Creando el proyecto  
Ng new angular bases

Ng serve -o

Añadiendo Bootstrap  
en el index.html añadir debajo de link

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

Creando componentes:

Ng g c pages/about  
Se agrega automaticamente al app.module aunque desde ionic versión 4 no es necesario ya que se hace con lazyload.

**Implementando rutas:**

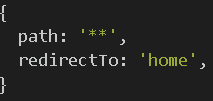
Creando un manejador de rutas 

Al ponerle el comando flat, crea el archivo en el mismo nivel del app sin crearle una carpeta.

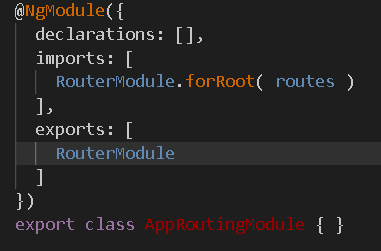
En el App component se crea la variable de rutas, importando el angular router.  
Esta contiene un arreglo de objetos que son las rutas, con el path y el componente al que se refiere.



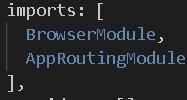
Y añadiendo una ruta por default cuando no exista la ruta escrita.



Para implementar el sistema de rutas se utiliza el modulo de rutas dentro de los imports del module de app routing y luego exportarlo:



E importarlo en el app.module

  
Recordando que desde ionic 4 no es necesario importar en el app.module ya que se utiliza el lazyload

Para poder visualizar las paginas dependiendo de lo escrito en el url y utilizar la configuración hecha en el routing module, hay que utilizar el tag en el app.component.html

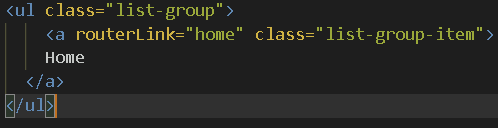


**RouterLink y MenuComponent**

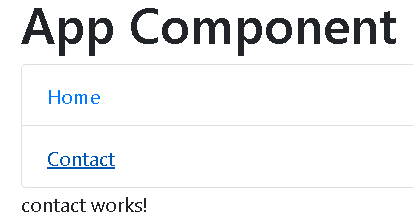
Creando un menú:



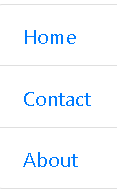
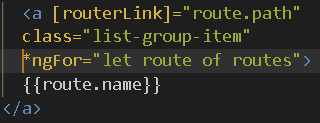
Dentro en el html se crea una lista ordenada con ayuda de Bootstrap y utilizando el routerlink podemos acceder al moudlo de rutas que creamos.



De esta forma si tenemos varias rutas, no recarga la pagina si no que se mueve dinámicamente.



Haciéndolo de una mejor forma se puede crear una variable con las rutas, repitiendo el tag a con un ngfor, usando la variable creada.



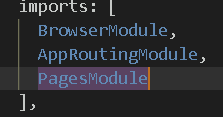
**Módulo de Páginas**

Para vaciar espacio en el app.module, creamos un modulo de paginas en la misma carpeta de pages





Así en el app.module ya no es necesario importarlos, si no que importamos el modulo de paginas en imports que contiene las paginas.

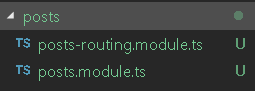


**LazyLoad**

Creando un nuevo componente que va a cargar de esta nueva manera.

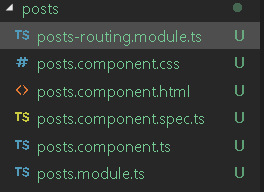
Se crea una carpeta posts dentro de pages y dentro un modulo y un modulo de rutas.  
(se agrega el parámetro routing para que se agregue un archivo de configuración de rutas).





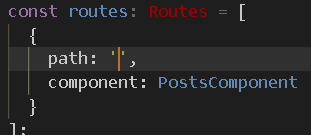
Ahora dentro de posts se crea un componente post (comando para que no se creen carpetas?)



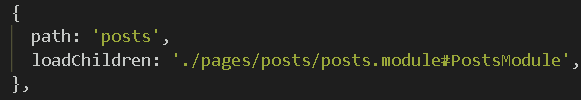


Ahora tenemos creado el componente de posts.

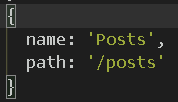
En el archivo posts-routing.module creamos las rutas hijas que tendrá este componente.  
Dejando en banco el path que luego será llenado en el archivo de app-routing.module

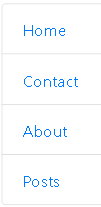


Modificamos el app routing module que teníamos originalmente para añadirle el post  
Utilizando un formato diferente con el atributo loadChildren.  
utilizando el pathy llegando al archivo postmodule, y añadiendo el carácter # para referirse al modulo a importar que contiene las rutas hijas que se utilizaran.



Para que se muestre en la pagina debemos agregarla al arreglo de menu con su path.





Ahora, La diferencia en usar lazy load, es que si no se ha seleccionado la función o el lugar donde se debe abrir esa pestaña o componente, esta no se va a cargar cuando se carguen los demás componentes del app.module.  
Al seleccionar la pestaña de posts se descarga el componente



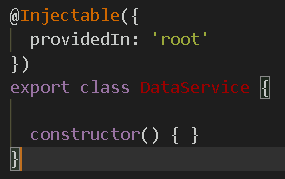
**Servicios y data externa**

En el html de posts creamos una lista ordenada con las clases de Bootstrap

Utilizando el mock de posts que viene en el curso se agrega al proyecto para poder iterar los posts en el html.

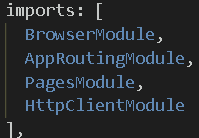
Necesitaremos un service para utilizar angular de la manera correcta en relación a recuperar datos externos.  
Creamos el servicio  


Se genera el servicio y vemos que tiene la propiedad injectable, providedIn : root. Esto significa que va a estar disponible en toda la aplicación globalmente y no va a estar importado en un modulo.



Es necesario realizar una petición http, ósea que debemos importarlo en el app.module.





\*Importante que sea de esta ruta y este modulo, recordando que los módulos van en los imports.

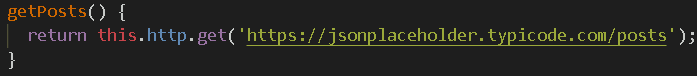
**¡IMPORTANTE!**

**En el app.module se importa el HttpClientModule,   
y en el servicio se importa el HttpClient  
ambos desde @angular/common/http**

Luego dentro del servicio data en el constructor importar el HttpClient

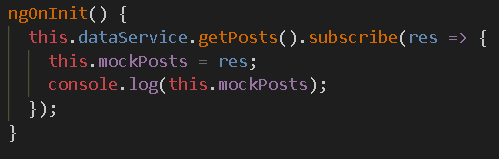


Ya podemos crear la función del servicio que pedirá la información para luego llamarla

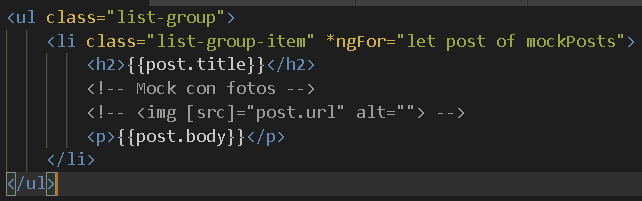


Recordando que para poder recuperar esta informacion, desde el componente que hará el llamado debe hacer un subscribe a esta función.

Ahora desde el componente de posts creado nos subscribimos al servicio creado para traer el arreglo de objetos mockeado de posts.



Y en el html creamos la lista que mostrara el arreglo recuperado del servicio



Al tener un subscribe en el ngOnInit este se va a ejecutar cada vez que la pagina sea cargada, lo que puede traer pedidos de información excesivos, data perdida sin necesidad. Por lo que es necesario cancelar la suscripción.

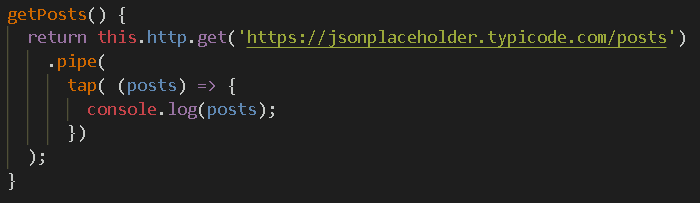
Una alternativa a esto es asignar a la variable que va a recibir la informacion del servicio, la función que llama al servicio local, y luego en el html estructurar la variable con un pipe “**async**” el cual hará que esta informacion se solicite para poder construir la vista.





Podemos implementar el operador tap de rxjs desde el servicio.  
El operador tap es conjutamente usado con pipe en cadena.

Esto lo implementaremos desde el servicio.



**@Input, Enviando información hacia un componente hijo**

Creando un nuevo componente para hacer reutilizable la lógica de los posts

El componente padre ahora va a ser **POSTS**, el cual hará el llamado al componente hijo desde el html con el tag que tenga. Este componente hijo **POST** será iterado según la cantidad de posts (en este caso) devuelva el servicio. Y se le pasará con la propiedad **INPUT** utilizada de la forma [Nombre de variable en hijo] = “variable en padre”.



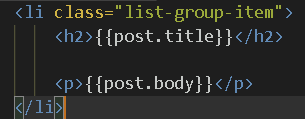
Previo a esto debemos haber creado el componente post.:



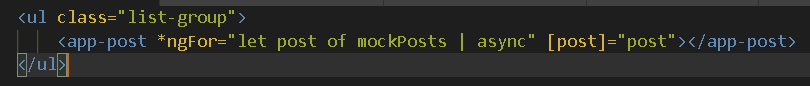
Ahora dentro de este componente en el component.ts declaramos la variable con el mismo nombre como la mandamos en el padre, Utilizando la propiedad @Input



Y en el html



Quedando así la estructura del padre:



**@Output, emitir eventos desde el componente hijo.**

Como prueba queremos que al hacer click en un componente hijo, su id sea impreso desde el componente padre.

**En el componente hijo**

Declaramos una variable de tipo output, que podrá ser vista desde afuera del componente cuando utilicemos su tag.

Para declararla es necesario importar la propiedad EventEmitter();



Podemos tipar el EventEmitter:

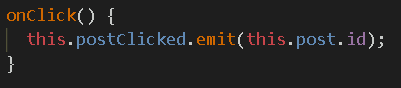


Ahora desde afuera podremos acceder a esta variable.

Creamos un método on click que sea llamado cada vez que sea clickeado el hijo



Para ahí emitir hacia el componente padre o donde se este escuchando la variable y enviar el valor.



**En el componente padre**

Ya podemos utilizar la variable evento creada y cuando detecte que esta fue emitida desde el hijo se llamará a la función del padre, en este caso postListener, con un parámetro evento que puede ser renombrado en el component.ts

