

Alejandro Ladislao Rodriguez Perez



Angular8 Welcome Pack for Santander guys.

TABLA DE CONTENIDO

[Introducción 3](#_Toc34731936)

[Preparando el Entorno 4](#_Toc34731937)

[Creacion de un nuevo proyecto 6](#_Toc34731938)

[Lanzar nuestra aplicación 7](#_Toc34731939)

[Creación de nuevos elementos 8](#_Toc34731940)

[Módulos: 8](#_Toc34731941)

[Componentes: 8](#_Toc34731942)

[Servicios: 9](#_Toc34731943)

[Interfaces: 9](#_Toc34731944)

[Añadiendo Angular-material 10](#_Toc34731945)

[CUERPO DE COMPONENTES 11](#_Toc34731946)

[Dando contenido a un componente 13](#_Toc34731947)

[Home-page 14](#_Toc34731948)

[Peticiones REST 14](#_Toc34731949)

[Interfaces de Usuario según nuestra API 14](#_Toc34731950)

[Datos visuales 19](#_Toc34731951)

[Conclusión 20](#_Toc34731952)

[Impresiones clave 20](#_Toc34731953)



# 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.

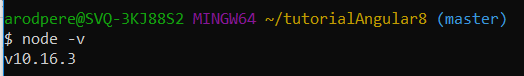
Como IDE usaremos VSC (Visual Studio Code) pero puedes elegir usar cualquiera que guste, como por ejemplo, WebStorm, Sublime Text or Atom.

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

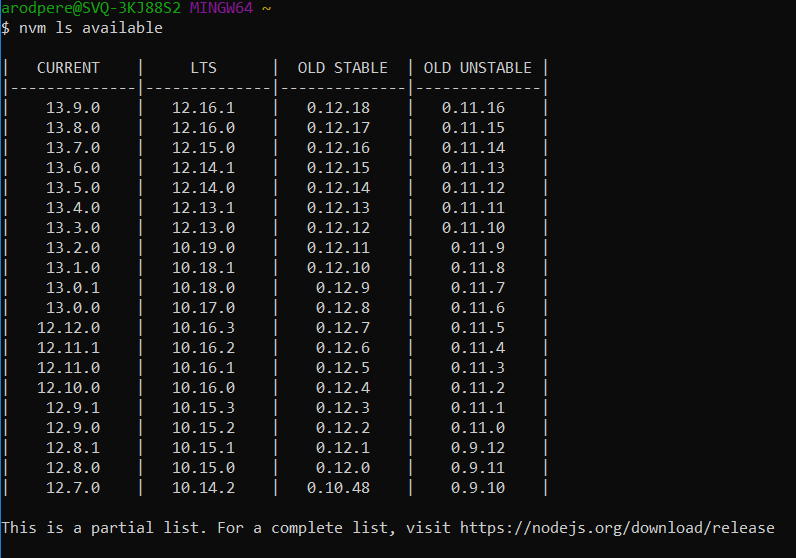
En este tutorial usaremos:

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

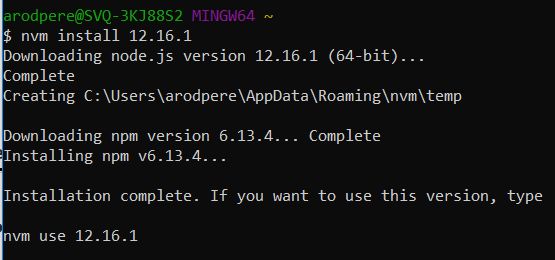
# Preparando el Entorno

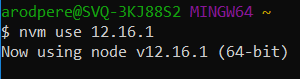
Para empezar doy por hecho que tienes instalado Git y tu respectivo IDE, en caso de que no lo tengas espero que no necesites que te lo explique este tutorial. Necesitaremos instalar los programas necesarios y sus dependencias, para ello vamos a empezar por comprobar que tenemos instalado Node.js, lo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

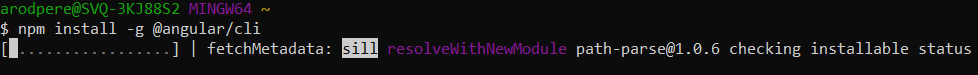
Elegimos una de todas las versiones que se nos listan para ser instalada:

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

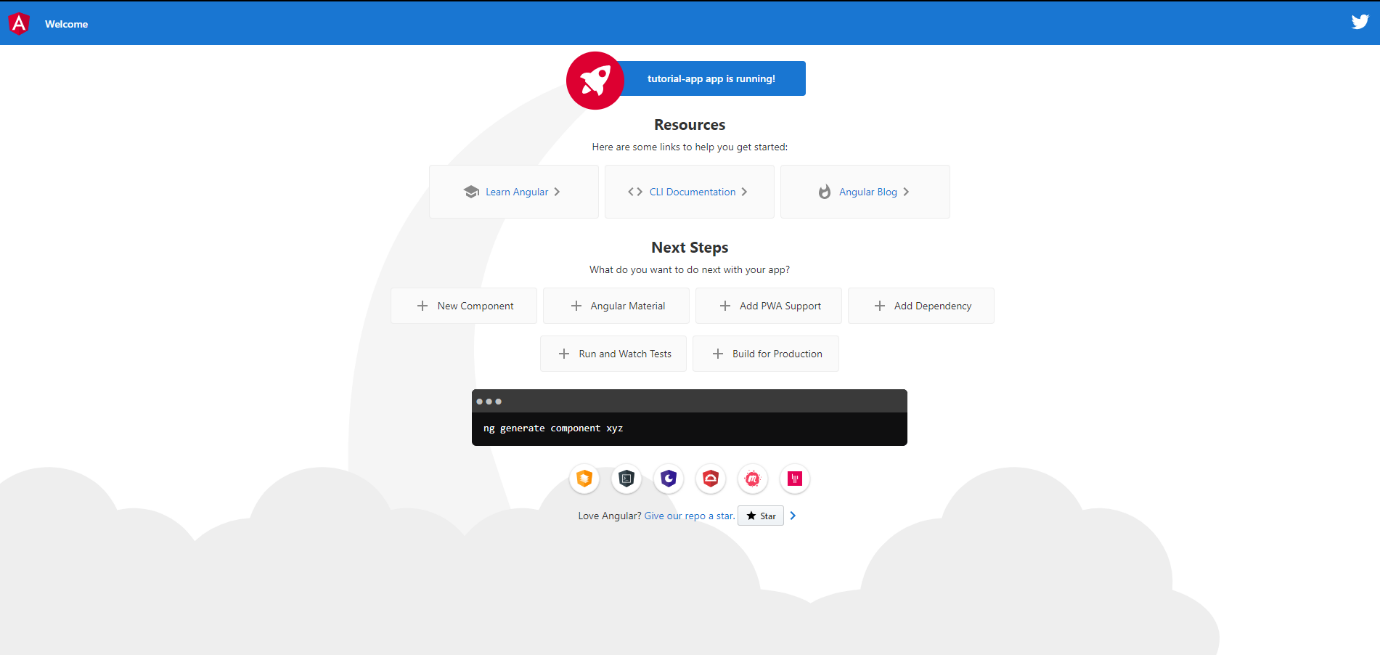
Una vez se haya instalado ya estamos listos para generar un nuevo proyecto.

# Creacion de un nuevo proyecto

## Lanzar nuestra aplicación

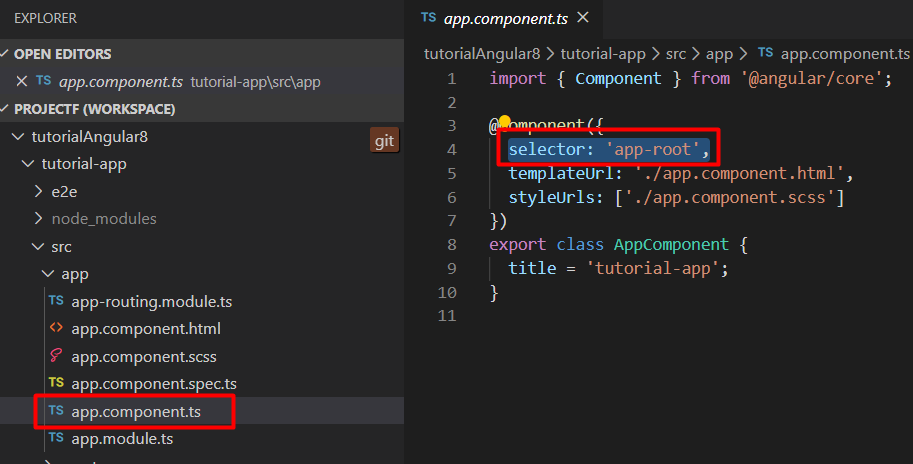
Para lanzar nuestra aplicación simplemente tendremos que movernos a la carpeta que hemos creado de nuestra nueva aplicación y lanzar:

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:

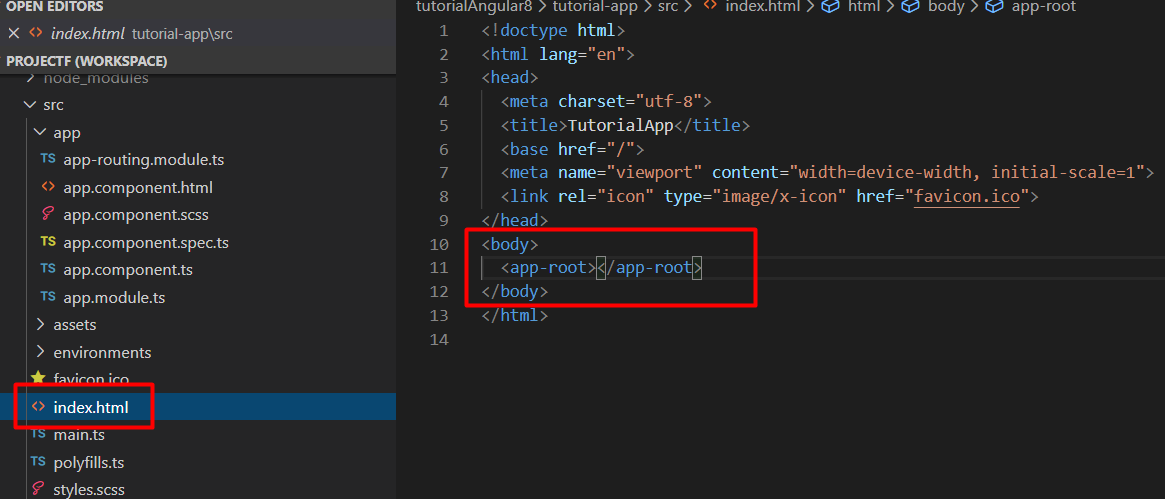
A continuación sigamos con la creación de nuevos elementos en nuestra aplicación y con el uso de **Angular-cli**.

## 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:

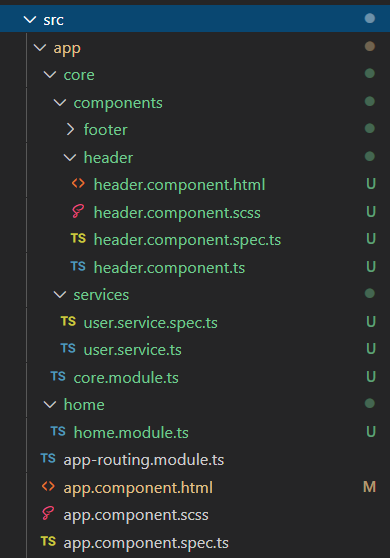
Las interfaces básicamente son una manera de saber el tipo de datos que estamos manejando, las creamos para poder tipar nuestro código TypeScript y tener más control sobre nuestra aplicación.

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.

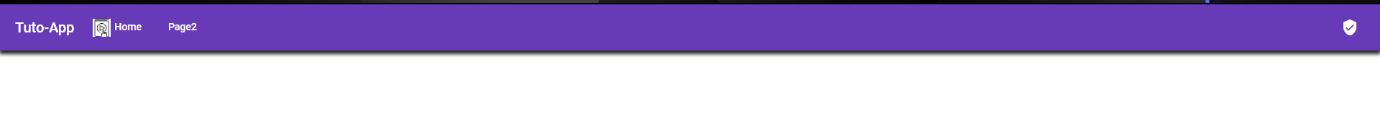
El comando que hemos lanzado te añadirá las siguientes configuraciones por defecto:

* 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:

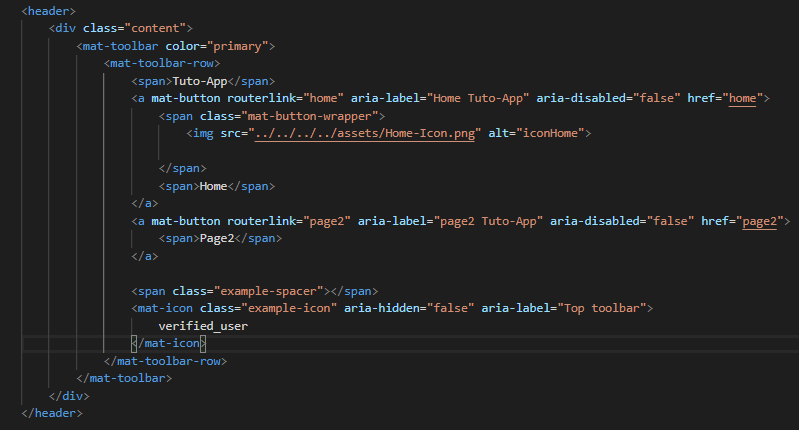
En este ejemplo hemos añadido los siguientes módulos de Angular Material:

* [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:

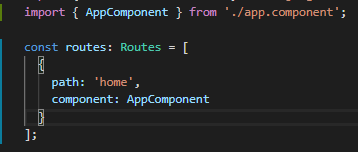


En mi caso he decidido darle algunos estilos y añadir algunas nuevas rutas de navegación aunque actualmente no tenga componente al que navegar, así que como prueba navegaremos a nuestro **app-component**, que es nuestro componente padre y podremos ver como se nos duplica nuestro header ya que es el componente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

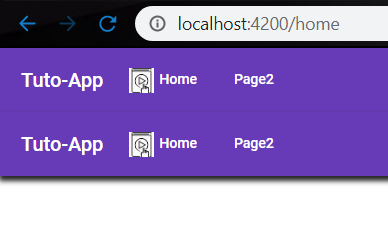
Para obtener un header similar debemos de tener un código como este (no olvidéis que también hay estilos aplicados, así que añadirlo a tu gusto):

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. Esto lo podemos hacer de la siguiente forma:

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.

## 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):

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

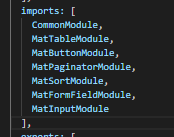
****Vamos a explicar lo que estamos añadiendo, para empezar podemos ver que tenemos:

* **“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.

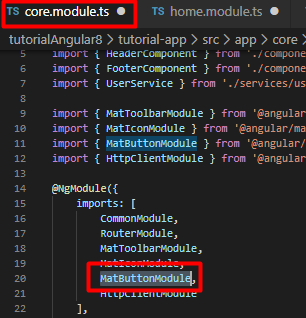
Para más información acerca de los distintos **selectores** que usamos en la estructura que nos facilita la documentación, podemos consultarla en su [API](https://material.angular.io/components/table/api).

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**.

**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 estamos importando dos veces el mismo módulo cuando no debería de ser necesario.

Para resolverlo lo que tenemos que hacer es irnos a nuestro modulo general, **app.module.ts**, añadirlo en este archivo y eliminarlo de ambos ficheros: **core.module.ts** y **home.module.ts**.

# 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:

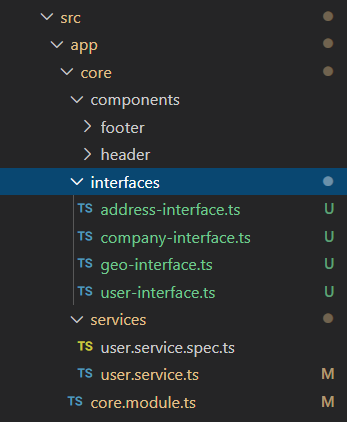
Para ello vamos a crear una carpeta interfaces en nuestro actual **core** y dentro crearemos las interfaces necesarias.

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.

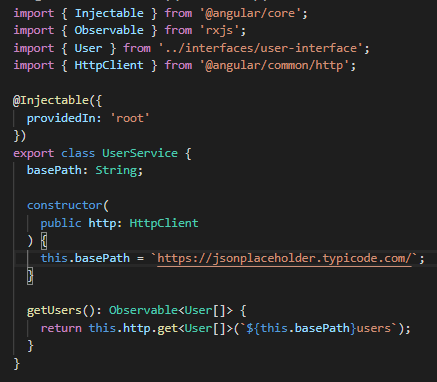
Como vemos si replicamos lo que estamos viendo nos damos cuenta de que a pesar de poner address y company como tipado nos da error por el simple hecho de que aun no la hemos creado, entonces lo que debemos de hacer es crear nuestras interfaces **Address** y **Company** y importarlas en nuestra interface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

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. 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, entonces nuestro código quedaría así:

Aquí podemos ver cómo hemos declaro 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.

# 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