

# PRÁCTICA *con* ANGULAR





## Índice

1. Intro.....	3
2. The environment (El entorno).....	4
3. Components.....	7
4. Services.....	11
5. Maquetación con Bootstrap.....	14
6. Directives.....	18
7. Pipes.....	20
8. Creando nuestra segunda petición.....	22
9. Funcionalidad Buscar Usuarios usando ngModel y Events.....	24
10. Registrar nuestra app en Github (Client ID y Client Secret).....	29
11. Routes.....	31



## Angular 2

### 1. Intro

Angular2 es un framework para desarrollar aplicaciones web creado por Google y desarrollado en Typescript para crear aplicaciones SPA (*Single Page Application*).

*Más sobre Angular2:*

[http://www.w3ii.com/es/angular2/angular2\\_overview.html](http://www.w3ii.com/es/angular2/angular2_overview.html)

*Más sobre SPA:*

[https://es.wikipedia.org/wiki/Single-page\\_application](https://es.wikipedia.org/wiki/Single-page_application)

<https://cink.es/blog/2013/10/07/spa-un-paradigma-de-arquitectura-de-aplicaciones-web-en-auge/>

*Más sobre Typescript:*

<https://www.genbetadev.com/javascript/hello-world-en-typescript-el-lenguaje-en-el-que-se-construira-angular-2>

En esta práctica desarrollaremos una aplicación sencilla para conocer algunas de las características de Angular. La aplicación consistirá en un buscador de perfiles de Github, desarrollado por Brad Traversy, donde mostraremos los repositorios de un usuario, así como otros datos relevantes del usuario buscado. Los datos que usaremos para el backend serán consumidos de la Api Rest de Github mediante el protocolo HTTP donde realizaremos diferentes tipos de petición (POST, PUT, GET, DELETE) y esta nos devolverá las respuestas en formato JSON.

*Api Rest de Github:* <https://developer.github.com/v3/repos/>

Es conveniente saber que es una Api Rest para aprovechar al máximo la práctica. Además es una de las arquitecturas más usadas como backend para aplicaciones SPA.

Esta arquitectura posee una gran ventaja, y es que podemos usar la misma Api Rest para proporcionar los datos a diferentes clientes, como puede ser Angular para la versión web, pero también podríamos desarrollar una aplicación nativa para android, ios o windows phone.

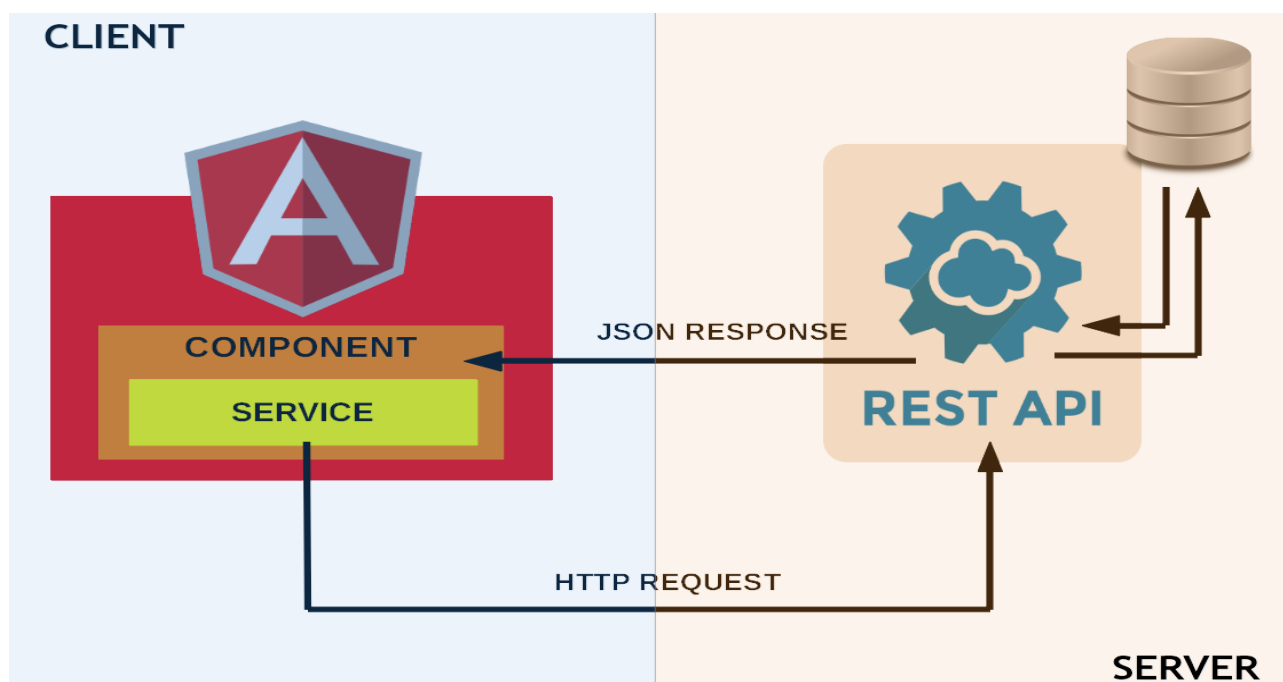


*Aquí tienes información acerca de las Api Rest:*

<http://asiermarques.com/2013/conceptos-sobre-apis-rest/>



La arquitectura de este proyecto será la siguiente:



## 2. The environment (El entorno)

Para desarrollar con Angular vamos a necesitar realizar una serie de instalaciones las cuales iremos viendo a lo largo de la práctica.

Lo primero que vamos a necesitar instalar es **NodeJS**, que se instalará junto con **Npm** (Node Package Module) como gestor de dependencias.

Su pagina oficial es: <https://www.npmjs.com/>

Es necesario conocer npm para desarrollar en Angular ya que se utiliza para la instalación/desinstalación de librerías de terceros, pero resumiendo, podemos decir que es el gestor de dependencias de NodeJS.

Para la descarga del instalador y más información sobre NodeJS: <https://nodejs.org/es/>





Una vez hayamos descargado e instalado NodeJS podemos comprobar que la instalación se ha ejecutado correctamente ejecutando en la consola los siguientes comandos:

```
C:\WINDOWS\system32\cmd.exe

C:\>node -v
v6.10.3

C:\>npm -v
3.10.10
```

Con el gestor de dependencias de npm vamos a proceder a instalar **Angular-CLI** (*Command-Line-Interface*) que es una herramienta desarrollada por Angular que te proporciona un interprete de línea de comandos de Angular que nos facilitará la creación de nuevos proyectos y la posibilidad de generar automáticamente la mayoría de los componentes de una aplicación Angular.

Más información sobre Angular-CLI:

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

<https://www.desarrolloweb.com/articulos/angular-cli.html>

Para instalar el módulo de manera global(-g) introduciremos el siguiente comando por consola:

```
C:\WINDOWS\system32\cmd.exe

C:\>npm install -g angular-cli
```

Existen varias maneras de iniciar un proyecto Angular, podemos crear los archivos y directorios manualmente siguiendo los pasos de su web oficial, podemos clonar el quickstart oficial de Angular disponible en Github, o bien, usando Angular-CLI como haremos nosotros para irnos familiarizando con los comandos que nos proporciona esta herramienta, ya que nos ahorrará mucho tiempo de desarrollo.

Nos ubicaremos en la ruta donde deseemos alojar el proyecto y ejecutamos el siguiente comando:

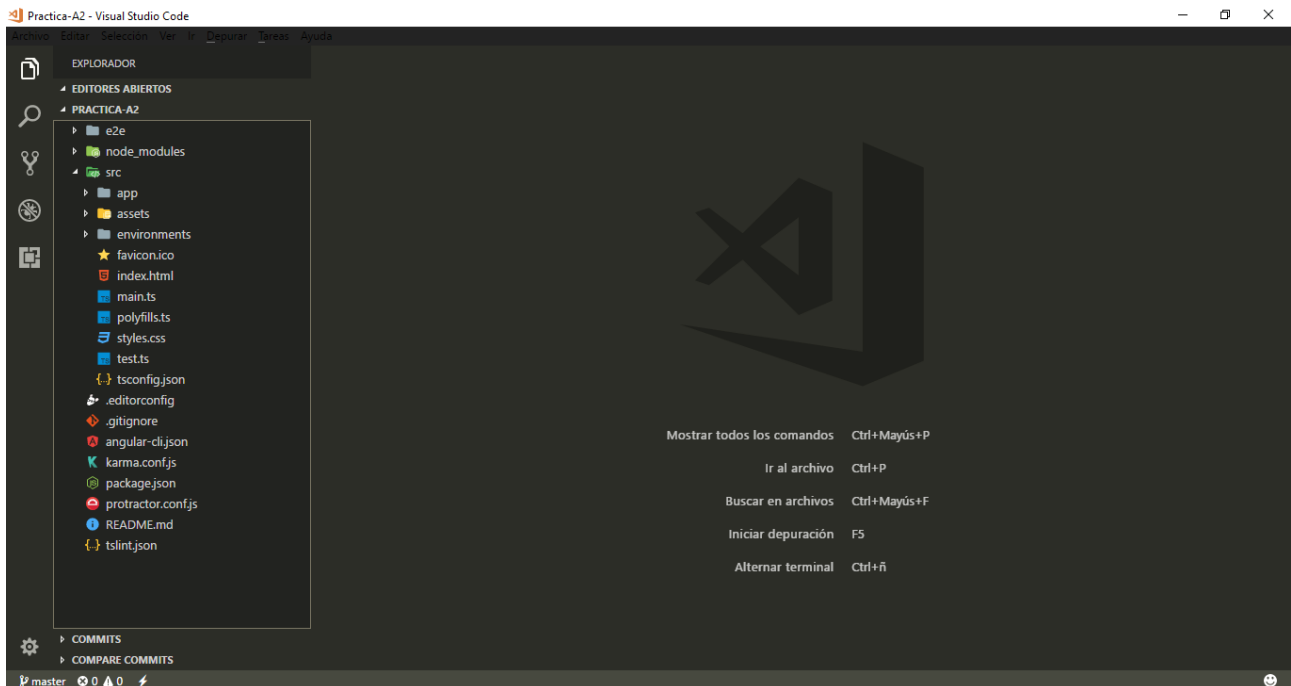
**ng new PROJECT-NAME**

```
C:\WINDOWS\system32\cmd.exe

workspace>ng new Practica-A2
```



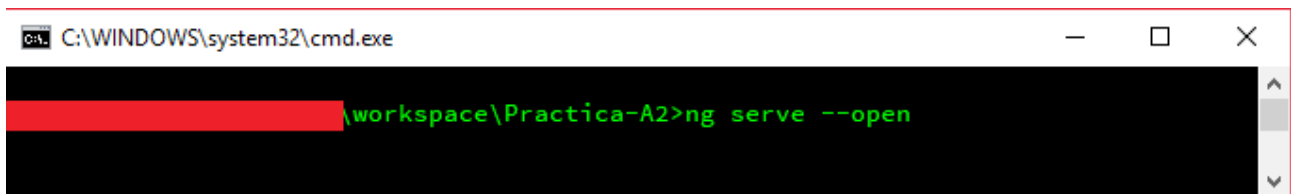
Ahora que ya tenemos nuestro proyecto creado vamos a abrirlo con un editor de texto. En esta práctica se usará **Visual Studio Code** aunque la elección es libre.



La estructura de directorios y ficheros cuando se genera un nuevo proyecto de Angular es compleja. Podemos leer una breve descripción acerca de ellos, o al menos de los más importantes en los *siguientes enlaces*:

<https://www.desarrolloweb.com/articulos/analisis-carpetas-proyecto-angular2.html>  
<https://programadmalditos.com/estructura-de-un-proyecto-angular-2/>

Ahora que ya tenemos una idea general de la estructura de directorios de Angular vamos a ubicarnos en la raíz del proyecto y vamos a arrancarlo ejecutando el siguiente comando:



El flag **-open** abrirá nuestro proyecto en el navegador que tengamos asignado por defecto.



## app works!

Listo, ya tenemos nuestro proyecto corriendo en el puerto 4200 que Angular usará por defecto.



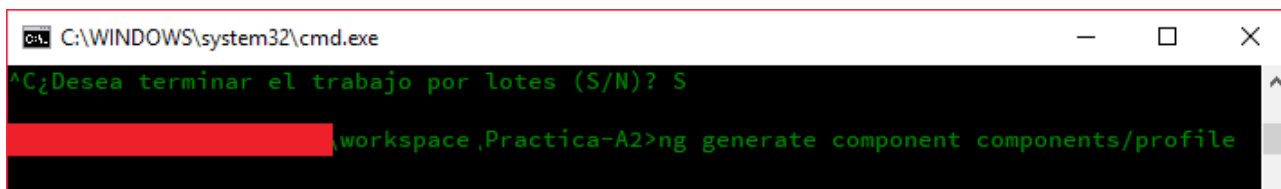
### 3. Components

Algo muy importante que debemos saber de Angular es que está basado en componentes:

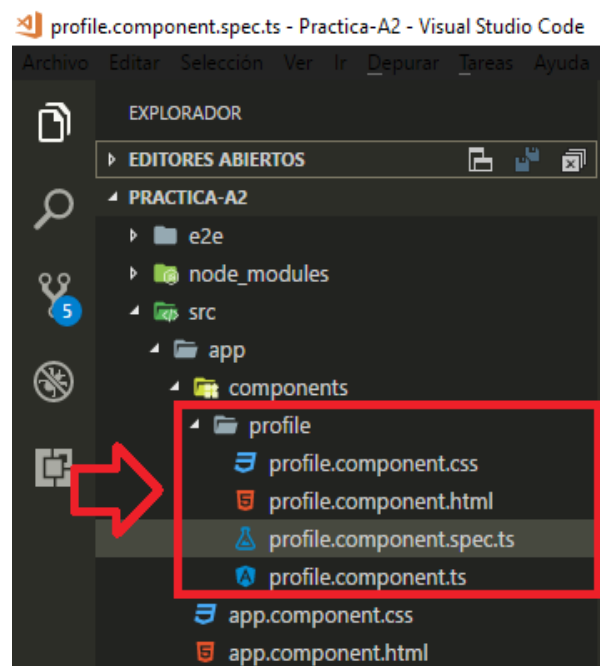
[http://www.w3ii.com/es/angular2/angular2\\_components.html](http://www.w3ii.com/es/angular2/angular2_components.html)

Un componente está compuesto generalmente por tres archivos, aunque también podrían ir en un solo fichero, dependiendo de la modularidad que le querramos dar y del tamaño del desarrollo. Estos ficheros son: un archivo **html**, un archivo **css** y un archivo **typescript**. Estos tres archivos conforman un componente, y este puede ser llamado con un selector (que definiremos en el archivo typescript) desde cualquier vista de cualquier otro componente dentro del proyecto.

Para ir comprendiendo esto vamos a crear nuestro primer componente con la herramienta Angular-CLI. Para ello primero vamos a parar nuestra aplicación de Angular situándonos en la consola y presionando **Control+C** y confirmando que deseamos terminar, una vez hayamos parado el servidor ejecutaremos el siguiente comando:



Esto nos creará nuestro componente **profile**, el cual podemos ver en nuestra estructura de directorios, dentro de la carpeta **src/app/components**, ya que le hemos indicado que nos cree una carpeta **components** donde iremos alojando los componentes que vayamos generando a lo largo de la práctica, y dentro de esta se encuentra el carpeta **profile** donde tenemos cuatro archivos. Un archivo **.css** (para desarrollar los estilos), un archivo **.html** (para desarrollar la estructura de la vista), un archivo **spec.ts** (destinado al testing, lo ignoraremos en esta práctica) y un archivo **.ts** (donde desarrollaremos la lógica del componente).





Vamos a ir viendo cada uno de los archivos que conforman nuestro componente profile, empezando por el archivo typescript (**lógica del componente**):

```
profile.component.ts x
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-profile',
5    templateUrl: './profile.component.html',
6    styleUrls: ['./profile.component.css']
7  })
8  export class ProfileComponent implements OnInit {
9
10   constructor() {}
11
12   ngOnInit() {}
13
14 }
```

Tal como se dice uno de nuestros enlaces sobre Typescript, este 'es **un superconjunto de JavaScript que esencialmente añade capacidades de POO** como es el tipado estático y objetos basados en clases.'

De modo que Typescript es un Javascript orientado a objetos, con todo lo que ello conlleva, como puede ser la herencia, las clases, los objetos, constructores, etc, lo que hace que las aplicaciones sean mucho más escalables y mantenibles que usando Javascript tradicional.

Volviendo al código, la primera línea que vemos es la de la importación del decorador **Component** y de un Interface llamado '**OnInit**' que luego se implementa en la exportación de la clase.

Como vemos las importaciones provienen en este caso de '**@angular/core**'. Estas son parte del repositorio principal de Angular y no requieren de una instalación previa con npm.

El decorador Component es un método que recibe un objeto JSON como parámetro. El contenido del decorador se denominan '**anotación**' y llamamos **metadatos** a cada uno de los atributos que lo componen.

Las anotaciones más relevantes de un Component son:

- **selector**: aquí indicaremos el nombre de la etiqueta por el cual vamos a llamar a nuestro componente desde cualquier otro html de la aplicación.
- **templateUrl**: aquí indicamos la ruta donde se encuentra nuestra vista en html. También podríamos escribir nuestro html dentro del decorador, indicándole la propiedad **template** en lugar de templateUrl, pero en nuestro caso vamos a separar la vista de la lógica en diferentes archivos.
- **styleUrls**: indicaremos la ruta donde se encuentra nuestro css para los estilos de este componente. Al igual que con las vistas, también podríamos escribir este código dentro del decorador.





Como hemos usado el NG-CLI para generar el componente, todas estas anotaciones ya estarán configuradas correctamente, al igual que nos habrá generado la declaración y exportación de la clase donde se encontrará toda la lógica del componente.

Es dentro de esta clase donde podremos realizar la declaración de variables y funciones que podremos, a posteriori, llamar o mostrar en la vista html asociada.

Como vemos exportamos la clase con la palabra clave **export** seguido de la palabra **class** y el nombre de la clase del componente. En este caso también implementaremos la interface **OnInit** y por tanto declararemos la función **ngOnInit()** dentro de la clase.

Esta función **ngOnInit** es parte del ciclo de vida de un componente. Este método es llamado después de inicializar las propiedades de entrada enlazadas. Es el primer método que se ejecuta cuando se llama al componente desde otra parte de la aplicación. Aunque en realidad es el segundo método que se ejecuta puesto que el constructor se ejecuta primero.

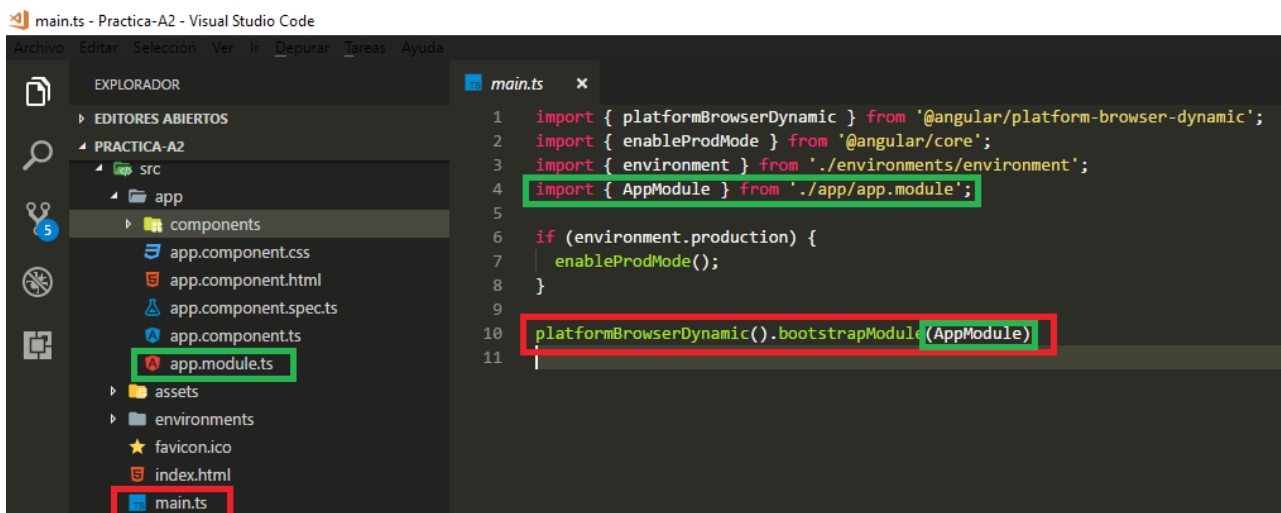
Como regla general usa **ngOnInit** cuando necesites acceder a alguna entrada del componente y puedes usar el constructor para inicializar las propiedades de tu clase.

Más información:

<http://www.luismdeveloper.com/angular2/angular-2-lo-basico-parte-1/>

Antes de concluir con el apartado de los Componentes, debemos saber que estos deben de ser importados en el archivo **app.module.ts**. Este fichero es muy importante puesto que va a contener todos los componentes y módulos que vayamos creando en nuestra aplicación para después ser exportados dentro del módulo **AppModule**, que será llamado desde el fichero **main.ts** como el primer (y único en la mayoría de los casos) módulo en el arranque de la aplicación. Además los componentes declarados en este archivo podrán ser llamados desde cualquier otro html de cualquier componente mediante su selector.

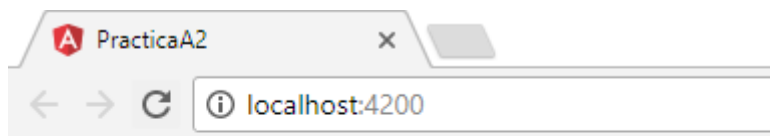
```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { FormsModule } from '@angular/forms';
4 import { HttpClientModule } from '@angular/http';
5
6 import { AppComponent } from './app.component';
7 import { ProfileComponent } from './components/profile/profile.component';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



Aparte de las anotaciones que hemos descrito existen otras importantes, algunas de ellas las iremos descubriendo durante la práctica.

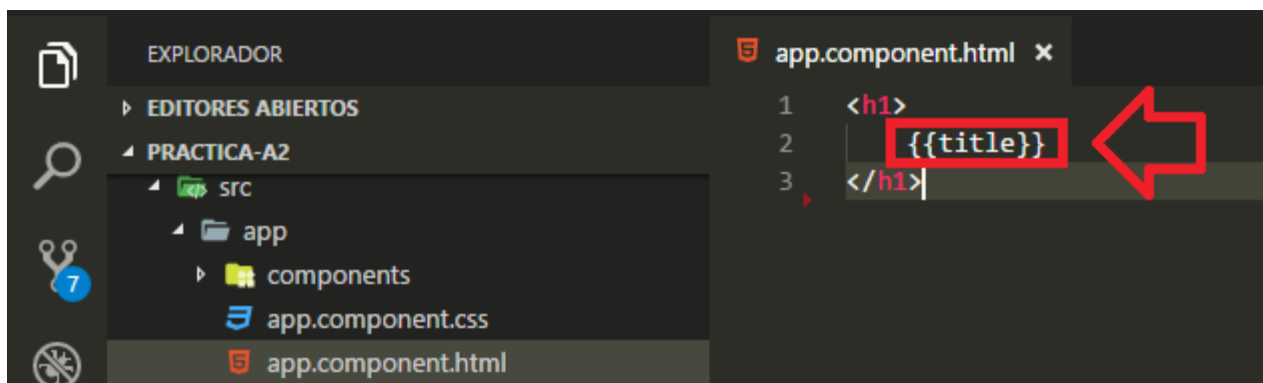
[http://www.w3ii.com/es/angular2/angular2\\_metadata.html](http://www.w3ii.com/es/angular2/angular2_metadata.html)

Ahora vamos a volver a arrancar la aplicación con **ng serve -open** y vamos dirigirnos al fichero **app.component.ts** y para cambiarle el valor a la variable **title**. Luego guardamos los cambios, y abrimos el navegador, viendo como se ha producido el cambio.



## GitHub Searcher

Ahora vamos a ir a **app.component.html** para ver como utilizamos esta variable para mostrarla en la vista.



Estas dobles llaves que vemos encerrando nuestra variable **title** están haciendo un **binding por interpolación**, o lo que es lo mismo, está enlazando el valor de la variable que tenemos en la lógica (.ts) a su vista (.html).



## 4. Services

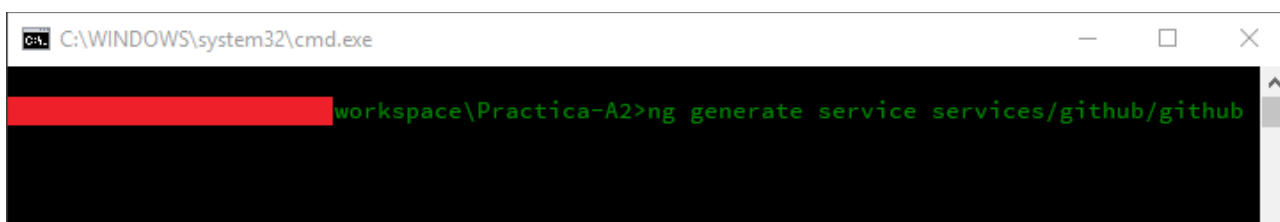
Los services serán la capa donde envolveremos mediante funciones las llamadas HTTP al API Rest de GitHub, para luego, desde uno o varios componentes, importar este servicio y realizar los diferentes tipos de petición según el caso (aunque en nuestro caso solo realizaremos peticiones de tipo GET).

Podemos ver una representación gráfica de esto en la página 2, donde se muestra la arquitectura del proyecto.

Más sobre services:

[http://www.w3ii.com/es/angular2/angular2\\_services.html](http://www.w3ii.com/es/angular2/angular2_services.html)

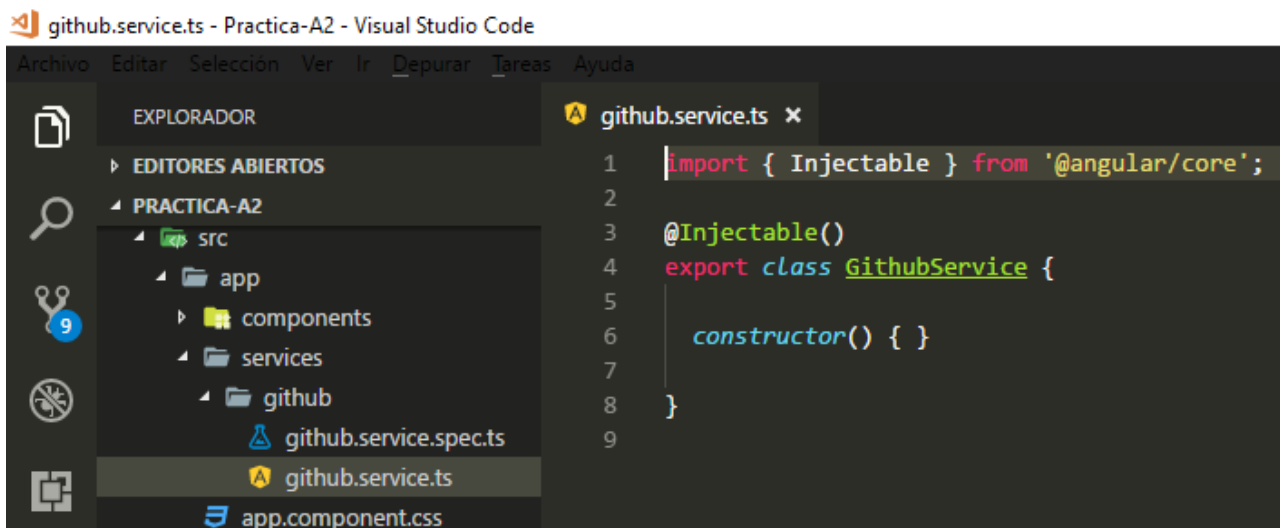
Una vez dicho esto, vamos a crear nuestro primer service usando nuevamente la herramienta NG-CLI. Para generar un nuevo service vamos a ejecutar el siguiente comando:



```
C:\WINDOWS\system32\cmd.exe
workspace\Practica-A2>ng generate service services/github/github
```

Con esto se nos habrá creado una carpeta en nuestro proyecto en la ruta **src>app>services/github/** y aquí dentro tenemos nuestro archivo **github.service.ts**

Su contenido es el siguiente:



```
github.service.ts - Practica-A2 - Visual Studio Code
Archivo  Editar  Selección  Ver  Ir  Depurar  Tareas  Ayuda
EXPLORADOR
  EDITORES ABIERTOS
    PRACTICA-A2
      src
        app
          components
          services
            github
              github.service.spec.ts
              github.service.ts
            app.component.css
github.service.ts x
1  import { Injectable } from '@angular/core';
2
3  @Injectable()
4  export class GithubService {
5
6      constructor() { }
7
8  }
9
```

Como vemos, el contenido que nos genera por defecto NG-CLI es la importación del decorador **Injectable**, que al igual que el decorador Component, también es un método el cual exporta una clase, y por tanto podremos importarla desde cualquiera de nuestros componentes (en nuestro caso tanto en el componente principal, que es el **app.component.ts** como el componente que hemos generado, que es **profile.component.ts**)

Lo que vamos a declarar dentro de la clase van a ser, principalmente funciones que usaremos para envolver las peticiones Http lanzadas al Api para obtener una respuesta de datos en formato JSON con las que poder formar la vista.



Primero deberemos de añadir unas cuantas líneas de código para realizar la importación de unos cuantos módulos que necesitaremos para realizar las peticiones **Http** y enviar **headers**(cabeceras) en dichas peticiones, además de un operador **map** para coger los datos de la respuesta **HTTP** en formato **JSON**.

github.service.ts - Practica-A2 - Visual Studio Code

```
Archivo Editar Selección Ver Ir Depurar Tareas Ayuda
EXPLORADOR
PRACTICA-A2
  e2e
  node_modules
  src
    app
      components
      services
        github
          github.service.spec.ts
          github.service.ts
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts

github.service.ts x
1 import { Injectable } from '@angular/core';
2 import { Http, Headers } from '@angular/http';
3 import 'rxjs/add/operator/map';
4
5 @Injectable()
6 export class GithubService {
7
8   private username: string;
9
10  constructor(private _http: Http) {
11    this.username = 'bradtraversy';
12  }
13
14  getUser() {
15    return this._http.get('http://api.github.com/users/' + this.username)
16      .map(res => res.json());
17  }
18
19 }
```

También hemos declarado una variable de tipo string para almacenar un username, que será el nombre del perfil que buscaremos en nuestra aplicación.

En el constructor hemos inicializado dicha variable y además hemos creado la función **getUser()** que como se puede ver, va a devolver la respuesta como **JSON** obtenido de la petición realizada a la URL que se indica como primer parámetro de la llamada al método **get** del objeto **\_http**. Con respecto al objeto **\_http**, vemos que la inyección de dependencias se produce en el constructor, pudiendo usar a posteriori dicho objeto para llamar a sus diferentes métodos.

Ahora que tenemos creado nuestro service vamos a realizar una prueba importándolo en nuestro componente **profile.component.ts** pero antes vamos a necesitar importarlo en nuestro **app.component.ts**, para ello, además de la ya conocida llamada a **import** indicándole el nombre del services y la ruta donde se encuentra, tendremos que añadir una anotación más en nuestro decorador Component para almacenar nuestro service, esta anotación es **providers** que recoge un array donde ir incluyendo los servicios que vayamos creando.

```
app.component.ts x
1 import { Component } from '@angular/core';
2 import { GithubService } from '../services/github/github.service';
3
4 @Component({
5   selector: 'app-root',
6   templateUrl: './app.component.html',
7   styleUrls: ['./app.component.css'],
8   providers: [ GithubService ]
9 })
10 export class AppComponent {
11   title = 'Git Hub Searcher';
12 }
```



Ahora que ya hemos importado y declarado el service en el **app.component.ts** vamos a importarlo en nuestro **profile.component.ts** y además llamar al método **getUser()** que hemos creado dentro del **GithubService**.

```

1  import { Component, OnInit } from '@angular/core';
2  import { GithubService } from '../services/github/github.service';
3  import 'rxjs/add/operator/map';
4
5  @Component({
6    selector: 'app-profile',
7    templateUrl: './profile.component.html',
8    styleUrls: ['./profile.component.css']
9  })
10 export class ProfileComponent implements OnInit {
11
12   constructor(private _githubService: GithubService) {
13     this._githubService.getUser().subscribe(user => {
14       console.log(user);
15     })
16   }

```

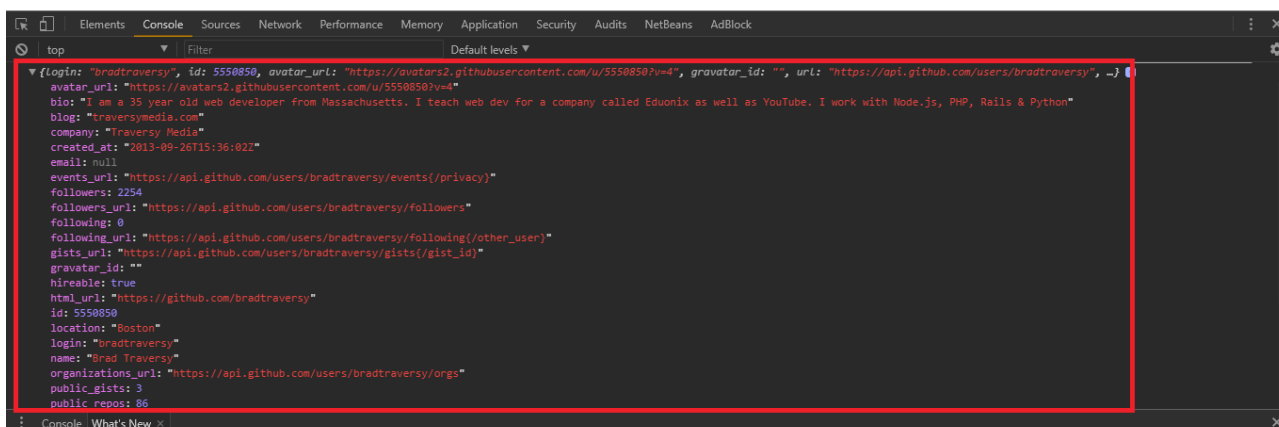
Ahora que ya tenemos importado nuestro **GithubService** en **ProfileComponent**, vamos a llamar a **ProfileComponent** con su *selector* desde la vista del componente principal, que no es otro que **AppComponent**.

De modo que cuando se inicialice la aplicación, el primer componente que se llamará será **AppComponent**, este llamará a **ProfileComponent**, el cual contiene un *Service* llamado **GithubService** que contiene un método llamado **getUser()** que realiza una petición **http** de tipo **get** al *Api Rest de Github* y esta devolverá un objeto **json** que recibiremos en el método **subscribe**. Si todo ha ido bien, el objeto json se almacenará en la variable *user*, que mostraremos en consola con un **console.log(user)**



## GitHub Searcher

profile works!





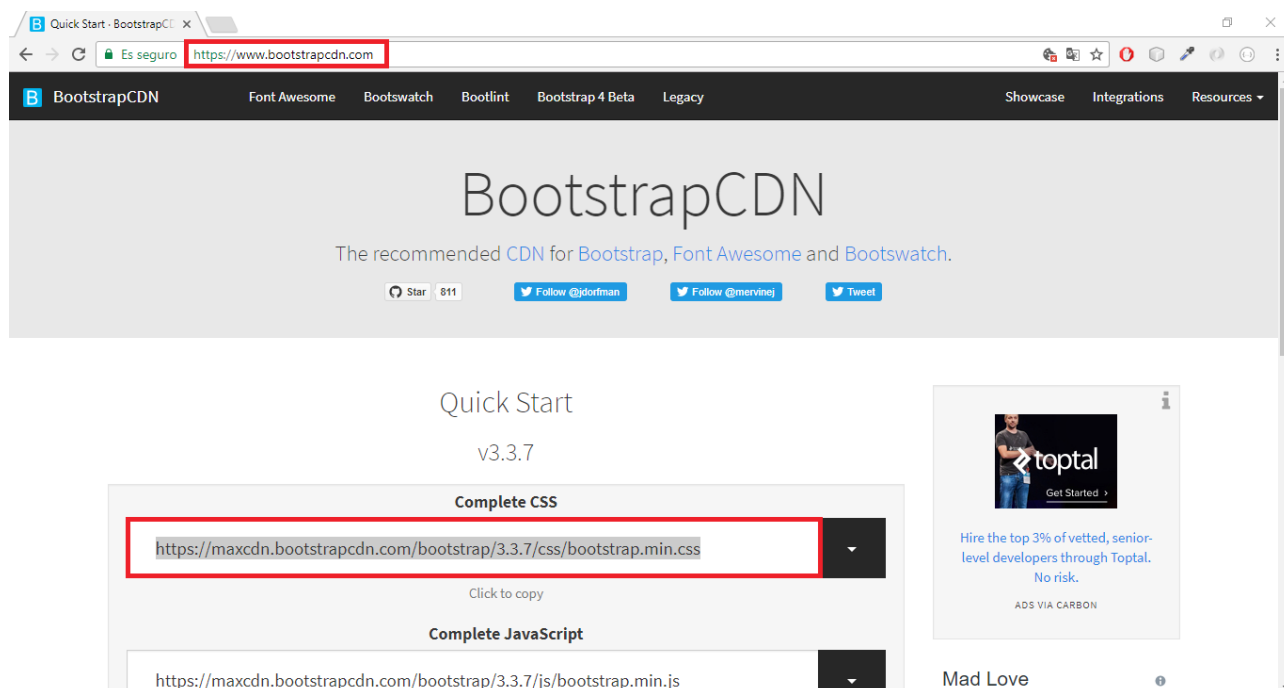
Si obtienes una respuesta como la que se muestra en la captura en la consola de tu navegador significa que todo ha ido bien y que podemos continuar con la práctica.

## 5. Maquetación con Bootstrap

Ahora que ya tenemos datos que poder mostrar en pantalla vamos a integrar bootstrap en nuestro proyecto para maquetar nuestra web de una manera rápida y sencilla.

Para ello simplemente vamos a copiar el enlace del archivo css que nos proporciona el cdn de bootstrap para llamarlo en nuestro index.html

cdn bootstrap: <https://www.bootstrapcdn.com/>



Como vemos en las capturas, simplemente copiamos el enlace de la página y lo pegamos en el atributo src de una nueva etiqueta link dentro de nuestro index.html





También vamos a ir a nuestro archivo **app.component.html** y vamos a quitar las etiquetas **<h1>** junto con la **interpolación de la variable title** y vamos a dejar únicamente el selector de nuestro ProfileComponent.

```
app.component.html x
1 | <app-profile></app-profile>
```

Una vez hecho esto, vamos a crear un **navbar** para nuestra aplicación. Para ello nos vamos a crear un nuevo componente mediante nuestra herramienta de **NG-CLI**, como hemos hecho anteriormente con el comando **ng generate component**:

```
C:\WINDOWS\system32\cmd.exe
C:\Users\erick\Desktop\workspace\Practica-A2>ng generate component components/navbar
```

Con el componente creado, vamos a dirigirnos al siguiente enlace, que es una plantilla de ejemplo de bootstrap, en su versión 3.3.2.

<https://getbootstrap.com/docs/3.3/examples/starter-template/>



## Bootstrap starter template

Use this document as a way to quickly start any new project.  
All you get is this text and a mostly barebones HTML document.

Una vez dentro del enlace, presionaremos **Control+u** para ver código de la página y copiaremos el código del navbar, tal como se muestra en la siguiente captura:





Este código vamos a pegarlo en nuestro **navbar.component.html** y vamos a aprovechar para realizarle algunos cambios, puesto que de momento no vamos a necesitar ninguno de los enlaces que contiene el navbar. También podemos cambiar el nombre del título.

```

1 <nav class="navbar navbar-inverse navbar-fixed-top">
2   <div class="container">
3     <div class="navbar-header">
4       <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" ar
5       <span class="sr-only">Toggle navigation</span>
6       <span class="icon-bar"></span>
7       <span class="icon-bar"></span>
8       <span class="icon-bar"></span>
9     </button>
10    <a class="navbar-brand" href="#">GithubSearch</a>
11  </div>
12  <div id="navbar" class="collapse navbar-collapse">
13    <ul class="nav navbar-nav">
14      <li><a href="#">Home</a></li>
15    </ul>
16  </div>
17 </nav>
18

```

**BORRAR** (indicando la eliminación de los enlaces en líneas 13-15)

Ahora vamos a llamar a este componente navbar desde nuestro componente principal, **app.component.html** con su selector, **app-navbar**.

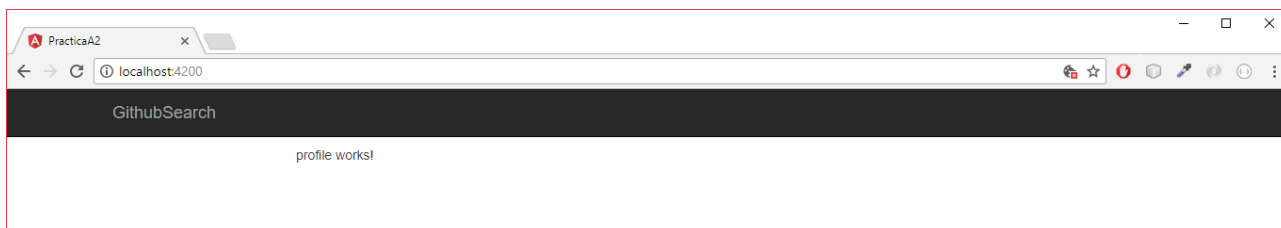
Además vamos a envolver nuestra etiqueta **<app-profile>** dentro del sistema de grid de bootstrap. Temporalmente vamos a añadirle también unas etiquetas **<br>** entre el navbar y el container para que el navbar no oculte el contenido del componente profile.

```

1 <app-navbar></app-navbar>
2
3 <br><br><br>
4
5 <div class="container">
6   <div class="row">
7     <div class="col-md-8 col-md-offset-2">
8       <app-profile></app-profile>
9     </div>
10  </div>
11 </div>

```

Una vez realizados los cambios nuestro navegador debe mostrar el siguiente resultado:







Vamos a dirigirnos ahora a nuestro **profile.component.ts** y vamos a añadirle unas cuantas líneas de código.

Para empezar vamos a declarar un array donde almacenar todos los datos que nos lleguen del usuario. Ese array se llamará **user**.

Luego vamos a añadir una línea dentro del método subscribe para indicar que deseamos almacenar en nuestro array user el contenido que nos llegue de la respuesta, en caso de que esta sea positiva.

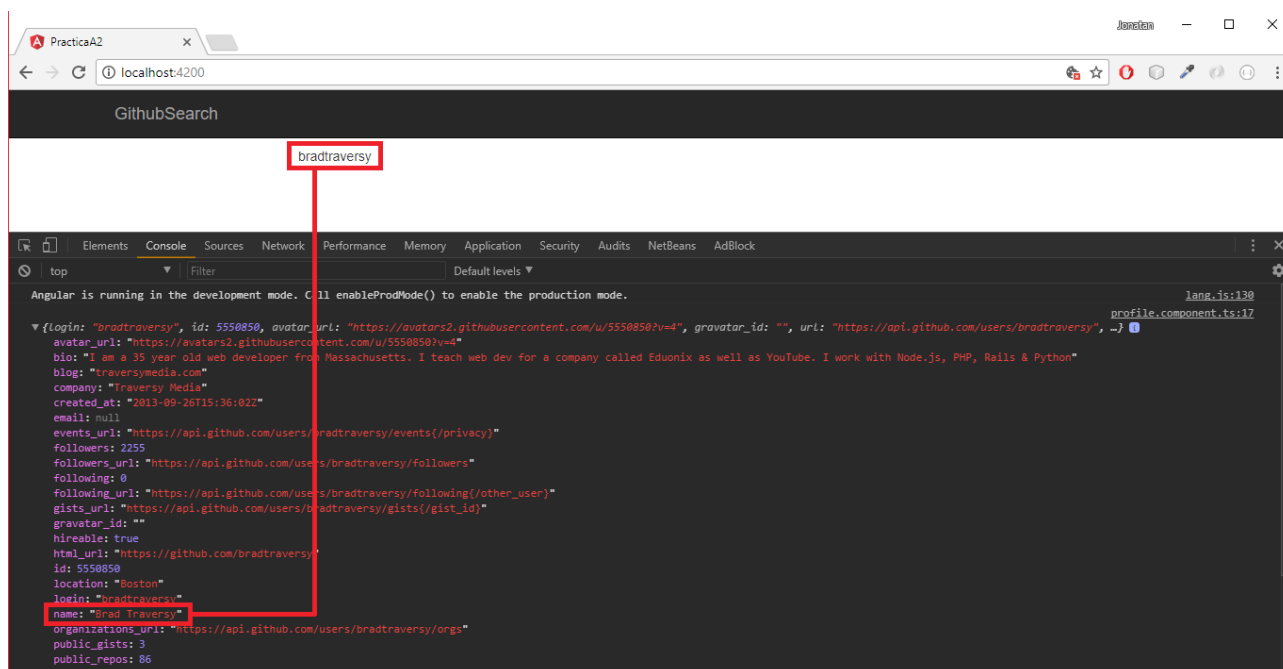
```
profile.component.ts x
1  import { Component, OnInit } from '@angular/core';
2  import { GithubService } from '../../services/github/github.service';
3  import 'rxjs/add/operator/map';
4
5  @Component({
6    selector: 'app-profile',
7    templateUrl: './profile.component.html',
8    styleUrls: ['./profile.component.css']
9  })
10 export class ProfileComponent implements OnInit {
11
12    public user: any = [];
13
14    constructor(private _githubService: GithubService) {
15      this._githubService.getUser().subscribe(user => {
16        this.user = user;
17        console.log(this.user);
18      })
19    }
20  }
```

Ahora nos vamos a su vista (**profile.component.html**) para mostrar por pantalla mediante interpolación, la propiedad name del objeto user que hemos recibido.

```
profile.component.html x
1  [{{ user.name }}]
```



Una vez hayamos guardado los cambios en el navegador se nos debe de mostrar lo siguiente:



Si se te muestra el nombre de la propiedad del objeto que estamos interpolando en la vista significa que podemos empezar a maquetar algunos de estos datos en un panel de bootstrap.

## 6. Directives

Antes de continuar con la práctica, debes saber que son las directivas en Angular y para que sirven. Haciendo un breve resumen podemos clasificar las directivas por tres tipos:

- Directivas **Component**: *selectores*.
- Directivas **Estructurales**: *ngFor, ngIf, ngSwitch*
- Directivas **Atributo**: *ngClass, ngStyle*

Más información sobre directivas:

[http://www.w3ii.com/es/angular2/angular2\\_directives.html](http://www.w3ii.com/es/angular2/angular2_directives.html)

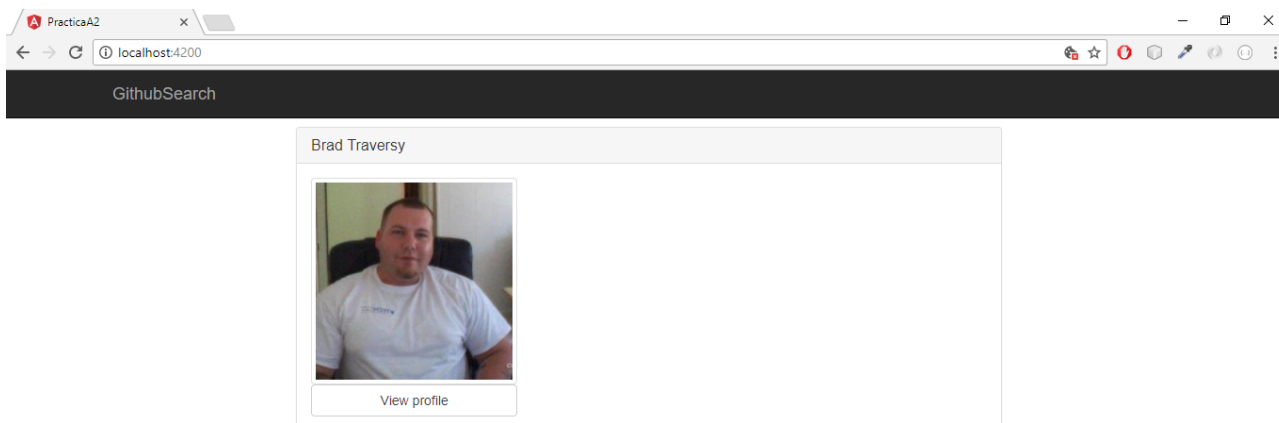


Vamos a empezar usando la directiva estructural **\*ngIf** para encerrar un fragmento de código que solo se mostrará si la variable `user` contiene datos. Dicha directiva la usaremos en este caso dentro de un `div`.

Dentro de este `div` vamos a crear un panel con las clases que nos proporciona bootstrap y vamos a añadir una imagen, la cual cargaremos usando la url que nos proporciona el objeto `user` con la propiedad **avatar\_url**, además de un botón que nos lleve al perfil de github del usuario, información que tendremos también dentro de la propiedad **html\_url** dentro de nuestro objeto `user`.

```
profile.component.html x
1  <div *ngIf="user">
2    <div class="panel panel-default">
3      <div class="panel-heading">
4        <h3 class="panel-title">{{ user.name }}</h3>
5      </div>
6      <div class="panel-body">
7        <div class="row">
8          <div class="col-md-4">
9            
10           <a class="btn btn-default btn-block" href="{{ user.html_url }}" target="_blank">View profile</a>
11          </div>
12          <div class="col-md-4">
13
14          </div>
15        </div>
16      </div>
17    </div>
18  </div>
```

Guardando los cambios realizados nuestro navegador debe mostrar lo siguiente:





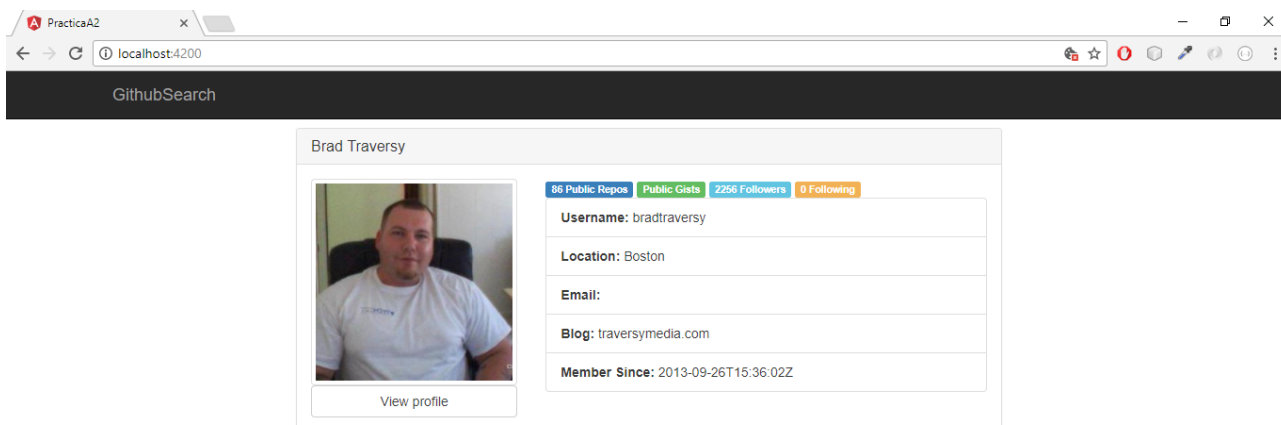
Vamos a completar el resto del panel interpolando unos cuantos datos que nos llegan desde nuestro objeto user y aprovecharemos algunas de las clases que nos proporciona bootstrap para maquetarlo.

```

profile.component.html x
1 <div *ngIf="user">
2   <div class="panel panel-default">
3     <div class="panel-heading">
4       <h3 class="panel-title">{{ user.name }}</h3>
5     </div>
6     <div class="panel-body">
7       <div class="row">
8         <div class="col-md-4">
9           
10          <a class="btn btn-default btn-block" href="{{ user.html_url }}" target="_blank">View profile</a>
11        </div>
12        <div class="col-md-8">
13          <div class="stats">
14            <span class="label label-primary">{{ user.public_repos }} Public Repos</span>
15            <span class="label label-success">{{ user.gists }} Public Gists</span>
16            <span class="label label-info">{{ user.followers }} Followers</span>
17            <span class="label label-warning">{{ user.following }} Following</span>
18          </div>
19          <ul class="list-group">
20            <li class="list-group-item"><strong>Username: </strong>{{ user.login }}</li>
21            <li class="list-group-item"><strong>Location: </strong>{{ user.location }}</li>
22            <li class="list-group-item"><strong>Email: </strong>{{ user.email }}</li>
23            <li class="list-group-item"><strong>Blog: </strong>{{ user.blog }}</li>
24            <li class="list-group-item"><strong>Member Since: </strong>{{ user.created_at }}</li>
25          </ul>
26        </div>
27      </div>
28    </div>
29  </div>
30 </div>

```

Con este código, el resultado en el navegador debe de ser el siguiente:



## 7. Pipes

Por último, vamos a usar un **pipe**, que es una funcionalidad de Angular que nos permite transformar los datos en la vista según deseemos. Existen una serie de pipes ya definidos en el core de angular y también podremos crear nuestros propios pipes.

*Más información sobre los pipes y como crear pipes personalizados:*

<https://angular.io/guide/pipes>

<https://victorroblesweb.es/2016/08/20/pipes-personalizados-angular-2/>

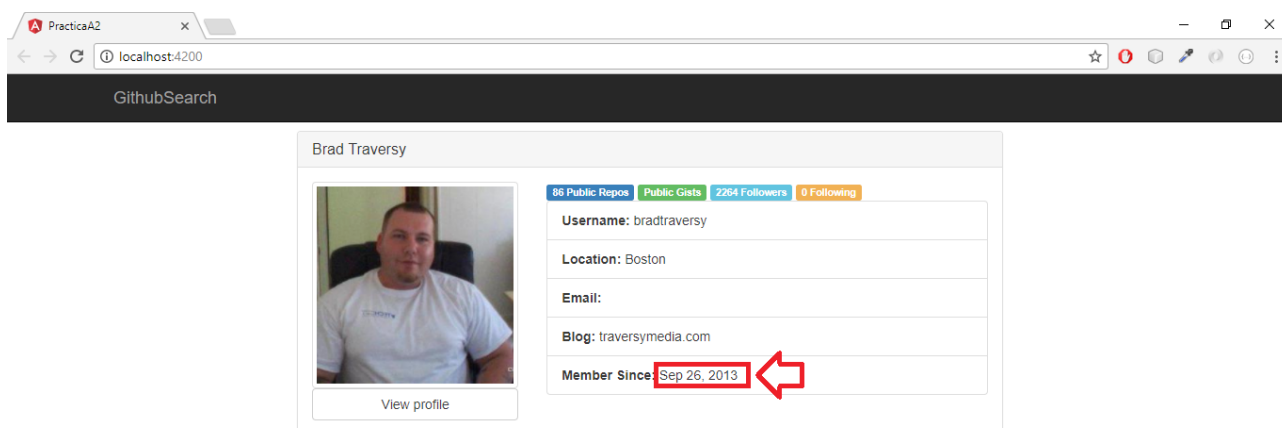


Un pipe se usa dentro de la interpolación de una variable, añadiendo el caracter "|" seguido del nombre del pipe. Para ver esto más claro vamos a usar el pipe **date** para dar un formato distinto a la fecha de registro que mostramos en nuestro panel de usuario.

En este caso, al tratarse de este pipe, ya está incluido en el core de angular y no es necesario ninguna importación. Simplemente lo añadiremos dentro de la interpolación de **user.created\_at**, tal y como muestra la siguiente captura:

```
profile.component.html x
1 <div *ngIf="user">
2   <div class="panel panel-default">
3     <div class="panel-heading">
4       <h3 class="panel-title">{{ user.name }}</h3>
5     </div>
6     <div class="panel-body">
7       <div class="row">
8         <div class="col-md-4">
9           
10          <a class="btn btn-default btn-block" href="{{ user.html_url }}" target="_blank">View profile</a>
11        </div>
12        <div class="col-md-8">
13          <div class="stats">
14            <span class="label label-primary">{{ user.public_repos }} Public Repos</span>
15            <span class="label label-success">{{ user.gists }} Public Gists</span>
16            <span class="label label-info">{{ user.followers }} Followers</span>
17            <span class="label label-warning">{{ user.following }} Following</span>
18          </div>
19          <ul class="list-group">
20            <li class="list-group-item"><strong>Username: </strong>{{ user.login }}</li>
21            <li class="list-group-item"><strong>Location: </strong>{{ user.location }}</li>
22            <li class="list-group-item"><strong>Email: </strong>{{ user.email }}</li>
23            <li class="list-group-item"><strong>Blog: </strong>{{ user.blog }}</li>
24            <li class="list-group-item"><strong>Member Since: </strong>{{ user.created_at | date }}</li>
25          </ul>
26        </div>
27      </div>
28    </div>
29  </div>
30</div>
```

Ahora la salida de nuestro navegador debe mostrarnos la fecha formateada:





## 8. Creando nuestra segunda petición

Ahora que ya tenemos nuestro panel mostrando los datos del usuario, vamos a crear otra función en nuestro **GithubService** para realizar una petición a la que llamaremos **getRepos** que nos devuelva los repositorios de un usuario. Para ello nos dirigiremos al fichero **github.service.ts** y añadiremos la siguiente función:

```
github.service.ts x
1  import { Injectable } from '@angular/core';
2  import { Http, Headers } from '@angular/http';
3  import 'rxjs/add/operator/map';
4
5  @Injectable()
6  export class GithubService {
7
8      private username: string;
9
10     constructor(private _http: Http) {
11         this.username = 'bradtraversy';
12     }
13
14     getUser() {
15         return this._http.get('http://api.github.com/users/' + this.username)
16             .map(res => res.json())
17     }
18
19     getRepos() {
20         return this._http.get('http://api.github.com/users/' + this.username + '/repos')
21             .map(res => res.json())
22     }
23
24 }
```

Ahora vamos a llamar a esta función desde el constructor de nuestro **profile.component.ts** para obtener los datos de los repositorios del usuario y mostrarlos debajo del panel de los datos personales. Estos datos los recogeremos dentro de otro array llamado **repos**, que deberemos declarar previamente.

```
profile.component.ts x
1  import { Component, OnInit } from '@angular/core';
2  import { GithubService } from '../services/github/github.service';
3  import 'rxjs/add/operator/map';
4
5  @Component({
6      selector: 'app-profile',
7      templateUrl: './profile.component.html',
8      styleUrls: ['./profile.component.css']
9  })
10  export class ProfileComponent implements OnInit {
11
12      public user: any = [];
13      public repos: any = [];
14
15      constructor(private _githubService: GithubService) {
16          this._githubService.getUser().subscribe(user => {
17              this.user = user;
18              console.log(this.user);
19          })
20
21          this._githubService.getRepos().subscribe(repos => {
22              this.repos = repos;
23          })
24      }
25  }
```



Ahora que ya tenemos los datos de los repositorios del usuario almacenados en el array **repos** vamos a crear otro panel en la vista con bootstrap de la misma forma que hicimos con los datos del usuario:

```

profile.component.html x
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

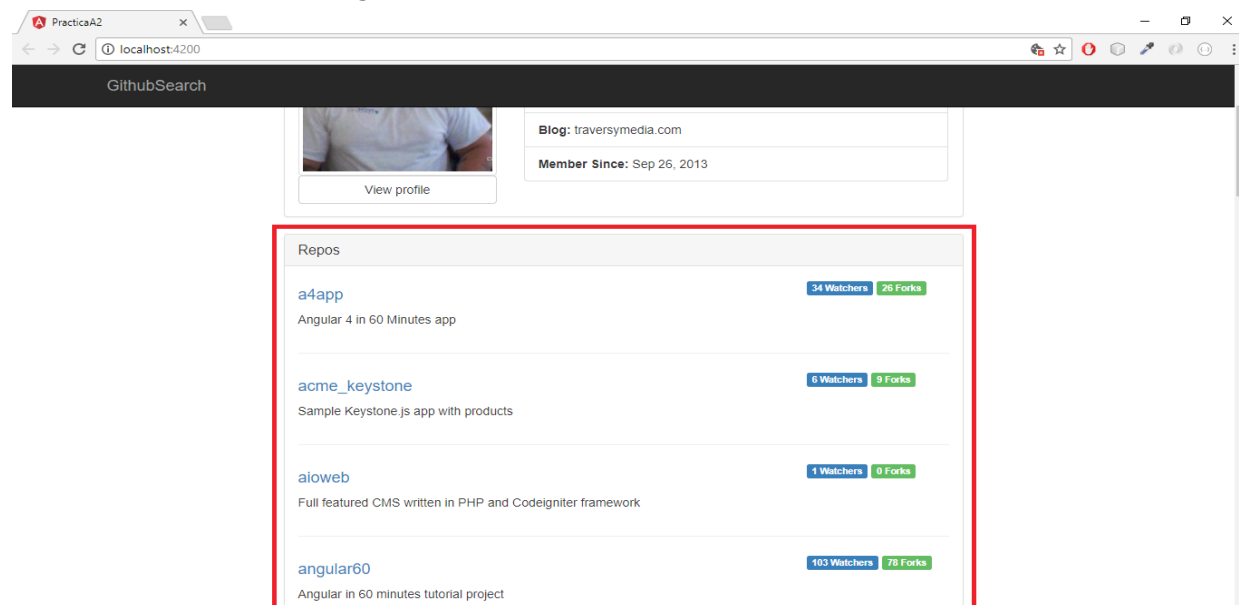
<li class="list-group-item"><strong>Email: </strong>{{ user.email }}</li>
<li class="list-group-item"><strong>Blog: </strong>{{ user.blog }}</li>
<li class="list-group-item"><strong>Member Since: </strong>{{ user.created_at | date }}</li>
</ul>
</div>
</div>
</div>
</div>
</div>

<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">Repos</h3>
  </div>
  <div class="panel-body">
    <div *ngFor="let repo of repos">
      <div class="row">
        <div class="col-md-9">
          <h4><a href="{{ repo.html_url }}" target="_blank">{{ repo.name }}</a></h4>
          <p>{{ repo.description }}</p>
        </div>
        <div class="col-md-3">
          <span class="label label-primary">{{ repo.watchers }} Watchers</span>
          <span class="label label-success">{{ repo.forks }} Forks</span>
        </div>
      </div>
      <hr>
    </div>
  </div>
</div>
</div>

```

Como vemos en la captura, hemos incluido un nuevo panel debajo del que ya teníamos (**rojo**), y además hemos hecho uso de otra directiva estructural de angular, como es **\*ngFor** para iterar nuestro array de repos y usando como variable local de iteración la variable **repo** (**verde**). Con esta variable **repo** vamos a ir sacando las propiedades del objeto e intercalarlas en la vista para mostrarlas por pantalla (**azul**).

El resultado debe ser el siguiente:





## 9. Funcionalidad Buscar Usuarios usando ngModel y Events

Ahora que ya tenemos creada la vista donde mostrar los datos del usuario, vamos a crear una funcionalidad que permita buscarlos e irlos mostrando dinámicamente.

Este buscador va a realizar una petición por cada vez que pulsemos una tecla dentro del mismo. Esto significa deberemos utilizar el **evento keyup** para realizar una petición cada vez que se llame a este evento. También utilizaremos una de las características más interesantes que nos proporciona Angular y este es el **Two-way data-binding**, que **nos permite modificar el valor de una propiedad de forma bidireccional**.

Esto significa que, por ejemplo, si creamos una variable en la lógica(.ts) y usamos la directiva **[(ngModel)]="variable/objeto"** dentro de un input en el html, si realizamos un cambio dentro del input en la vista, dicho cambio se realizará también en la lógica.

Para comprender esto mejor vamos a empezar añadiendo un formulario en nuestro **profile.component.html** para realizar las peticiones del buscador de usuarios.

```
profile.component.html x
1 <div class="row">
2   <div class="col-md-12">
3     <form>
4       <div class="form-group">
5         <input class="form-control" type="text" name="username" [(ngModel)]="username" (keyup)="searchUser()" placeholder="Enter a Github" />
6       </div>
7     </form>
8   </div>
9 </div>
10
11 <div *ngIf="user">
12   <div class="panel panel-default">
13     <div class="panel-heading">
14       <h3 class="panel-title">{{ user.name }}</h3>
15     </div>
16     <div class="panel-body">
17       <div class="row">
18         <div class="col-md-4">
19           
20           <a class="btn btn-default btn-block" href="{{ user.html_url }}" target="_blank">View profile</a>
21         </div>
22         <div class="col-md-8">
23           <div class="stats">
24             <span class="label label-primary">{{ user.public_repos }} Public Repos</span>
25             <span class="label label-success">{{ user.gists }} Public Gists</span>

```

Como vemos, usamos **[(ngModel)]="username"** para que se realice el enlace bidireccional del que hemos hablado con la variable **username(verde)**, que todavía no hemos declarado en la lógica del componente.





De modo que vamos a declarar la variable `username` en **`profile.component.ts`** junto con una nueva función **`searchUser()`**, que será la que se llame cada vez que se pulse una tecla dentro del input de búsqueda de usuarios(**azul**).

```
profile.component.ts x
6   selector: 'app-profile',
7   templateUrl: './profile.component.html',
8   styleUrls: ['./profile.component.css']
9   })
10  export class ProfileComponent implements OnInit {
11
12    public user: any = [];
13    public repos: any = [];
14
15    public username: string;
16
17    constructor(private _githubService: GithubService) {
18      this._githubService.getUser().subscribe(user => {
19        this.user = user;
20      })
21
22      this._githubService.getRepos().subscribe(repos => {
23        this.repos = repos;
24      })
25    }
26
27    searchUser() {
28      this._githubService.updateUser(this.username);
29
30      this._githubService.getUser().subscribe(user => {
31        this.user = user;
32      })
33
34      this._githubService.getRepos().subscribe(repos => {
35        this.repos = repos;
36      })
37    }
}
```

Como podemos ver en la captura, creamos la función **`searchUser()`** para envolver la llamada al método **`updateUser(username)`** del objeto **`_githubService`**.



Dicha función aun no ha sido creada, así que vamos a crearla en nuestro fichero **github.service.ts**

```
github.service.ts x
1  import { Injectable } from '@angular/core';
2  import { Http, Headers } from '@angular/http';
3  import 'rxjs/add/operator/map';
4
5  @Injectable()
6  export class GithubService {
7
8      private username: string;
9
10     constructor(private _http: Http) {
11         this.username = 'bradtraversy';
12     }
13
14     getUser() {
15         return this._http.get('http://api.github.com/users/' + this.username)
16             .map(res => res.json())
17     }
18
19     getRepos() {
20         return this._http.get('http://api.github.com/users/' + this.username + '/repos')
21             .map(res => res.json())
22     }
23
24     updateUser(username) {
25         this.username = username;
26     }
27
28 }
```

Como vemos la función **updateUser** se encarga de asignar a la variable username el username que le llega como parámetro.



No debemos olvidarnos que en nuestro **app.module.ts** debemos importar el módulo **FormsModule** desde el @angular/forms, ya que usamos la etiqueta **<form>** dentro de la vista del componente profile.

```
app.module.ts x
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4  import { HttpClientModule } from '@angular/http';
5
6  import { AppComponent } from './app.component';
7  import { ProfileComponent } from './components/profile/profile.component';
8  import { NavbarComponent } from './components/navbar/navbar.component';
9  import { AboutComponent } from './components/about/about.component';
10
11 import { RouterModule, Routes } from '@angular/router';
12 import { appRouterProviders, routing } from './app.routes';
13
14 @NgModule({
15   declarations: [
16     AppComponent,
17     ProfileComponent,
18     NavbarComponent,
19     AboutComponent
20   ],
21   imports: [
22     BrowserModule,
23     FormsModule,
24     HttpClientModule,
```



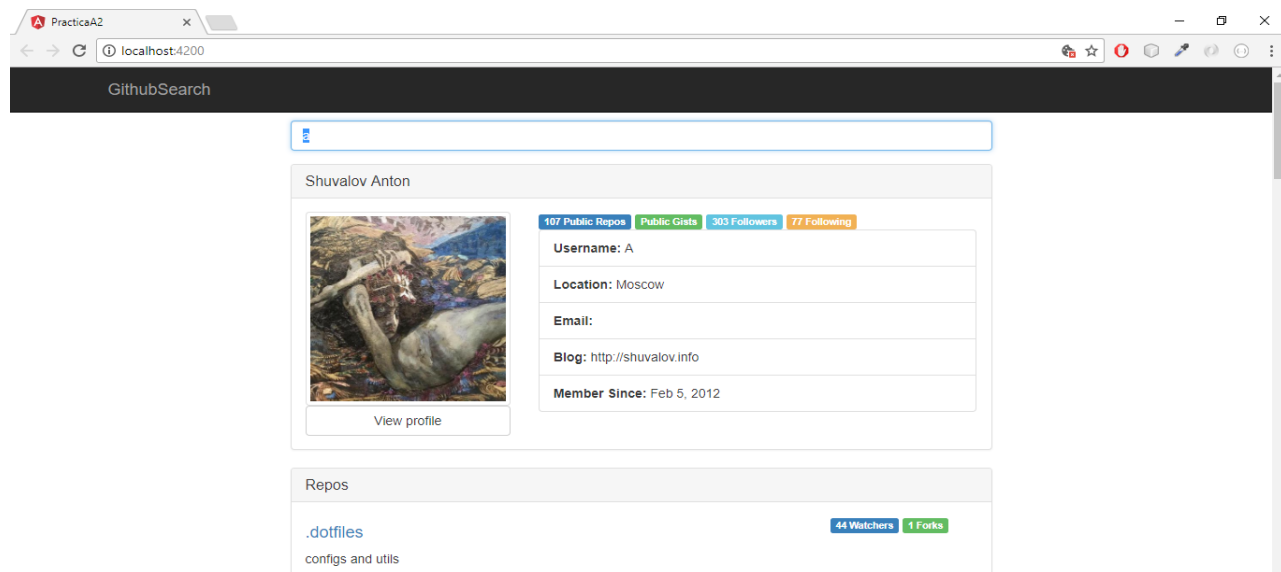
Ahora vamos a realizar un cambio en el constructor de **profile.component.ts**, donde vamos a borrar las llamadas con el objeto **\_githubService** y vamos a inicializar la variable **user** a **false**, para no mostrar ningún usuario antes de empezar a buscarlo.

```
profile.component.ts x
1  import { Component, OnInit } from '@angular/core';
2  import { GithubService } from '../services/github/github.service';
3  import 'rxjs/add/operator/map';
4
5  @Component({
6    selector: 'app-profile',
7    templateUrl: './profile.component.html',
8    styleUrls: ['./profile.component.css']
9  })
10 export class ProfileComponent implements OnInit {
11
12   public user: any = [];
13   public repos: any = [];
14
15   public username: string;
16
17   constructor(private _githubService: GithubService) {
18     this.user = false;
19   }
```

Con estos cambios, deberíamos de ver en nuestro navegador algo como esto:



Si introducimos caracteres dentro del input, irá mostrando los usuarios conforme a las coincidencias que vaya encontrando.





## 10. Registrar nuestra app en Github (Client ID y Client Secret)

La aplicación ya funciona, pero si seguimos realizando peticiones al Api de Github sin un *Client Id* y un *Client Secret*, nos acabará dejando de dar una respuesta por seguridad al cabo de un número determinado de peticiones. De modo que si queremos que la aplicación no nos deje de funcionar tenemos que registrarla en Github.

<https://github.com/settings/applications/new>

Register a new OAuth application

Application name

Something users will recognize and trust

Homepage URL

The full URL to your application homepage

Application description

Application description is optional

This is displayed to all users of your application

Authorization callback URL

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Register application Cancel

Deberemos rellenar los campos, y presionar en "**Register application**".

Register a new OAuth application

Application name

Practica-A2

Something users will recognize and trust

Homepage URL

http://localhost:4200

The full URL to your application homepage

Application description

Practica Angular2 - GithubSearcher

This is displayed to all users of your application

Authorization callback URL

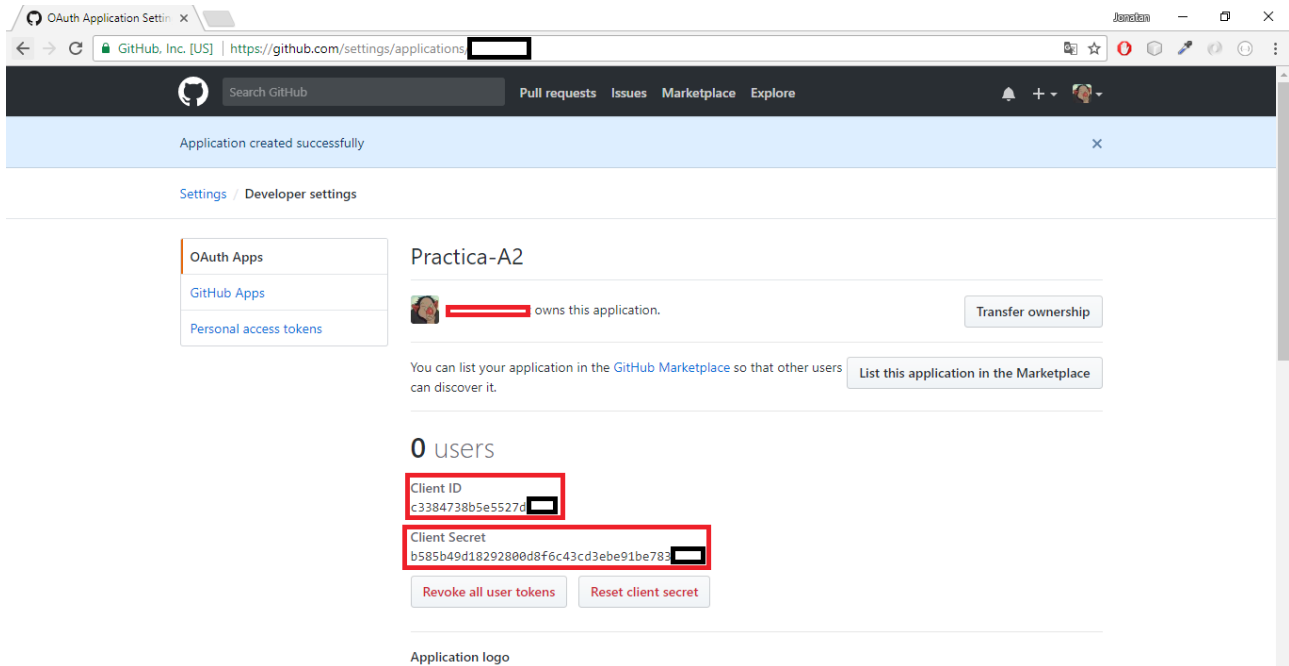
http://localhost:4200

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Register application Cancel



Esto registrará la aplicación y nos dará un **Client ID** y un **Client Secret** que usaremos concatenándolos en las rutas de las peticiones al Api.



Copiaremos el contenido de *Client ID* y *Client Secret* para crear dos variables en nuestro **github.service.ts** puesto que es aquí desde donde se realizan las peticiones al Api. Este Client ID y Client Secret debe ser único por aplicación.

```
github.service.ts
1  import { Injectable } from '@angular/core';
2  import { Http, Headers } from '@angular/http';
3  import 'rxjs/add/operator/map';
4
5  @Injectable()
6  export class GithubService {
7
8      private username: string;
9      private client_id = 'c3384738b5e5527d769b';
10     private client_secret = 'b585b49d18292800d8f6c43cd3ebe91be7836467';
11
12     constructor(private _http: Http) {}
13
14     getUser() {
15         return this._http.get('http://api.github.com/users/' + this.username + '?client_id=' + this.client_id + '&client_secret=' + this.client_secret)
16             .map(res => res.json());
17     }
18
19     getRepos() {
20         return this._http.get('http://api.github.com/users/' + this.username + '/repos?' + client_id + this.client_id + '&client_secret=' + this.client_secret)
21             .map(res => res.json());
22     }
23
24     updateUser(username) {
25         this.username = username;
26     }
27 }
```

Perfecto, ya hemos terminado.



## 11. Routes

Vamos a concluir la aplicación creando un nuevo componente **about** para usar el sistema de rutas de angular y navegar entre un componente y otro.

```
C:\WINDOWS\system32\cmd.exe
Practica-A2>ng generate component components/about
```

Ahora vamos a realizarle alguna modificación en la vista (**about.component.html**).

```
about.component.html x
1 <p>
2   Más información sobre Angular en: https://angular.io/
3 </p>
```

Para configurar el sistema de router de Angular debemos de hacer unas cuantas cosas. Una de las cosas de deberemos tener, y que por defecto, **NG-CLI** ha declarado en nuestro **index.html** es la etiqueta **<base href>** para decirle al enrutador como redactar las URL de navegación.

```
index.html x
1 <!doctype html>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6   <title>PracticaA2</title>
7   <base href="/">
8
9   <meta name="viewport" content="width=device-width, initial-scale=1">
10  <link rel="icon" type="image/x-icon" href="favicon.ico">
11  <!-- css bootstrap -->
12  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
13 </head>
14
15 <body>
16   <app-root>Loading...</app-root>
17 </body>
18
19 </html>
```

Ahora debemos importar los servicios de enrutamiento de Angular, que no es parte del core de Angular, por lo que deberemos importar los módulos de **RouterModule** y **Routes** desde su propio paquete '@angular/router'.

Esto lo haremos en un nuevo archivo que llamaremos **app.routes.ts** junto con unas cuantas líneas más y lo situaremos dentro de la carpeta **app**.



```
app.routes.ts x
1
2 import { ModuleWithProviders } from '@angular/core';
3 import { RouterModule, Routes } from '@angular/router';
4
5 import { AboutComponent } from '../components/about/about.component';
6 import { ProfileComponent } from '../components/profile/profile.component';
7
8 const routes: Routes = [
9   { path: '', component: ProfileComponent },
10  { path: 'about', component: AboutComponent }
11 ]
12
13 export const appRouterProviders: any = [];
14 export const routing: ModuleWithProviders = RouterModule.forRoot(routes);
```

Como vemos en la captura, hemos importado los módulos de angular para el enrutamiento(rojo) y también los componentes que vamos a usar para navegar entre ellos(verde).

Además declaramos un array de tipo **Routes**, al que llamaremos **routes**, y en su interior contendrá un objeto JSON por cada componente. En la propiedad **path** debe ir un string que indique el nombre de la URL con el que cargar el componente, que ira en la segunda propiedad llamada **component**(naranja).

Por último importaremos un array vacío **appRouterProviders** y y un objeto routing de tipo **ModuleWithProviders**(azul).





Ahora vamos a importar estos arrays dentro de **app.module.ts**

```
app.module.ts x
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4  import { HttpClientModule } from '@angular/http';
5
6  import { AppComponent } from './app.component';
7  import { ProfileComponent } from './components/profile/profile.component';
8  import { NavbarComponent } from './components/navbar/navbar.component';
9  import { AboutComponent } from './components/about/about.component';
10
11  import { RouterModule, Routes } from '@angular/router';
12  import { appRouterProviders, routing } from './app.routes';
13
14  @NgModule({
15    declarations: [
16      AppComponent,
17      ProfileComponent,
18      NavbarComponent,
19      AboutComponent
20    ],
21    imports: [
22      BrowserModule,
23      FormsModule,
24      HttpClientModule,
25      routing
26    ],
27    providers: [ appRouterProviders ],
28    bootstrap: [AppComponent]
29  })
30  export class AppModule { }
```

Ahora ya podemos incluir la etiqueta `<router-outlet>` que será el lugar donde se cargará el componente que se indique según la URL. Sustituiremos la etiqueta de `<app-profile>` por la de `<router-outlet>` dentro de **app.component.html**

```
app.component.html x
1  <app-navbar></app-navbar>
2
3  <br><br><br>
4
5  <div class="container">
6    <div class="row">
7      <div class="col-md-8 col-md-offset-2">
8        <router-outlet></router-outlet>
9      </div>
10    </div>
11  </div>
```



Ahora solo tenemos que usar la directiva **routerLink** para cambiar el nombre de la URL y con esto, poder navegar de un componente a otro.

```
1 <nav class="navbar navbar-inverse navbar-fixed-top">
2   <div class="container">
3     <div class="navbar-header">
4       <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar">
5         <span class="sr-only">Toggle navigation</span>
6         <span class="icon-bar"></span>
7         <span class="icon-bar"></span>
8         <span class="icon-bar"></span>
9       </button>
10      <a class="navbar-brand" href="#">GithubSearch</a>
11    </div>
12    <div id="navbar" class="collapse navbar-collapse">
13      <ul class="nav navbar-nav">
14        <li><a routerLink="">Home</a></li>
15        <li><a routerLink="about">About</a></li>
16      </ul>
17    </div>
18  </div>
19 </nav>
```

Listo, ya tenemos los enlaces para poder navegar entre componentes.



Y con esto concluimos la práctica.