

Alejandro Ladislao Rodriguez Perez



Angular8 Welcome Pack for Santander guys.

TABLA DE CONTENIDO

[Introducción 3](#_Toc34833831)

[Preparando el Entorno 4](#_Toc34833832)

[Creacion de un nuevo proyecto 6](#_Toc34833833)

[Lanzar nuestra aplicación 6](#_Toc34833834)

[Creación de nuevos elementos 7](#_Toc34833835)

[Módulos: 7](#_Toc34833836)

[Componentes: 7](#_Toc34833837)

[Servicios: 8](#_Toc34833838)

[Interfaces: 8](#_Toc34833839)

[Añadiendo Angular-material 9](#_Toc34833840)

[CUERPO DE COMPONENTES 10](#_Toc34833841)

[Dando contenido a un componente 12](#_Toc34833842)

[Home-page 13](#_Toc34833843)

[Adaptación del componente 17](#_Toc34833844)

[Peticiones REST 19](#_Toc34833845)

[Interfaces de Usuario según nuestra API 19](#_Toc34833846)

[Datos visuales 22](#_Toc34833847)

[Conclusión 23](#_Toc34833848)

[Impresiones clave 23](#_Toc34833849)



# Introducción

En este “Welcome Pack” abordaremos como crear un front-end con Angular8, crearemos algunas páginas para poder tener una navegación, también crearemos una tabla de contenido rellenándola con peticiones REST, es decir, haremos un CRUD, y nos centraremos en que la Tabla tenga filtrado de contenido, paginación y ordenación por columnas.

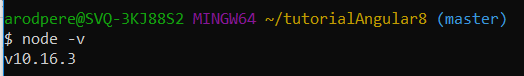
Como IDE usaremos VSC (Visual Studio Code) pero puedes elegir usar cualquiera que guste, como por ejemplo, WebStorm, Sublime Text or Atom.

Para poder tener una visión más real sin tener que crearnos o levantarnos un backend usaremos JSONPlaceholder para poder hacer peticiones REST.

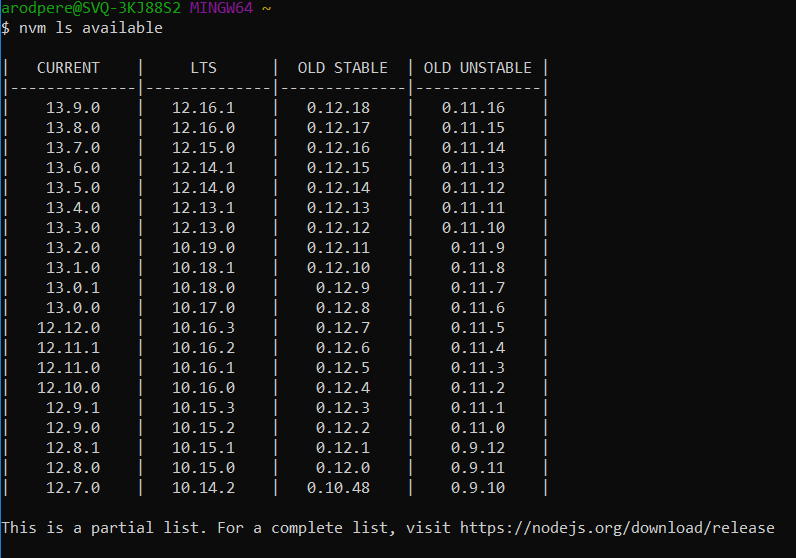
En este tutorial usaremos:

* VSC
* Git
* Node.js
* Angular-Cli
* Angular-material
* JSONplaceholder.typicode.com como API
* TypeScript
* Karma
* Jasmine

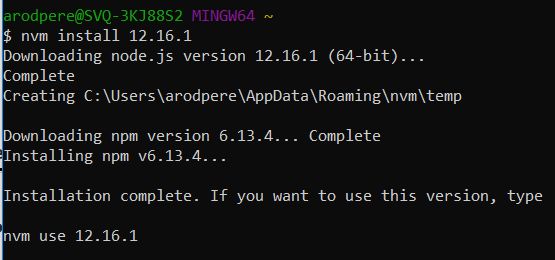
# Preparando el Entorno

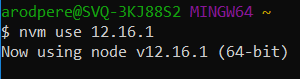
Para empezar doy por hecho que tienes instalado Git y tu respectivo IDE, en caso de que no lo tengas espero que no necesites que te lo explique este tutorial. Necesitaremos instalar los programas necesarios y sus dependencias, para ello vamos a empezar por comprobar que tenemos instalado Node.js, lo podemos comprobar de la siguiente manera:

En caso de que no lo tengamos instalado podemos descargarlo en [nodejs.org](https://nodejs.org/). También debes de tener en cuenta de que si estas usando varias versiones de Node o necesitas varias versiones de Node o necesitas ir cambiando porque tengas más de un proyecto o cualquier razón, es muy aconsejable instalarse **nvm** para poder instalar versiones diferentes e ir usando una u otra, aquí tienes un ejemplo de cómo funciona una vez instalado:

nvm ls available

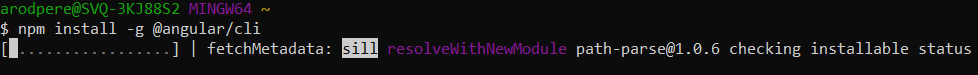
Elegimos una de todas las versiones que se nos listan para ser instalada:

nvm install 12.16.1

Usando el siguiente comando tal y como se muestra en la imagen previa, declaramos que versión de Node vamos a usar.

nvm use 12.16.1

Una vez tenemos instalado Node.js y nvm en caso de ser necesario, vamos a instalar Angular cli que nos ayudará a crear nuevos proyectos y generar parte de la aplicación entre otras cosas, para ello hacemos un:

npm install –g @angular/cli

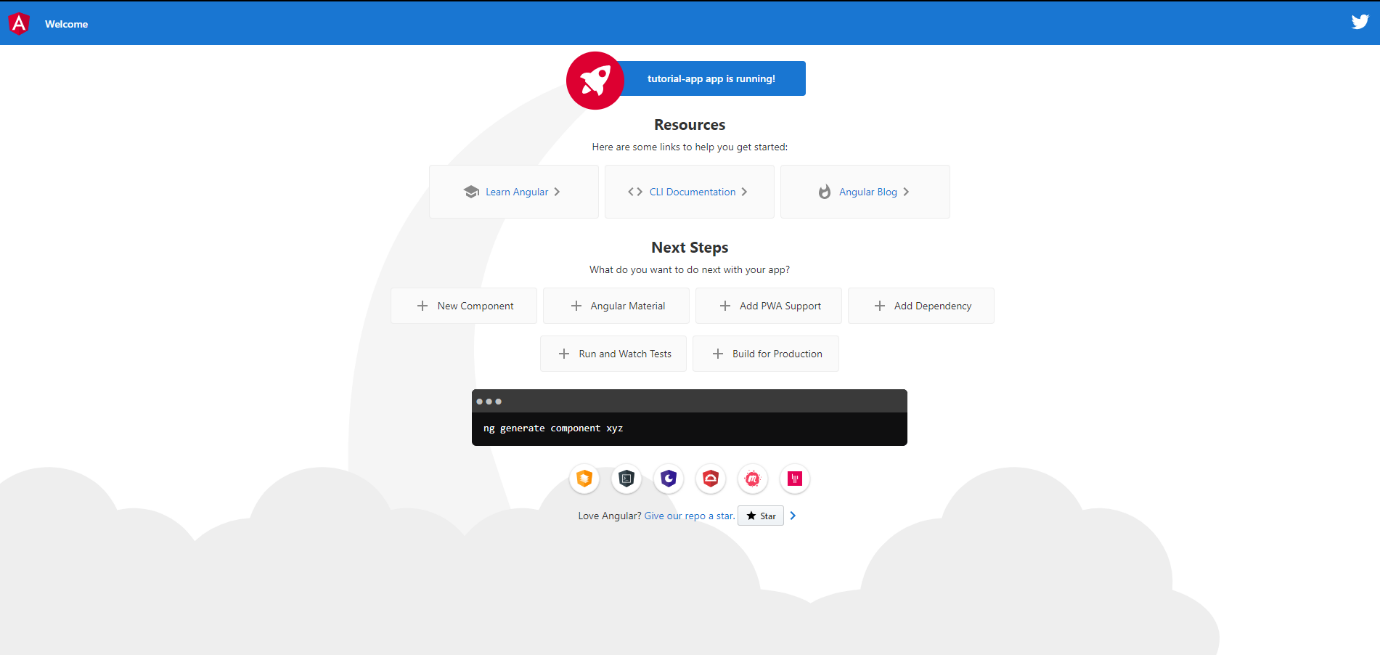
Una vez se haya instalado ya estamos listos para generar un nuevo proyecto.

# Creacion de un nuevo proyecto

## Lanzar nuestra aplicación

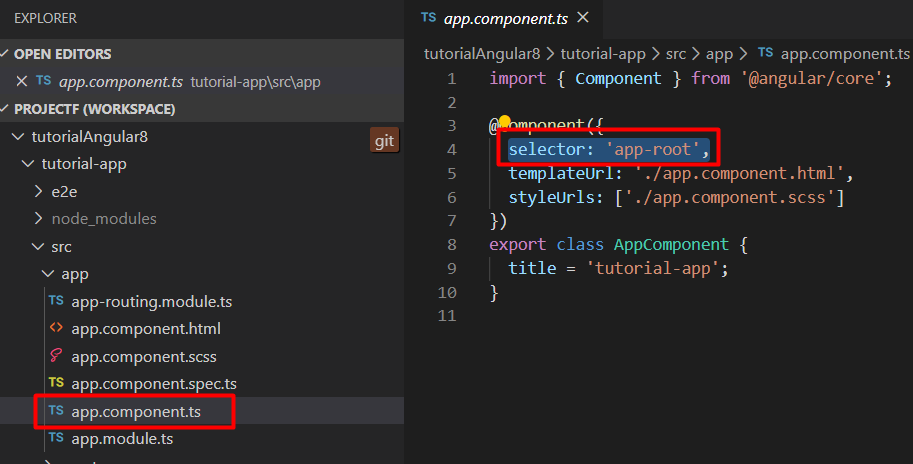
Para lanzar nuestra aplicación simplemente tendremos que movernos a la carpeta que hemos creado de nuestra nueva aplicación y lanzar:

ng serve --open

La opción --open es para que se nos abra en el navegador directamente nuestra aplicación y no tener que ir al navegador y buscar la url en donde se nos levanta nuestra app.

La primera vez que veamos nuestra aplicación veremos algo como esto que es lo que contiene actualmente nuestro **‘index.html’** y nuestro **‘app.component.html’**. Podemos darnos cuenta de esto mirando las siguientes imágenes de ambos ficheros.

En esta imagen podemos observar como el selector de nuestro actual y único componente es **‘app-root**’.

Y en nuestro index.html vamos a encontrar una etiqueta llamándolo como la siguiente:

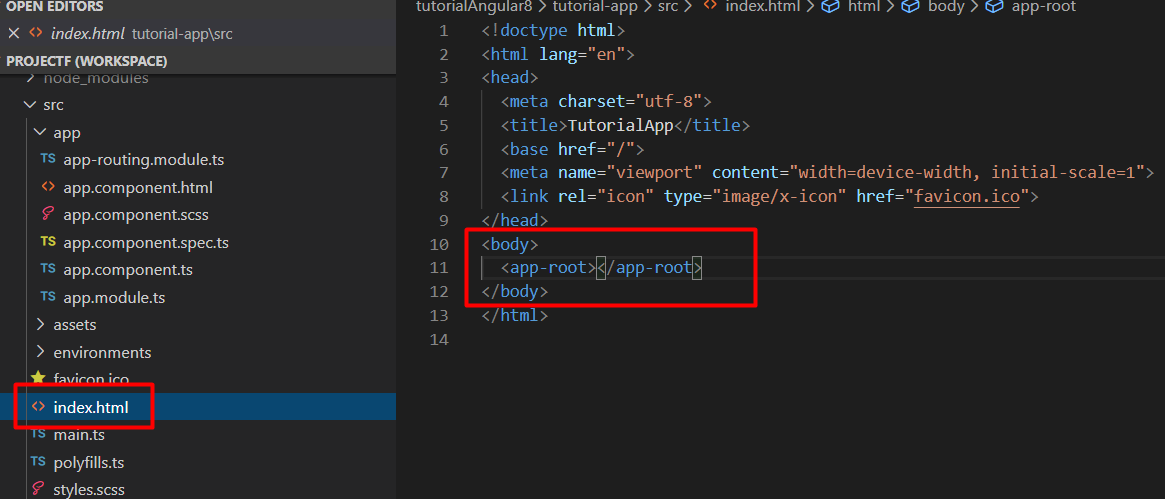
A continuación sigamos con la creación de nuevos elementos en nuestra aplicación y con el uso de **Angular-cli**.

## Creación de nuevos elementos

Módulos:

Los módulos en cuanto a código se refiere es una clase con un decorador llamado @NgModule. Su propósito es: Organizar las partes de nuestra aplicación. Ordenando la misma en bloques.

Para generar un módulo nuevo usaremos Angular-cli ya que nos generará el código necesario, no podemos olvidarnos de agregarlo al **‘app.module.ts’** en la parte de **imports**, sino no funcionará nuestra aplicación. El comando es el siguiente:

ng generate module my-module

Una vez hemos creado nuestro modulo podemos agregarle otros elementos como componentes o servicios. Así mantenemos nuestra aplicación organizada y ordenada en bloques.

Componentes:

Los componentes en cuanto a código se refiere es donde tendremos localizados nuestra estructura html, estilos, typescript con su respectiva lógica y sus tests.

Para generar un módulo nuevo usaremos Angular-cli ya que nos generará los ficheros y el código necesario aparte de importarse en nuestro respectivo modulo. Lo más recomendable es crearlo a partir de una carpeta componentes y ahí dentro meter los componentes que queramos crear. El comando es el siguiente:

ng generate component moduloCreado/components/nuevoComponente

La ruta es totalmente adaptable a la arquitectura que mantenga tu proyecto.

Recuerda que en caso de tener nuestra aplicación estructurada con módulos debemos añadir en nuestro modulo creado, **‘modulocreado.module.ts’** en la parte de **exports** nuestro nuevo componente para poder usarlo en otras partes de nuestra aplicación.

Servicios:

Los servicios básicamente son unos proveedores de datos que mantienen la lógica de acceso a ellos y la operativa relacionada con el negocio y también con los datos dentro de la aplicación. Los servicios deberán de ser consumidos por los componentes, que delegarán en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos.

Para generar un servicio nuevo usaremos Angular-cli por las mismas razones que hemos leído previamente, ficheros, código e importaciones automáticas (aunque por defecto). El comando es el siguiente:

ng generate service moduloCreado/services/nuevoServicio

La ruta es totalmente adaptable a la arquitectura que mantenga tu proyecto.

Recuerda que en caso de seguir la misma estructura deberemos de añadir a nuestro modulo creado nuestro nuevo servicio en la parte de **providers**.

Interfaces:

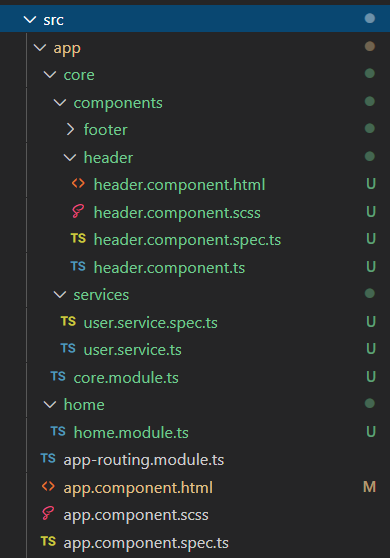
Las interfaces básicamente son una manera de saber el tipo de datos que estamos manejando, las creamos para poder tipar nuestro código TypeScript y tener más control sobre nuestra aplicación.

Para generar una interfaz de nuevo usaremos Angular-cli, aunque personalmente prefiero añadirle la extensión ‘-interface.ts’ para que sean más reconocibles. El comando es el siguiente:

ng generate interface myInterface

Una vez generada no hay mucho misterio, declararla e importarla donde vaya a usarse.

Al final de haber creado nuestros nuevos elementos tendremos una estructura como esta (Sin las interfaces porque aún no tenemos ningún tipo de dato al que poder crearle una interfaz):

Como podemos ver tenemos una estructura bastante organizada y modulada, en caso de ir creciendo nuestra aplicación, gracias a esta arquitectura de carpetas podremos mantener un orden y obtendremos una facilidad muy alta de entender toda nuestra estructura. Esto nos permitirá en un futuro no perder tiempo intentando buscar o entender que es lo que está pasando en nuestra aplicación y en nuestro código.

## Añadiendo Angular-material

Para este tutorial vamos a usar Angular-material, para añadirlo a la aplicación solo necesitamos usar el siguiente comando:

ng add @angular/material

Recuerda que debes de importar el módulo de **‘BrowserAnimationsModule’** en la aplicación para habilitar las animaciones de Angular material.

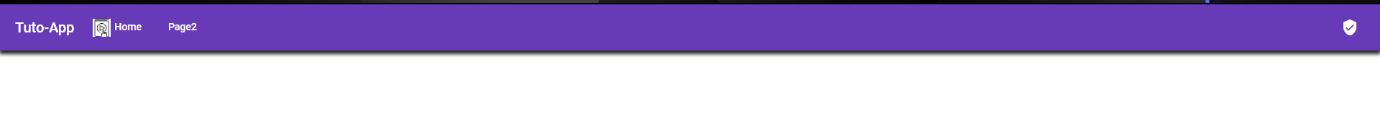
El comando que hemos lanzado te añadirá las siguientes configuraciones por defecto:

* Añadir el proyecto a dependencias en el package.json.
* Añadir la fuente ‘Roboto’ en nuestro index.html.
* Añadir la fuente de icono de material design en nuestro index.html.
* Añadirá unos pocos **estilos CSS** como:
  + Eliminar los márgenes del body.
  + Establecer una altura de 100% en nuestro html y body.
  + Y establecerá por defecto la fuente ‘Roboto’ por defecto en nuestra aplicación.

# CUERPO DE COMPONENTES

Ahora que tenemos todo preparado y listo vamos a comenzar a darle un poco de estilo y agregar algunos módulos de material para introducir algunos elementos a nuestro componente header.

Todo lo que Angular Material ofrece lo podéis ver en su página web -> [@angularMaterial](https://material.angular.io)

Este es el resultado final que queremos conseguir:

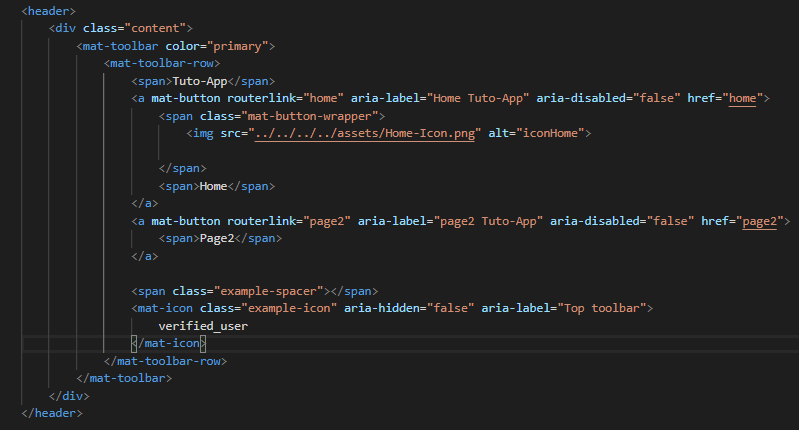
En este ejemplo hemos añadido los siguientes módulos de Angular Material:

* [MatToolbarModule](https://material.angular.io/components/toolbar/overview)
* [MatIconModule](https://material.angular.io/components/icon/overview)
* [MatButtonModule](https://material.angular.io/components/button/overview)

Para que nuestra aplicación sea consciente y conozca los nuevos módulos debemos agregarlos a nuestro respectivo **module.ts** en la parte de **imports** que es donde van los módulos. Deberiamos de quedar así:



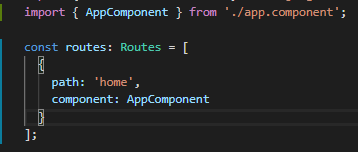
En mi caso he decidido darle algunos estilos y añadir algunas nuevas rutas de navegación aunque actualmente no tenga componente al que navegar, así que como prueba navegaremos a nuestro **app-component**, que es nuestro componente padre y podremos ver como se nos duplica nuestro header ya que es el componente que siempre queremos mostrar.

Para obtener un header similar debemos de tener un código como este (no olvidéis que también hay estilos aplicados, así que añadirlo a tu gusto):

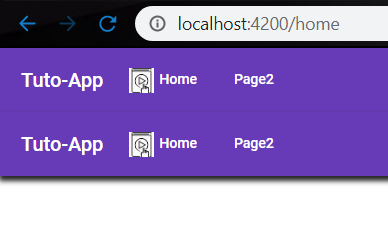
Como podemos observar tenemos las respectivas etiquetas y directivas que nos ofrecen nuestros nuevos módulos importados, **‘mat-toolbar’**, **’mat-icon’**, **’mat-button’**. También estamos haciendo uso del **routing** de Angular para que al hacer clic en los enlaces naveguemos al componente que hemos configurado en nuestro routing.

## Routing

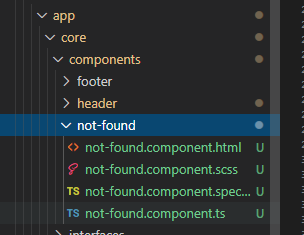
Debemos ir a nuestro fichero **‘app-routing.module.ts’** y ahí debemos a añadir nuestro **path** con el nombre que queremos darle y el **componente** que queremos cargar.

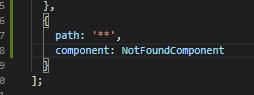


Al hacer esto lo que conseguimos es que cuando presionemos el botón el cual tenemos configurado con el **routerLink** navegue al componente que hemos puesto, deberíamos ver algo como esto:

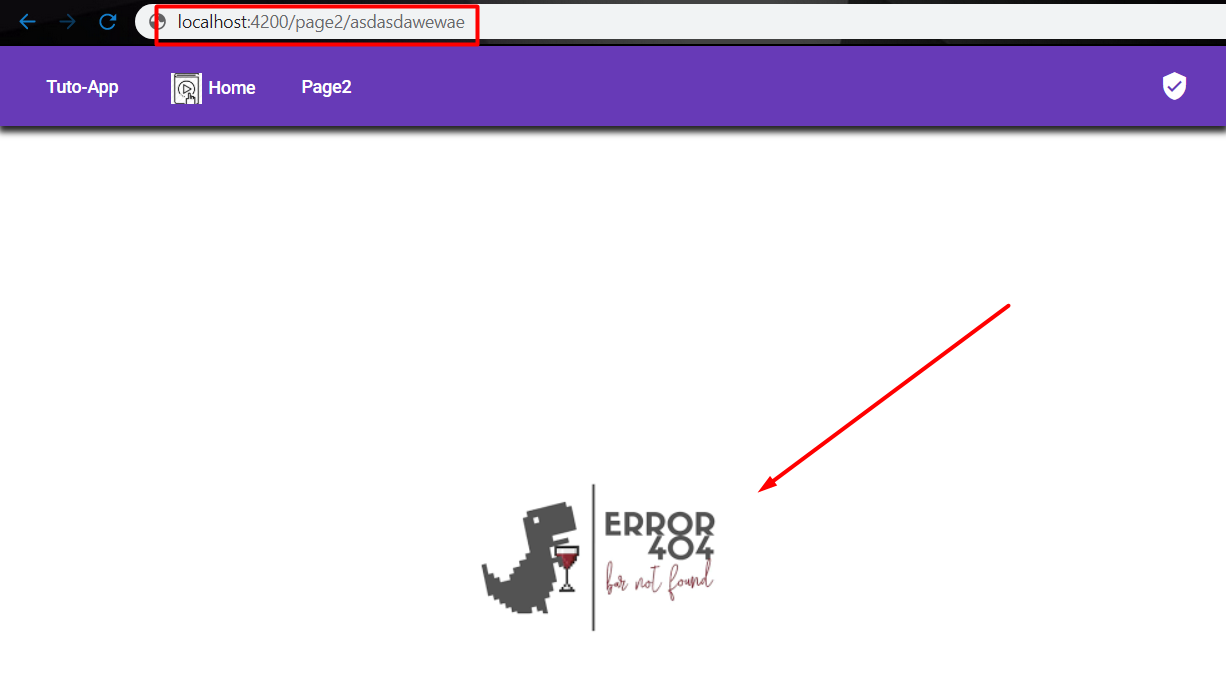
Al hacer clic en nuestro Home, vemos como nuestra aplicación web navega a **‘/home’** y nos muestra nuestro componente header dos veces porque nuestro **app-component** es lo único que contiene en este momento, si queremos que navegue a otro componente primero debemos de crearlo.

NotFoundComponent

Dado que no tenemos contenido para todas las rutas, y en caso de que el usuario intente navegar a una ruta que no existe en nuestra aplicación vamos a crear un componente para controlar el error 404 de no encontrar una ruta en nuestra aplicación.

Se tratará de un componente **core**, así que lo crearemos en nuestro módulo de core usando angular cli.

Luego para suceda lo que hemos estado hablando simplemente debemos ir a nuestro **app.routing.module.ts** y añadir que cualquier ruta que no exista previamente en nuestra web cargue nuestro componente **NotFoundComponent**.

Añadiendo cualquier imagen o diseño que queramos darle cuando naveguemos a cualquier ruta que no existe en nuestro routing se nos cargará ese componente tal y como vemos.

## Dando contenido a un componente

Como ya teníamos creada nuestra estructura previamente vamos a usar esa estructura para rellenar y darle contenido a nuestra web. Por ahora vamos a crear dos nuevos componentes en el módulo de home, **welcome-page** y **home-page**:

 **Welcome-page** será nuestra página principal, la que se encargará de dadle la bienvenida a nuestros usuarios de la manera más linda que podamos o que se nos ocurra, por ahora yo solo añadiré una imagen de bienvenida.

**Home-page** será donde navegaremos con nuestro header y donde se encontrará nuestra tabla de contenidos obteniendo toda la lista de usuarios.

Home-page

Para añadir una tabla, poder ordenarla y tener un paginado para dicha tabla usaremos los siguientes componentes de Angular material:

* [MatTableModule](https://material.angular.io/components/table/overview)
* [MatSortModule](https://material.angular.io/components/sort/overview)
* [MatPaginatorModule](https://material.angular.io/components/paginator/overview)

Como hemos hecho anteriormente vamos a añadirlo a nuestro respectivo módulo, [home.module.ts](http://home.module.ts) para seguir manteniendo nuestra modularización, todos estos módulos.

Añadamos la tabla de ejemplo para luego adaptarla a nuestro requerimiento, para ello vamos a usar una tabla que podemos encontrar en Angular material, solo necesitamos usar ese código en nuestra aplicación e importar los módulos dependientes de ese código, una vez lo tengamos agregado pasaremos a adaptarlo a nuestra aplicación, pero por ahora veamos que nos quede de la siguiente forma (recuerda los estilos):

1. **home-page.component.html:**

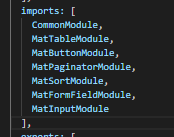
****Vamos a explicar lo que estamos añadiendo, para empezar podemos ver que tenemos:

* **“mat-form-field”:** Este se trata del módulo **MatFormFieldModule**, y dentro contiene un **MatInputModule**, por lo tanto es una manera de agrupar el contenido del mismo form que estamos creando, el cual incluye un input para poder realizar una búsqueda usando la función de **applyFilter($event)** que luego veremos a fondo.
* **“mat-table”:** Se trata de la tabla que estamos pintando al declararle la directiva de **‘mat-table’** lo que hacemos es usar nuestro módulo de Angular material **MatTableModule** el cual trabaja con dataSource para pintar los datos en la tabla.
* **“mat-paginator”:** Esta etiqueta se trata del módulo **MatSortModule** el cual es el encargado de paginar toda la data que imprimimos luego en nuestra tabla.

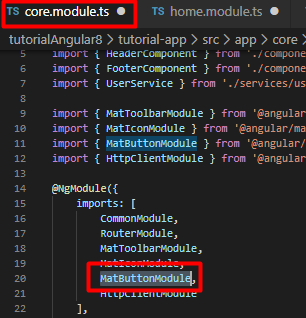
Para más información acerca de los distintos **selectores** que usamos en la estructura que nos facilita la documentación, podemos consultarla en su [API](https://material.angular.io/components/table/api).

Al pillar el **HTML** del ejemplo podemos darnos cuenta de que hay varias etiquetas y directivas que no tenemos importadas, que pertenecen a nuevos módulos por lo tanto nuestra aplicación va a empezar a lanzar errores al introducir elementos desconocidos, así que vamos a importar los nuevos módulos en **home.module.ts**.

1. **home.module.ts:**

Lo más sencillo para ir añadiendo módulos que aún no conoces y no estás familiarizado con su uso en Angular es agregar sus etiquetas e ir resolviendo los errores que te lanza la aplicación realizando una búsqueda, o realizando una búsqueda previa de como agregar ese modulo.

Al final deberíamos de tener nuestro modulo como la imagen que tenemos al lado:

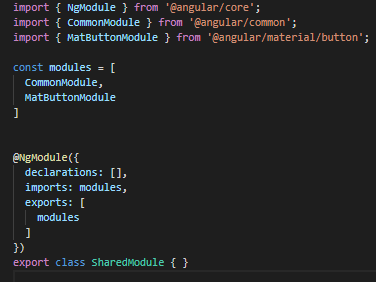
Sin embargo, al verlo podemos darnos cuenta de que en nuestro **core.module.ts** también tenemos importado el **MatButtonModule** ya que lo hemos necesitado previamente allí.

Esto debe de llamarnos la atención ya que es la clave para darnos cuenta de que empezamos a necesitar un módulo shared el cual contenga todos los modulos que vamos a compartir.

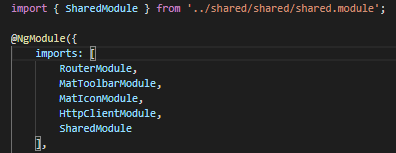
Para resolverlo lo que tenemos que hacer es crear un nuevo módulo shared, en una carpeta shared, y agregar ahí los módulos que vamos a compartir, una vez lo tengamos solo necesitamos importar ese módulo shared en los módulos que vayamos a usar.

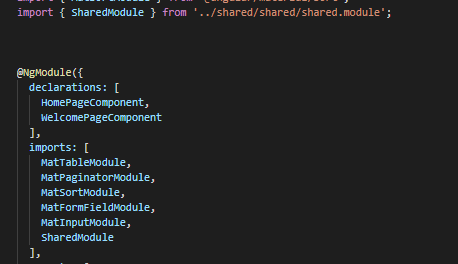
Al final deberíamos de tener algo así:

shared.module.ts:





core.module.ts:

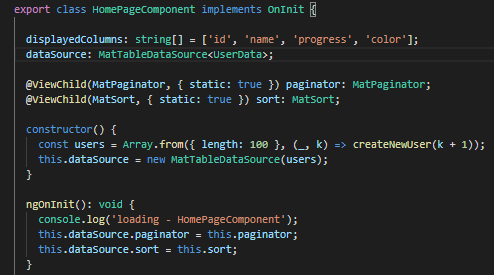


home.module.ts:

1. **home-page.component.ts:**

Lo primero que podemos ver en el ejemplo que nos ofrece Angular-material es que se crea una interfaz y unos datos mockeados para el ejemplo, esta no es la manera recomendable asi que luego adaptaremos este código a nuestro caso.

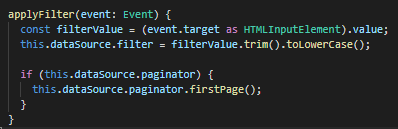
Luego podemos observar como declara un par de variables que usamos previamente en el html, **displayedColumns** y **dataSource**, dataSource es la variable mas importante ya que es la q va a almacenar todos nuestros datos y es el objeto con el cual vamos a trabajar a la hora de usar la **paginación**, **ordenación** y **filtrado** de datos.

Como vemos en la imagen esta declarando dos **ViewChild**, uno para nuestro paginator y otro para la ordenación, vamos a lo importante, que es un ViewChild, Angular trata nuestro HTML como un **template** y cada elemento, cada div, nav, etc lo trata como un **view**, para acceder a ellos antiguamente usabamos un id y este id lo llamamos en nuestro js(ahora ts) haciendo un getElementById, pero ahora al tratarse de Angular el que interpreta como views podemos acceder a ellos usando el ViewChild, necesitamos acceder a ellos para luego mas adelante añadirlos a nuestro dataSource para asi poder trabajar con los datos para su paginación y ordenación.

En el **constructor** lo que hacemos es generar un **Array** de usuarios con la función que nos ha dado la el ejemplo y luego instanaciamos como **MatTableDataSource** nuestro dataSource añadiendole los usuarios que hemos generado.

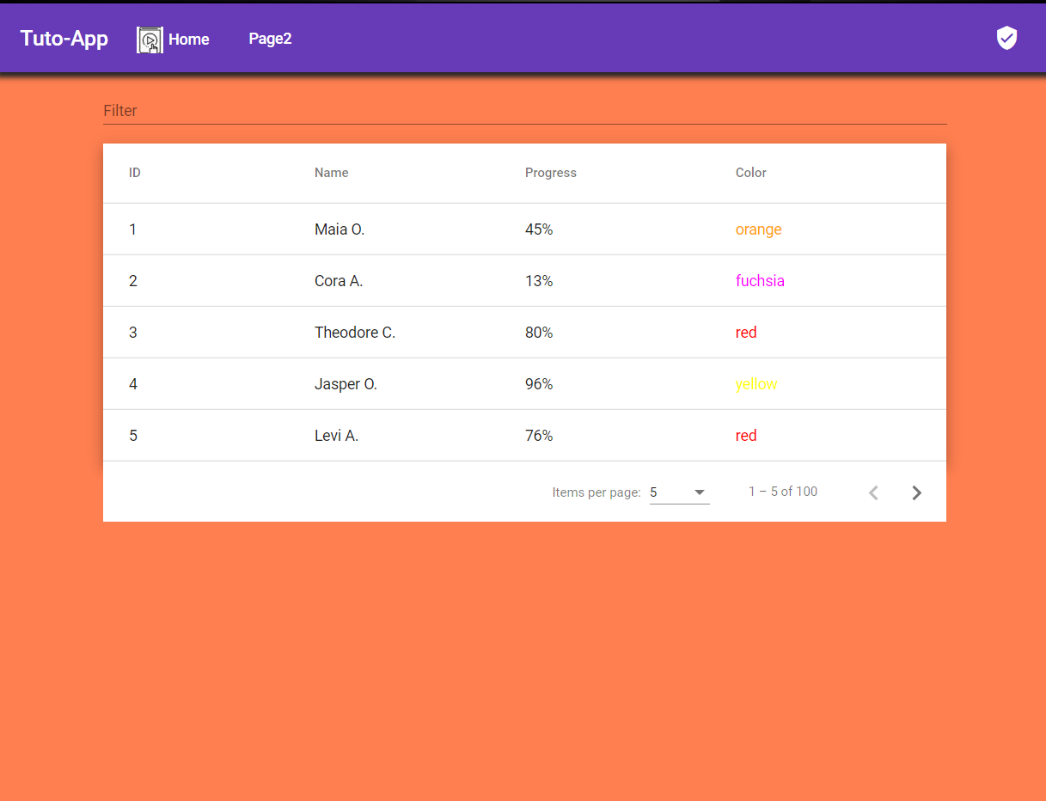
Luego en el **OnInit** lo único que hacemos es inicializar los valores de paginación y ordenación de nuestro dataSource.

Por ultimo vamos a explicar la funcion **applyFilter** el cual recibe el evento que le hemos pasado desde nuestro HTML. Lo primero que hace es recoger la propiedad target de event para realizarle un casting y pillar el value de este mismo, acto seguido elimina los espacios del inicio y el final y lo setea todo a minúsculas para añadirselo a nuestro **dataSource.filter** que es el que se encargará de realizar el trabajo de la busqueda en nuestro contenido.routing

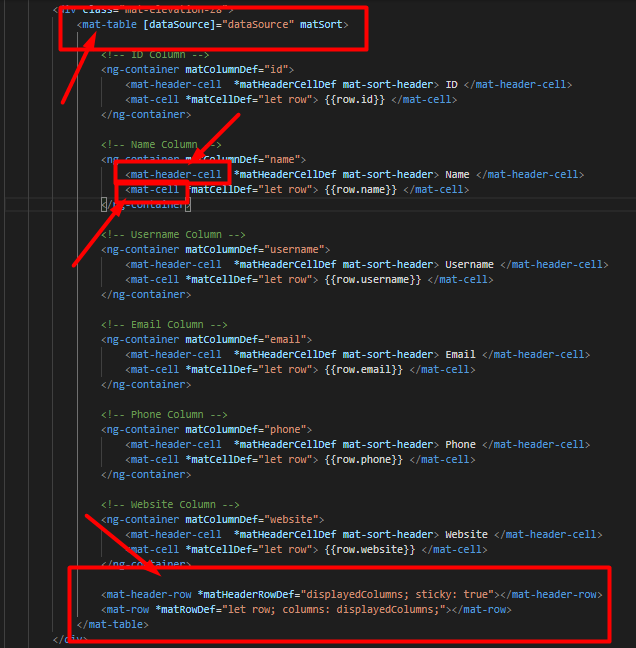
El if que tenemos simplemente lo que hace es setear nuestro paginador a la primera página en caso de que no lo este.

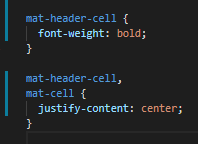
Adaptación del componente

Una vez ya sabemos y comprendemos que es lo que nos ofrece el ejemplo de Angular Material vamos a adaptarlos a nuestra aplicación y a nuestras necesidades.

Actualmente después de todo el proceso que hemos hecho, cuando naveguemos a nuestra pagina Home tendremos una vista algo similar a esta, dado que supongo al igual que yo, tu le has dado los estilos que más te hayan gustado.

Más adelante en las peticiones REST veremos como actualizamos nuestro código para esperar la llamada de datos que realizamos desde el controlador, asi que lo que vamos a preparar es nuestra Tabla para cuando realicemos esa petición de datos nos pinte los datos bien. Para ello debemos de observar en nuestra API los datos que nos van a traer y que campos queremos pintar en nuestra Tabla. En nuestro caso por ahora pintaremos, el **ID**, **name**, **username**, **email**, **phone** y **website**. Los campos address y company los dejaremos para mas adelante.

Para ello vamos a volver a nuestro HTML y vamos a adaptarlo para esas columnas, adaptarlo a esas columnas no hay mayor misterio que añadirlas, pero en caso de que queramos centrar nuestro header en nuestra tabla vamos a tener que hacer un cambio más especifico en nuestro HTML.

Como vemos tendremos que cambiar nuestro table por un mat-table como etiqueta y no como atributo, lo mismo sucederá con nuestros th y nuestro td, tendremos que poner los atributos mat-x-x como etiquetas para poder manejarlas desde nuestra hoja de estilos y poder manejarla a nuestro antojo.

Una vez tengamos las columnas añadidas y el HTML actualizado no olvideis actualizar la propiedad de **displayedColumns** de la cual nos estamos alimentado de nuestro **home-page.component.ts**. Con esto ya tenemos el componente adaptado y preparado para pintar la data que nos devolvera la llamada a nuestra API.

# Peticiones REST

## Interfaces de Usuario según nuestra API

Ya que vamos a realizar peticiones **REST** deberíamos de tener un control tipado sobre la estructura de datos que nos va a ir devolviendo nuestra API, para ello vamos a comprobar que nuestra API respecto a **USER** tiene la estructura que estamos viendo a nuestro lado:

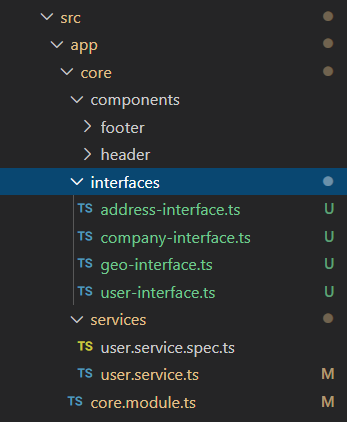
Para ello vamos a crear una carpeta interfaces en nuestro actual **core** y dentro crearemos las interfaces necesarias.

Aquí podemos ver que tenemos una interfaz padre que sería la **User**, y luego podemos observar que contiene dos objetos más, esto significa que debemos de hacer una interfaz por cada objeto para tener un control tipado de cada uno de ellos, esos dos objetos son el **address** que contiene a su vez otro objeto hijo llamado **geo** y **company**.

Para crear la interfaz de **User** podríamos pensar que nos basta con duplicar lo que estamos viendo en el objeto, pero no es tan simple, para ello debemos de seguir cierto criterio.

Como vemos si replicamos lo que estamos viendo nos damos cuenta de que a pesar de poner address y company como tipado nos da error por el simple hecho de que aun no la hemos creado, entonces lo que debemos de hacer es crear nuestras interfaces **Address** y **Company** y importarlas en nuestra interface **User**.

Vamos a ver que cuando creemos nuestra interfaz **Address** tendremos otro objeto **geo** que no tenemos creada la interfaz, la creamos igualmente y la importamos en la misma interfaz de **Address**.

Al final lo que tendremos será una estructura de interfaces que podremos usar más adelante.

Dado que hemos decidido usar [JSONplaceholder](https://jsonplaceholder.typicode.com/) como **API** vamos a preparar un **CRUD** de peticiones **REST** a dicha API, para ello debemos de tener en cuenta de que esto siempre debe ir en un servicio, ya que son los encargados del manejo de datos.

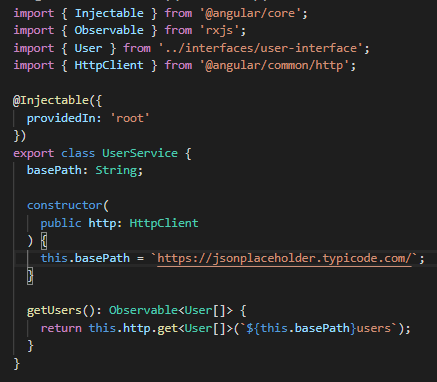
## Desde Servicio

Por lo tanto las peticiones **REST** se hacen con el módulo **HttpClient**, él se encarga de facilitarnos las llamadas y lo que devuelve es un **Observable** al cual podemos subscribirnos para recoger los datos en caso de que la petición salga exitosa o lance un error y poder controlarlo.

Para ello debemos importarlo en nuestro module.ts correspondiente en la sección de **imports** ya que se trata de un módulo (**HttpClientModule**), una vez tenemos esto lo que tenemos que hacer es irnos al servicio e iniciar nuestro **HttpClient.**

Peticiones GET

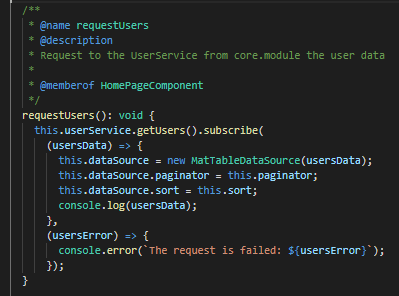
Vamos a realizar por ejemplo y **GET** de todos los Usuarios que nos ofrece la API, para esto sabemos que nos devolverá un **Array** de Usuarios.

Aquí podemos ver cómo hemos declarado también una variable para guardar la url base que a la que vamos a ir haciendo peticiones, como sabemos devuelve un **Observable** de **Array de Users** por lo tanto le hemos creado una interfaz para ello previamente.

Peticiones POST

Peticiones DELETE

## Desde Controlador

Para acceder a nuestro servicio donde tenemos la petición REST devolviendo un Observable desde un controlador solo tenemos que instanciar el servicio en nuestro controlador y luego subscribirnos a la función que hace la petición. Cuando realizamos esta subscripción debemos de controlar tanto la respuesta positiva que nos devuelve los datos como el error.

En este caso debemos de actualizar el código para ‘setear’ nuestro dataSource, paginator y ordenación cuando nos devuelva los datos la petición. Recuerda llamar a la función en tu **OnInit**.

# Datos visuales

Insertar listas/gráficos/datos

# Vista desde arriba de un apretón de manos sobre papeles empresarialesConclusión

Tiempo de hacer una conclusión. ¿Cuál es su conclusión? ¿Cómo sintetizaría toda la información en algo que aún el CEO más ocupado quiera leer? ¿Cuáles son los aportes clave? ¿Cómo aborda su producto/servicio/metodología de manera única los problemas que surgen del presente estudio?

## Impresiones clave

* Impresión núm. 1
* Impresión núm. 2
* Impresión núm. 3