campo 'template' del fichero `src/app/posts/container/posts/posts.component.ts`

```
<app-post-list [posts]="posts"></app-post-list>
```

. Una vez hayamos editado este fichero, deberemos dirigirnos

al fichero `src/app/posts/components/post-list/post-list.component.ts` de nuestro proyecto donde ejecutaremos el comando `ng g c posts/components/post-list`

. Finalmente deberemos editar el archivo

```
@Input() posts: any[]
```

donde añadiremos a la ya existente sentencia

'import' el módulo 'Input' de '@angular/core' y añadiremos la propiedad

```
<div *ngFor="let post of posts" class="mb-3">  
  <app-post-item [post]="post"></app-post-item>  
</div>
```


2.5 Creando los Componentes Presentacionales

A continuación crearemos el componente 'PostItemComponent', para ello nos dirigiremos nuevamente desde una terminal a directorio raíz de nuestro proyecto donde ejecutaremos la orden:

```
ng g c posts/components/post-item
```

Una vez generado nuestro componente

src/app/posts/components/post-item/post-item.component.ts donde añadiremos (como hicimos anteriormente) el 'import' del módulo 'Input' del módulo '@Input()' posts: any[] ar/core' y añadiremos la propiedad y actualizaremos la sección 'template' tal y como se muestra en

la siguiente diapositiva

2.5 Creando los Componentes Presentacionales

```
<!-- The row and the col make sure the content is always centered -->
<div class="row">
  <div class="col-md-8 offset-md-2">
    <!-- The card is where the message is shown -->
    <div class="card">
      <div class="card-body">
        <!-- We use the Bootstrap 'media' component to show an avatar
with content -->
        <div class="media">
          <img class="avatar mr-3 rounded" [attr.src]="post?.profile?.
avatar">
          <div class="media-body">
            <!-- The full name of the author is used to navigate to the
post detail -->
            <h5>
              <a [routerLink]="post?.profile?.id"> {{post?.profile?.
fullName}} </a>
              <span class="date float-right text-muted">

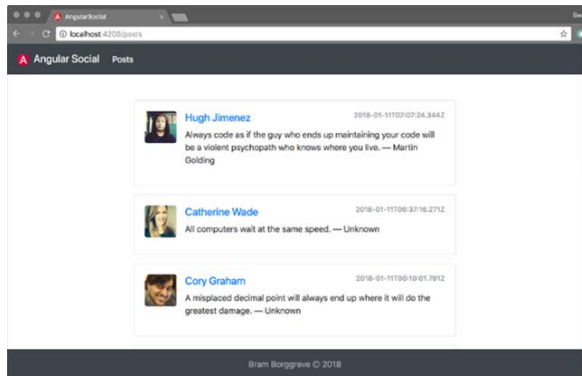
                </span>
            </h5>
            <!-- The text of the post is shown in a simple paragraph
tag-->
            <p>{{post?.text}}</p>
          </div>
        </div> <!-- End media -->
      </div>
    </div> <!-- End card -->
  </div>
</div> <!-- End row-->
```

2.5 Creando los Componentes Presentacionales

Finalmente actualizaremos la propiedad 'styles' de nuestro componente de manera que quede de la siguiente forma:

```
styles: [`  
  img.avatar {  
    height: 60px;  
    width: 60px;  
  }  
  span.date {  
    font-size: small;  
  }  
`],
```

Ahora al recargar nuestra aplicación nos deberá mostrar una página como la vista en la imagen.



2.5 Creando los Componentes Presentacionales

A continuación crearemos el componente 'ProfileItemComponent', para ello de manera similar a lo realizado anteriormente, editaremos `src/app/posts/container/posts/profile.component.ts`

donde deberemos `<app-profile-item [profile]="profile"></app-profile-item>` }
.

y nos dirigiremos nuevamente desde

una terminal a directorio

```
ng g c posts/components/profile-item
  create src/app/posts/components/profile-item/profile-item.
component.spec.ts (664 bytes)
  create src/app/posts/components/profile-item/profile-item.component.ts
(273 bytes)
  update src/app/posts/posts.module.ts (846 bytes)
```

2.5 Creando los Componentes Presentacionales

Una vez ejecutada la instrucción anterior, editaremos el archivo “/src/app/posts/components/profile-item/profile-item.component.ts” donde importaremos nuevamente el módulo ‘Input’ perteneciente a ‘@angular/core’ y añadiremos la propiedad `@Input() posts: any[]` a continuación deberemos actualizar la propiedad ‘styles’ de la misma manera que lo haremos en la siguiente diapositiva. y actualizaremos la sentencia ‘template’ como

```
styles: [`  
  img.avatar {  
    height: 60px;  
    width: 60px;  
  }  
  span.date {  
    font-size: small;  
  }  
`],
```

2.5 Creando los Componentes Presentacionales

```
<div class="row">
  <div class="col-md-8 offset-md-2">
    <div class="card mb-3" *ngFor="let post of profile.posts">
      <div class="card-body">
        <div class="media">
          <img class="avatar mr-3 rounded" [attr.src]="profile?.avatar">
          <div class="media-body">
            <h5>
              <a [routerLink]="profile?.id"> {{profile?.fullName}} </a>
              <span class="date float-right text-muted">

            </span>
            </h5>
            <p>{{post?.text}}</p>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

2.6 Creando Resolutores para Recopilar Datos usando el Router

En esta sección crearemos manualmente 2 clases que actuarán como resolutores y configuraremos nuestro Router para usarlos. Posteriormente actualizaremos nuestros componentes contenedores para utilizar los datos proporcionados por los resolutores.

Antes de continuar deberemos saber qué es exactamente un resolutor, un **resolutor** es una clase que podemos utilizar para buscar los datos que usaremos en nuestro componente *antes* de que nuestro componente sea mostrado. En nuestro ejemplo los resolutores se encargan de extraer los datos de la API y proporcionárselos a los componentes.

2.6 Creando Resolutores para Recopilar Datos usando el Router

Para crear nuestro primer resolutor, abriremos un terminal y nos dirigiremos a la ruta raíz de nuestro proyecto donde ejecutaremos la orden

```
ng g class posts/resolvers/posts-resolver
```

y una vez generado `src/app/posts/resolvers/posts-resolver.ts` donde

importaremos las siguientes dependencias al inicio del documento:

```
import { Injectable } from '@angular/core'
import { Resolve } from '@angular/router'
import { PostsService } from '../services/posts.service'
```


2.6 Creando Resolutores para Recopilar Datos usando el Router

Una vez importadas las dependencias necesarias, deberemos añadir el decorador '@Injectable' a la clase 'PostsResolver' y haremos que la clase implemente la interfaz 'Resolve<any>', de modo que quedara de la siguiente manera:

```
@Injectable()  
export class PostsResolver implements Resolve<any> {  
}
```

A continuación deberemos añadir la línea `private postsService: PostsService` como parámetro del método 'constructor()'. Finalmente crearemos un nuevo método

'resolve' con el cuerpo siguiente:

```
resolve() {  
  return this.postsService.getPosts()  
}
```

2.6 Creando Resolutores para Recopilar Datos usando el Router

El resolutor previamente creado es el encargado de recuperar todos nuestros posts, ahora crearemos un segundo resolutor encargado de identificar el perfil. Para ello nuevamente nos dirigiremos desde una terminal al directorio raíz de nuestro proyecto donde ejecutaremos la instrucción

Una vez generado nuestro resolutor, editaremos el archivo `src/app/posts/resolvers/profile-resolver.ts` donde importaremos las siguientes dependencias:

```
import { Injectable } from '@angular/core'
import { ActivatedRouteSnapshot, Resolve } from '@angular/router'
import { PostsService } from '../services/posts.service'
```

2.6 Creando Resolutores para Recopilar Datos usando el Router

El siguiente paso, análogamente a lo realizado anteriormente, dejaremos la cabecera de la clase de la siguiente forma:

```
@Injectable()
export class ProfileResolver implements Resolve<any> {
}
```

Deberemos añadir también la sentencia `private postsService: PostsService` como parámetro del método 'constructor()'.

Finalmente deberemos crear bajo el constructor un método 'resolve' con el siguiente cuerpo:

```
resolve(route: ActivatedRouteSnapshot) {
  return this.postsService.getProfile(route.params['profileId'])
}
```

2.6 Creando Resolutores para Recopilar Datos usando el Router

Finalmente solo nos quedara importar nuestros resolutores, para ello deberemos editar el archivo `src/app/posts/posts-routing.module.ts`

donde importaremos nuestros resolutores con las siguientes lineas:

```
import { PostsResolver } from '../resolvers/posts-resolver'  
import { ProfileResolver } from '../resolvers/profile-resolver'
```

y actualizaremos nuestras rutas con la propiedad 'resolve' y la llamada a los resolutores:

```
{ path: '', component: PostsComponent, resolve: { posts: PostsResolver }  
},  
{ path: ':profileId', component: ProfileComponent, resolve: { profile:  
  ProfileResolver } },
```

2.6 Creando Resolutores para Recopilar Datos usando el Router

Cabe la posibilidad de encontrarnos con un error por el cual nada se mostraria por pantalla al recargar nuestra aplicacion, para solventarlo deberemos editar el archivo `src/app/posts/posts.module.ts` donde importaremos nuestros resolutores como hicimos en la diapositiva anterior y los referenciaremos en el array 'providers' que deberá quedar de la siguiente forma:

```
providers: [PostsService, PostsResolver, ProfileResolver]
```

2.6 Creando Resolutores para Recopilar Datos usando el Router

Para concluir esta sección deberemos actualizar nuestro PostsComponent para leer los datos de los resolutores, para ello editaremos el archivo `src/app/posts/container/posts/posts.component.ts`

```
import { ActivatedRoute } from '@angular/router';
```

módulo:

y eliminaremos el import del `private route: ActivatedRoute` nos será necesario. También deberemos actualizar el constructor añadiendo la línea

```
ngOnInit() {  
  // como parametro. Finalmente actualizaremos el método 'ngOnInit()',  
  // reemplazando su contenido  
  this.route.data  
    .map(data => data['posts'])  
    .map(data => data['items'])  
    .subscribe((result: any) => this.posts = result)  
}
```

2.6 Creando Resolutores para Recopilar Datos usando el Router

El ultimo paso será actualizar nuestro ProfileComponent de manera similar a la diapositiva anterior, para ello abriremos el archivo `src/app/posts/container/profile/profile.component.ts`

donde borrarémos la referencia a PostsService (pues ya no lo utilizaremos) y actualizaremos el método `private route: ActivatedRoute` reemplazando la línea

y por último actualizaremos el método `'ngOnInit()'` reemplazando su contenido por:

```
this.route.data
  .map(data => data['profile'])
  .subscribe((result: any) => this.profile = result)
```

Parte 3: Renderizado del Servidor

- Generando la Aplicación Servidor
- Añadiendo Dependencias para la Aplicación Servidor
- Implementando un Servidor Web
- Añadiendo Metadatos Dinámicos

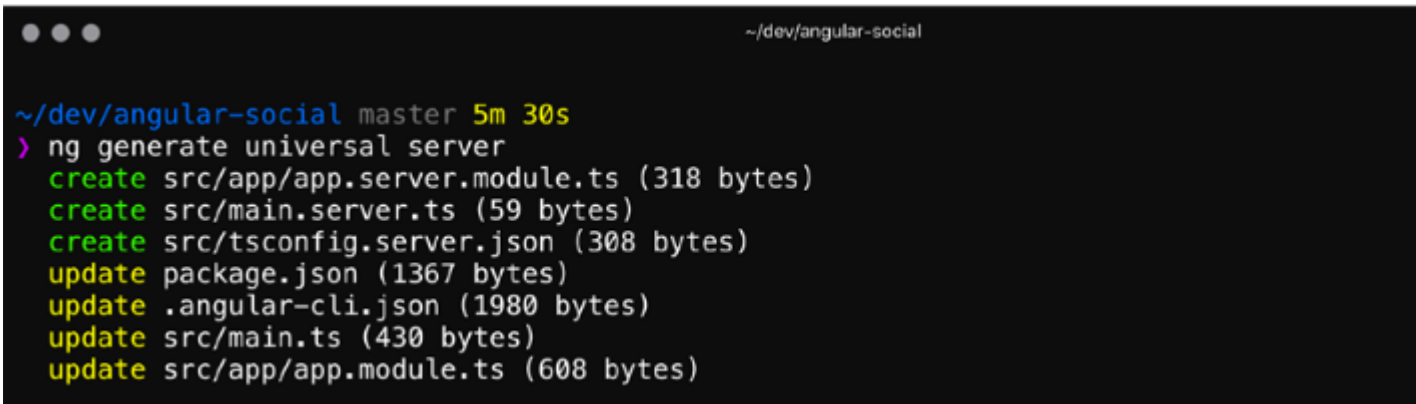
3. Renderizado del Servidor

En esta sección aprenderemos a:

- Renderizar la aplicación que hemos ido construyendo como ejemplo
- Añadir Angular Universal a nuestra aplicación y configurar una segunda aplicación en la configuración de Angular CLI
- Implementar un servidor web que hospede nuestra aplicación
- Añadir metadatos dinamicos a nuestra aplicación

3.1 Generando la Aplicación Servidor

Para generar nuestra aplicación servidor simplemente deberemos dirigirnos desde una terminal a la ruta raíz de nuestra aplicación y ejecutar la orden *'ng generate universal server'* la cual nos mostrará una salida como la mostrada en la imagen:

A terminal window with a dark background and light text. The title bar at the top shows three window control buttons on the left and the path '~ /dev/angular-social' on the right. The terminal content shows a prompt '~ /dev/angular-social master 5m 30s' followed by the command 'ng generate universal server'. The output consists of six lines: 'create src/app/app.server.module.ts (318 bytes)', 'create src/main.server.ts (59 bytes)', 'create src/tsconfig.server.json (308 bytes)', 'update package.json (1367 bytes)', 'update .angular-cli.json (1980 bytes)', and 'update src/main.ts (430 bytes)'. The final line is 'update src/app/app.module.ts (608 bytes)'.

```
~ /dev/angular-social master 5m 30s
> ng generate universal server
create src/app/app.server.module.ts (318 bytes)
create src/main.server.ts (59 bytes)
create src/tsconfig.server.json (308 bytes)
update package.json (1367 bytes)
update .angular-cli.json (1980 bytes)
update src/main.ts (430 bytes)
update src/app/app.module.ts (608 bytes)
```

3.1 Generando la Aplicación Servidor

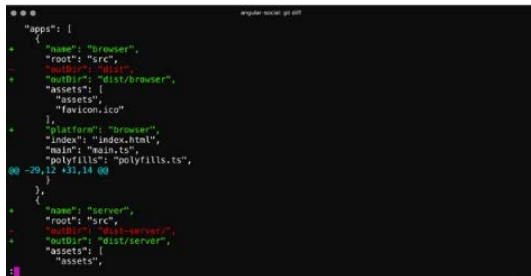
El siguiente paso a seguir para renderizar nuestro servidor será generar la aplicación de angular universal, para ello nos dirigiremos nuevamente a la ruta raíz de nuestro proyecto a través de una terminal donde ejecutaremos la instrucción `ng generate universal server` tras lo cual deberemos ejecutar la orden `'npm install'` para instalar las dependencias creadas en el fichero `'package.json'`

3.1 Generando la Aplicación Servidor

Ahora centraremos nuestros esfuerzos en hacer nuestra aplicación más consistente, para ello ejecutaremos la siguiente serie de instrucciones en el directorio raíz de nuestro proyecto:

```
ng set apps.1.outDir=dist/server
ng set apps.0.outDir=dist/browser
ng set apps.0.name=browser
ng set apps.0.platform=browser
```

El efecto de estas instrucciones se verá reflejado en el archivo `'.angular-cli.json'` que se mostrará de la siguiente manera:



```
***
angular-cli.json
{
  "apps": [
    {
      "name": "browser",
      "root": "src",
      "outDir": "dist",
      "outDir": "dist/browser",
      "assets": [
        "assets",
        "favicon.ico"
      ],
      "platform": "browser",
      "index": "index.html",
      "main": "main.ts",
      "polyfills": "polyfills.ts",
      "tsConfig": "tsconfig.json",
      "scripts": [],
      "styles": [
        "styles.css"
      ],
      "assets": [
        "assets",
        "favicon.ico"
      ]
    },
    {
      "name": "server",
      "root": "src",
      "outDir": "dist/server",
      "outDir": "dist/server",
      "assets": [
        "assets",
        "favicon.ico"
      ]
    }
  ]
}
```

3.2 Añadiendo Dependencias para la Aplicación Servidor

Para asegurarnos de que nuestra aplicación funciona correctamente, deberemos comprobar que angular carga correctamente sus dependencias 'zone.js' y 'reflect-metadata' así como activar el modo producción.

Para ello nos dirigiremos al archivo `src/main.server.ts`, donde deberemos añadir al inicio las siguientes líneas para importar las dependencias requeridas:

```
import 'zone.js/dist/zone-node';
import 'reflect-metadata';

import { enableProdMode } from '@angular/core';
import { environment } from '../environments/environment';
```

Adicionalmente activaremos el modo producción en función del entorno en que trabajemos:

```
if (environment.production) {
  enableProdMode();
}
```

3.2 Añadiendo Dependencias para la Aplicación Servidor

Finalmente añadiremos la aplicación servidor a la configuración de nuestro Angular CLI. Para ello abriremos una terminal dentro de nuestro proyecto donde instalaremos la dependencia de Angular Universal ejecutando la orden `npm install --save @nguniversal/module-map-ngfactory-loader`

`src/app/app.server.module.ts` editaremos el archivo

donde deberemos importar la siguiente línea:

```
import { ModuleMapLoaderModule } from '@nguniversal/module-map-ngfactory-loader';
```

Por último deberemos añadir la referencia al módulo recién importado (`ModuleMapLoaderModule`) en el array `'imports'` del documento.

3.3 Implementando un Servidor Web

Ahora que nuestras aplicaciones han sido creadas, crearemos un simple servidor que las hospede, para ello crearemos un simple servidor en Node.js basado en Express.js. Definiremos nuestro servidor en un archivo TypeScript el cual llamaremos 'server.ts'.

El primer paso a seguir para este fin será instalar las dependencias necesarias para el servidor, para ello abriremos una terminal dentro de nuestro proyecto donde ejecutaremos la orden:

```
npm install --save ts-node @nguniversal/express-engine
```

3.3 Implementando un Servidor Web

Una vez instaladas las dependencias procederemos a la creación del archivo 'server.ts', para ello crearemos dicho archivo en el directorio raíz de nuestro proyecto y lo construiremos de la siguiente manera:

```
import * as express from 'express';
import { join } from 'path';
import { ngExpressEngine } from '@nguniversal/express-engine';
import { provideModuleMap } from '@nguniversal/module-map-ngfactory-loader';
const PORT = process.env.PORT || 8080;
const staticRoot = join(process.cwd(), 'dist', 'browser');
const { AppServerModuleNgFactory, LAZY_MODULE_MAP } = require('./dist/server/main.bundle');
const app = express();
app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));
app.set('view engine', 'html');
app.set('views', staticRoot);
app.get('*', express.static(staticRoot));
app.get('*', (req, res) => res.render('index', { req }));
app.listen(PORT, () => console.log(`Server listening on http://localhost:${PORT}`));
```


3.3 Implementando un Servidor Web

Ahora que hemos creado nuestro servidor, deberemos añadir al archivo 'package.json' un script para inicializar nuestro servidor, para ello editaremos dicho archivo donde deberemos localizar el objeto 'scripts'.

Una vez localizado dicho objeto, deberemos reemplazar la propiedad existente 'start' por lo siguiente:

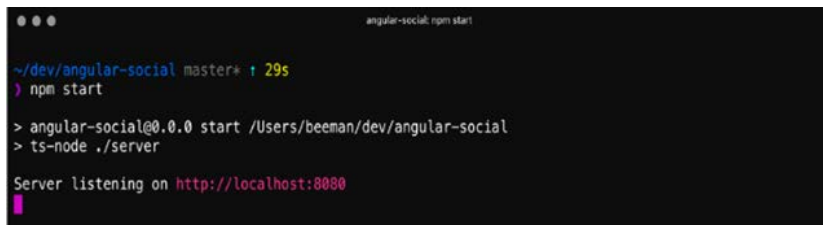
```
"start": "ts-node ./server",
```

3.3 Implementando un Servidor Web

Para concluir esta sección iniciaremos nuestro servidor, para ello nos dirigiremos desde una terminal al directorio raíz de nuestra aplicación y ejecutaremos las siguientes instrucciones:

```
$ npm run build  
$ npm start
```

La salida del segundo comando debe ser similar a la mostrada en la imagen:

A terminal window titled 'angular-social: npm start' with three window control buttons in the top-left corner. The terminal shows the following text:

```
~/dev/angular-social master* ↑ 29s  
> npm start  
  
> angular-social@0.0.0 start /Users/beeman/dev/angular-social  
> ts-node ./server  
  
Server listening on http://localhost:8080
```

A pink cursor is visible at the end of the last line.

3.3 Implementando un Servidor Web

Para verificar que nuestro servidor funciona correctamente, nos dirigiremos desde nuestro navegador a la dirección `http://localhost:8080`, donde deberá cargarse nuestra aplicación.

Para comprobar en mayor profundidad el correcto funcionamiento de nuestra aplicación, podemos comprobar que todos los componentes se hayan cargado correctamente, para ello desde el menú de Chrome nos dirigiremos a Ver -> Herramientas de Desarrollador -> Ver Código Fuente

3.4 Añadiendo Metadatos Dinámicos

En esta sección añadiremos a nuestra aplicación metadatos dinámicos que actualizarán nuestros componentes contenedores para, por ejemplo, añadir diferentes títulos y metadatos que faciliten la posibilidad de localización de nuestra aplicación.

3.4 Añadiendo Metadatos Dinámicos

Lo primero que haremos para añadir los metadatos deseados será crear un servicio 'UiService', para ello abriremos una terminal dentro de nuestro proyecto donde ejecutaremos la orden

Una vez generado este servicio, editarensrc/app/ui/services/ui.service.ts

donde añadiremos las siguientes líneas:

```
private appColor = '#C3002F';  
private appImage = '/assets/logo.svg';  
private appTitle = 'Angular Social';  
private appDescription = 'Angular Social is a Social Networking App  
built in Angular';
```

Deberemos también de importar los modulos 'Title' y 'Meta' de la siguiente forma:

```
import { Meta, Title } from '@angular/platform-browser';
```

3.4 Añadiendo Metadatos Dinámicos

A continuación deberemos inyectar en el método constructor los parámetros `private titleService: Title, private metaService: Meta` para utilizar los módulos previamente importados.

Finalmente deberemos crear un nuevo método `'setMetaData(config)'` cuyo cuerpo se muestra en la siguiente diapositiva.

3.4 Añadiendo Metadatos Dinámicos

```
// Get the description of the config, or use the default App Description
const description = config.description || this.appDescription
// Get the title of the config and append the App Title, or just use the
App Title
const title = config.title ? `${config.title} - ${this.appTitle}` :
this.appTitle;

// Set the Application Title
this.titleService.setTitle(title);

// Add the Application Meta tags
this.metaService.addTags([
  { name: 'description', content: description },
  { name: 'theme-color', content: this.appColor },
  { name: 'twitter:card', content: 'summary' },
  { name: 'twitter:image', content: this.appImage },
  { name: 'twitter:title', content: title },
  { name: 'twitter:description', content: description },
  { name: 'apple-mobile-web-app-capable', content: 'yes' },
  { name: 'apple-mobile-web-app-status-bar-style', content: 'black-
translucent' },
  { name: 'apple-mobile-web-app-title', content: title },
  { name: 'apple-touch-startup-image', content: this.appImage },
  { property: 'og:title', content: title },
  { property: 'og:description', content: description },
  { property: 'og:image', content: this.appImage },
]);
```

3.4 Añadiendo Metadatos Dinámicos

Para concluir esta sección, añadiremos metadatos a nuestros componentes 'PostsComponent' y 'ProfileComponent'.

Para añadir metadatos a 'PostsComponent' deberemos abrir el archivo `src/app/posts/container/posts/posts.component.ts`

```
import { UiService } from '../../../../ui/services/ui.service';
```

```
private uiService: UiService; e inyectar
```

como parametro del metodo constructor la línea

.

Ahora, al igual que hicimos previamente deberemos crear un método 'setMetaData(posts)' cuyo cuerpo se muestra en la siguiente diapositiva.

3.4 Añadiendo Metadatos Dinámicos

```
const { itemsPerPage, itemsTotal } = posts['counters']
const description = `Showing ${itemsPerPage} from ${itemsTotal} posts`
const title = 'Posts List'

this.uiService.setMetadata({ description, title })
return posts;
```

Finalmente deberemos actualizar la sentencia `‘.map’` del método `‘ngOnInit()’` con el siguiente código:

```
.map(data => this.setMetadata(data['posts']))
```

3.4 Añadiendo Metadatos Dinámicos

Para concluir y de forma análoga a lo realizado anteriormente, añadiremos metadatos al

‘ProfileComponent’, para ello editaremos `src/app/posts/container/profile/profile.component.ts`

```
import { UiService } from '../../../../ui/services/ui.service';
```

donde importaremos la sentencia

```
private uiService: UiService;
```

e inyectaremos el parámetro

en el método constructor.

A continuación crearemos de nuevo un método al cual llamaremos ‘setMetaData(profile)’ cuyo cuerpo se muestra en la siguiente diapositiva.

3.4 Añadiendo Metadatos Dinámicos

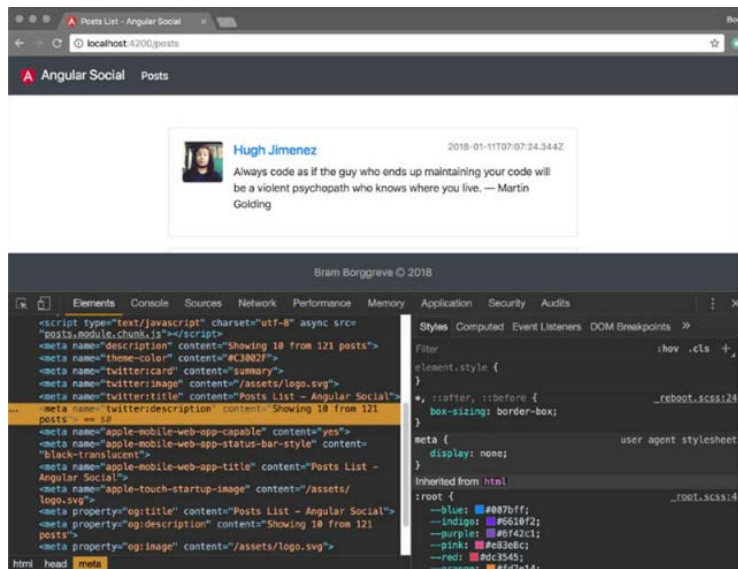
```
const { fullName, posts } = profile;
const description = `${fullName} posted ${posts.length} posts.`;
const title = `Posts by ${fullName}`;
this.uiService.setMetadata({ description, title });
return profile;
```

Finalmente deberemos actualizar el método 'ngOnInit' al siguiente código el cual transmitirá los datos extraídos de nuestra API a través del método previamente definido:

```
this.route.data
  .map(data => this.setMetadata(data['profile']))
  .subscribe((result: any) => this.profile = result)
```

3.4 Añadiendo Metadatos Dinámicos

Finalmente para comprobar que nuestros metadatos se han añadido correctamente podemos utilizar la herramienta de Google Chrome **Inspeccionar Elemento** que nos deberá mostrar una salida similar a la mostrada en la imagen:



Parte 4: Operadores de Servicio

- ¿Que es un Operador de Servicio?
- ¿Que es una Aplicación Web Progresiva?
- Instalando Dependencias
- Activando el Operador de Servicio
- Configurando el Operador de Servicio

4. Operadores de Servicio

En esta sección:

- Exploraremos Operadores de Servicio y PWAs
- Añadiremos un Operador de Servicio a nuestra aplicación
- Configuraremos el Operador de Servicio para convertir nuestra aplicación en una aplicación web progresiva
- Exploraremos cómo depurar un Operador de Servicio

4.1 ¿Qué es un Operador de Servicio?

Un Operador de Servicio es un script que el navegador ejecuta en segundo plano y que actúa como un proxy de red que gestiona las peticiones a la red.

Se sitúa entre la red y el dispositivo que almacena el contenido de la aplicación en su memoria caché, facilitando el uso offline de la aplicación al usuario.

Adicionalmente a almacenar datos en caché, puede sincronizar los datos de la API en segundo plano añadiendo por ejemplo notificaciones de inserción.

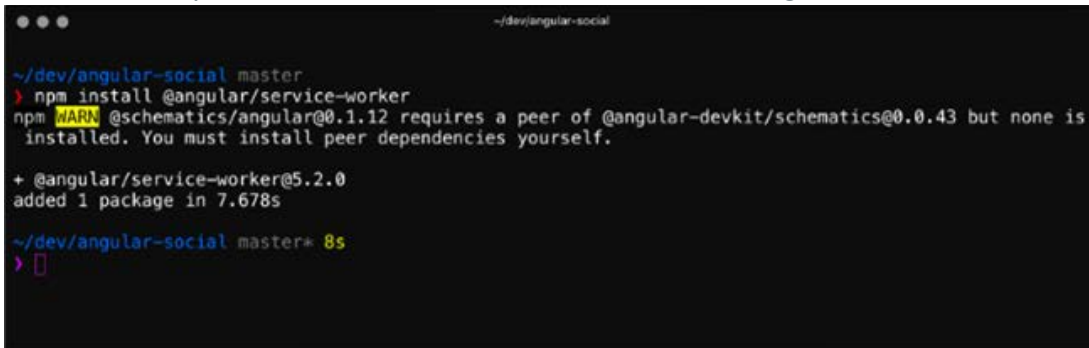
4.2 ¿Qué es una Aplicación Web Progresiva?

Una Aplicación Web Progresiva (PWA por sus siglas en inglés) es un término utilizado para aplicaciones web que se comportan de manera similar a aplicaciones móviles nativas. De este modo, al igual que estas últimas, las PWAs permiten al usuario inicializar la aplicación offline, almacenando en caché los elementos de la interfaz y las llamadas a la API para mostrar la página de inicio, de este modo el usuario puede interactuar a nivel básico con la aplicación hasta que la conexión sea restablecida. Una vez la conexión se restablece, la PWA recupera los datos del servidor y actualiza la aplicación de forma que el usuario interactúe con la última versión de la aplicación.

4.3 Instalando Dependencias

Para empezar a trabajar con los Operadores de Servicio, deberemos instalar las dependencias necesarias para ello abriremos una terminal en el directorio raíz de nuestro proyecto donde ejecutaremos la `npm install @angular/service-worker`

Cuando la instalación se complete, la terminal nos deberá mostrar la siguiente salida:

A terminal window with a dark background and light-colored text. The title bar at the top reads '~ / dev / angular - social'. The prompt is '~ / dev / angular - social master'. The user enters 'npm install @angular/service-worker'. The output shows a warning from npm about a peer dependency, followed by the installation of @angular/service-worker@5.2.0. The prompt then changes to '~ / dev / angular - social master* 8s' and the user enters a command that results in a red error message.

```
~ / dev / angular - social
~ / dev / angular - social master
> npm install @angular/service-worker
npm WARN @schematics/angular@0.1.12 requires a peer of @angular-devkit/schematics@0.0.43 but none is
  installed. You must install peer dependencies yourself.

+ @angular/service-worker@5.2.0
added 1 package in 7.678s

~ / dev / angular - social master* 8s
> 
```

4.4 Activando el Operador de Servicio

Ahora que hemos instalado la dependencia necesaria es hora de activar el Operador de Servicio, para ello lo primero que debemos hacer será abrir una terminal en el directorio de nuestro proyecto donde

ejecutaremos el comando `ng set apps.0.serviceWorker=true`

. Una vez ejecutado nos

`src/app/app.module.ts`

donde importaremos las

siguientes sentencias:

```
import { ServiceWorkerModule } from '@angular/service-worker'
import { environment } from '../environments/environment'
```

Deberemos

añadir el módulo `(ServiceWorkerModule)` al array `(imports)`:

```
ServiceWorkerModule.register('/ngsw-worker.js', {enabled: environment.
production}),
```

4.4 Activando el Operador de Servicio

Finalmente deberemos crear la configuración de nuestro Operador de Servicio.

Para ello crearemos el archivo `src/ngsw-config.json` cuyo cuerpo se muestra en la siguiente diapositiva.

Una vez creado dicho archivo abriremos una terminal en el directorio de nuestra aplicación donde ejecutaremos la orden `npm run build:browser`

4.4 Activando el Operador de Servicio

```
{
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico",
          "/index.html"
        ],
        "versionedFiles": [
          "/*.bundle.css",
          "/*.bundle.js",
          "/*.chunk.js"
        ]
      }
    },
    {
      "name": "assets",
      "installMode": "lazy",
      "updateMode": "prefetch",
      "resources": {
        "files": [
          "/assets/**"
        ]
      }
    }
  ]
}
```

4.5 Configurando el Operador de Servicio

En la sección anterior hemos añadido el archivo de configuración del Operador de Servicio pero realmente todavía no hemos realizado ninguna configuración.

En esta sección añadiremos 2 tipos de configuración: grupo de activos y grupo de datos.

En el grupo de activos especificaremos cómo queremos que nuestro Operador de Servicio se encargue de los activos(hojas de estilo, imágenes, archivos JS externos...)de nuestra aplicación.

En el grupo de datos especificaremos cómo queremos que nuestro Operador de Servicio guarde en cache los datos de la API de la cual estamos extrayendo los datos.

4.5 Configurando el Operador de Servicio

Para configurar el grupo de activos abriremos el archivo `src/ngsw-config.json`

donde deberemos localizar el array 'assetGroups' donde añadiremos los objetos:

```
{
  "name": "externals",
  "installMode": "prefetch",
  "updateMode": "prefetch",
  "resources": {
    "urls": [
      "https://ajax.googleapis.com/**",
      "https://fonts.googleapis.com/**",
      "https://fonts.gstatic.com/**",
      "https://maxcdn.bootstrapcdn.com/**"
    ]
  }
},
{
  "name": "avatars",
  "installMode": "prefetch",
  "updateMode": "prefetch",
  "resources": {
    "urls": [
      "http://localhost:3000/avatars/**",
      "https://packt-angular-social.now.sh/avatars/**"
    ]
  }
}
```

4.5 Configurando el Operador de Servicio

Para configurar el grupo de datos deberemos editar el archivo `src/ngsw-config.json` donde crearemos un nuevo array 'dataGroups' en el cual añadiremos el objeto:

```
{
  "name": "rest-api",
  "urls": [
    "http://localhost:3000/api/**",
    "https://packt-angular-social.now.sh/api/**"
  ],
  "cacheConfig": {
    "strategy": "freshness",
    "maxSize": 100,
    "maxAge": "1h",
    "timeout": "5s"
  }
}
```