Angular Resumen

Angular es un framework para front-end de Javascrip, que nos permite crear aplicaciones web conocidas como Single Pages Aplications. Esta escrito en TypeScript.

Lo que vamos a necesitar para desarrollar una aplicación con Angular es un modulo que se llama **Angular/CLI** (es una interfaz de línea de comandos para proyectos Angular) que agregamos de manera global al Visual Studio Code mediante el comando

**npm install -g @angular/cli**

De esta manera estamos incorporando a Node.JS una herramienta (Angular/CLI) que nos permitirá generar proyectos en Angular, a través de la línea de comando. Con el comando

**ng new myfirstapp**

crearía mi primera aplicación. Lo que hace es crearme una carpeta con el nombre que le dia a la aplicación u dentro de ella me crea el esqueleto de la misma.

Dentro de una consola de Visual Studio Code y ubicado dentro de la carpeta del nuevo proyecto, ejecuto el comando

**ng serve**

me crea un servidor y ei accedo en un browser a la URL <http://localhost:4200/> puedo ver una pagina inicial del proyecto.

Lo que se crea con el proyecto es una estructura de directorios y archivos iniciales y un componente principal (el componente app) un componente esta definido dentro de una carpeta y esta compuesto por archivos del tipo TypeScript, html y css.

De allí en más, puedo a armar la pagina creando mis componentes propios que se invocaran desde el componente principal ‘app’. Para crear un componente propio utilizo el comando Angular

**ng generate component mi-componente**

generalmente el componente que cree, tendrá tres archivos de los tipos indicados anteriormente y allí debere agregar código. En el .html escribiré la etiquetas que conformaran el componente en la página. En el css, ira el estilo de para lo contenido en el html. En el .ts el código typeScrip con el comportamiento dinámico del componente. En éste ultimo también existirá el ‘selector’, que es el nombre con el cual debiera de utilizarlo en la página principal como etiqueta html. Es decir, en el archivo .html de la página principal lo llamare en formato de etiqueta

**<mi-componente></mi-componente>**

Mis componentes se ‘colgaran’ del componente principal (app) o desde otros componentes propios y se le podrán pasar paramentros desde estos como atributos en la etiqueta del .html del contenedor.

Los parámetros se pasan como un atributo en la etiqueta del componente (contenido) desde el html del contenedor. Se utilizan [] para encerrar al nombre del atributo y se le asigna el valor.

Supongamos que en la clase definida para el componente (contenedor), en el archivo .ts correpondiente al contenedor se declara una propiedad

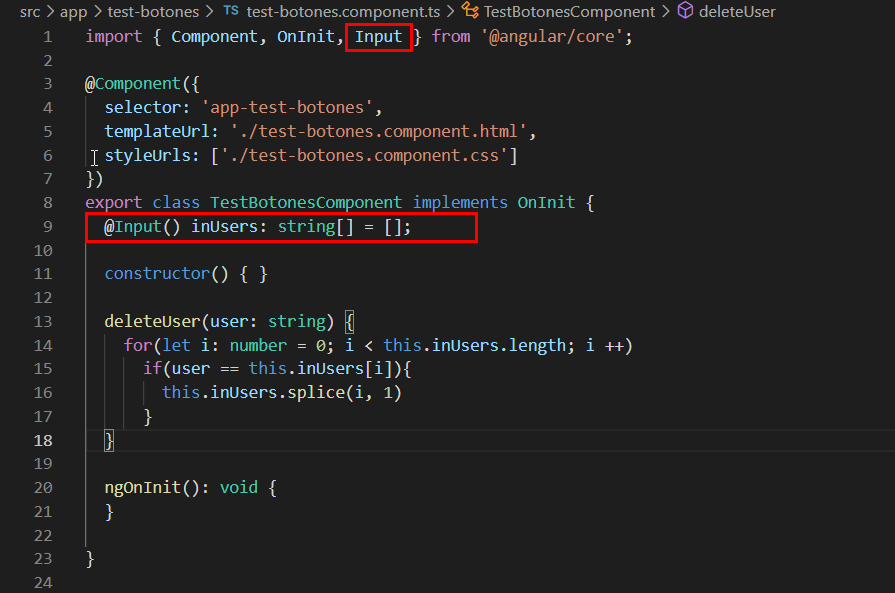
users: string[] = ['Jose', 'Carlos', 'Fernando', 'Luis', 'Ricardo']

y quiero que se la conozca en uno de los componentes que se utiliza dentro del archivo .html, al componente debo indicárselo de la siguiente manera

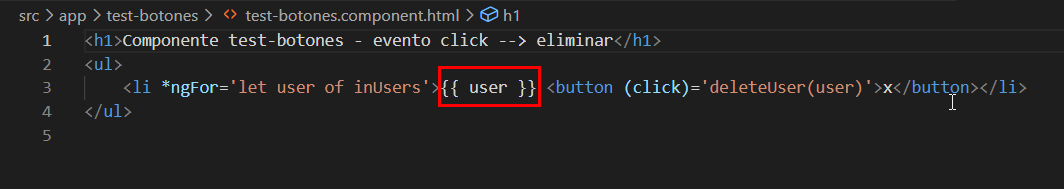
<app-test-botones [inUsers]='users'></app-test-botones>

A la etiqueta del componente le agrego un atributo con el nombre del argumento que quiero pasar, el uso de [] indicara que es un argumento de entrada. Y ecomo valor del atributo le doy la propiedad de la clase que defini anteriormente.

Por otro lado en el componente que recibirá dicho argumento, en su archivo .ts debo agregar el decorador @Input de Angular usandolo de la siguiente manera



Esto haría que se viese como una propiedad dentro de la clase del nuevo componente (TestBotones) y podría mencionárselo en el archivo .html del componente



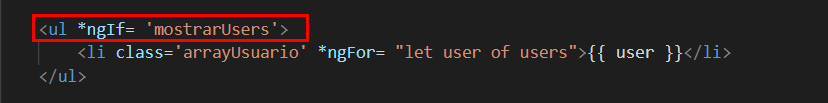
dentro del archivo .html para referenciar a una variable (propiedad) debe hacérselo entre doble llave {}, de esta manera el html reemplazara por el valor de la variable.

En este caso se esta utilizando la una directiva de Angular ngFor que nos permite ciclar que se utiliza como se muestra en el ejemplo, a manera de atributo de una etiqueta (en este caso un <li>) y lo que hace es ciclar por los elementos del arreglo inUsers a manera de un for each, y cada elemento lo mantiene en la variable user que luego utiliza para el ítem list.

Asi como el ngFor existe la directiva ngIf que se utiliza como condicional, supongamos que definimos una propiedad dentro de la clase que define a un componente (en el archivo .ts), supongamos que esa propiedad es de tipo boolean, y luego de acuerdo a la validación del el condicional ngIf

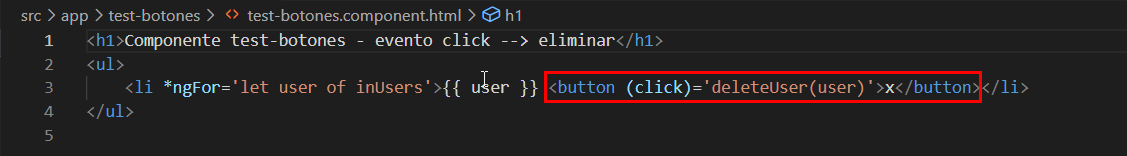


En el html por ejemplo en una lista desordenada <ul></ul> de acuerdo al el valor de esta propiedad utilizando una directiva condicional ngIf se mostrará o no la lista de ítems del arreglo de usuarios.

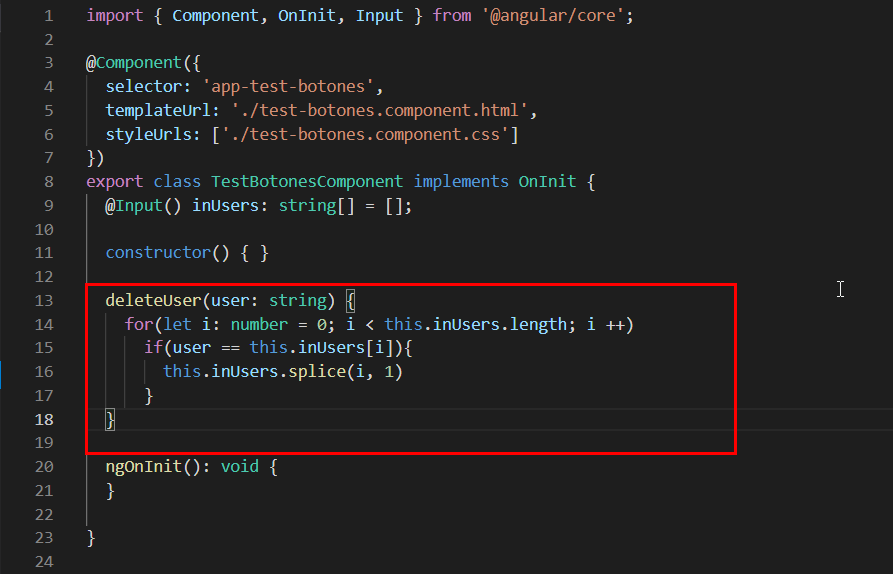


Tambien veamos como se maneja los eventos, por ejemplo el click de un botón, el submit de un formulario. En este caso el comportamiento del evento o el código será parte del archivo .ts, allí estará definido como un método de la clase que define al componente que estemos creando. Por otro lado se lo invocara desde el archivo .html del componente.

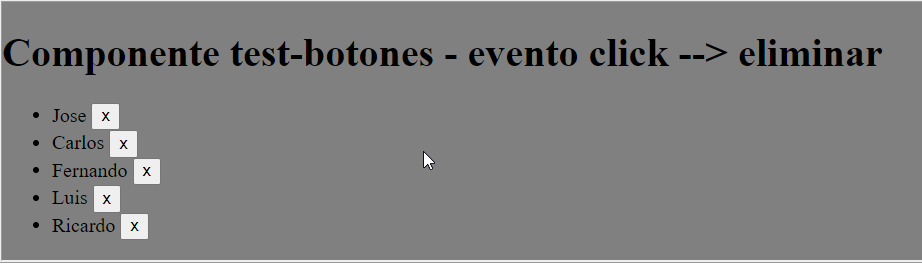
Supongamos que creamos una lista desordenada con los suarios del array y a cada uno le colocamos un botón al lado y definimos su evento click al que le asociamos un método deleteUser. En el archivo .html lo veriamos de esta manera



En la etiqueta correspondiente al ítem list <li></li> devinimos una etiqueta button <button></button> y en este para definir el evento click los hacemos entre () Para que entienda que es un evento y le damos el nombre del método deleteUser(user) que en este caso además lleva como parámetro el nombre del usuario que deseamos eliminar en el arreglo de usuarios. En el archivo .ts veríamos el código del método deleteUser() de la siguiente manera

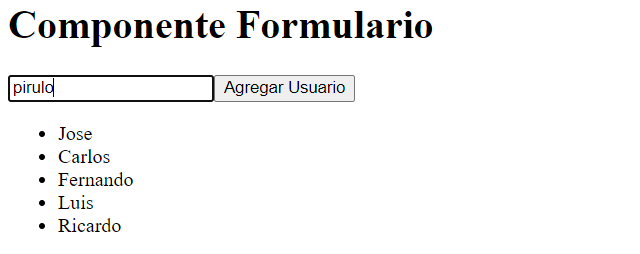
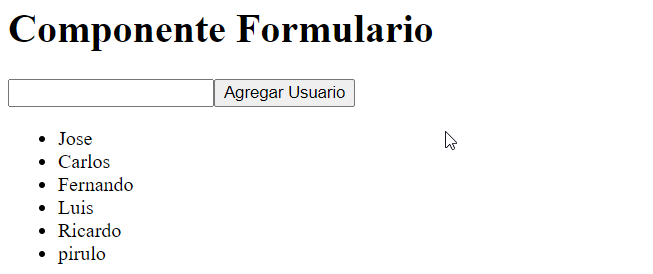


allí utilizando código TypeScript se recorre el arreglo de usuarios hasta encontrar el usuario a eliminar y obtenida la posición del elemento utilizando el método splice(posición, cantidad) se elimina el elemento del arreglo

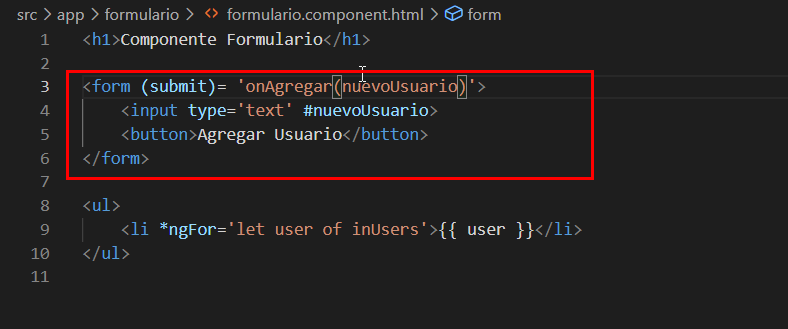


Cada vez que oprima en un botón x el elemento correspondiente desaparrecera de la pantalla, si recargo la pagina obviamente volverá a aparecer.

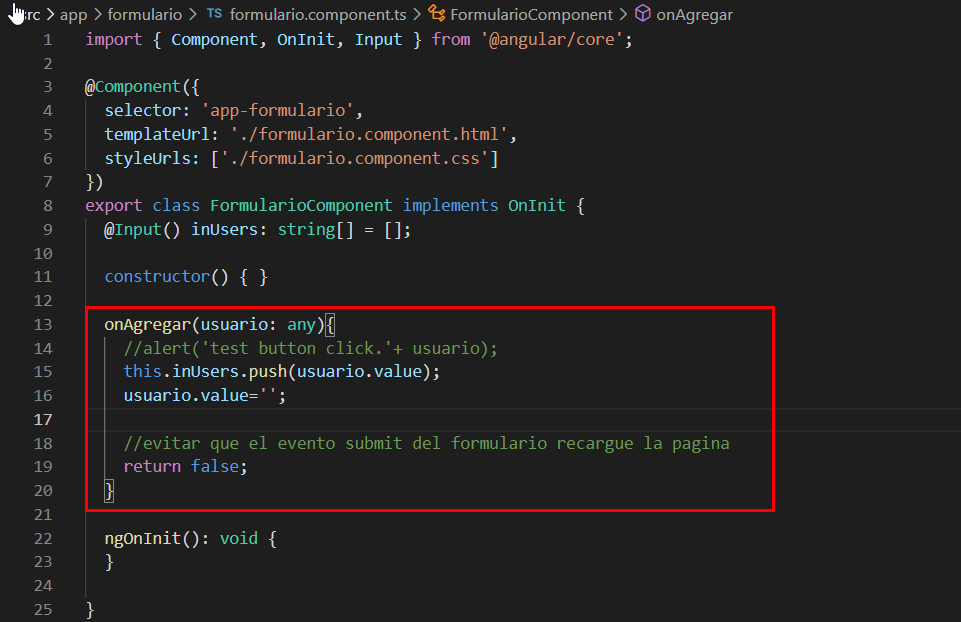
Veamos ahora el caso de un formulario que agrega un usuario a la lista de usuarios, que se mostraría de la siguiente manera

y que al darle al botón Agregar Usuario, lo agrega a la lista y limpia el input. Por un lado en el archivo .html defino un <form></form> con un <input></input> y un <button></button>. El evento del click en el botón Agregar Usuario no va a ser el evento click() de un botón, sino que sera el evento submit() del formulario, y se le asocia el metodo onAgregar(nuevoUsuario) que está definido en la clase que define al componente en el archivo .ts



ahora, es necesario pasar el contenido del <input></input> como argumento al evento para agregar el usuario al arreglo. Para esto utilizamos #nuevoUsuario



Algo que hay que tener en cuenta en este punto es que no estoy pasando el valor del <input></input>, o sea el string contenido, sino el tipo del argumento seria string. El tipo de lo que paso es any porque recibe el objeto que denomine como #nuevoUsuario, y para obtener el valor hago referencia a usuario.value en el método del evento onAgregar. Podria haber pasado como argumento solo el valor, haciendo la llamada al método como ‘onAgregar(nuevoUsuario.value)’, pero si paso el argumento de esta manera no puedo hacer que en el método se limpie el input asignando el string vacio.

Otra cosa a tener en cuenta es que cada vez que se ejecuta el evento submit() de un formulario, se refresca la pagina, esto es debido a que generalmente los formularios son utilizados para enviar datos a un servidor, no siendo este el caso, para evitar este comportamiento utilizamos el return false sobre el final del método.

Por último voy a agregar los botones para eliminar en cada <li></li> y utilizo el método onDelete(user) que ya mencione anteriormente.

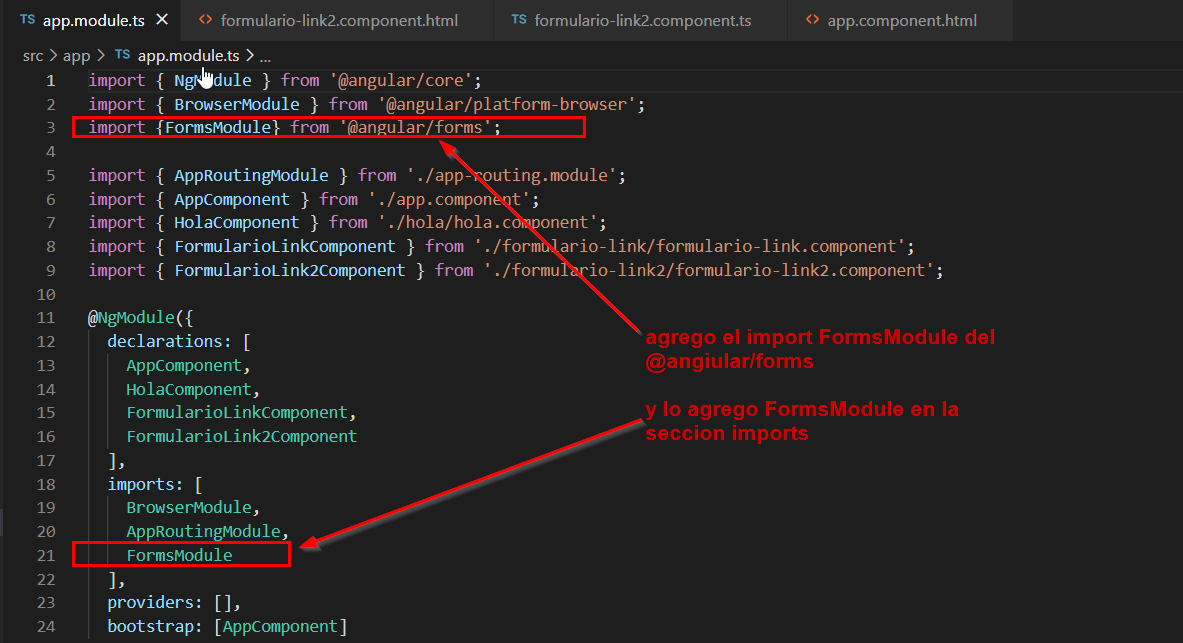


# ngModel - Doble enlace de datos (two-way binding)

Sincroniza un control de un formulario con una variable, cuando editemos un control de un formulario (por ejemplo un cuadro de texto) entonces se actualiza el valor de una variable vinculada con el nuevo valor del control (y al revés).

Angular proporciona esta característica, que por un lado muestra los datos mientras estos van siendo modificados.

En principio, para poder utilizar esta funcionalidad de Angular debemos activarla, para esto vamos a tener que importar el modulo de formularios FormsModule de Angular desde @angular/forms. Además en el app.module.ts debemos agregarlo en la sección de imports. Esto último se hace en el archivo app.module.ts



En el control dentro del component.html, por ejemplo un <input> tenemos que agregar una propiedad que indicara el modelo de dato, que asociara el dato con una variable (propiedad de la clase) del componente. Esta propiedad es ngModel y se escribe con la siguiente notación [(ngModel)] y se le asocia la variable en cuestion

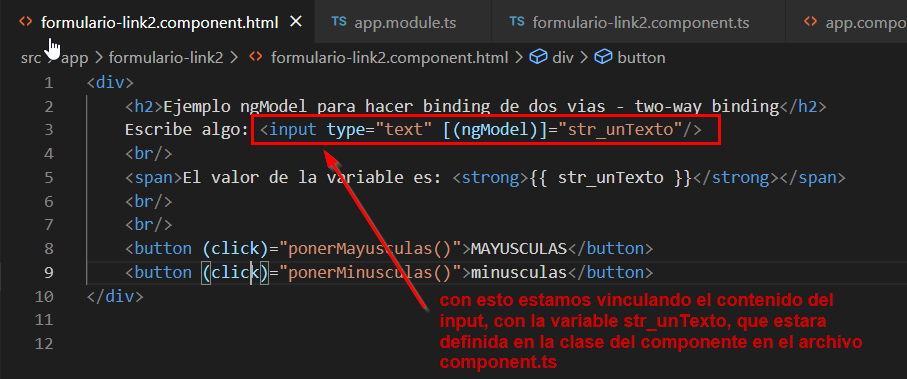
**<input type=”text” name=”nombre\_control” [(ngModel)]=”nombre\_variable”>**

Donde

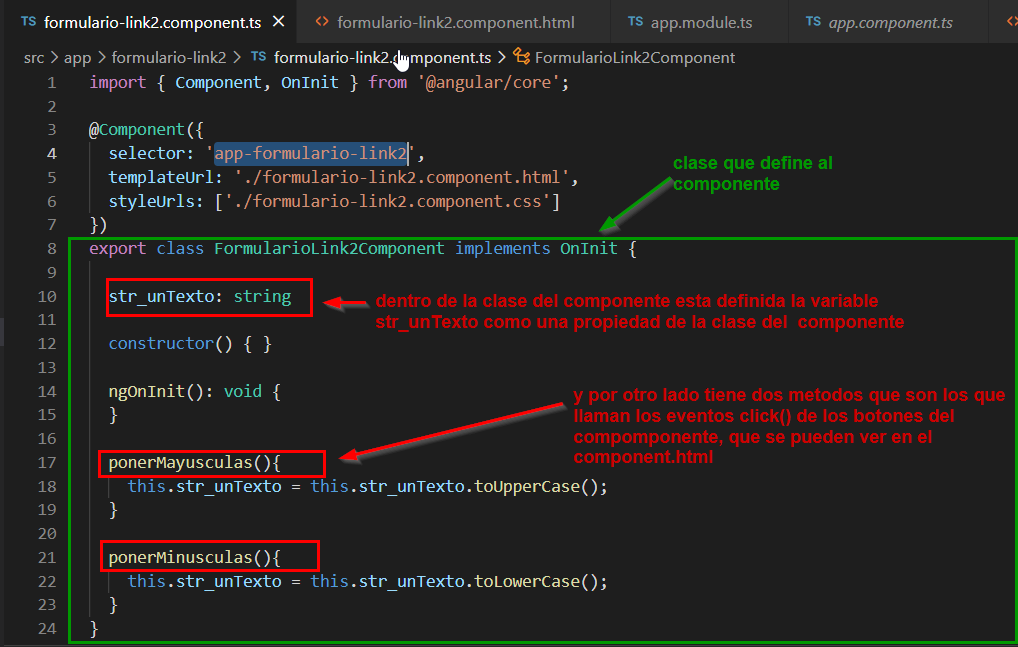
type=”text” 🡪 indica que el imput sera un cuadro de texto

name=”nombre\_control”” 🡪 solo para identificar al control con un nombre determinado por si hay varios mas distinguirlos

[(ngModel)]=”nombre\_variable” 🡪bincula enel conreo con la variable, si se modifica algo en el cuado se modificara la variable, si se modifica la variable en un método se modificara el contenido del cuadro de texto



Nota: [(ngModel)] debe utilizarse de esa manera (sin espacios) y como se menciono debe de importarse el modulo y y el paquete ya que esta funcionalidad no esta activa cuando se crea un proyecto Angular.

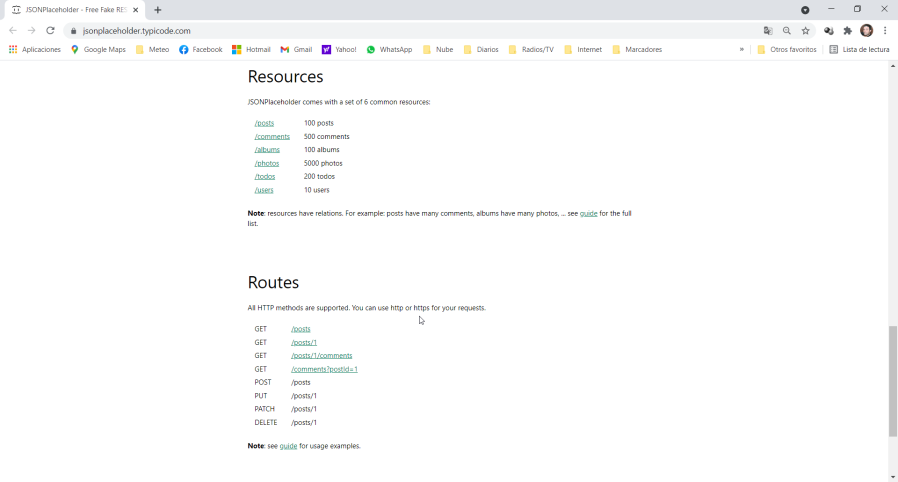


El formulario en la pagina se veria con un cuadro de texto que asocia a la variable, un párrafo que muestra el valor actual de la variable y dos botones que cambian el texto de la variable de mayúscula a minúscula. Si lo pruebo puedo ver que el valor que ingreso es el que toma la variable y que los cambios al ejecutarse los eventos de los botones se reflejan tanto tanto en la variable como en el input

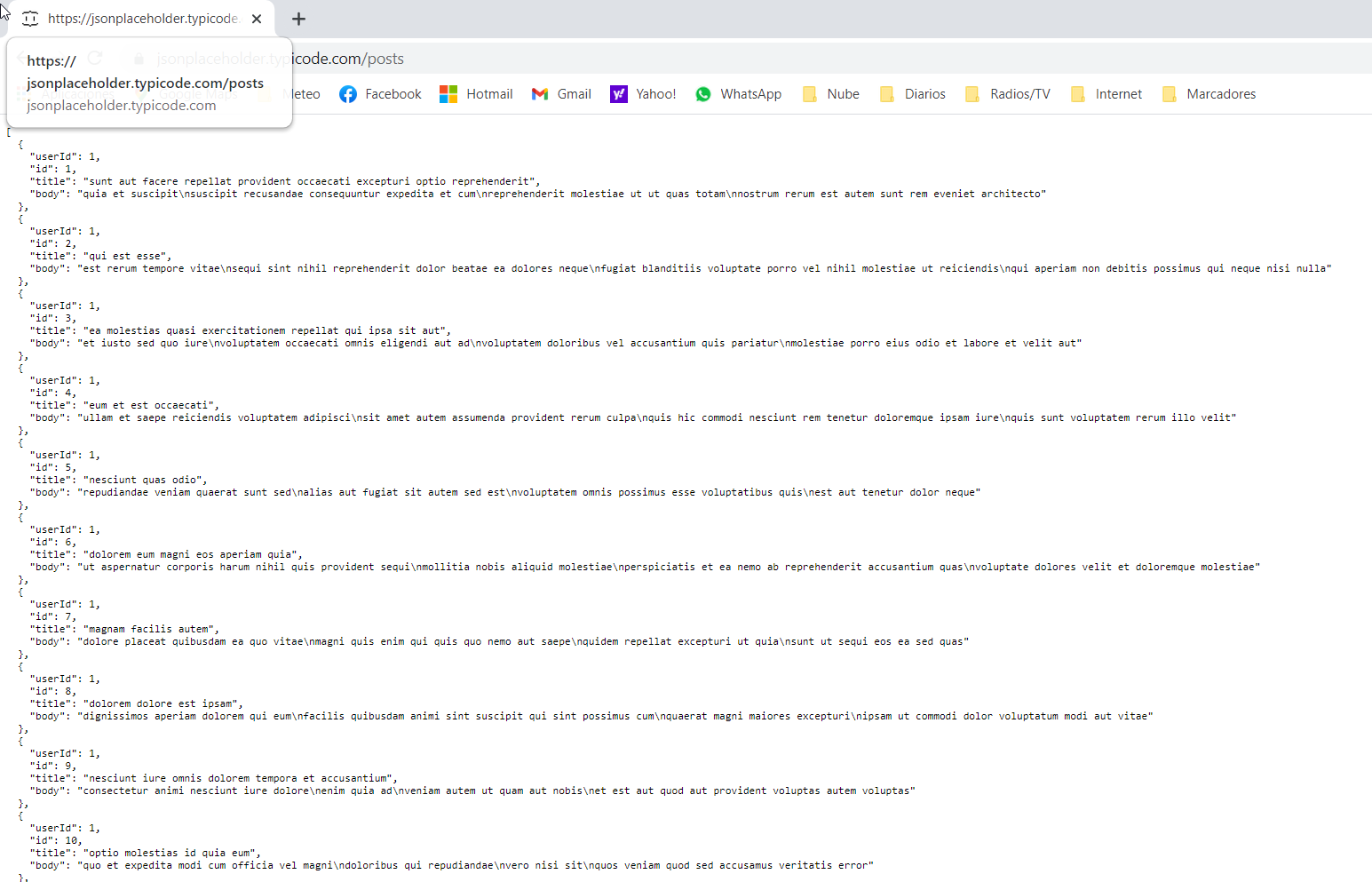
Todo esto que vemos ocurre en el frontend, los datos que agregamos, eliminamos o cambiamos en la pagina están en la memoria local, si yo refresco (recargo) la pagina, la misma vuelve a su estado original, con los datos que tenia originalmente, esto es asi porque una aplicación front end no se encarga de almacenar los datos (puede hacerse algo parecido a una persistencia pero en la memoria local del navegador, pero no es lo usual). La persistencia de los datos se logra cuando actuamos con el backend de la aplicación.

# Uso de JSON Placeholder

JSONPlaceholder es una API REST en línea gratuita que uno puede utilizar siempre que necesite datos falsos que sirven para probar cosas localmente.



Como se ven hay varios tipos de objetos Json (post, comentarios, fotos, etc) almacenados en arreglos

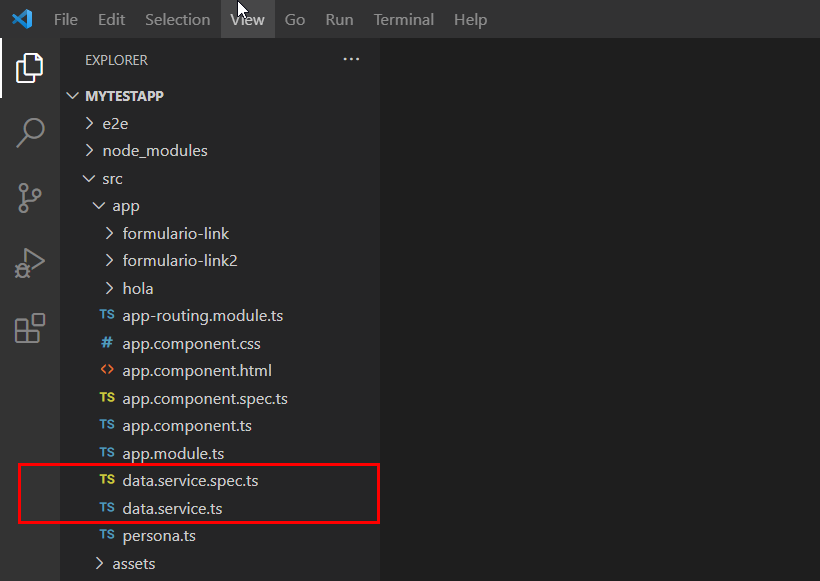


# Crear un servicio

Vamos a crear un servicio en Angular que tomara los datos desde JSONPlaceholder y luego poder mostrarlos como parte del front end de la aplicación. Entonces ahora vamos a crear en nuestra aplicación un servicio de la siguiente manera

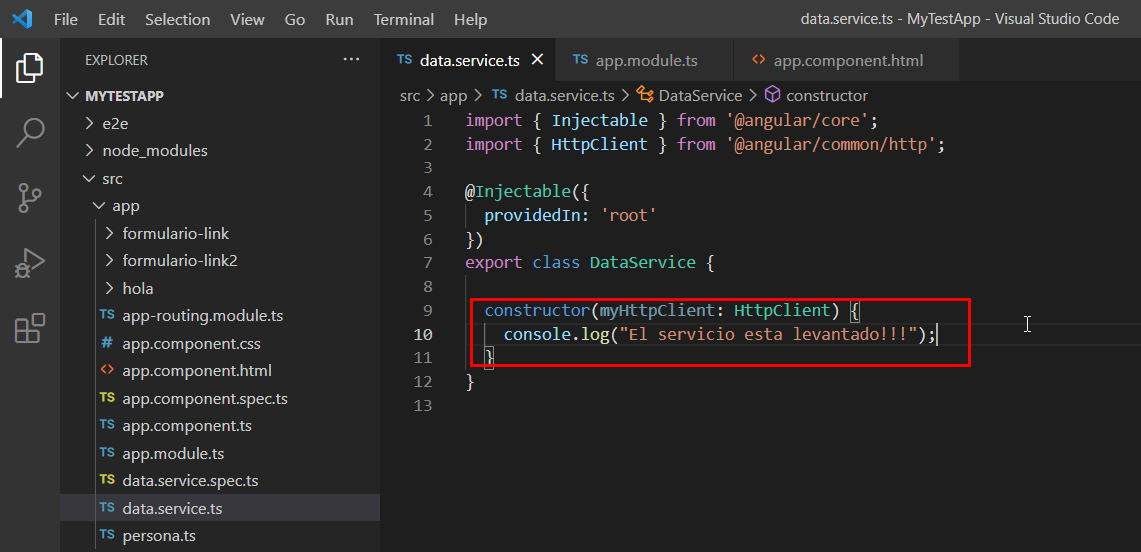
**ng generate service data**

este servicio nos brindara una clase en la que crearemos métodos que podremos utilizar en cualquier parte de la aplicación. En nuestro caso será el servicio lo llamamos data y nos creara una clase llamada DataService (Data por el nombre que le dimos al servicio)

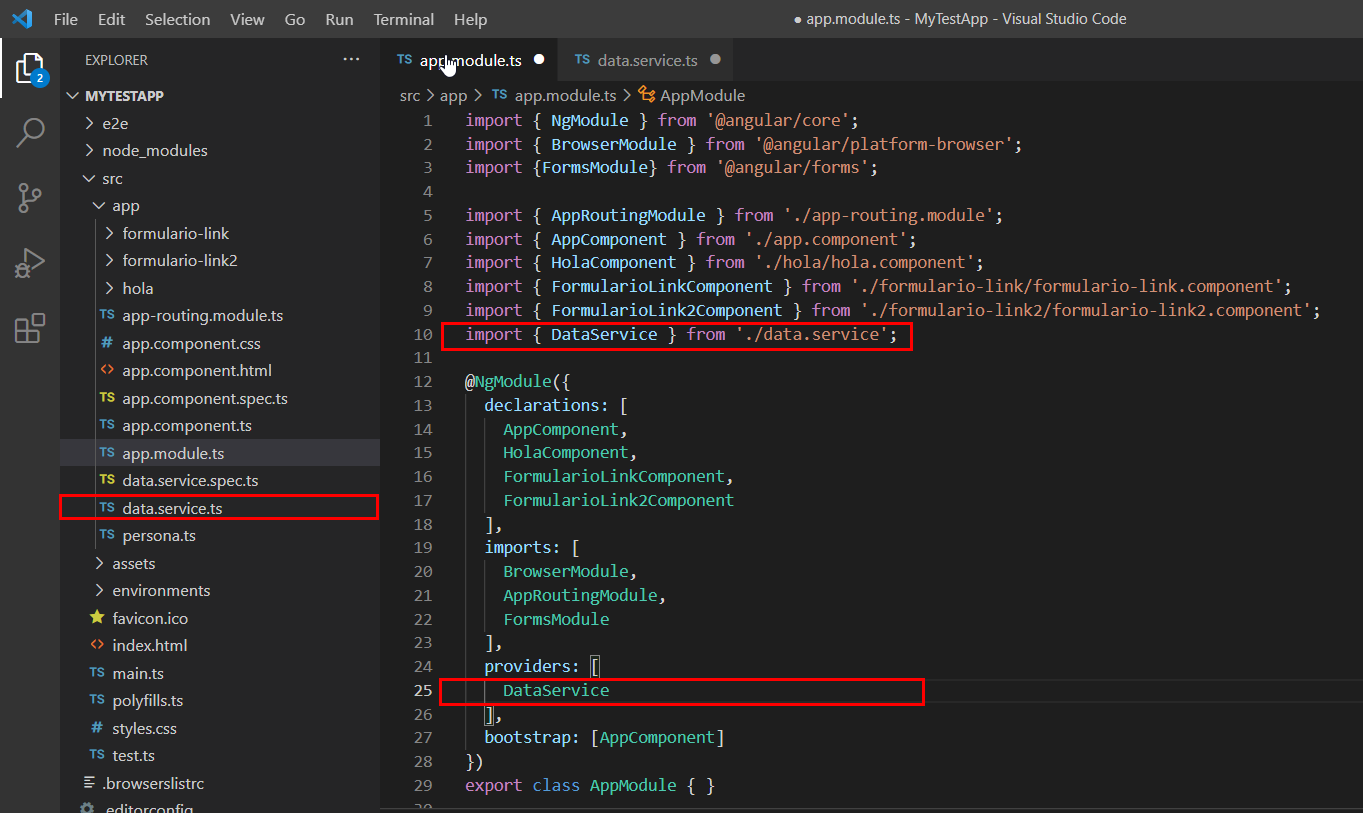


De los archivos que nos crea en la aplicación, el que nos interesa es el service.ts ya que el otro es para hacer tests sobre los servicios. Además vamos a necesitar en este caso importar la biblioteca http desde @angular/common/http y desde esta biblioteca indicamos que utilizaremos el método HttpClient, que es un cliente http que nos permite hacer peticiones desde Angular.

Este método que importamos lo vamos utilizar como miembro de la clase y para esto lo pasamos como argumento al método constructor de la clase (inyectamos el objeto HttpClient).



Por otro lado, a diferencia de cola creación de un componente con el **ng generate**, que automaticamente importa el modulo del componente y lo agrega en la sección Declarations del app.module.ts, para un servicio este trabajo debemos hacerlo manualmente nosotros. LO importamos y debemos declararlo, pero un servicio no es un componente que se delcare en la sección declarations, ni es un modulo que lo declaremos en la sección imports, este se lo declara en la sección providers

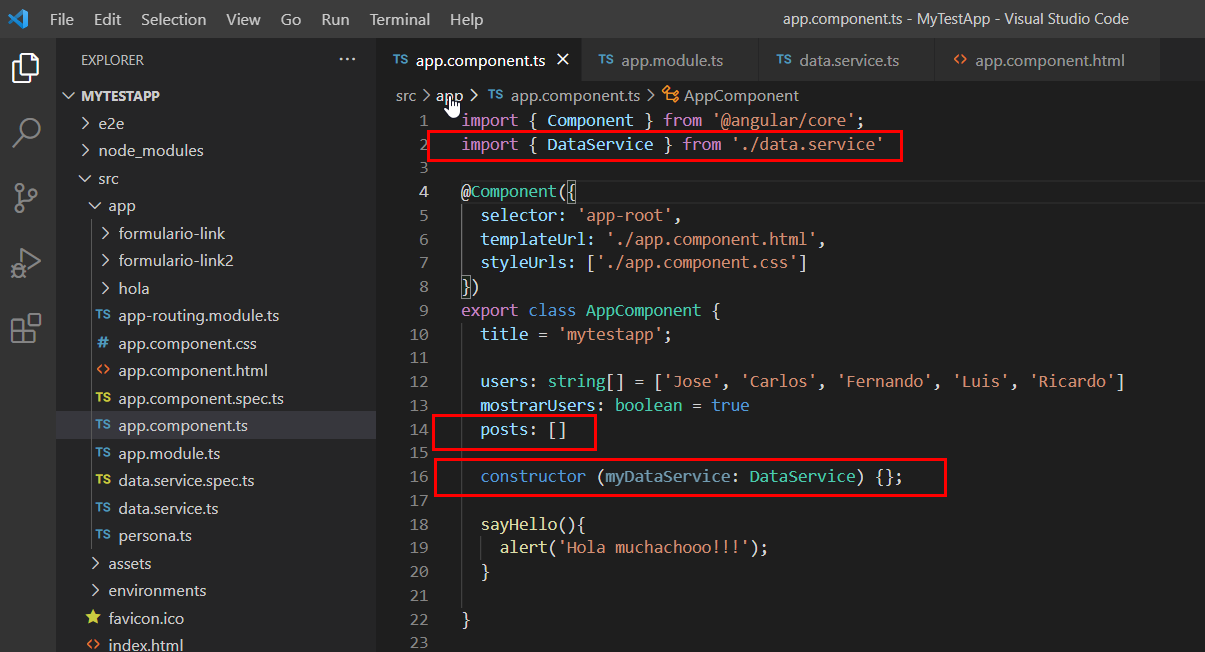


Veamos como lo vamos a utilizar.

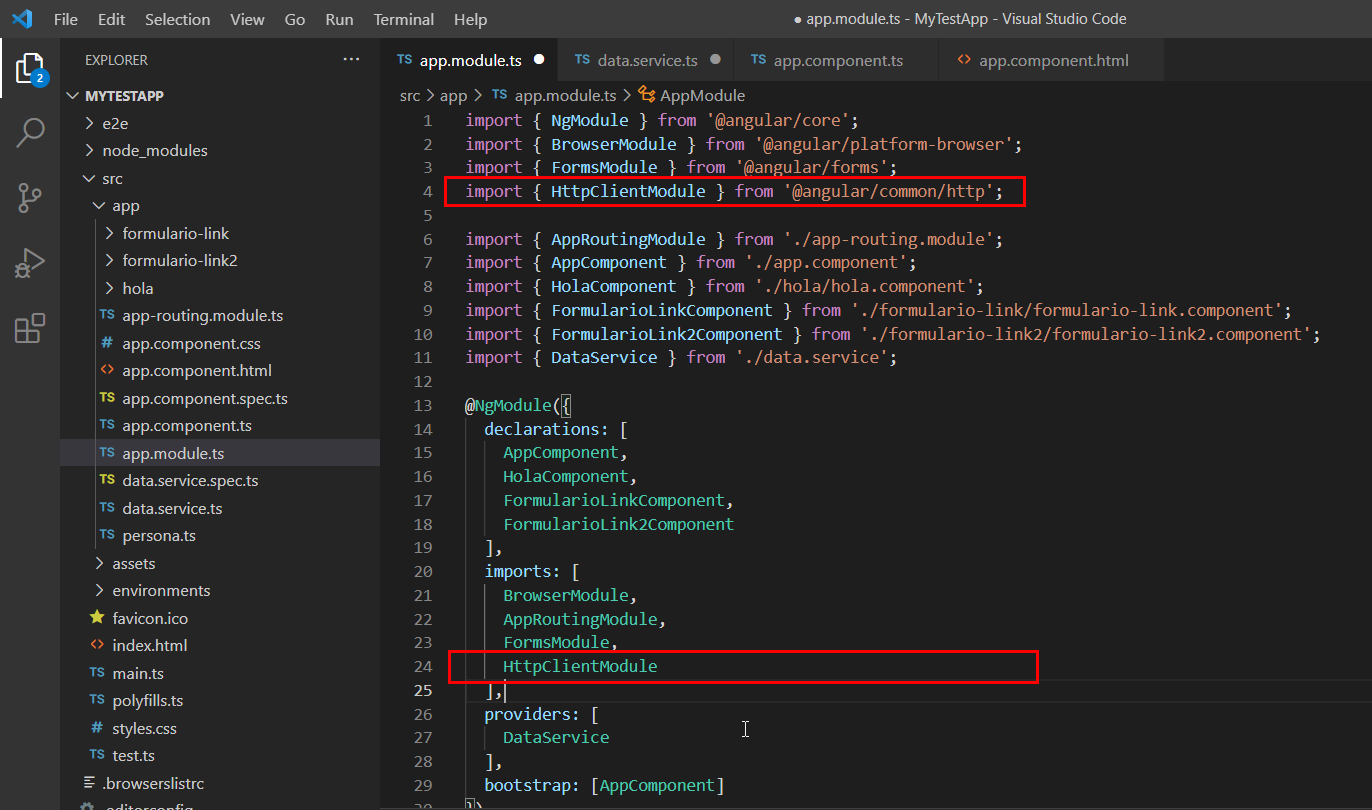
Lo que vamos a hacer en el app.component.html es listar todos los objetos del resource Post de JSONPlaceholder (debajo de las llamadas a los componentes que habíamos definido anteriormente).

En el app.component.ts dentro de la clase vamos a definir una propiedad posts del tipo [] donde quedaran almacenados los objetos que retorne el servicio.

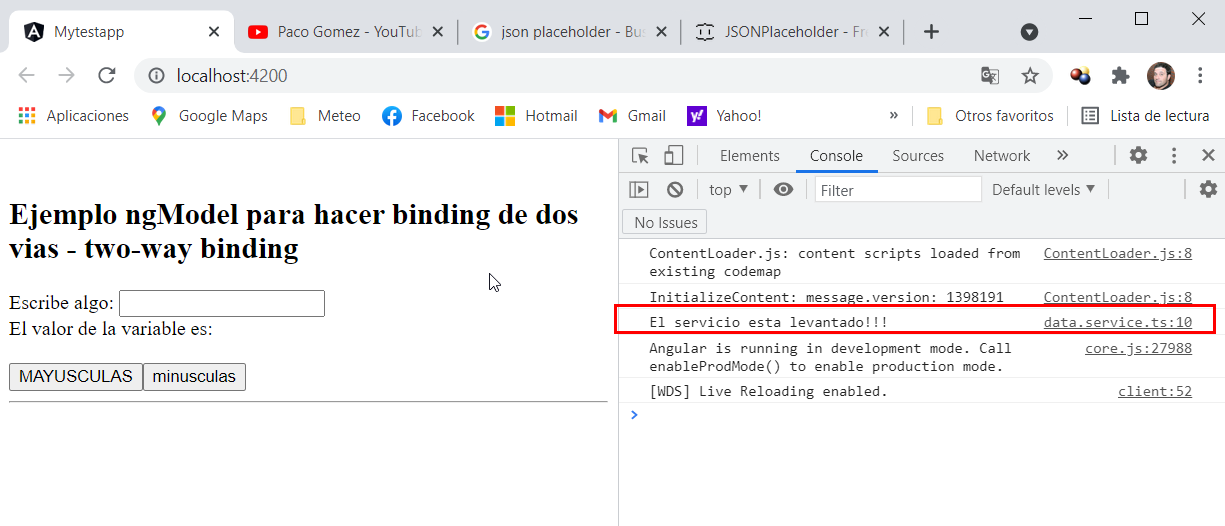
Vamos a necesitar el servicio que obtiene los datos al requerir utilizarlo aquí vamos a tener que importarlo también dentro del app.component.ts. El servicio lo vamos a utilizar dentro de la clase AppComponent, para la cual vamos a definir un constructor (ya que inicialmente no lo tiene) y a este método le definimos un argumento del tipo DataService (inyectamos el objeto DataService). El argumento lo definimos como private para que solo tenga alcance dentro de la clase. No va a ser necesario escribir nada más aquí para su uso.



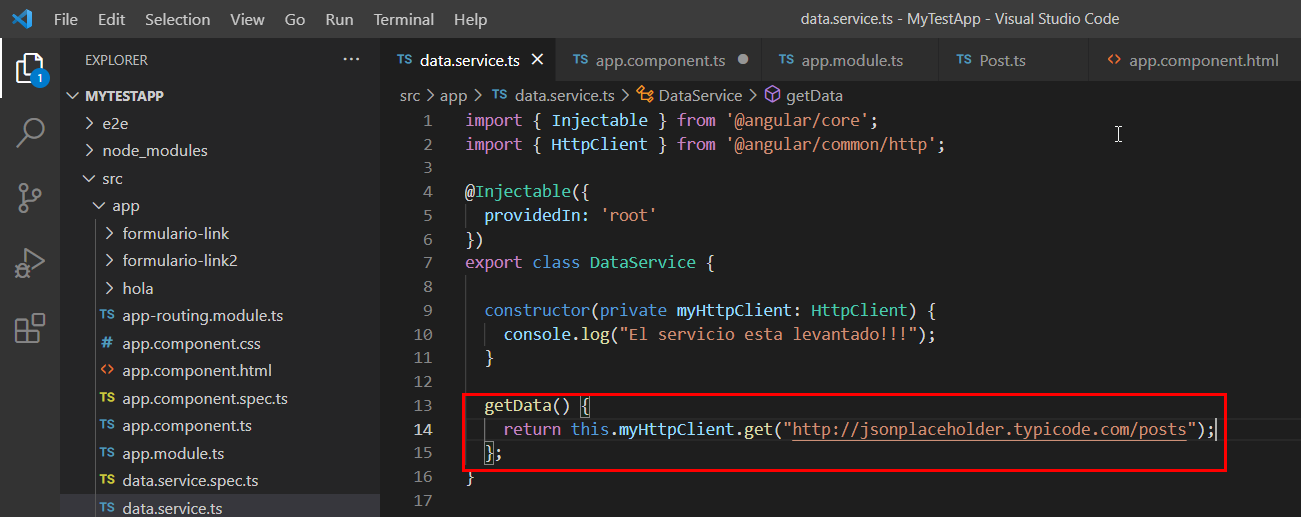
Pero al intentar ejecutar, esperando ver en la consola del browser el mensaje “el servidor esta levantado!!!” en logar de eso nos va a mostrar un error que veremos desde la consola del navegador y esto es porque al haber importado DataService y haberlo delarado en los providers en el app.module.ts no reconoce allí el HttpClient que utiliza DataService y para que lo haga habrá que importar el modulo HttpClientModule dentro del archivo app.module.ts y además declararlo en la sección imports



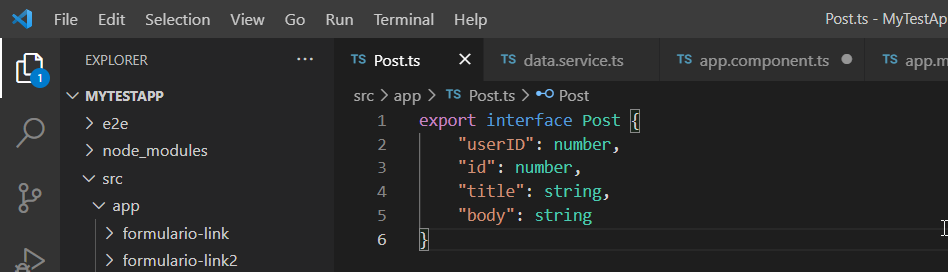
Hecho esa corrección veremos en la consola el mensaje de que el servicio Http de Angular que definimos para el objeto AppComponent esta levantado.



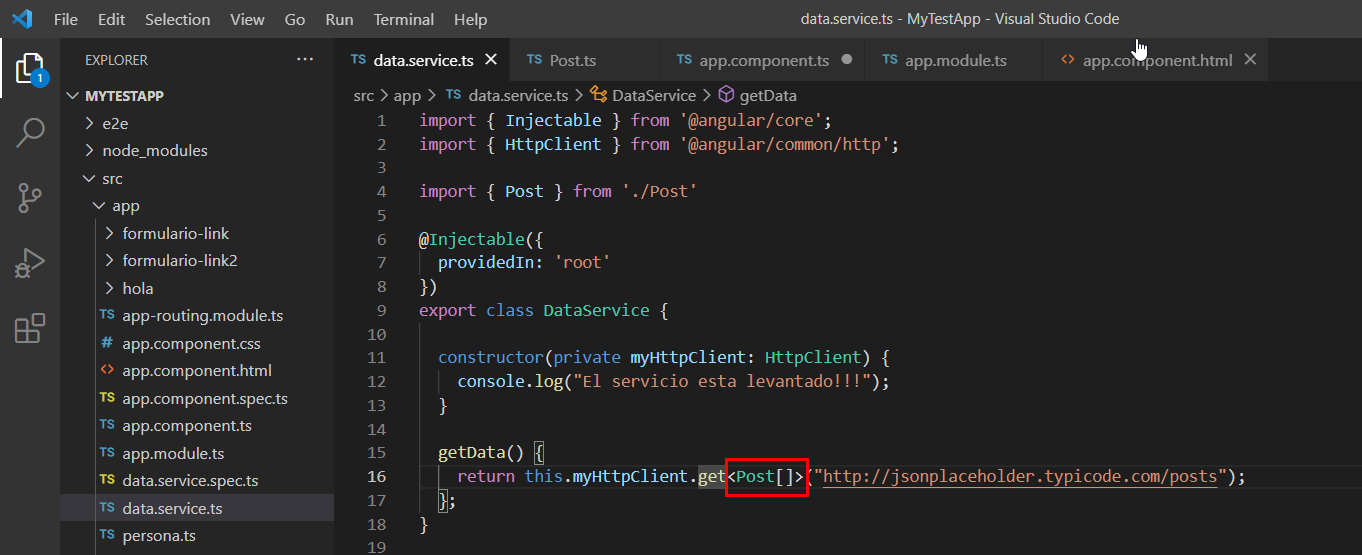
viendo que el servicio ya responde pues esta creado y que el mismo se levanta al iniciar la aplicación cuando se ejecuta el constructor del componente que crea el objeto myDataService que a su vez crea en su constructor el objeto myHttpClient de Angular. Tenemos que crear un método para utilizar el servicio que ya existe, el método lo llamaremos getData() y es el que utilizando el objeto servicio (myHttpClient) traerá los datos de Post de la pagina JSONPlaceholder mediante un get(URL)



por otro lado el get() nos retornara un arreglo de objetos Json del tipo post, vamos a definir un tipo propio Post en nuestra aplicación para eso creamos un archivo post.ts y allí definimos un tipo propio a partir de una interfaz Post, que exportamos

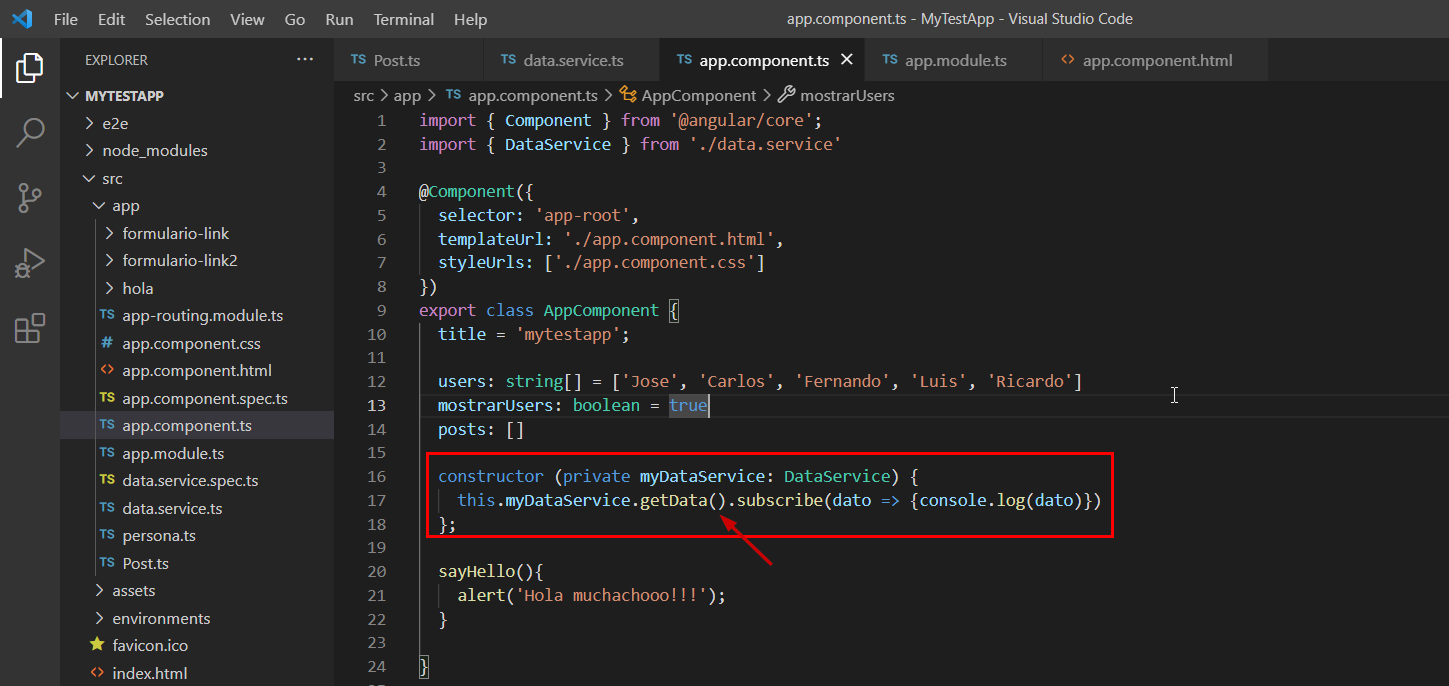


Definido el tipo de dato ahra podre hacer mas explicita la definición del método get() indicando el tipo de retorno del mismo de la siguiente manera



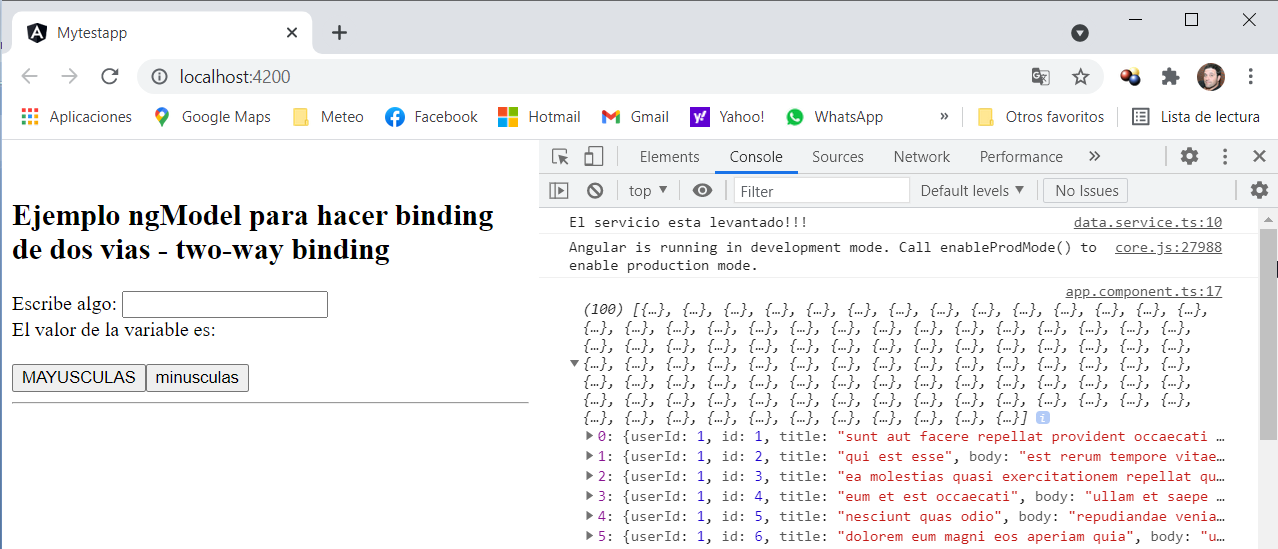
entonces vamos a llamar el método getData() en el constructor del componente como lo muestra la siguiente figura.

El método get() del http de Angular retorna un objeto observable (en este caso el arreglo de objetos post), los objetos observables http se deben consumir mediante el método subscribe() para que se lancen. En este caso observable llamara a una función callback dato => (console.log(dato)), esto es por cada objeto del arreglo de posteos imprimirá por consola, es una especie de loop por cada post.

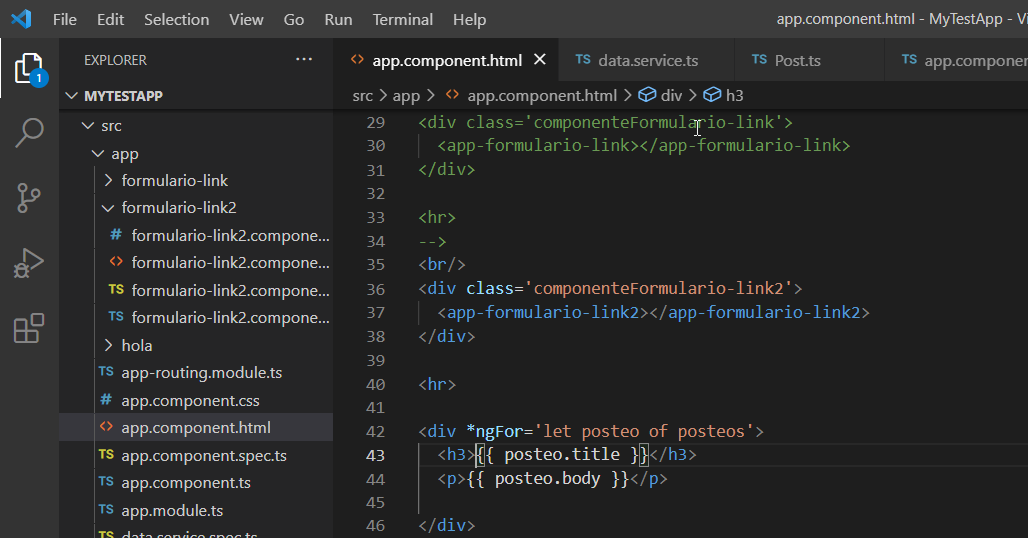


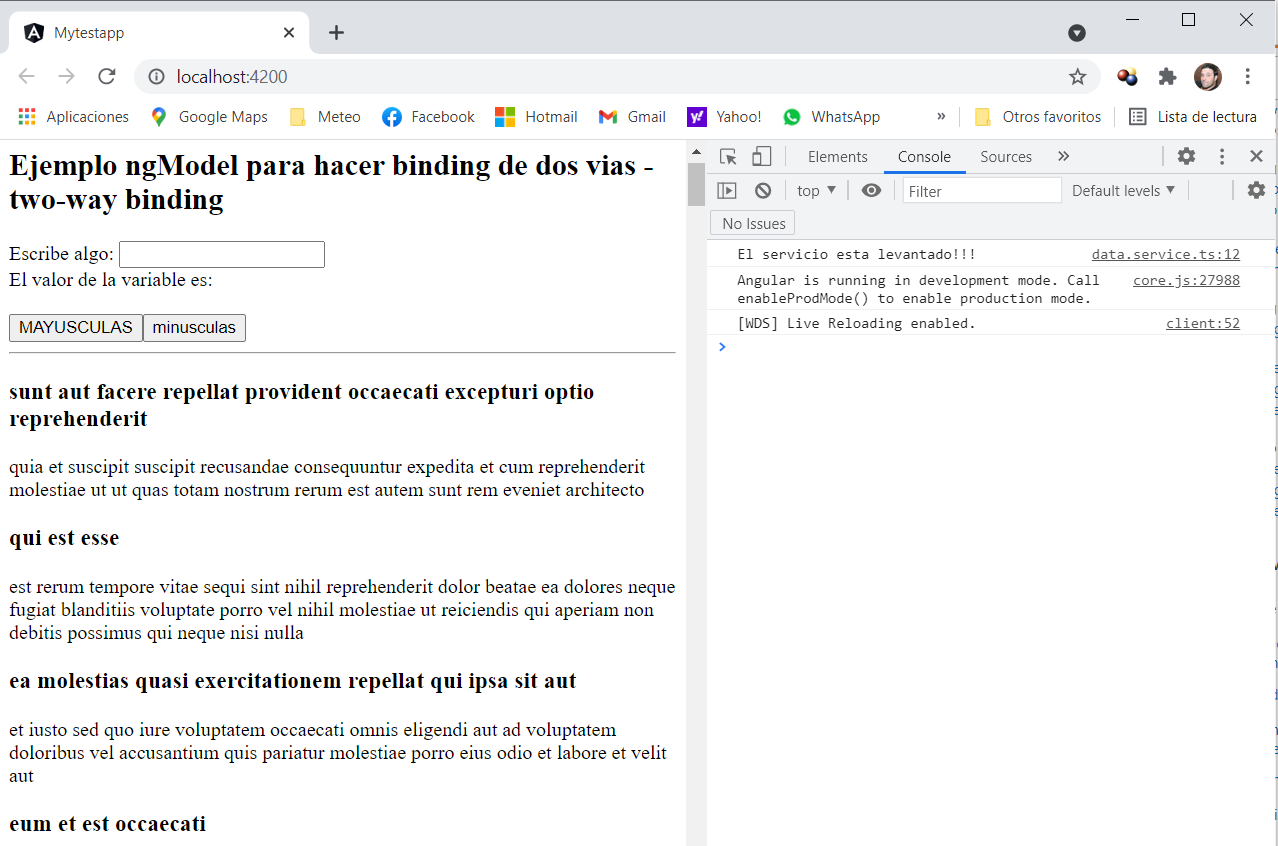
**Nota**: El método get() de http retorna un objeto observable. Los observables de http, han de consumirse mediante el método subscribe() para que realmente se lancen. Dicho método subscribe admite hasta tres callbacks para responder a tres sucesos posibles. Retorno de datos correcto, retorno de un error y señal de finalización. La sintaxis original ofrece tres argumentos opcionales para enviarles las funciones callback susbcribe(data, err, end). En este ejemplo solo hemos usado el primero.

Si observamos en la consola del navegador se puede ver que allí imprimió los objetos retornados por el getData(), en realidad por el get() que llama el anterior.



hasta aquí todo bien, pero aun no hemos mostrado nada en la pagina y ahora vamos por eso, lo que vamos a hacer es asignar a la propiedad posts[] que habíamos definido en la clase Component, el resultado del get<Post[]>(URL). Entonces en propiedad posts de la clase componente, ahora tendremos todos los datos retornados por el servidor. Solo nos queda mostrarlos por la pagina o sea que en el component.html podremos hacerlo recurriendo por ejemplo a un \*ngFor dentro de una etiqueta <div> y listar los valores de titulo y cuerpo del objeto posteo

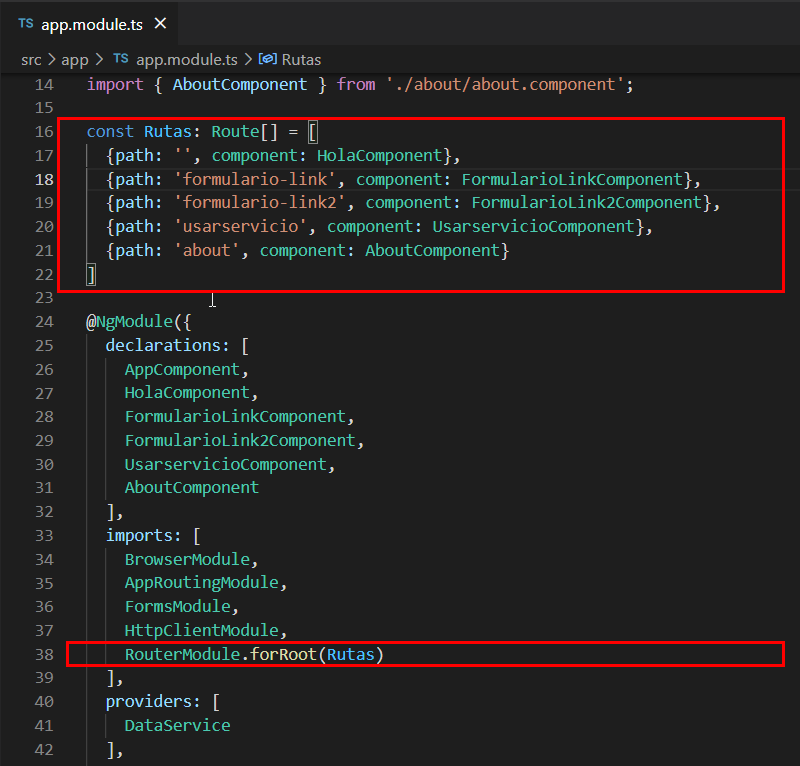




# Enrutador Router

Vamos a ver ahora como trabajar por ejemplo cuando nuestra pagina tiene multiples pestañas o una barra de navegación, para eso vamos a ver el concepto de Router o Enrutador, un router nos permitirá acceder a diferentes navegaciones sin que se refresque nuestra pagina.

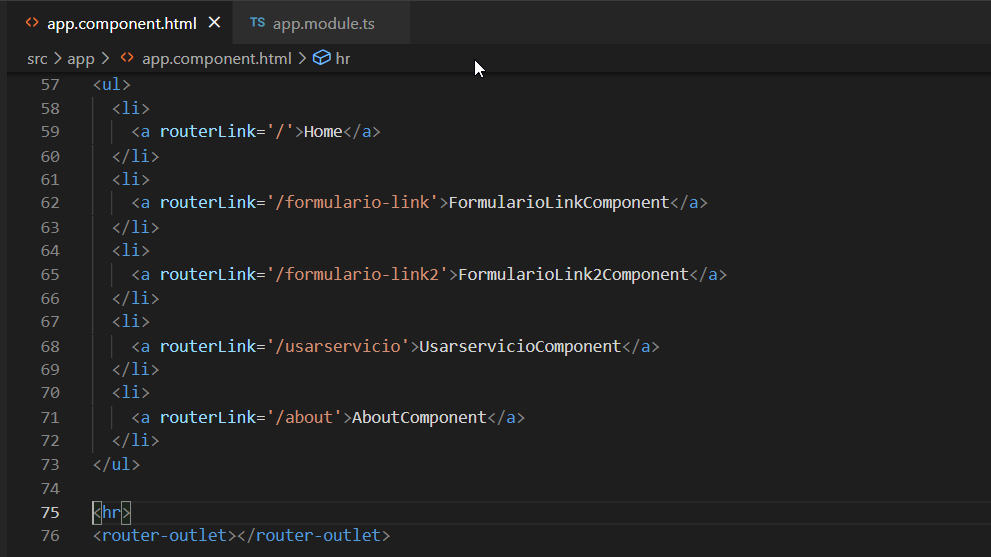
La funcionalidad del enrutador la vamos a conseguir importando la librería Router de Angular y específicamente el modulo RouterModule y Route en el archivo app.module.ts. Ademas en la sección Imports vamos a necesitar declarar RouteModule.forRoot(Routes) donde Routes será una constante a definir dentro del archivo app.module.ts que será una arreglo de objetos Route (arreglo de rutas). Como estarán formadas esas rutas, bueno la aplicación tiene una url con la ruta ‘/’ será la ruta root (el home de mi aplicación), luego podría tener una ruta ‘/about’ que cuando la selecciono me deriva seguramente a sección de la pagina ‘Acerca de…’



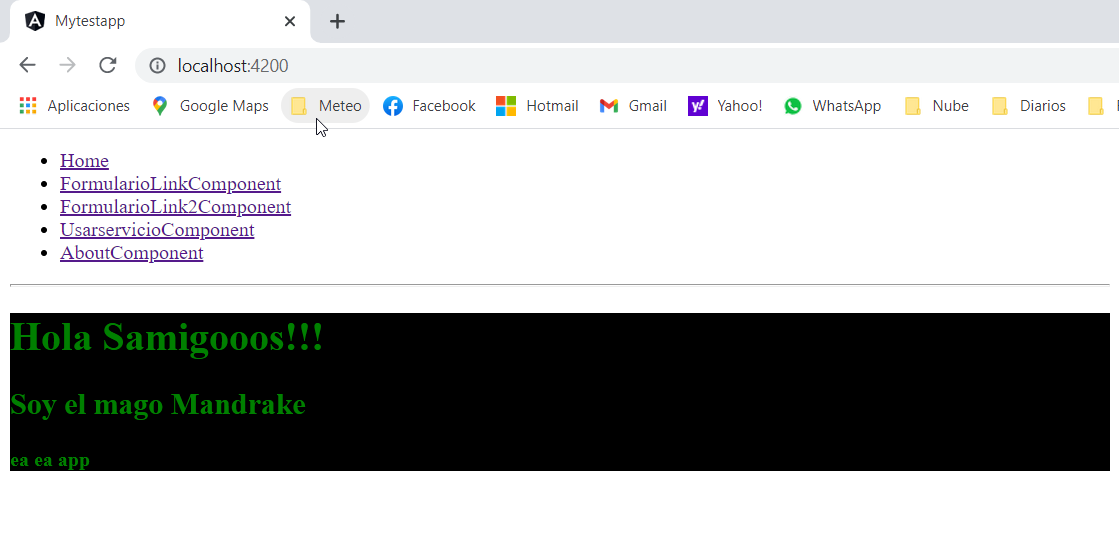
lo que muestra la imagen anterior es como debiera de quedar el app.module.ts, al que agregé una const Rutas, y defini en la sección imports el RouterModule.forRoot(Rutas).

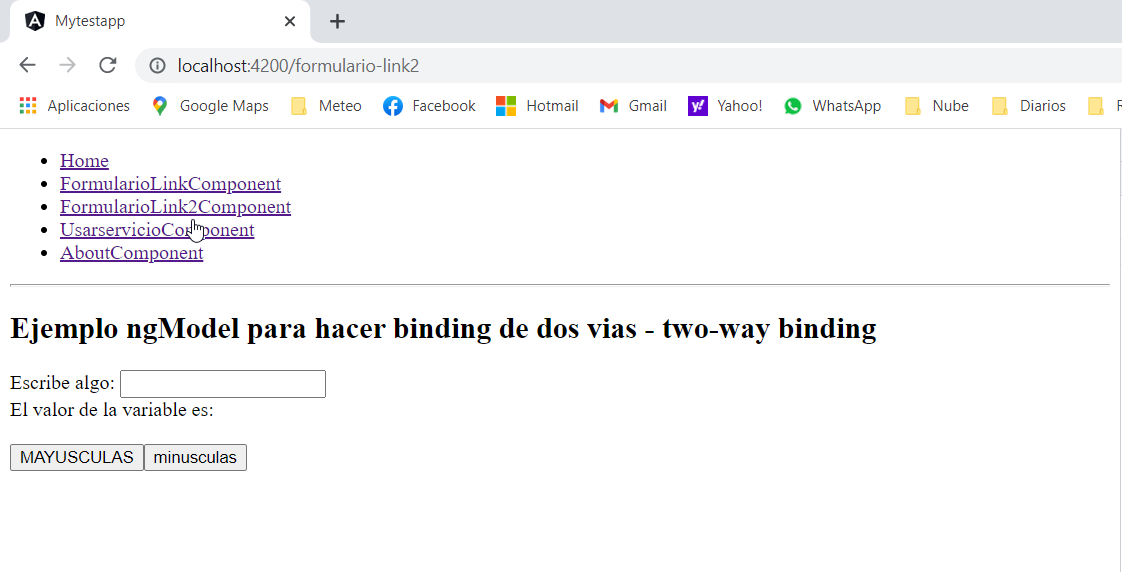
El arreglo Rutas contendrá los objetos que representaran las diferentes rutas de la aplicación, conformado por path, component. En path asocio el nombre que poseera la ruta (para conformar la url, en todas está implícito el ‘/’) y a component lo asocio con un componente de los que he creado para la aplicación. Como que cada componente seria una pagina.

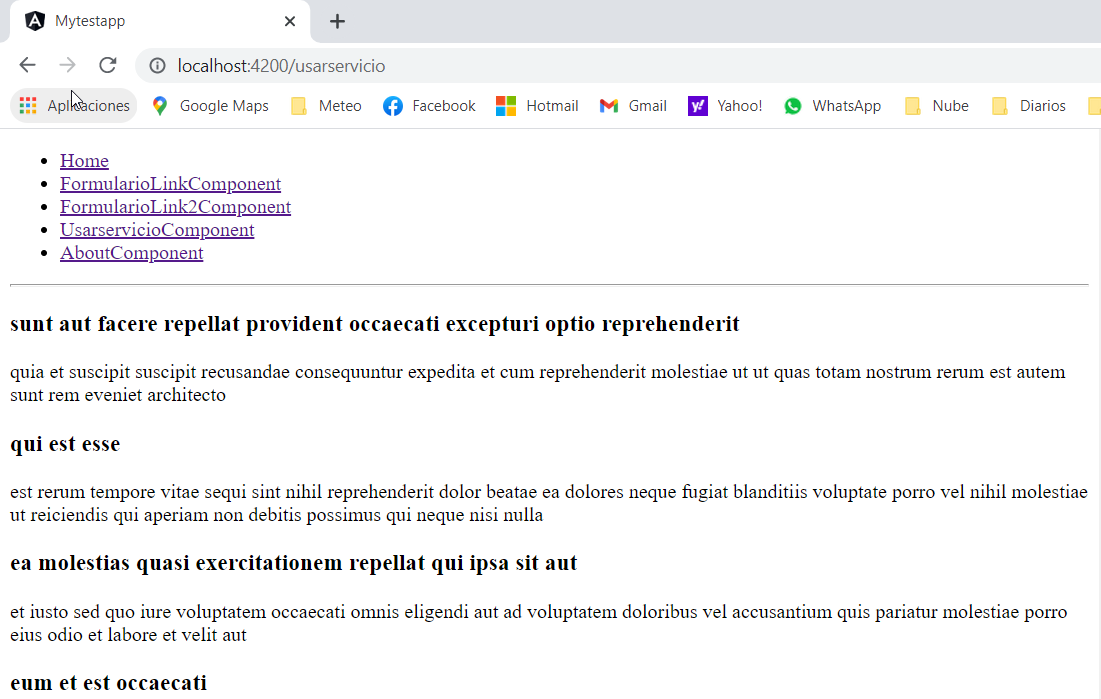
En el app.component.html es necesario agregar una etiqueta <router-outlet></router-outlet> para que Angular pueda mostrar las diferentes rutas dependiendo de lo que este solicitando el navegador y además agregare una lista desordenada <ul></ul> con ítems <li></li> que funcionaran a modo de un navegador. Cada ítem contendrá una referencia a una pagina, lo que de otra manera se veria como un <a href=></a> en este caso cambiando el atributo ‘href’ por uno ‘routerLink’ que referirá a las rutas definidas anteriormente en el app.module.ts.

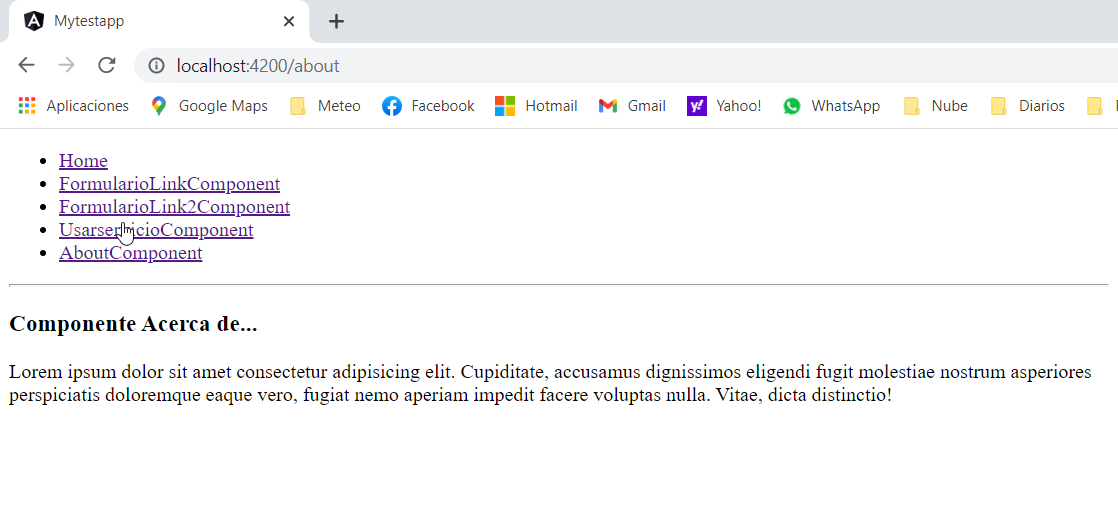


El resultado a la vista será el que se muestra mas abajo que nos permite navegar por lo que parecen diferentes paginas. La velocidad con que trabaja es debido a que se trata de una aplicación Angular cuyas paginas (o la creación de las mismas) no son obtenidas desde un servidor, sino que son creadas desde la misma aplicación del front end. LO único que viene desde un servidor son los datos que se cargan en el UsarservicioComponent que provienen de JSONPlaceholder.









# Ver curso de angular Paco Gomez

<https://www.youtube.com/c/PacoGomez/videos>

<https://appdividend.com/2018/10/03/angular-ngmodel-directive-example-tutorial/>