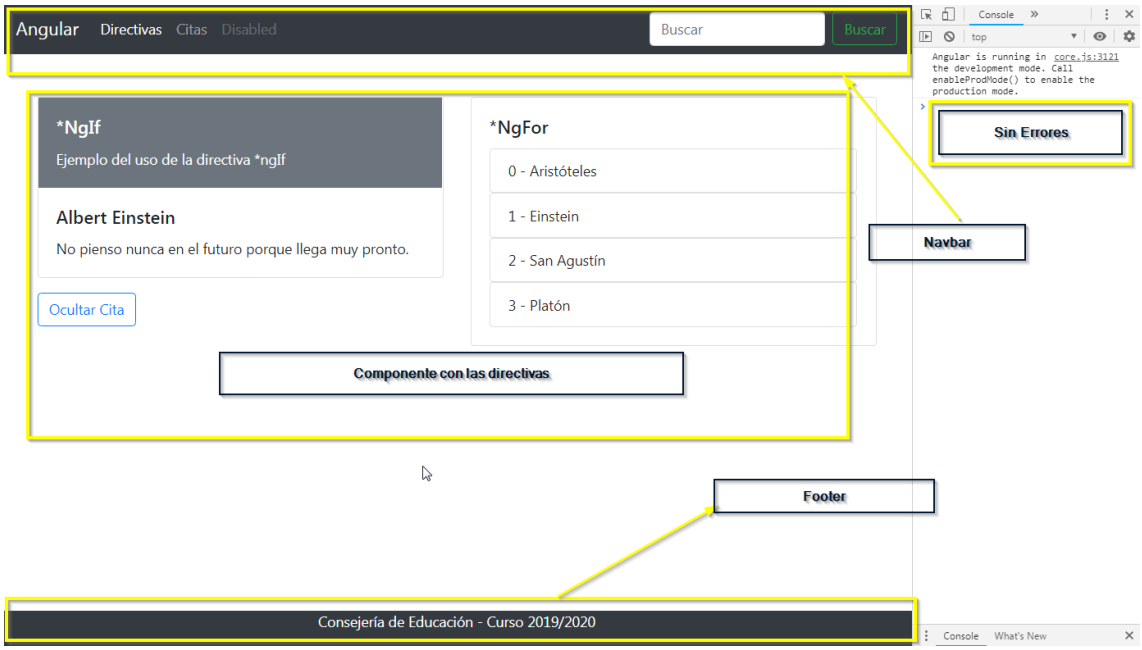
## Los Servicios

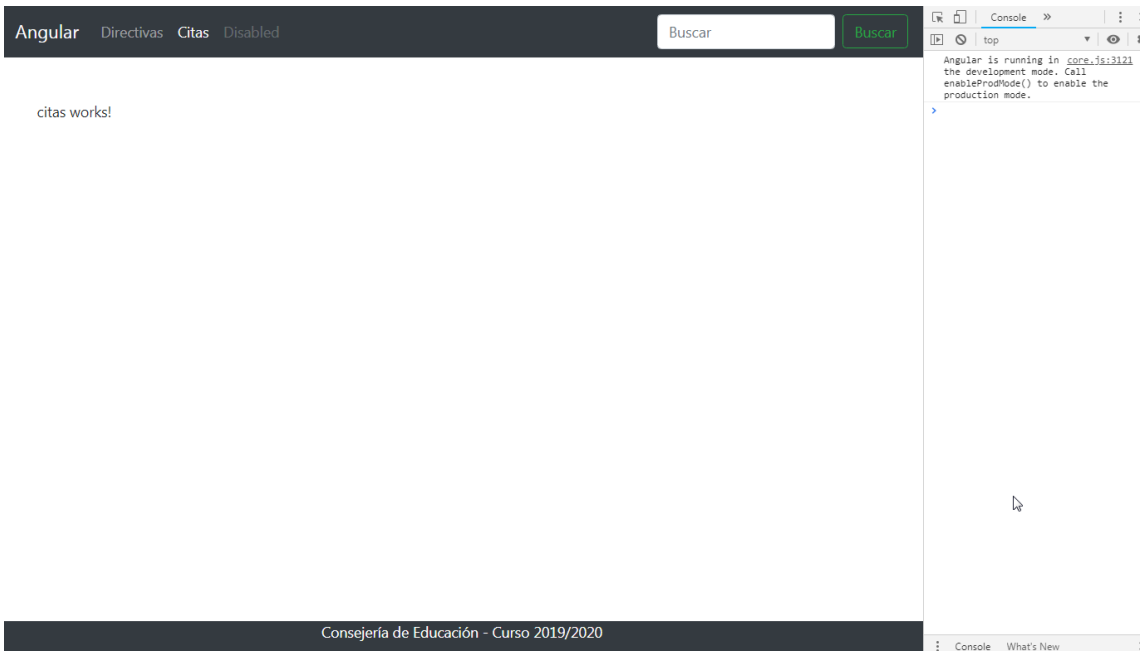
Nuestra aplicación tiene actualmente esta apariencia.

Por una parte, tenemos un navbar, un footer,  un componente con las directivas y otro componente con el cuerpo de la página ( body ) donde tenemos las directivas. Tendríamos que haber llamado en lugar de body, directivas, pero bueno, iremos así.

Además, se puede ver que es responsive y que si abrimos las herramientas de desarrollo ( F12 en Chrome ) no tenemos herrores.




Al navegar a “Citas”, vemos que se cambia el cuerpo de la pantalla por el componente citas y tenemos :




Aquí en este componente, vamos a crear una lista con Citas de autores famosos.

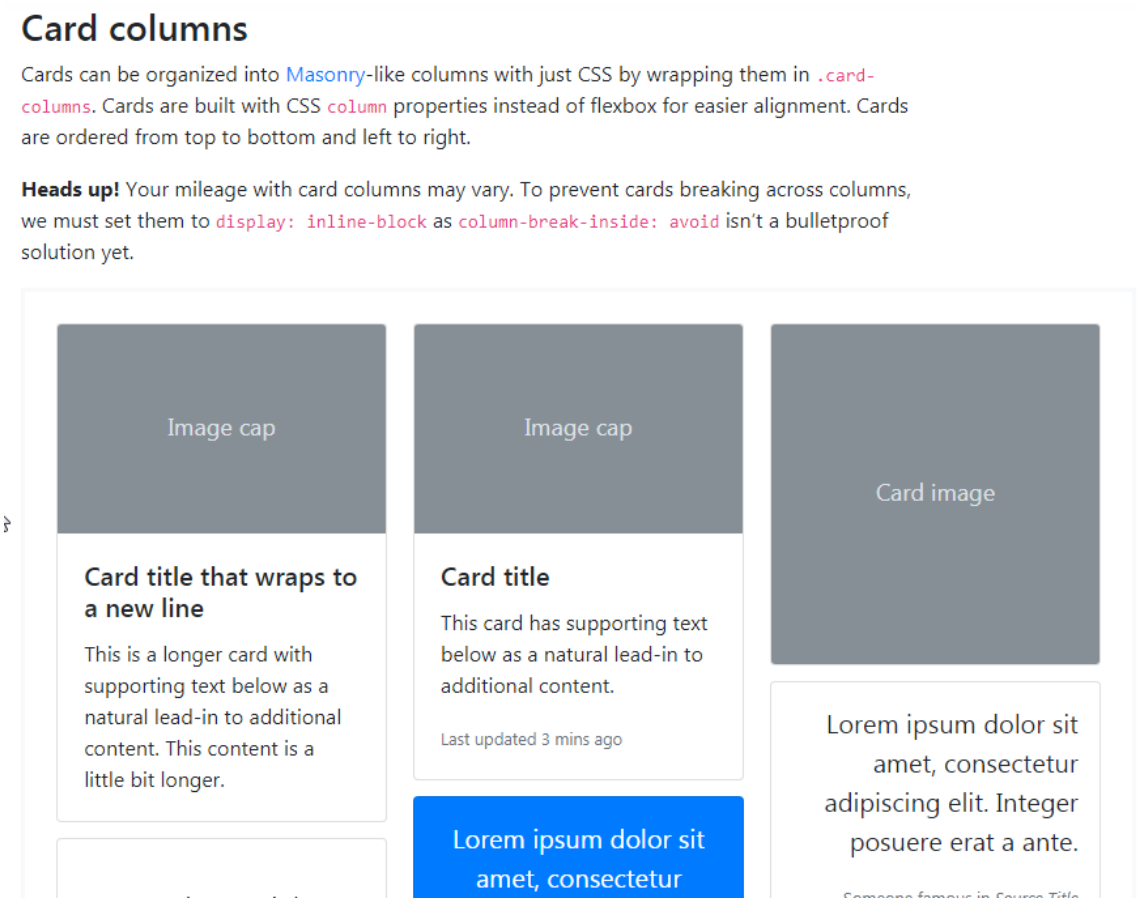
Pues vamos a ello.

Lo primero, vamos a abrir el fichero citas.component.html y donde pone

```
<p>
  citas works!
</p>
```

Lo vamos a cambiar por alguna cosita de bootstrap, para poner por ejemplo una foto del autor de la frase o cita , para ello, vamos a buscar primero por https://getbootstrap.com/ en la carpeta documentation la palabra “cards”. Buscamos una tarjeta que nos guste y que tenga una imagen.

Por ejemplo:




La primera está bien.

Más abajo está el código:

```
<div class="card-columns">
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title that wraps to a new line</h5>
      <p class="card-text">This is a longer card with supporting text below
as a natural lead-in to additional content. This content is a little bit
longer.</p>
    </div>
  </div
```

Bueno, pues lo ponemos en el fichero html de las citas y probamos.

Nos tiene que quedar el código así:

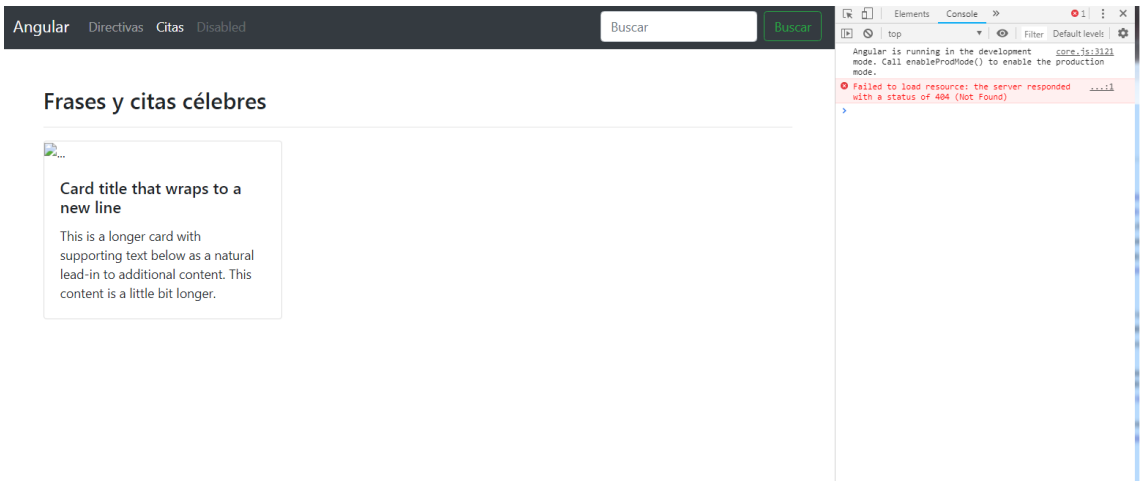
Ponemos un titulito:

```
<h3>Frases y citas célebres</h3>
<hr/>
```

Ponemos el código anterior, cerrando el div del class=”card-columns” y quedaría todo el código así:

```
<h3>Frases y citas célebres</h3>
<hr/>

<div class="card-columns">
    <div class="card">
        <img src="..." class="card-img-top" alt="...">
        <div class="card-body">
            <h5 class="card-title">Card title that wraps to a new
line</h5>
            <p class="card-text">This is a longer card with supporting
text below as a natural lead-in to additional content. This content is a
little bit longer.</p>
        </div>
    </div>
</div>
```

Probamos la aplicación y nos sale algo como esto:

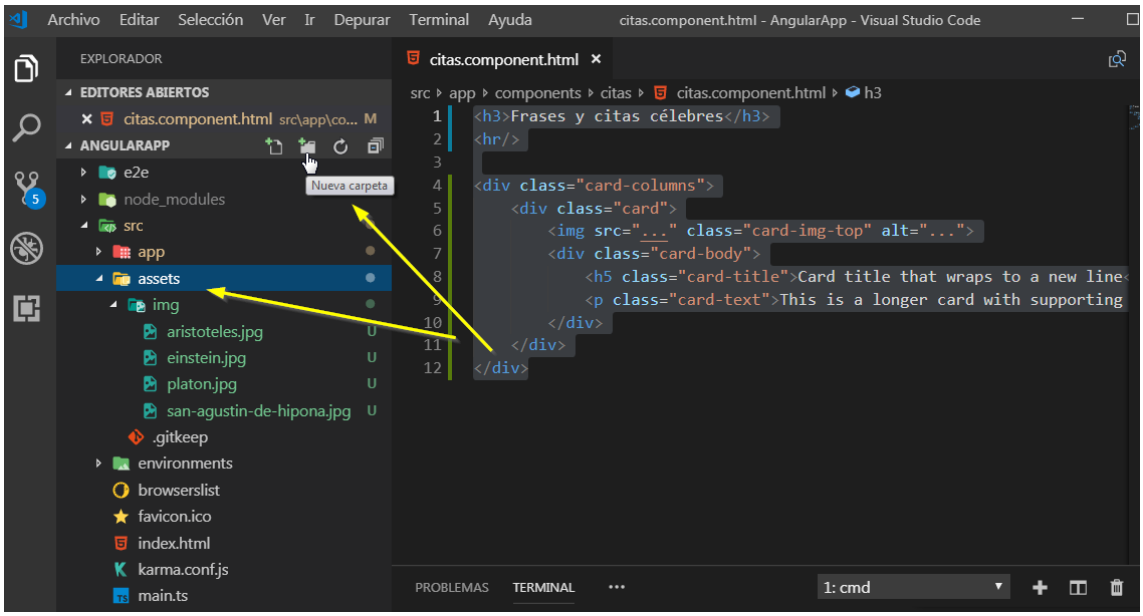



Podemos ver que hay un error en la consola.

Es debido, a que busca una imagen y ni siquiera tiene una ruta el parámetro src.

Vamos a poner imágenes.

En la carpeta assest, vamos a crear una carpeta llamada img y dentro de ella, vamos a copiar algunas imágenes. Yo he copiado ( irán en el fichero del proyecto ) cuatro imágnes.

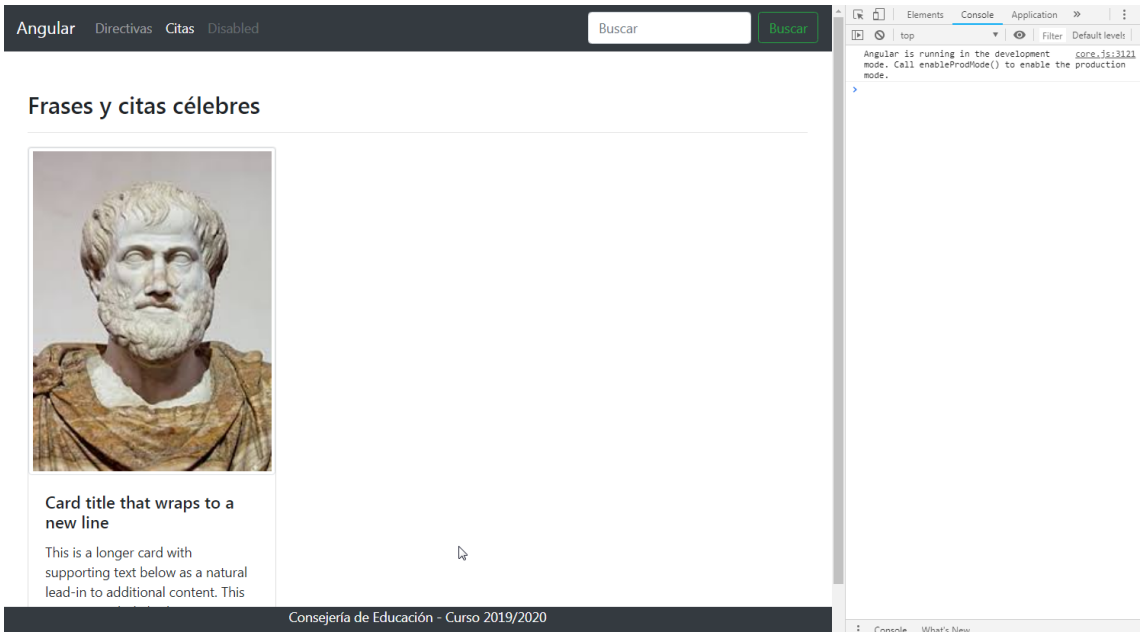



Ahora, en el código, vamos a poner bien el parámetro src y además, en el div donde está la clase card-columns vamos a poner un margen inferior para que no quede pegado el footer si se ponen muchas card ( prueba verás a lo que me refiero )

Quedando entonces el código así:

```
<h3>Frases y citas célebres</h3>
<hr/>

<div class="card-columns mb-5">
    <div class="card">
        <img src="/src/assets/img/aristoteles.jpg" class="card-img-top
img-thumbnail" alt="...">
        <div class="card-body">
            <h5 class="card-title">Card title that wraps to a new
line</h5>
            <p class="card-text">This is a longer card with supporting
text below as a natural lead-in to additional content. This content is a
little bit longer.</p>
        </div>
    </div>
</div>
```

Y nuestra aplicación así y sin errores:

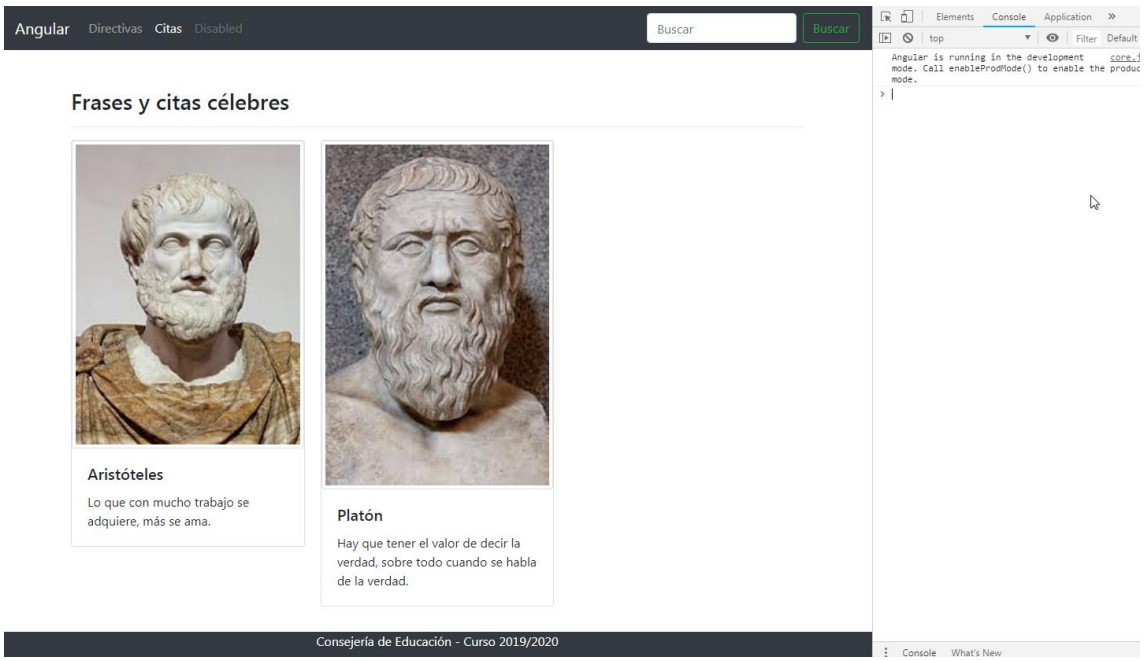


Vamos a poner a mano una frase célebre de Aristóteles y alguna de Platón para ir rellenando y luego lo iremos sacando de nuestro servicio.

De momento, dejemos el código así:

```
<h3>Frases y citas célebres</h3>
<hr/>

<div class="card-columns mb-5">
    <div class="card">
        <img src="/src/assets/img/aristoteles.jpg" class="card-img-top
img-thumbnail" alt="...">
        <div class="card-body">
            <h5 class="card-title">Aristóteles</h5>
            <p class="card-text">
                Lo que con mucho trabajo se adquiere, más se ama.</p>
        </div>
    </div>

    <div class="card">
        <img src="/src/assets/img/platon.jpg" class="card-img-top img-
thumbnail" alt="...">
        <div class="card-body">
            <h5 class="card-title">Platón</h5>
            <p class="card-text">
                Hay que tener el valor de decir la verdad, sobre todo
cuando se habla de la verdad.</p>
        </div>
    </div>




</div>
```

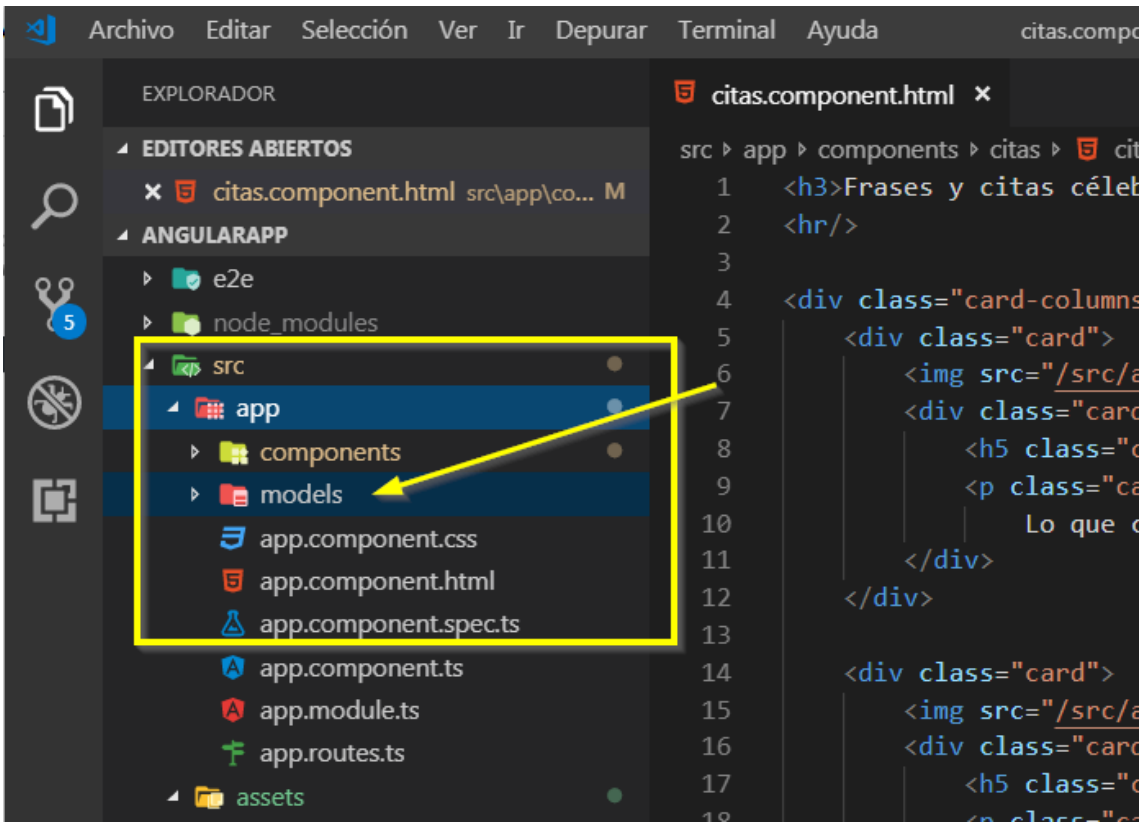
Y nuestra aplicación, tendrá esta pinta:




Lo que vamos a hacer ahora, es crear un **servicio**, que nos devuelva los datos para rellenar nuestra página de citas célebres. Este servicio nos va a devolver los datos de un fichero en formato json. Más tarde en otro documento, crearemos otro servicio que nos devolverá los datos de una backend realizado en java. Este nuevo servicio que haremos más adelante, tendrá también los métodos crear, borrar, modificar y listar.

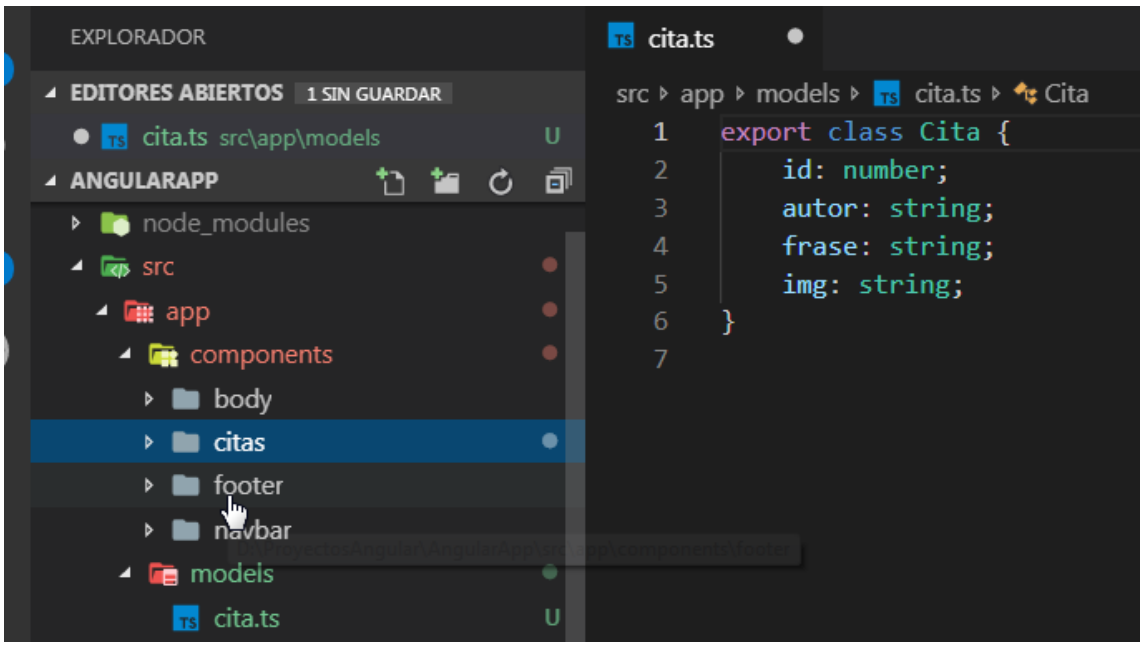
De momento, vamos a irnos preparando para nuestro primer servicio, pero como "buenas prácticas ", vamos  crear una clase "Cita" con las propiedades que tiene que tener.

Para ello, en VSC, debajo de app, vamos a crear una carpeta llamada **models.**




Dentro de esa carpeta, vamos a crear una archivo, que le llamaremos cita.ts

Y pondremos los atributos que de momento vamos a utilizar. Nos tiene que quedar así:




Si algún atributo de la clase, es opcional, por ejemplo pudiera ser la imagen si no la tenemos, basta con poner ese atributo con una interrogación. Por ejemplo:

```
img?: string;
```

Antes de hacer el servicio, vamos a ver su necesidad, creando en el componente citas, un array de objetos de la clase Cita.

Por eso , en citas.component.ts vamos a crear este array, haciendo:


Creamos una variable de tipo array de tipo Cita e importamos la clase que hemos creado anteriormente y la cargamos con unos valores, quedando el fichero citas.component.ts de la forma:

```
import { Component, OnInit } from '@angular/core';
import { Cita } from 'src/app/models/cita';

@Component({
  selector: 'app-citas',
  templateUrl: './citas.component.html',
  styleUrls: ['./citas.component.css']
})
export class CitasComponent implements OnInit {

  citas: Cita[] = [
    {
      id: 1,
      autor: 'Aristóteles',
      frase: 'Lo que con mucho trabajo se adquiere, más se ama.' ,
      img: 'assets/img/aristoteles.jpg'},
    {
      id: 2, autor: 'Platón',
      frase: 'Hay que tener el valor de decir la verdad, sobre todo
cuando se habla de la verdad.' ,
      img: 'assets/img/platon.jpg'
    },
```

```
    {
      id: 3, autor: 'Einstein',
      frase: 'Dar ejemplo no es la principal manera de influir sobre los
demás; es la única manera.',
      img: 'assets/img/einstein.jpg'
    },
  ];

  constructor() { }

  ngOnInit() {
  }

}
```

Además, en la vista, vamos a utilizar el ngFor, para que se muestren todas nuestras frases de forma dinámica.

Y nos quedará el citas.component.html así:

```
<h3>Frases y citas célebres</h3>
<hr/>

<div class="card-columns mb-5">
    <div class="card" *ngFor="let cita of citas">
        <img [src]="cita.img" class="card-img-top img-thumbnail">
        <div class="card-body">
            <h5 class="card-title">{{ cita.autor }}</h5>
            <p class="card-text">
                {{ cita.frase }} </p>
        </div>
    </div>

</div>
```

Ahora toca quitar la variable citas y llevar todas las “citas” a un fichero en formato json.

Creamos el fichero citas.json.ts en la carpeta citas y además, importaremos la clase cita en ese fichero y crearemos una constante con todas las citas que tenemos en citas.component.ts.

Al lío.

El fichero, simula los datos que nos vienen de un api rest.

Primero en la carpeta citas, el fichero citas.json.ts y con el contenido:

```
import { Cita } from 'src/app/models/cita';

export const CITAS: Cita[] = [
    {
      id: 1,
      autor: 'Aristóteles',
      frase: 'Lo que con mucho trabajo se adquiere, más se ama.' ,
      img: 'assets/img/aristoteles.jpg'},
    {
      id: 2, autor: 'Platón',
      frase: 'Hay que tener el valor de decir la verdad, sobre todo
cuando se habla de la verdad.' ,
      img: 'assets/img/platon.jpg'
```

```
    },
    {
      id: 3, autor: 'Einstein',
      frase: 'Dar ejemplo no es la principal manera de influir sobre los
demás; es la única manera.',
      img: 'assets/img/einstein.jpg'
    },
  ];
```

Ahora, en nuestro citas.component.ts, quitamos todo el contenido del array de citas y lo importamos del fichero que hemos creado y nos quedará así:

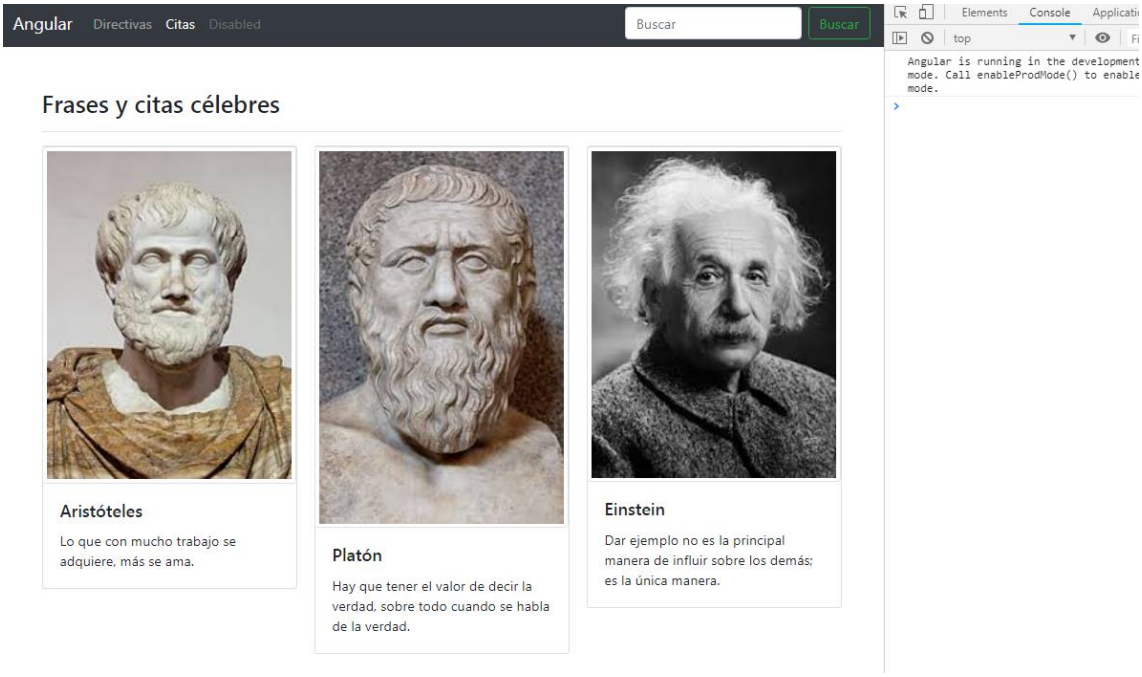
```
import { Component, OnInit } from '@angular/core';
import { Cita } from 'src/app/models/cita';
import { CITAS } from './citas.json';

@Component({
  selector: 'app-citas',
  templateUrl: './citas.component.html',
  styleUrls: ['./citas.component.css']
})
export class CitasComponent implements OnInit {

  citas: Cita[];

  constructor() { }

  ngOnInit() {
    this.citas = CITAS;
  }

}
```

Lo que hemos hecho es:

a. Importar la constante CITAS que tiene el contenido del fichero
b. Crear una variable citas de tipo array de Citas.

c. En el método ngOnInit() que es el que se ejecuta cuando se inicializa el componente, asignamos el valor de la constante a la propiedad citas que es la que se utiliza en la vista. Se podría haber hecho en el constructor, pero para ir viendo poco a poco como va funcionando este método y los que veremos.

   Nuestra aplicación tiene que seguir funcionando así:



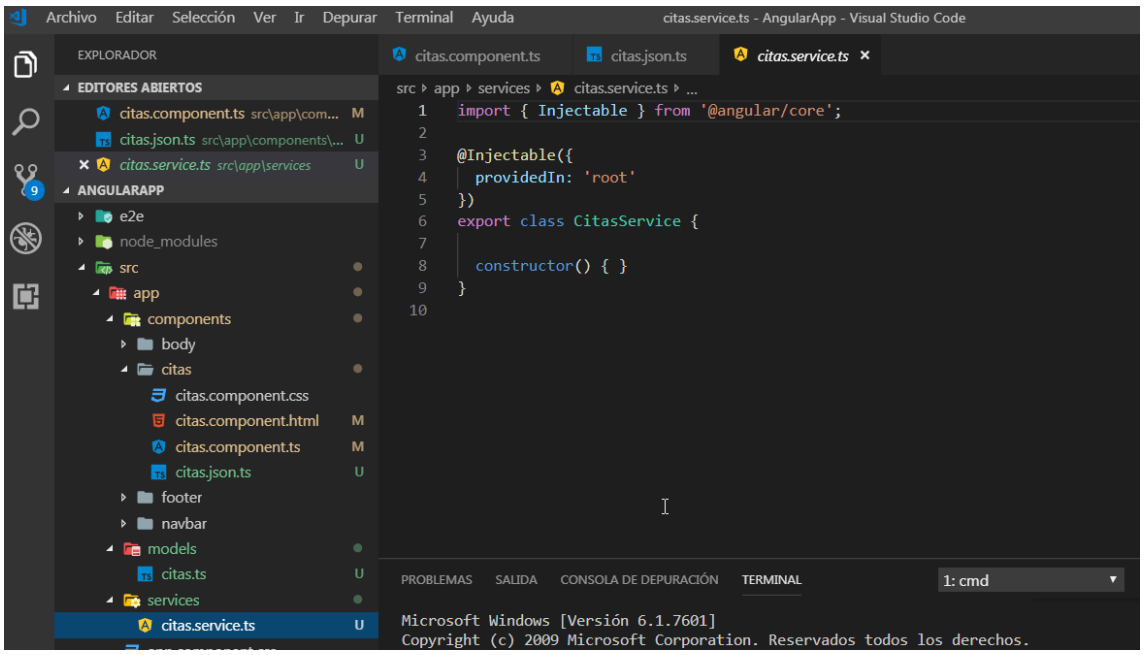

Ahora vamos a quitar lo referente a los datos de nuestro ts del componente citas y lo vamos a mover a nuestro servicio citas. El servicio, es una clase especializada en devolver los datos.

Por lo tanto, vamos a crear el servicio citasService, se puede hacer manual, pero lo haremos con el CLI. En el terminal de VSC decimos y escribimos. (angular) ng ( genera) g ( un servicio ) s ( en la carpeta services y que se llame citas) services/cita ( pero no me generes el archivo de prueba unitarias ) --spec=false ( si este pequeño curso tiene algo de éxito, hacemos unas pruebas)

Entonces:

ng g s services/citas --spec=false

y nos generará esto:




El decorador @Injectable, configurado con la propiedad provideIn = root, lo pone de forma automática en las versiones 6 y 7 de Angular. Si no lo pone, hay que ir al fichero app.module.ts y añadir el servicio que hemos creado en

```
providers: [],
```

Se puede por lo tanto no dar de alta en el providers o darlo de alta si en el servicio se ha creado únicamente con la anotación @Inyectable.

En el citas.service.ts tenemos que tener este código:

```
import { Injectable } from '@angular/core';
import { CITAS } from '../components/citas/citas.json';


@Injectable({
  providedIn: 'root'
})
export class CitasService {

  constructor() {
    console.log('Entrando en el servicio');
   }
```

```
  getCitas() {
    return CITAS;
  }
}
```

¿Qué es lo que hemos hecho?

a. Hemos importado CITAS que es el fichero del json
b. Hemos puesto en el contructor un console.log, para ver si es verdad que pasa por aquí cuando inyectemos el servicio en nuestra clase ts de las citas.
c. Hemos creado un método que nos devuelve todas las Citas. Sería mucho mejor, poner en el método que devuelve un array de Citas de la forma:

```
d.   getCitas(): Cita[] {
e.     return CITAS;
f.   }
```

Y claro, lo importamos

```
import { Citas } from '../models/citas.js';
```

pero Bueno….

g. Luego, en el fichero citas.component.ts, hemos dejado el código así:

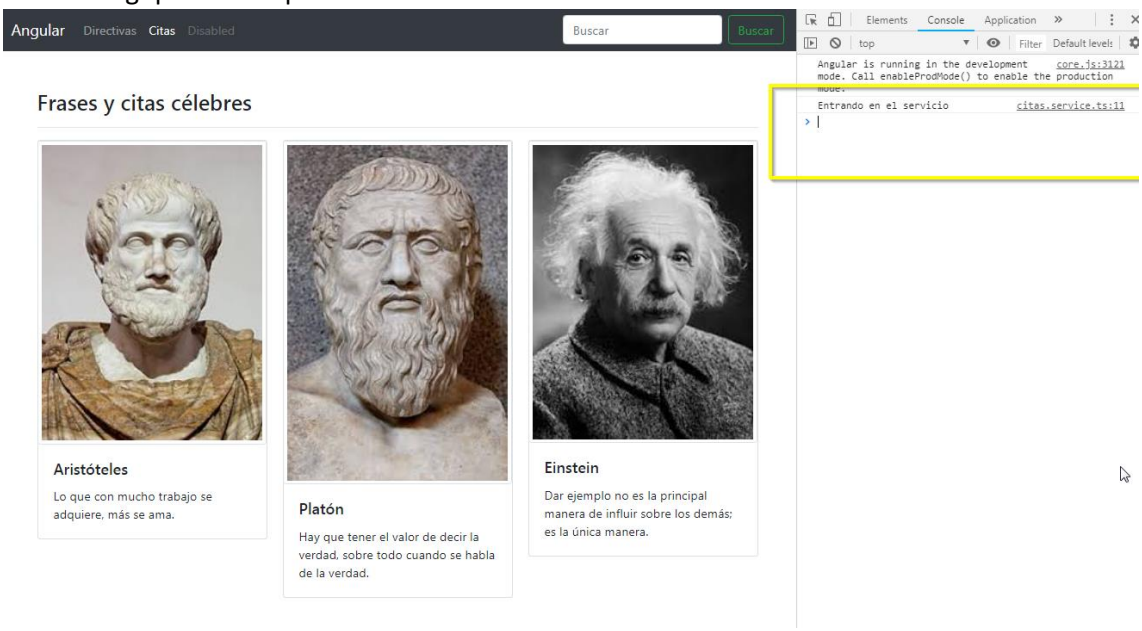
```
h. import { Component, OnInit } from '@angular/core';
i. import { CitasService } from '../../services/citas.service';
j. import { Cita } from 'src/app/models/cita';
k.
l. @Component({
m.   selector: 'app-citas',
n.   templateUrl: './citas.component.html',
o.   styleUrls: ['./citas.component.css']
p. })
q. export class CitasComponent implements OnInit {
r.
s.   citas: Citas[];
t.
u.   constructor(private citasService: CitasService) {
v.     this.citas = citasService.getCitas();
w.   }
x.
y.   ngOnInit() {
```

```
z.    }
aa.
bb.}
cc.
```

¡Mucho más limpio!
Y ¿Qué es lo que hemos hecho?

Únicamente tenemos un atributo citas que es un array y en el el constructor de la clase, inyectamos el servicio y asignamos el valor de las citas de nuestro servicio a la propiedad.

Si probamos, todo tiene que funcionar y además se muestra en la consola el console.log que hemos puesto en el constructor del servicio:




Pero estas clases servicio, normalmente, como he puesto más arriba, se utilizarán para las llamadas a los servicios Rest que implementaremos más adelante.
Ahora, el método getCitas, es un método síncrono y no funcionaría en un escenario real, donde la petición tiene que ser asíncrona para no bloquear la aplicación cliente mientras se espera la respuesta. Además en el servidor se pueden hacer varias peticiones al mismo tiempo y que no estén sincronizadas entre sí y que se ejecuten en tiempo real. Esto son los métodos reactivos, que reaccionan en tiempo real y a través de flujos de datos (stream ) de entrada y salida. Lo vamos a ver un poco mas abajo,

cuando tengamos que "suscribirnos" a un stream de datos para capturar el parámetro que nos viene por url.

**Rutas con Parámetros**

Vamos a terminar esta versión de nuestra aplicación y vamos a navegar a una página ( ruta ) de detalle de una cita utilizando los parámetros en la petición, vamos a crear un componente llamado cita donde tendremos el detalle de la cita en cuestión.
De forma, que cuando demos click en un Ver Mas, nos lleve a una página del detalle.

Al lío:

Primero, en nuestra página de la vista, vamos a poner un botón para navegar a la cita que hemos pinchado.
Hay dos formas, o de forma programática al dar a un botón o con un <a> y el enlace.

Vamos a poner las dos formas.
Primero el botón y ponemos algo así en el componente de la vista citas.component.html

```
<button (click)="verCita()" class="btn btn-outline-primary">Ver
Mas...</button>
```

Esto lo que hace es mostrar un botón y cuando se pulsa click, ejecuta una función de citas.component.ts donde navegaremos a la cita.

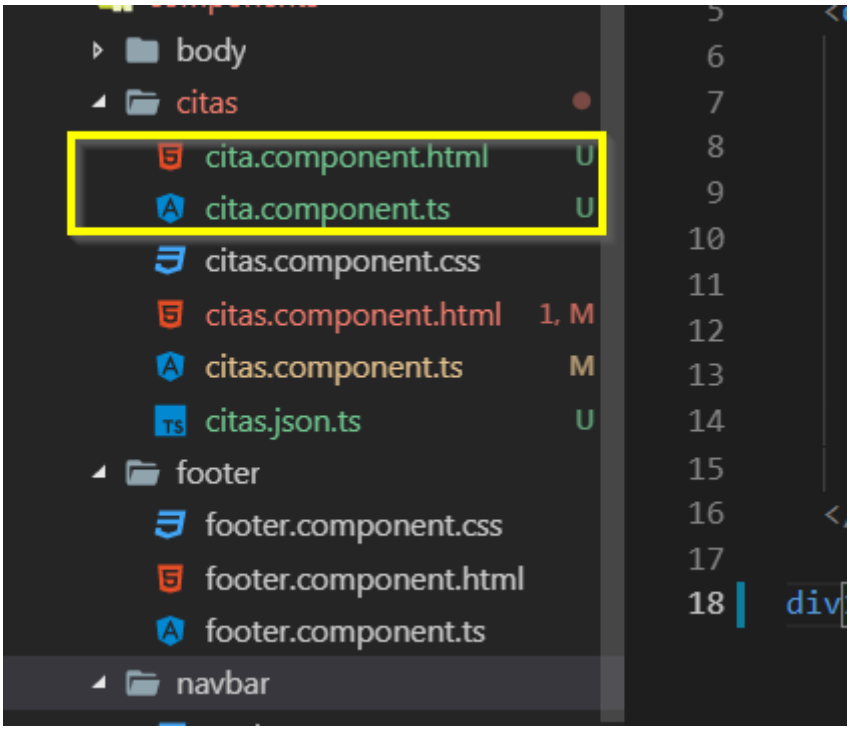
Otra forma es utilizar esto:

```
<a href="" class="btn btn-outline-primary">Ver Mas...</a>
```

Donde ponemos un link con la misma clase del botón.
En el href hay que poner a la página donde tiene que ir.

Como vemos, estamos creando la forma de navegación a una página, pero no tenemos esa página. Por lo tanto, lo primero que vamos a hacer es crear un componente cita, que le crearemos dentro de la carpetas citas y será ese componente el que nos muestre el detalle de la cita individual. Bueno, la verdad es que no hay mucho que mostrar, pero para ver el funcionamiento de la navegación y el envío y recepción de parámetros, es un buen ejemplo.
Al lío.
Abrimos la terminal de VSC y decimos y escribimos.

(Angular, créame un componente llamado cita en la carpeta citas, pero no me hagas otra carpeta, además, no me generes el fichero de pruebas "spec" y tampoco la hoja de estilos )
ng g c components/citas/cita --flat --spec=false –is




Podemos ver que en cita.component.html tenemos

```
<p>
  cita works!
</p>
```

Claro, que para navegar a este componente, hay que darlo de alta en el fichero de las rutas, y nos quedará así:
En app.routes.ts lo agregamos, pero además poniendo que recibirá un parámetro:

```
{ path: 'cita/:id', component: CitaComponent },
```

He importamos el componente

```
import { CitaComponent } from './components/citas/cita.component';
```

tiene que quedarnos así:

```
import { RouterModule, Routes } from '@angular/router';
import { BodyComponent } from './components/body/body.component';
import { CitasComponent } from './components/citas/citas.component';
import { CitaComponent } from './components/citas/cita.component';

const APP_ROUTES: Routes = [
    { path: 'body', component: BodyComponent },
    { path: 'citas', component: CitasComponent },
    { path: 'cita/:id', component: CitaComponent },
    { path: '**', pathMatch: 'full', redirectTo: 'body' }
];

export const APPROUTING = RouterModule.forRoot(APP_ROUTES);
```

lo que decimos es que cuando se ponga cita/44, pues tiene que ir al componente cita y mandando el parámetro 44
Podemos probar a ver si nuestra ruta funciona, escribiendo en el navegador y teniendo el servidor de angular levantado lo siguiente:

http://localhost:4200/cita/88


Esto nos dice que por lo menos el componente funciona y que la ruta también.
Ahora, vamos a dar vida a nuestro botón y a el enlace.

En el botón hemos puesto que cuando se haga click, nos ejecute el método verCita(), pues lo hacemos en el ts del componente citas.

```
<button (click)="verCita( cita.id )" class="btn btn-outline-primary">Ver
Mas...</button>
            <br><br>
```

```
            <a [routerLink]="['/cita', cita.id ]" class="btn btn-outline-
primary">Ver Mas...</a>
```

En el botón, hemos puesto que ejecute el método verCita y enviamos como parámetro el id de la cita.
Luego, el método verCita() del componente, hace esto:

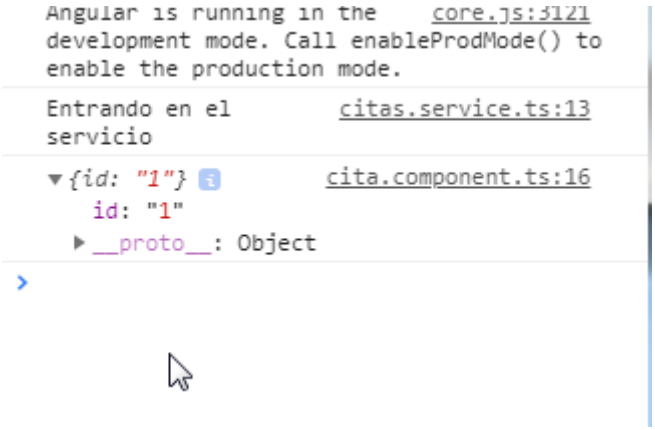
```
import { Component, OnInit } from '@angular/core';
import { CitasService } from '../../services/citas.service';
import { Cita } from 'src/app/models/cita';
import { Router } from '@angular/router';

@Component({
  selector: 'app-citas',
  templateUrl: './citas.component.html',
  styleUrls: ['./citas.component.css']
})
export class CitasComponent implements OnInit {

  citas: Cita[];

  constructor(private citasService: CitasService,
              private router: Router) {
    this.citas = citasService.getCitas();
  }

  ngOnInit() {
  }
  verCita(id: number) {
    this.router.navigate(['/cita', id]);
  }
}
```

Navega a "/cita", enviando el id que se recibe como parámetro. Pero para capturar el parámetro hacemos uso del componente Router que inyectamos en el constructor.

En el caso del <a> utilizamos el routerLink de angular y como podéis ver se envían dos parámetros, el primero con la ruta y el segundo con el parámetro.

Si no hubiese que enviar ningún parámetro, bastaría con poner la primera parte y si son 5 parámetros pues se ponen separados por comas.
En cualquier caso, cuando se navegue a cita.component, el constructor tiene que capturar el parámetro que nos viene por la url y llamar al servicio para que nos devuelva la cita que queremos.
Al lío:
En cita.component.ts, inyectamos el componente ActivatedRoute
Este componente ActivatedRoute tiene un método llamado params, que nos devuelve un "Observador", ¿Qué es esto?, es lo que adelanté a la hora de llamar a un servicio asíncrono, esto está pendiente de cambios y nosotros nos "suscribimos" para recibir esos cambios. Veamos, nos vamos a suscribir y cuando tengamos el stream de los datos recibidos por el observalos, los vamos a mostrar en la consola.

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

@Component({
  selector: 'app-cita',
  templateUrl: './cita.component.html',
  styles: []
})
export class CitaComponent implements OnInit {

  idCita: number;

  constructor(private activatedRoute: ActivatedRoute) {

    this.activatedRoute.params.subscribe ( data => {
      console.log ( data );
    });
   }

  ngOnInit() {
  }

}
```




Estamos imprimiendo en la consola, los datos que nos trae el observador.

Pero realmente, nos hace falta únicamente el id para llamar al servicio y que nos traiga los datos del id que nos han mandado.

```
console.log ( data.id );
```

Pues ese será el id de la cita que queremos.
Inyectaremos el servicio en este componente, crearemos una variable de tipo Cita y la rellenaremos con los datos que vengan de un método que tenemos crear en el servicio y que nos devuelva la cita como tal.

Al lío.
Primero en el servicio citas.service.ts, vamos a crear un método que al recibir un id, devuelva la cita que tiene ese id. Hemos hecho lo siguiente:

```
import { Injectable } from '@angular/core';
import { CITAS } from '../components/citas/citas.json';
import { Cita } from '../models/cita.js';

@Injectable({
  providedIn: 'root'
})
export class CitasService {

  citas: Cita[];

  constructor() {
```

```
    console.log('Entrando en el servicio');
    this.citas = this.getCitas();
   }

  getCitas(): Cita[] {
    return CITAS;
  }

  getCita( i: number ): Cita {

    console.log(this.citas);
    const respondCitas = this.citas.filter( c => {
        return c.id === Number(i);
        } );

    return respondCitas[0];

  }
}
```

Hemos creado un atributo de la clase para tener el array de las citas,

Luego , hemos creado el método getCita, que recibe un número .

En el método, hacemos un filtro del array de las citas y que devuelva un array con todos los elementos que cumplen una condición. Como puedes ver, aunque i sea number, lo he tenido que poner Number ( i ) porque al ser la igualación === , me da problemas.

Luego, devuelve la primera posición del array.

(Claro que esto tendría que tener validaciones, que exista, etc….)

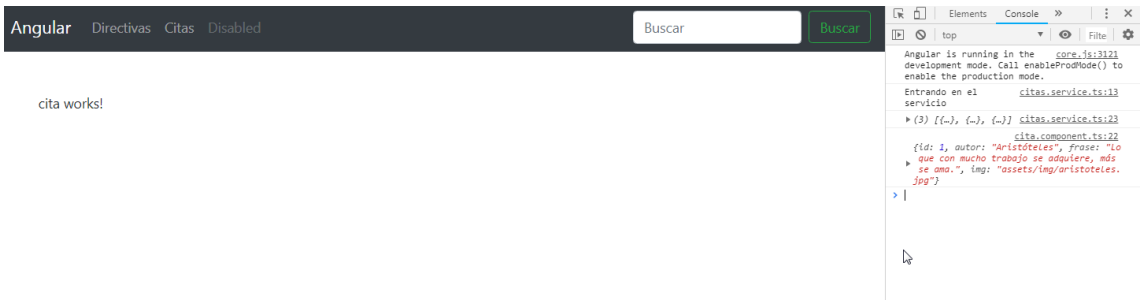
En el código del componente cita.component.ts, hemos capturado el id de los parámetros de la url y hemos inyectado el servicio para llamar al servicio y traernos la cita.

De momento, únicamente hacemos un console.log a ver si funciona

```
import { Injectable } from '@angular/core';
import { CITAS } from '../components/citas/citas.json';
import { Cita } from '../models/cita.js';
```

```
@Injectable({
  providedIn: 'root'
})
export class CitasService {

  citas: Cita[];

  constructor() {
    console.log('Entrando en el servicio');
    this.citas = this.getCitas();
   }

  getCitas(): Cita[] {
    return CITAS;
  }

  getCita( i: number ): Cita {

    console.log(this.citas);
    const respondCitas = this.citas.filter( c => {
        return c.id === Number(i);
        } );

    return respondCitas[0];

  }
}
```

Y debería funcionar:



Por cierto, el constructor del servicio, solo se ejecuta una vez aunque se esté inyectando en otros componentes.

Ahora, que ya tenemos en la variable cita la que nos ha traído el servicio, pues vamos construir el html.

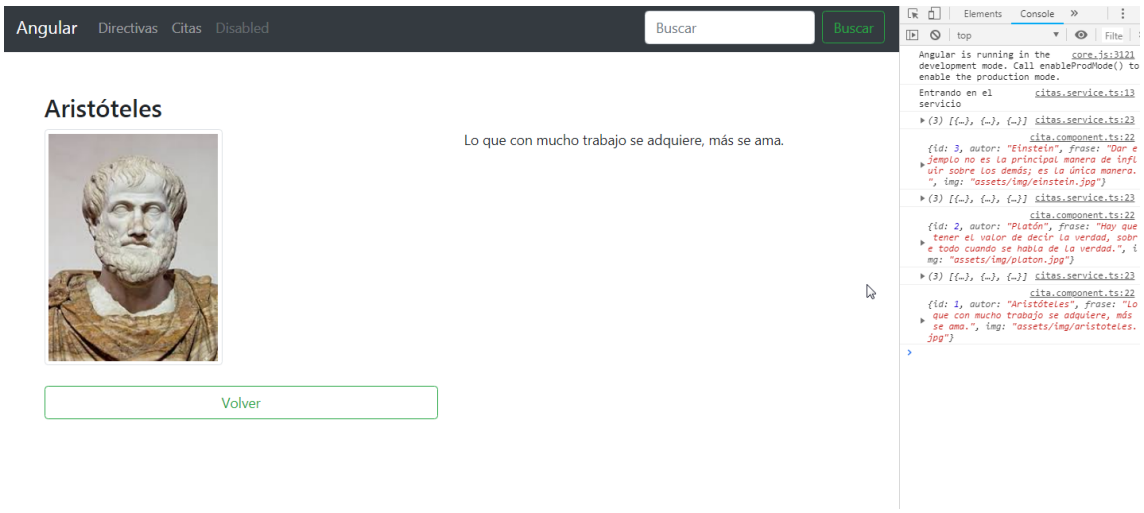
Únicamente, vamos a poner un título, la imagen a la izquierda y el texto.

Para el ejemplo nos sirve, y un botón regresar.

```
<h3>
    {{ cita.autor }}</h3>

<div class="row">
    <div class="col">
        <img [src]="cita.img" class="img-thumbnail">
        <br>
        <br>
        <a [routerLink]="['/citas']" class="btn btn-outline-success  btn-
block">Volver</a>
    </div>
    <div class="col ">
        {{ cita.frase }}
    </div>


</div>
```

No sé si esto dará alguna duda, espero que no.

La página nos tiene que quedar así:




Para ir terminando, vamos a dar “vida” al buscar que hemos puesto en el navbar.

Para ello, primero en el navbar.component, en el botón vamos a ponerlo de tal forma que según se pulse “Buscar” dispare la función del ts.

Para ello, vamos a hacer una referencia la objeto input con #buscarTexto y ponemos en el botón el evento click, llamando a la función del ts

```
<input class="form-control mr-sm-2" type="text" placeholder="Buscar"
aria-label="Search" #textoBuscar>
            <button (click)="buscarCita(textoBuscar.value)" class="btn
btn-outline-success my-2 my-sm-0" type="button">Buscar</button>
```

También hemos cambiado el type del button  de search a button. Para que no haga submit. Los formularios los veremos en otro “capítulo”.

Ahora la función en el navbar.component.ts para ir viendo si funciona:

```
  buscarCita( texto: string ) {
    console.log(texto);
  }
```

Muy bien, como es lógico, tenemos que crear la función en el servicio para buscar las citas, pues vamos con ello.

Quedaría así:

```
import { Component, OnInit } from '@angular/core';
import { CitasService } from '../../services/citas.service';
import { Cita } from 'src/app/models/cita';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})
export class NavbarComponent implements OnInit {

  citasEncontradas: Cita[] = [];

  constructor(private citasService: CitasService) {}

  ngOnInit() {
  }

  buscarCita( texto: string ) {
    console.log(texto);
    this.citasEncontradas =  this.citasService.buscarCitas( texto );
    console.log( this.citasEncontradas );
  }

}
```

Ahora, vamos a crear otro componente, para sacar las citas encontradas aunque más tarde lo cambiaremos, pero primero por partes vamos a crear un componente para mostrar el resultado de la búsqueda.

1. Cremos el componente

ng g c components/citas/mostrarCitas --flat --spec=flase –is

2. Agregamos la ruta nueva en app.routes

```
3. { path: 'mostrar/:termino', component: MostrarCitasComponent },
```

He importamos

```
import { MostrarCitasComponent } from './components/citas/mostrar-
citas.component';
```

En el mostrar.ts tenemos que coger el termino que viene en la url….

Entonces, hemos puesto el mostrarcitas.ts con este contenido

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { CitasService } from '../../services/citas.service';
import { Cita } from 'src/app/models/cita';

@Component({
  selector: 'app-mostrar-citas',
  templateUrl: './mostrar-citas.component.html',
  styles: []
})
export class MostrarCitasComponent implements OnInit {


  citas: Cita[] = [];

  constructor(private activatedRoute: ActivatedRoute,
              private citasService: CitasService) {}

  ngOnInit() {
    this.activatedRoute.params.subscribe( params => {
      this.citas = this.citasService.buscarCitas( params['termino']);
      console.log(this.citas);
    });
  }

}
```

Y para el html, vamos a copiar todo lo de mostrar todas las citas..

Quitando los botones, pero además, vamos a poner un alert, cuando no haya datos en lo que se ha buscado.

Buscamos el alert en bootstrap.

De forma que el html del compoente mostrar quedará así:

```
<h3>Resultado de la Búsqueda: <small>{{ terminoDeBusqueda }}</small></h3>
<hr/>

<div *ngIf="citas.length==0" class="row">
    <div class="col">
        <div class="alert alert-secondary" role="alert">
            No se han encontrado datos con el valor {{ terminoDeBusqueda
}}
        </div>
    </div>
</div>
<div *ngIf="citas.length > 0" class="card-columns mb-5">
    <div class="card" *ngFor="let cita of citas">
        <img [src]="cita.img" class="card-img-top img-thumbnail">
        <div class="card-body">
            <h5 class="card-title">{{ cita.autor }}</h5>
            <p class="card-text">
                {{ cita.frase }} </p>

        </div>
    </div>

</div>
```

Y el ts e mostarcitas, quedará así:

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

```
import { CitasService } from '../../services/citas.service';
import { Cita } from 'src/app/models/cita';

@Component({
  selector: 'app-mostrar-citas',
  templateUrl: './mostrar-citas.component.html',
  styles: []
})
export class MostrarCitasComponent implements OnInit {


  citas: Cita[] = [];
  terminoDeBusqueda: string;

  constructor(private activatedRoute: ActivatedRoute,
              private citasService: CitasService) {}

  ngOnInit() {
    this.activatedRoute.params.subscribe( params => {
      this.citas = this.citasService.buscarCitas( params['termino']);
      this.terminoDeBusqueda = params['termino'];
      console.log(this.citas);
    });
  }

}
```

No sé si algo necesitará aclaración, ya sabéis donde estamos.

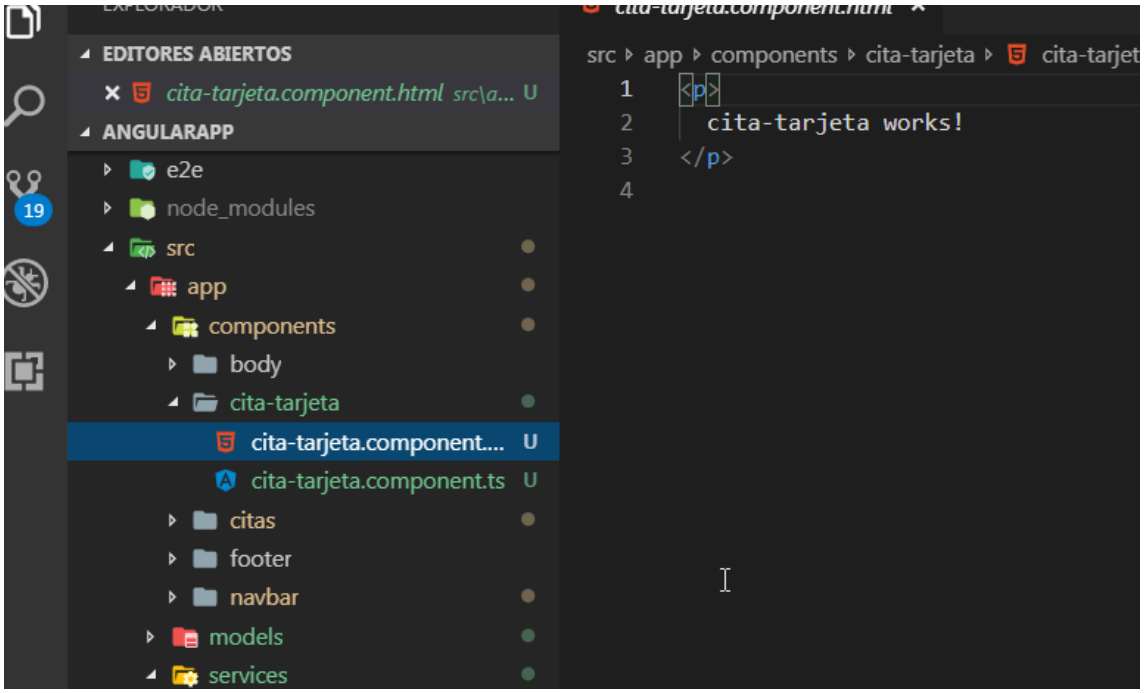
Pero, hay algo que seguramente alguno se está preguntando y es el tema de reaprovechamiento del código.

Hemos copiado y pegado las tarjetas de nuestras “citas célebre”. Pero ¿No sería bueno, tener un componente especializado en mostrar esas tarjetas y que se aproveche en la página que muestra todas las citas  y además en la que devuelve la búsqueda?

**Bienvenidos al @Input  @Output**

Vamos a crear un componente que le llamaremos citaTarjeta y será el que vamos a aprovechar.

ng g c components/citaTarjeta –spec=false –is




Lo que haremos es que donde se utilizan las tarjetas, se utilice este componente….

En el cita-tarjeta.ts, necesitamos la cita de cada tarjeta….

¿Qué hacemos?,

Pues en el el html de las citas lo dejaremos así:

```
<h3>Frases y citas célebres</h3>
<hr/>

<div class="card-columns mb-5">
    <app-cita-tarjeta [cita]="cita" *ngFor="let cita of citas"></app-
cita-tarjeta>
</div>
```

Lo que hacemos es poner el selector del componente citaTarjeta y con un ciclo for, vamos a ir enviando al parámetro cita, (se pone entre corchete) la cita que va en el for y el ts de la tarjeta, cogerá ese valor con @input y todo lo demás igual. También el ts, ponemos la función VerCita que teníamos en citas.component.ts y debe funcionar genial.

Así quedará el cita-tarjeta.compontent.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { Cita } from 'src/app/models/cita';
import { Router } from '@angular/router';

@Component({
  selector: 'app-cita-tarjeta',
  templateUrl: './cita-tarjeta.component.html',
  styles: []
})
export class CitaTarjetaComponent implements OnInit {

  @Input() cita: Cita  ;

  constructor(private router: Router) { }

  ngOnInit() {
  }

  verCita(id: number) {
    this.router.navigate(['/cita', id]);
  }

}
```

Ahora, para terminar vamos a poner esta tarjeta en el componente de la búsqueda.

Debe quedarnos así:

```
<h3>Resultado de la Búsqueda: <small>{{ terminoDeBusqueda }}</small></h3>
<hr/>
```

```
<div *ngIf="citas.length==0" class="row">
    <div class="col">
        <div class="alert alert-secondary" role="alert">
            No se han encontrado datos con el valor {{ terminoDeBusqueda
}}
        </div>
    </div>
</div>
<div *ngIf="citas.length > 0" class="card-columns mb-5">
    <app-cita-tarjeta [cita]="cita" *ngFor="let cita of citas"></app-
cita-tarjeta>

</div>
```

Y todo lo demás tiene que funcionar y ahora si que estamos reaprovechando componentes.

Hemos visto como enviar un dato desde un componente padre a un hijo y recibirlo con @Input.

Vamos a terminar haciendo lo contrario, para que desde un componente hijo, le mandemos algo al componente padre. Esto del @Input @Output, es muy utilizado en todo lo de angular.

Ahora, la comunicación del hijo al padre será emitiendo un evento y el padre estará esperando a que el hijo.

Ahora, cuando damos el botón VerCita, la está ejecutando del hijo. Vamos comentarla, es decir en cita-tarjeta.component.ts lo dejaremos así:

```
verCita(id: number) {
    //this.router.navigate(['/cita', id]);
  }
```

Ahrora, si damos a ver Mar, no va a funcionar.

Vamos a que funcione la función VerCita del padre, como estaba antes, llamando a la función Vercita del padre.

En cita-tarjeta.component.ts, ponemos:

```
import { Component, OnInit, Input, Output, EventEmitter } from
'@angular/core';
import { Cita } from 'src/app/models/cita';
import { Router } from '@angular/router';
```

```
@Component({
  selector: 'app-cita-tarjeta',
  templateUrl: './cita-tarjeta.component.html',
  styles: []
})
export class CitaTarjetaComponent implements OnInit {

  @Input() cita: Cita  ;
  @Output() citaSeleccionada: EventEmitter<number>;

  constructor(private router: Router) {
    this.citaSeleccionada = new EventEmitter();
   }

  ngOnInit() {
  }

  verCita(id: number) {
   this.citaSeleccionada.emit( this.cita.id);
  }
}
```

Lo que hemos hecho es que declaramos un @Output con el evento que estará escuchando el padre.

En este componente, en la función VerCita, emitimos el id y en el componente ponemos lo siguiente

```
<app-cita-tarjeta (citaSeleccionada)="verCita( $event)" [cita]="cita"
*ngFor="let cita of citas"></app-cita-tarjeta>
```

Que lo que hace es que cuando se hace click en “ver mas” se dispare el evento citaSeleccionada y llama a la función del padre.

No sé si esto último lo he explicado bien.

Lo mejor es practicarlo. Únicamente está puesto para el componente de ver todas las citas, no para el de mostrar.

**Git**

Como siempre lo subimos a un repositorio

Git status

Git add .

Git commit –m “Servicios, rutas con parámetros e input output”

Git push origin master.