

ANGULAR

INDICE

Contenido

INDICE.....	1
1-Qué es Angular y sus ventajas. Cómo se instala.	5
Qué es Angular	6
Ventajas de Angular	10
Qué es mejor Angular o JQuery.....	13
Cómo instalar Angular.....	17

npm install -g	
@angular/cli.....	18
NUEVO PROYECTO.....	19
ng new	
NOMBRE_APP	20
npm install	21
ng serve --open	22
ng build	25
Cómo es la estructura de	
una app creada con	
Angular	27
El mejor IDE para programar	
en Angular	58
Conclusiones.....	60

2-Angular - Qué es un componente web y cómo crear rutas con el router- outlet.....	63
 Qué es un componente.....	63
 Creando componentes en Angular	70
 Método automático	71
 ng generate component navbar ..	72
 Método manual.....	76
 crear un controlador vacío	77

Cómo crear rutas en Angular. Sistema de routing	83
Router outlet	103
Componentes hijos	107
¿Y en el navbar cómo podemos poner links a páginas de nuestra web?.....	111
Conclusiones.....	113
¿Te ha gustado el artículo? ♥	¡Error!
Marcador no definido.	

¿Quieres aprender Angular gratis?¡Error! Marcador no definido.

1-Qué es Angular y sus ventajas. Cómo se instala.

■ Contenidos del artículo

- Qué es Angular**

- **Qué es mejor Angular o JQuery**
- **Cómo instalar Angular**
- **Cómo es la estructura de una app creada con Angular**
- **El mejor IDE para programar en Angular**
- **Conclusiones**

Qué es Angular

Angular es un framework para la creación de páginas web SPA mantenido por Google.

SPA es el acrónimo de 'Single Page Application' o lo que es lo mismo, cuando un usuario entra en una web SPA, se carga todo a la vez en una misma página y Angular lo que hace por debajo es cambiar la

**vista al navegar por la página
para que de la apariencia de
una web normal.**

¿Qué ventajas tiene?

- **Velocidad de carga lenta la primera vez que se abre la web, pero luego navegar por la web es instantáneo debido a que se carga toda la web de golpe.**
- **Cómo SPA es una página solo hay una ruta que**

**tiene que enviar el
servidor.**

- **Aplicaciones modulares y
escalares.**

Ventajas de Angular

- **Lenguaje Typescript, tiene una sintaxis muy parecida a Java, con tipado estático.**
- **Sigue el patrón MVC, con la vista separada de los controladores.**
- **Basado en componentes, es decir, podemos escribir componentes**

**web con vista y lógica
para después reutilizarlos
en otras páginas.**

- **Comunidad muy grande
con multitud de tutoriales
y librerías.**
- **Inyección de
dependencias, un patrón
de diseño que se basa en
pasar las dependencias
directamente a los**

objetos en lugar de crearlas localmente.

- **Programación reactiva, la vista se actualiza automáticamente a los cambios.**
- **Dispone de asistente por línea de comandos para crear proyectos base (Angular cli).**

- **Se integra bien con herramientas de testing.**
- **Se integra bien con Ionic, para adaptar aplicaciones web a dispositivos móviles.**

Qué es mejor Angular o Jquery

Si ya conoces Jquery seguramente quieres ver para

qué sirve Angular, por si te interesa cambiarte. La realidad es que aunque JQuery se sigue utilizando mucho, está en desuso. En los tiempos en los que vivimos, los desarrolladores frontend se están pasando a otros frameworks como Angular o React. El por qué de esto radica en que estos

frameworks ofrecen un conjunto de herramientas completas para hacer una web. Es decir, mientras que JQuery te da ciertas facilidades y ahorras código Javascript, con Angular, por ejemplo, tienes a tu disposición muchas más cosas, como por ejemplo, formas de definir rutas para la

web, reactividad en las vistas, protección de rutas, etc.

Como conclusión diría que JQuery viene bien para proyectos muy sencillos si quieres trabajar con Javascript sin más, pero a la hora de hacer una web completa te recomiendo un framework completo como Angular o ReactJS

Cómo instalar Angular

Para instalarlo tenemos que disponer de Node y NPM instalados en el equipo. Si no lo tienes instalado, puedes descargar las dos herramientas desde aquí:

<https://www.npmjs.com/get-npm>.

**Una vez instalado NPM
ejecuta la terminal y escribe:**

npm install -g @angular/cli

**Este comando instalará
Angular cli de forma global en
nuestro equipo. Angular cli es
la herramienta de consola de
Angular que nos ayudará en la
programación con Angular.**

NUEVO PROYECTO

- 1. ng new nombre_app**
- 2. cd carpeta (muévete en la terminal con el comando cd a la carpeta)**
- 3. (npm install)
(opcional??)**
- 4. ng serve --open**

Para que Angular cli cree un proyecto vacío de base para crear una aplicación con Angular, escribe:

ng new NOMBRE_APP

Cambia NOMBRE_APP por el nombre que le quieras dar a tu aplicación.

Ahora muévete en la terminal con el comando `cd` a la carpeta que se acaba de crear y ejecuta:

`npm install`

Este comando servirá para que se instalen en el proyecto las dependencias que hagan falta.

Por último para ejecutar la aplicación web que acabamos de crear simplemente:

ng serve --open

El flag `--open` sirve para que se abra automáticamente el navegador web con la web.

Por defecto Angular se ejecuta en el puerto 4200.

Si todo ha ido bien, veremos una web como ésta (si no se te ha abierto el navegador, abre <http://localhost:4200/>):

Welcome to app!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

**Angular cli también permite
compilar una web para subirla**

**a producción, para hacerlo
introduce este comando:**

ng build

**De esta forma Angular se
encargará de comprimir todos
los archivos Typescript en
archivos Javascript
entendibles por el navegador.**

Estructura de una app creada con Angular

Cuando generamos un proyecto con Angular cli nos genera la siguiente estructura (en mi caso he llamado a la aplicación tutoApp:

▲ TUTOAPP


▸  e2e


▸  node_modules

▲  src


▲  app


 app.component.css


 app.component.html

 app.component.spec.ts


 app.component.ts

 app.module.ts


▸  assets


▸  environments

★ favicon.ico


 index.html


 main.ts


 polyfills.ts


 styles.css

 test.ts

 tsconfig.app.json


 tsconfig.spec.json


 typings.d.ts


 .angular-cli.json

 .editorconfig

 .gitignore


 karma.conf.js

 package-lock.json



 package.json

 protractor.conf.js

 README.md


 tsconfig.json


 tslint.json

-  **e2e:** Esta carpeta por el momento no nos es útil, aquí se encuentra el código para escribir tests end to end que prueben la aplicación
-  **node_modules:** En esta carpeta se encuentran las librerías de angular y sus dependencias, cuando

**instalemos librerías se
añadirán aquí.**

**Generalmente no hay que
tocar nada de esta
carpeta.**

-  **src: Aquí se
encuentran los archivos
que componen nuestra
aplicación**

 **app: Aquí se donde se van
a encontrar los componentes,**

vistas, y servicios de la app.

Por el momento hay un componente llamado app con sus respectivos archivos (css, html controlador, tests, etc)

app.module.ts: En este archivo se especifica los componentes que vamos a usar en la app web. Cuando creemos un componente

tenemos que importarlo en este archivo.

favicon: El favicon de la web

- **index.html: Punto de entrada a nuestra web, este archivo se carga en todas las webs, por lo que puedes poner código para que se incluya en todas las vistas.**

- **main.ts: Algunas configuraciones de Angular, de momento no nos hace falta tocarlo.**
- **polyfills.ts: Configuraciones y código que se ejecutará antes de que se inicie la app. De momento tampoco nos hace falta tocarlo.**

- **styles.css: Estilos css globales que se aplicarán en toda las vistas de la página.**
- **test.ts: Configuración para los tests. No es útil de momento.**
- **tsconfig.app.json, tsconfig.spec.json y typings.d.ts: Lo mismo que el anterior.**

- **{ } .angular-cli.json:**

Archivo de configuración de la app.

- **.editorconfig:**

Configuraciones a la hora de desarrollar, por ejemplo, como van a ser las identaciones.

- **.gitignore: Archivo para que git ignore ciertas**

carpetas que no hace falta subir, como node_modules (cuando te bajas el proyecto ejecutas npm install para que descargue las dependencias en node_modules).

- **{ } karma.conf.js: Más configuraciones para los**

tests, esta vez los de Karma.

- **{ package-lock.json:**
Árbol de dependencias
que se crea
automáticamente
- **{ package.json: Archivo**
con las dependencias
instaladas y los
comandos que se pueden
ejecutar con npm

- **protractor.conf.js:**
Configuración para protractor, una herramienta para realizar tests en el navegador.
- **README.md Archivo readme con información de la aplicación.**
- **tsconfig.json:**
Configuración para

Typescript, el lenguaje de Angular.

- **tslint.json: Configuración del linter de TypeScript (un linter sirve para hacer comprobaciones del estilo del código que escribimos).**

package.json:

- **metadatos{nombre,versión,como se ha hecho, descripción y datos}**
- **scripts**
 - **star preconfigurada
ng serve compila
antes ejec y abre
navegador.**

- **build compila y prepara para distribuir**
- **test para ejecutar test**
- **lint comprobación estática de la calidad del código**
- **e2e test de integración**

- **dependencias:**
dependencias de
ejecución {las necesitará
el navegador.
- **Core es el núcleo**
principal.
- **common enriquece html.**
- **Forms para formularios.**
- **platform-browser para**
preparar que se pueda
ejecutar en el navegador

(también es posible ejecutar en servidor).

- **Router permite tener SPA gestionando rutas del lado del cliente sin necesidad de recargar la página.**
- **http comunicación con los servidores y APIS**
- **core gestiona zonas de la pantalla**

- **rxjs librería enorme**
reactive extensions,
programación reactiva
mediante emision de
estados...
- **zone gestiona zonas de la**
pantalla y gestiona
canvios
- **devdependencies:**
dependencias del
desarrollador {todas las

**herramientas que
necesitaremos nosotros
como desarrolladores
pero que no necesitará el
usuario: el propio cli en
forma local, compilador y
servicio para el lenguaje
(html y typescript)**

{ } .angular-cli.json: Archivo de configuración de la app. Configura al propio CLI. Guía para conocer qué se ha creado.

- src (se puede cambiar) continen el código fuente.**
- Dist: directorio de distribución . Si hacemos npm run build ejecutará a ng build el cual va a**

compilar el código y generarlo para su distribución en un servidor. Generará un html i múltiples JS, dependencias, todo... Crea una carpeta DIST que es la carpeta que hay que poner en un servidor de ejecución. La configuración de la

carpeta de origen y de destino configurará muchas otras cosas.

- **Prefix: por defecto es app. Se puede cambiar por un nombre que nos recuerde el proyecto propio..**



typings.d.ts para

incorporar librerías Js a TS.

Tiene Intelligence para

gestionar tipos, clases,etc

para poder utilizar las

librerías JS como si hubieran

sido programadas en TS.



polyfills.ts tratar que todos los navegadores entiendan una base común de JS y, en caso de no poder, hacer algo al respecto:Útil cuando se programa para navegadores antiguos o problemáticos.

index.html es el fichero índice de la aplicación que genera (aparece en el archivo **.angular.cli.json** sección de **index** ,desde donde es posible utilizar otro nombre si lo necesitáramos).

index.html contiene etiqueta nueva **<app-root>Loading...</app-root>** que inyecta los scripts

transpilados . Si abrimos el inspector del navegador veremos etiquetas script:

- **inline.bundle.js lleva código propio de WEBPACK. Es un cargador de módulos dinámico en el lado del navegador.**
- **Polyfills: se genera aunque no esté**

**descomentado el tesxto
del navegador
correspondiente:**

- **Styles: compilación
de todos los estilos CSS,
SCSS, LESS o lo que
fuera se acaban
importando aquí como
módulos de JS.**

- **Vendor: todo el código de terceros (de package.json)**
- **Main: todo el código que tu programes.**

Ahora si abres el archivo app.component.ts situado en la carpeta src/app y cambias el string de:

title = 'app';

por:

```
title = 'my wonderful app';
```

por poner un ejemplo, si ahora abres la página (si no tienes funcionando el comando ng serve, ejecútalo), verás que ahora en la página pone

Welcome to my wonderful app!

Como ves, existe una variable llamada title (Typescript tiene inferencia de tipos y no hace falta que especifiques de que tipo es la variable) que automáticamente se pinta en el html, para ello si abres el archivo app.component.html verás que en la 4ª linea hay:

Welcome to {{ title }}!

**Con {{ nombre_variable }}
puedes pintar variables
creadas en el controlador
(archivos ts) de su
correspondiente componente
(en este caso el componente
es app).**

El mejor IDE para programar en Angular



Sin duda te recomiendo que instales el vscode. Si no lo conocías, es un editor de textos muy avanzado open source mantenido por Microsoft. Es muy recomendable porque en los últimos tiempos ha mejorado mucho, es ligero y muy

adaptable a los requisitos de todo el mundo mediante extensiones. Además cuenta con una comunidad en auge, por lo que es una buena opción a futuro. Te recomiendo si lo instalas que eches un vistazo a esta extensión pensada para Angular, ofrece más soporte, snippets, iconos, etc.

Conclusiones

Resumiendo, hemos visto qué es Angular, como instalarlo, cómo crear el esqueleto de una app, y una idea aproximada de para qué sirven los archivos y carpetas que crea por defecto. Como he dicho si que te recomiendo a que aprendas Angular porque es una apuesta segura

de futuro. Con Angular vas a aprender además, Typescript, un lenguaje de programación que aporta muchas ventajas respecto a Javascript plano.

Te animo a que pruebes y cambies cosas del código para que vayas viendo como funciona Angular y Typescript. Si quieres encontrar más info

**de Angular lo puedes hacer en
su página oficial:**

<https://angular.io/>.

2-Angular - Qué es un componente web y cómo crear rutas con el router-outlet

Qué es un componente

Como vimos anteriormente, Angular se basa en componentes. Un componente se basa en la creación de

fragmentos con vista, estilos y controladores de forma que puedan ser reutilizadas en distintas partes de la web. Por ejemplo, pongamos que creamos un componente que reciba una lista de elementos y los pinte en el html. Una vez creado podemos añadir este componente de lista que hemos creado en varios sitios

de tal forma que le podemos pasar los elementos que queremos pintar para que los pinte. Esto ofrece la ventaja de tener el código modular, es decir, si tenemos que efectuar un cambio en la manera en la que visualizamos las listas, por ejemplo, solo lo tenemos que realizar una vez para todas las listas.

Un componente se puede componer de varios archivos:

- **vista,**
- **estilos,**
- **controladores,**
- **servicios,**
- **etc.**

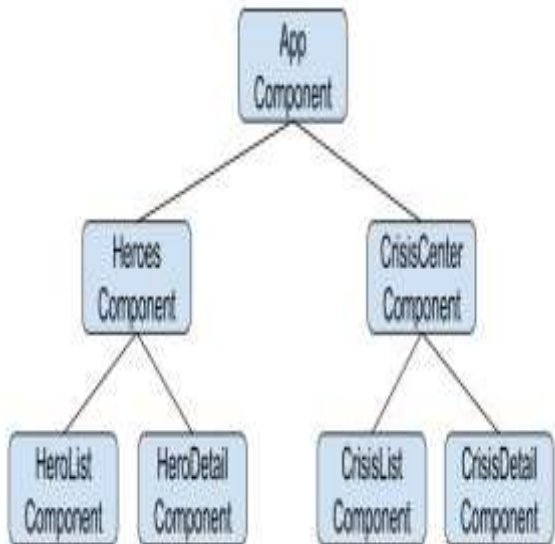
La vista (html) y los estilos (css), definen qué y cómo queremos representar la web.

En los controladores se encuentra la lógica de los componentes. Desde este archivo podemos inicializar las variables para la vista, actualizarlos, llamar a otros archivos, crear funciones, etc.

Desde los servicios es donde se hacen las llamadas para gestionar los datos, por ejemplo guardar datos, es

**decir, desde los controladores
es mejor no gestionar datos
directamente, sino que lo
mejor es delegar estas
funciones a los servicios.**

Por ejemplo una estructura de componentes con componentes padres e hijos puede ser la siguiente:



Creando componentes en Angular

Para crear los componentes en Angular existen dos maneras de hacerlo:

- I. automático**
- II. manual.**

I-Método automático de crear componentes

Si estamos usando Angular cli, en nuestro proyecto,

**existe un comando para
generar componentes:**

**Por ejemplo, imaginemos que
queremos crear un
componente para mostrar el
navbar en todas las páginas:**

**ng generate component
navbar**

**Si navegamos ahora a la
carpeta app del proyecto**

**veremos que Angular cli ha
creado por nosotros una
carpeta llamada navbar.**

**Dentro de la carpeta navbar
se ha creado un**

- **archivo css**
- **archivo html**
- **archivo .ts**
(controlador)
- **.spec.ts (tests).**

Otro detalle a tener en cuenta es que Angular ha importado por nosotros el nuevo componente en el archivo `app.module.ts`.

Método manual

Como podrás imaginar, este método consiste en crear los archivos que te hagan falta manualmente para el componente. Además, tendrás que importarlo manualmente en el archivo `app.module.ts`.

Angular recomienda separar los componentes en carpetas

según su funcionalidad, de esta forma será más fácil localizarlos y el código será más fácil de mantener.

crear un controlador vacío

Para crear un controlador vacío (archivo .ts) la estructura es la siguiente (por ejemplo para el componente de navbar):

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

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

export class

```
NavbarComponent {  
    constructor() { }  
}
```

**Importamos 'Component' de
@angular/core**

**Llamamos a @Component y le
pasamos tres cosas (de
momento):**

- **selector:** El selector es el nombre que va a tener la etiqueta html que se crea con el componente, para el navbar será `<app-navbar></app-navbar>`, es decir desde el html de cualquier otro componente, poniendo esa etiqueta se pintará el navbar. Angular tiene una convención de

nombre para el selector, kebab-case (el nombre de los selectores tiene que ser una palabra seguida de un guión y otra palabra: app-ejemplo).

- **templateUrl: La url de la vista html asociada al componente.**

- **styleUrl:** La url del estilo CSS asociado al componente.

Por último hacemos export class y el nombre de la clase.

Cómo crear rutas en Angular.

Sistema de routing

Con esto sabemos crear componentes sueltos (todavía sin funcionalidad) pero qué pasa si hemos creado un componente para una página entera, es decir, ¿cómo se crean páginas en Angular, y cómo asignarles una ruta en la página?

Para ello necesitamos hacer uso del routing de Angular, necesitamos el archivo con las rutas.

- I. crear**
app.routing.ts
- II. escribir código**
- III. importar las rutas en el archivo**
app.module.ts,

IV. en

app.component.html

quitar el html que

viene por defecto

para poner una

etiqueta especial:

<router-

outlet></router-

outlet>

I- crear app.routing.ts :

Creamos un archivo al mismo nivel que el app.module.ts y lo llamamos app.routing.ts.

II- escribir código

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

```
import {  
  AppComponent }  
from  
'./app.component';  
  
import {  
  ItemListComponent }  
from './item-list/item-  
list.component';  
  
import {  
  ItemDetailComponen  
t } from './item-
```


detail/item-

detail.component';

const appRoutes = [

{ path: "",

component:

ItemListComponent,

pathMatch: 'full'},

{ path: 'item/2',

component:

```
ItemDetailComponent,  
t, },  
];  
export const routing  
=  
RouterModule.forRoot  
t(appRoutes);
```

Explicación del Código:

***Una vez creado importamos las rutas de Angular:**

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

***creamos una variable con las rutas:**

```
const appRoutes = [  
  { path: "", component:  
    ItemListComponent,  
    pathMatch: 'full'}  
];
```

- **path:** La ruta a que queremos configurar
- **component:** Componente asociado a esa ruta. Para que funcione tienes que importar el componente en

la parte de arriba, por ejemplo:

```
import { ItemListComponent  
} from './item-list/item-  
list.component';
```

- **pathMatch: Esto es opcional, significa que toda la ruta URL tiene que coincidir.**

***Ahora, imaginemos que queremos crear una página para mostrar en detalle los items de la lista, entonces necesitamos que Angular genere una ruta para cada item, eso se puede hacer de la siguiente manera:**

{ path: 'item/:id', component: ItemDetailComponent }

:id indica que se generarán rutas con distintos id, luego dentro del controlador del detalle de item, recogeremos este valor y mostraremos el item correspondiente.

Para recoger este valor, dentro del componente de

detalle del item, tenemos que incluir en el constructor:

```
this.myId =  
activatedRoute.snapshot.params['id'];
```

this.myId es una variable que he creado dentro de el componente.

myId tendrá el valor que aparece en la ruta, es decir, si

**navegamos a la ruta /item/2,
myId tendrá valor 2.**

**Y si queremos una página 404
que aparezca cuando una ruta
no coincide con alguna de las
anteriores:**

```
{ path: '**', component:  
PageNotFoundComponent }
```

Una vez creadas todas las rutas, al final del fichero de routing introducimos:

```
export const routing =  
RouterModule.forRoot(appRoutes);
```

El fichero completo, sin la ruta a la página 404, quedaría de la siguiente manera:

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

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

```
import { ItemListComponent }  
from './item-list/item-  
list.component';
```

```
import { ItemDetailComponent  
} from './item-detail/item-  
detail.component';
```

```
const appRoutes = [  
  { path: '', component:  
ItemListComponent,  
pathMatch: 'full'},  
  { path: 'item/2', component:  
ItemDetailComponent, },  
];
```

```
export const routing =  
RouterModule.forRoot(appRou  
tes);
```

**Ahora, tenemos que importar
las rutas en el archivo**

app.module.ts,

**para ello importamos la ruta y
lo añadimos, esta vez en la
parte de imports:**

import { routing } from

'./app.routing';

...

imports: [

BrowserModule,

routing

1,

Router outlet

Si pruebas las páginas con estos cambios te darás cuenta de que todavía no se muestran las nuevas rutas, esto pasa porque en el archivo `app.component.html` que es el primer componente que se carga, tenemos que quitar el `html` que viene por

defecto para poner una etiqueta especial:

<router-outlet></router-outlet>

router-outlet es una etiqueta especial en Angular que sirve para mostrar los componentes hijos de un componente. Por defecto todos los componentes son hijos del componente AppComponent, por lo que si incluimos esta

etiqueta dentro de la vista de AppComponent, se renderizará cada uno de los componentes del routing dependiendo de la página en la que nos encontremos. Si nos encontramos en la ruta, por ejemplo, /item/2 se renderizará en el lugar de router-outlet el componente de ItemDetailComponent

**como definimos en el
routing.+**

**Para incluir un componente
dentro de otro se usa el
selector que definas dentro
del componente que quieres
renderizar, por ejemplo para el
navbar:**

<app-navbar></app-navbar>

**Como hemos dicho antes,
como todos los componentes
son hijos del AppComponent,
si incluyes el navbar en la
vista del AppComponent (
además del router-outlet),
el navbar se visualizará en
todas las páginas de la web.**

Componentes hijos

**Para que un componente
tenga componentes hijos**

asociados, lo tenemos que especificar en el routing:

```
{  
    path: 'admin', component:  
AdminComponent,  
    children: [  
        { path: '', component:  
MainComponent, },  
        { path: 'settings',  
component:  
SettingsComponent },  
    ]
```

},

En este ejemplo, la página de admin tiene dos componentes hijos. En la ruta /admin/ se cargará el componente MainComponent y en la ruta /admin/settings, el componente SettingsComponent

Los componentes hijos se dibujarán en el router outlet que coloquemos en el padre, es decir para que se renderizen estos componentes, tenemos que poner en la vista del AdminComponent un router-outlet. Todo lo que incluyamos en la vista del AdminComponent se

visualizará también en las dos páginas hijas.

¿Y en el navbar cómo podemos poner links a páginas de nuestra web?

Ya no podemos usar el atributo href, de no ser que queramos navegar a una página fuera de la web.

**Para poner un link ahora
tenemos que usar:**

```
<a routerLink="/list"  
routerLinkActive="active">Lin  
k</a>
```

**De esta manera, por ejemplo,
navegaremos a la url list, y si
la tenemos configurada en el
app.routing.ts se cargará su
vista dentro del router-outlet
visto anteriormente.**

Conclusiones

Con esto ya sabemos cómo crear componentes y como asignarlos a rutas, aunque ésto es solo una parte de todo lo que se puede hacer con rutas y componentes.

Lo que yo recomiendo es separar cada cosa en varios componentes, por ejemplo,

- **un componente para mostrar la cabecera de una página (en azul),**
- **otro para mostrar toda la lista de elementos (en verde),**

- otro componente por cada categoría de la lista (en rojo) y
- otro por cada ítem de la lista (azul oscuro).

My Shopping List

Total Number of Items: 9

Food

1. Apple
2. Bread
3. Cheese

Clothes

1. Shirt
2. Pants
3. Hat

Supplies

1. Pen
2. Paper
3. Glue

Para páginas pequeñas esta estructura es más tediosa de programar pero para páginas más grandes es la mejor estructura ya que al estar todo dividido en componentes es más fácil de mantener ya que cada cosa funciona independiente.

**Si quieres obtener más
información sobre el routing o
sobre otros aspectos de
Angular, visita la
documentación oficial de
Angular (en inglés):**

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