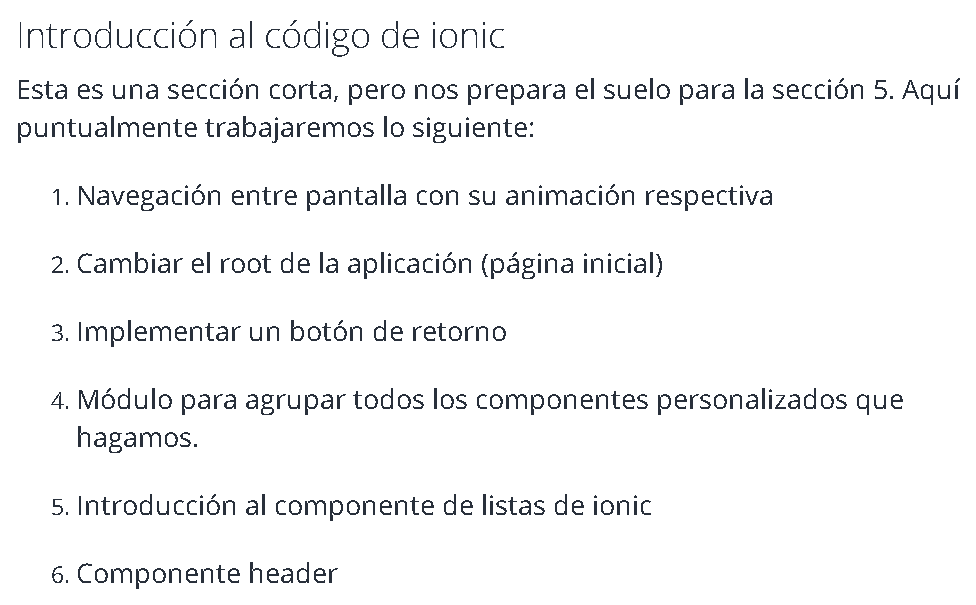
**Bases Ionic**

**Capitulo 1**



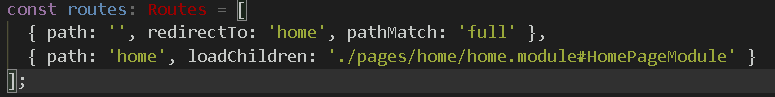
Para crear una aplicación



En el proyecto borramos la carpeta home que se crea por defecto, para crear nuestra propia pagina de inicio.



Crea la pagina en la ruta y actualiza automáticamente el app.routing.module con la ruta hacia este, con lazy-load.

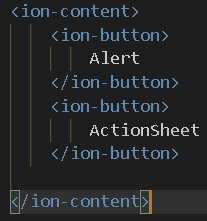


**Navegación entre paginas**

Creando paginas:

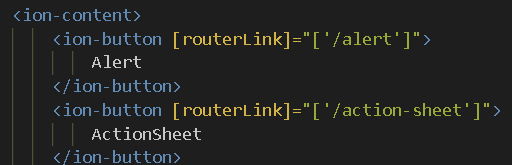
Se crean dentro de pages : alert y action sheet  


Creamos dentro del home 2 botones para poder navegar entre los nuevos componentes creados



Para añadirle la navegación

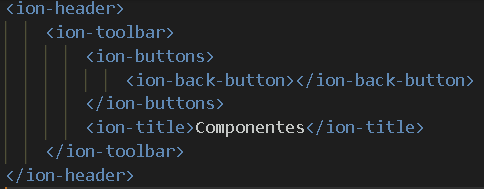
Se puede manejar de igual forma como en angular con el routerlink y el nombre ya agregado a la lista del modulo de routing.



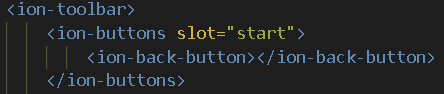
**BackButton**

Con esto se puede navegar, pero necesitamos agregar un botón de back para poder regresar al componente anterior.

Dentro del header -> toolbar de cada pagina agregamos un ion buttons y dentro un ion-back button, el cual viene programado para regresar a la pagina anterior.



Para posicionar los botones agregados en el toolbar se le agrega la propiedad slot y la posición



Y para que cargue automáticamente esta propiedad





Podemos personalizar la palabra que tenga este botón con la propiedad text



**Header**

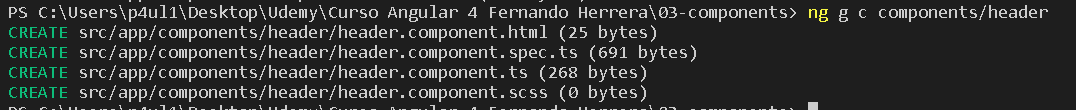
Dentro de app se crea una carpeta de components y dentro un modulo que va a manejar todos los componentes de la aplicación. Esto para no duplicar código por ejemplo con el header y solo hacer el llamado a este con su tag y personalizando un titulo.

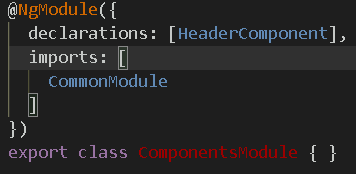


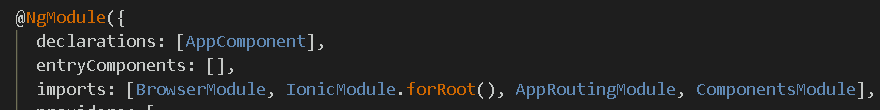
Para luego crear el componente del header



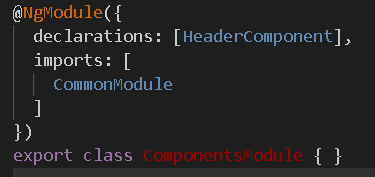
Verificar si no se modificó el module de componentes, ya que este componente debe estar declarado ahí para poder ser utilizado. Si no está, agregarlo manualmente.





Recordar que hay que añadir el nuevo modulo que contiene el header, al app.module en los imports.Esto va a lanzar un error ya que como todo esta cargado de forma LazyLoad, el header no ha sido cargado. Es necesario cargarlo en el modulo de la pagina donde se vaya a usar, en este caso y momento: alert y action sheet.

Ahora falta una configuración, si vemos el components.module, nos hace falta decirle que esta configuración va a poder ser usada fuera de este modulo.





Verificar también que este modulo de components tenga el IonicModule en los imports, ya que el código que tengan los componentes seguramente tendrá nomenclaturas de ionic.



Funciona correctamente el header después de pegarle el código que tenían los headers individuales. Ahora falta hacer dinámico el titulo pasandolo como un input, para que reciba este valor desde el componente que lo llama.

En el header.component.html





Y en los componentes donde se usa el tag se le pasa el valor:

Ejemplo AlertComponent  

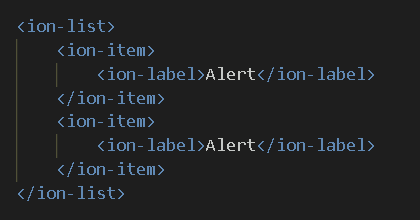

También se le puede añadir la propiedad al header de no border para que no tenga borde de sombra.





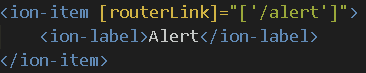
**ION-LIST**

Es una lista de elementos parecida a angular con la estructura





Al cambiar la estructura poniendole el router link al elemento redirecciona a la pagina que apuntemos y pone una flecha al final del ítem para hacer notar que es un elemento que redirecciona a otro lugar, pero esto solo sucede en iphone. Para que aparezca en Android hay que agregarle otra propiedad al ítem “detail”.



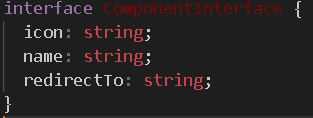




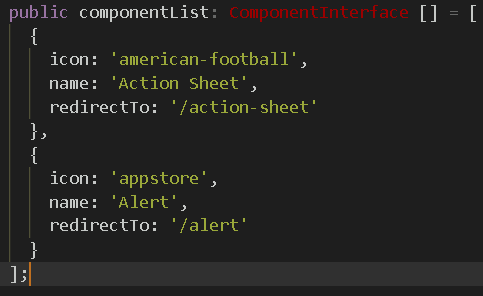
**Automatizando la lista**

Ahora para hacer la lista de los componentes que tenemos, no debe hacerse manual, si no creando un arreglo de componentes para recorrer y se muestre en una lista desde el [home.page.ts](http://home.page.ts)

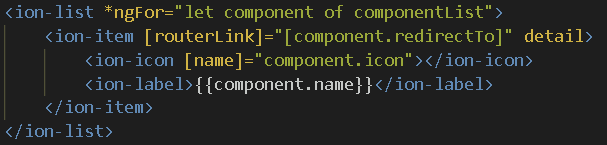
Para esto creamos una interfaz dentro del [home.page.ts](http://home.page.ts) que va a ser de tipo componente.

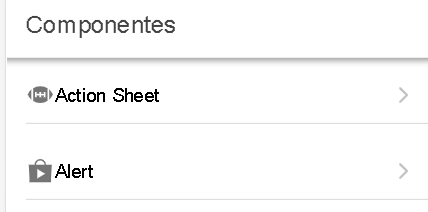


Y dentro de la clase del home creamos el arreglo de este tipo con la informacion de los componentes creados.



Ahora dentro del html podemos iterar en base a este arreglo



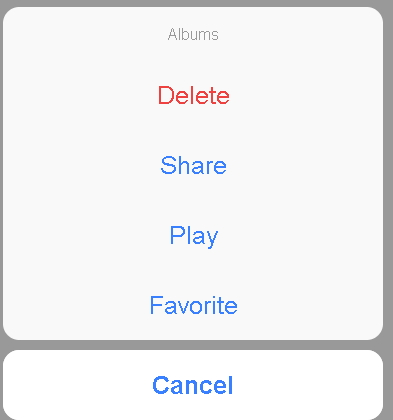


Arreglando los iconos, podemos definir su posición con la propiedad que usamos anteriormente “slot”.



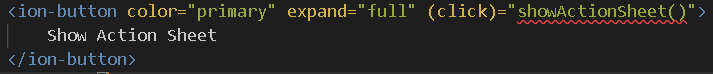


**Ion-action-sheet**



En el html del action sheet creamos un botón el cual abrirá este action sheet

La propiedad expand se refiere al tamaño del botón.





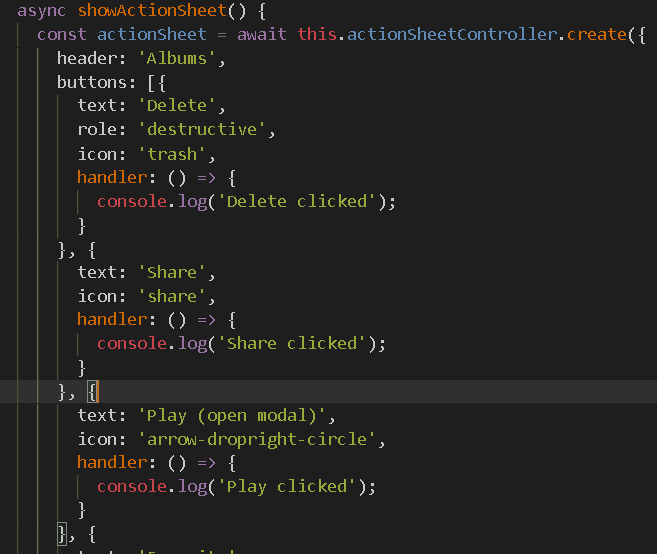
Creamos la función en el componente, esta empezará con el tipado **async** ya que es una función que regresa una promesa:



Seguido se declara una constante que tendrá el action sheet, para esto siendo una promesa se declara el valor que se le asignara con un tipado de await, lo que hace que espere que se cree el actionsheet y luego se le asigne, siendo así una función asíncrona.



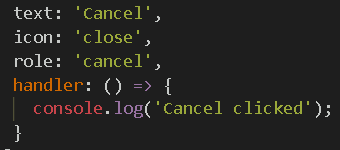
Añadiéndole luego el contenido del action sheet:



Y finalmente esperando que la variable actionSheet declarada este lista, se muestra:



Es importante saber que en el handler existen objetos con role:  
Cuando este dice Cancel, significa que no solo al presionarlo se activa, también presionando fuera del action-sheet.

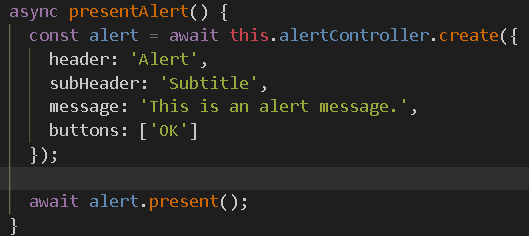


Se puede forzar también a que no se pueda tocar fuera del action-sheet para cerrarlo, o sea que es obligatorio seleccionar uno. Esto, agregándole la propiedad backDropDismiss: false.



Alert

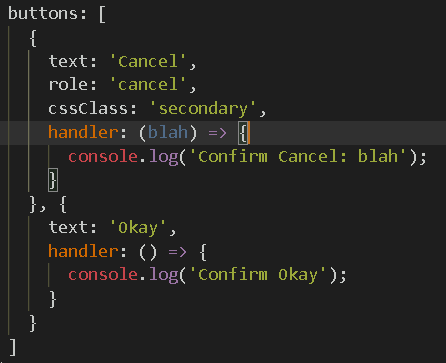
Importando el AlertController en el constructor, y luego creamos la función que llamara al alert

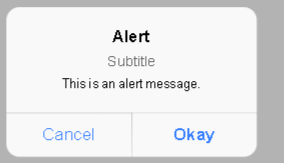


Esta función se llama desde el html creado para el alert

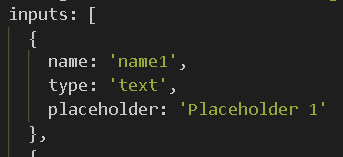


Añadiendole el arreglo de botones con sus acciones

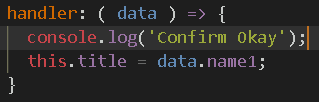




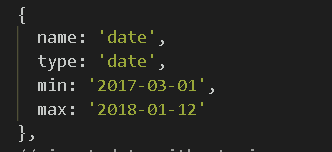
Podemos también tener inputs dentro del alert, para esto se declara un arreglo de inputs a los cuales para acceder a sus valores se podrá mediante el nombre que se le asigne.



Dentro del botón de confirmar:



O con fechas



**Avatars**

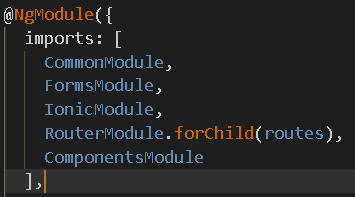
Creamos la pagina para avatar



Después de crea la pagina, declararla en el arreglo de componentes de home, recordando que la propiedad declarada redirectTo hará referencia hacia el nombre declarado en el routing.module que se genera automáticamente al generar la pagina con ionic.



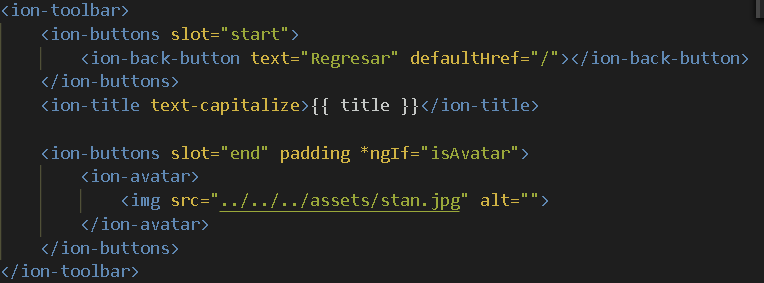
Recordando añadir al modulo de avatar el Components module, el cual tiene declarado el HeaderComponent, para poder utilizar el app-header en el html.



La pagina de avatar esta lista y como estamos utilizando el header componentizado, el poner un avatar en el header no seria practico crear un header para esto. Es necesario añadirle lapropiedad de avatar al header, y que este lo pueda activar dependiendo de los parámetros que este reciba según la necesidad del componente.



El header component quedaría así:



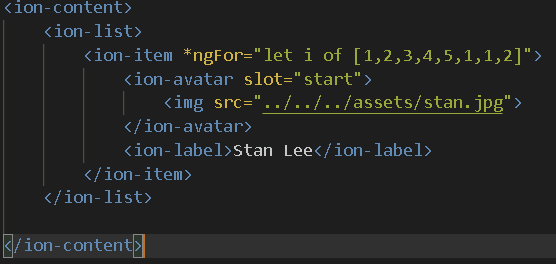
Con el código del avatar dependiendo de la variable que lo activa, y esta esta declarada como un @Input

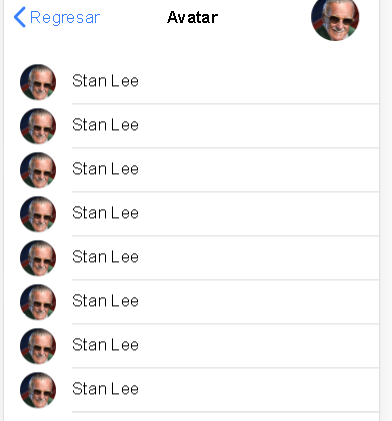


Luego en el componente que lo este llamando se le pasa el valor, en este caso desde el avatar component.



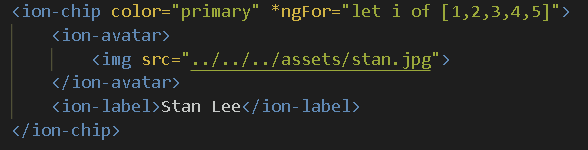
**Lista con avatars**





**Chips**





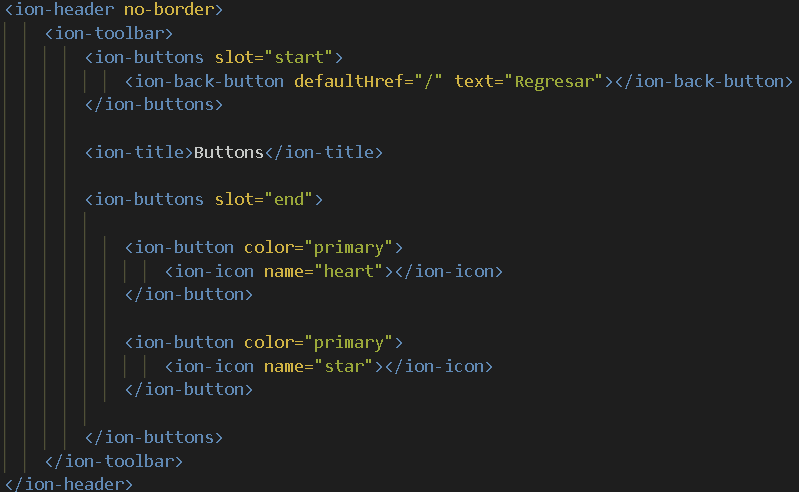
**Ion-button**

Creamos una nueva pagina para botones



La agregamos al arreglo de componentes en el home. No importamos el components module para tener el header, por que se usaran unos casos de pruebalocal con el header.

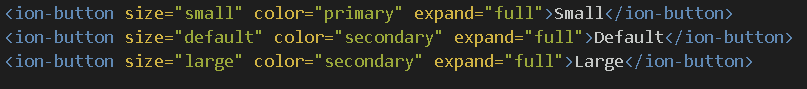
Dentro del header declaramos un grupo de botones que serán visualizados en el

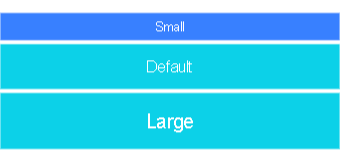




Ahora dentro del content

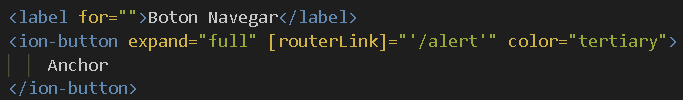
Los botones normales





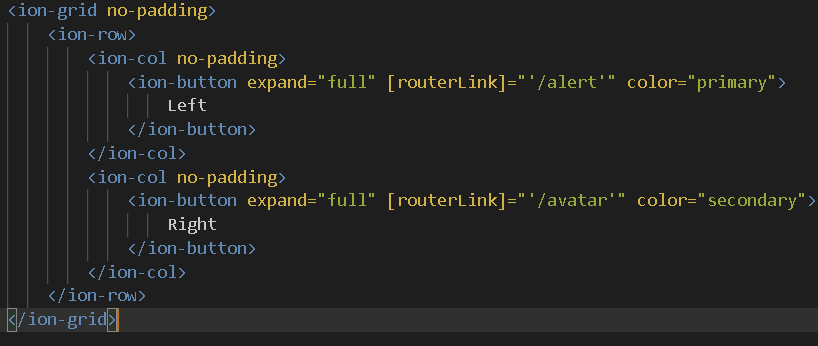
**Boton para navegar**

Utilizamos el routerlink



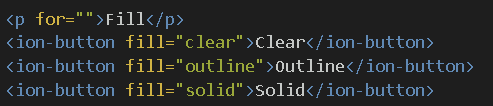


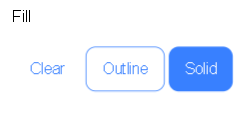
Botones en la misma línea (ROW + COL)





Fill

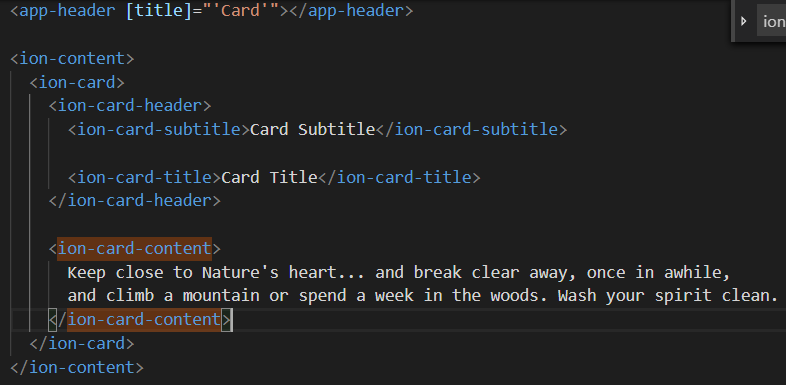


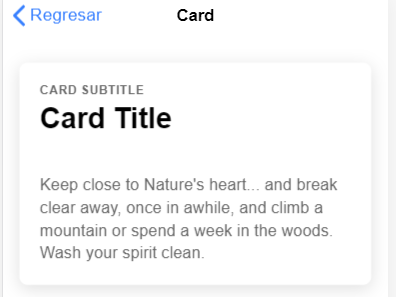


**Cards**

Creamos un nuevo componente para card  


Agregarla al arreglo de paginas de home.  
Importamos el ComponentsModule para utilizar el header y luego utilizaremos el código provisto por la documentación de ionicframework para la vista.

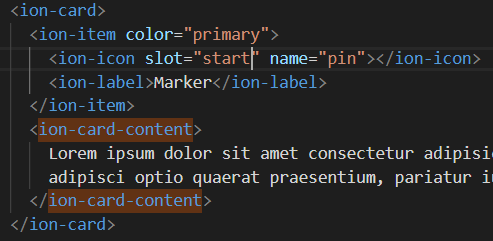


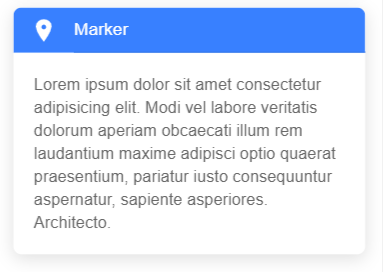


Sabemos que estos elementos se visualizan de formas diferentes en andriod y ios, podemos unificar su comportamiento con la propiedad **mode.** Pudiendo poner md para Android y ios para Ios.

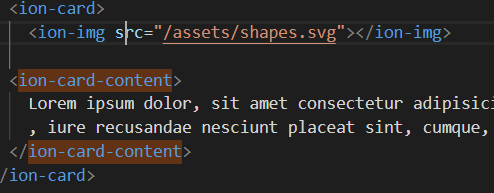


Otro ejemplo de card





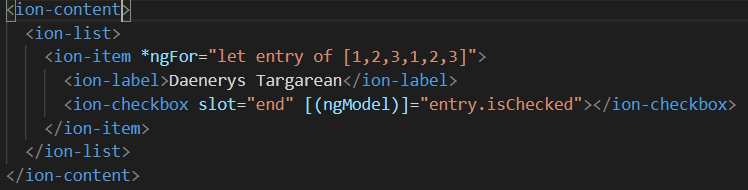
Ejemplo 3

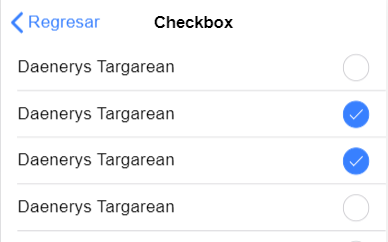




Ion-checkbox

Creando el componente y se repiten los pasos anteriores con el header y routing en home.

Usando la documentación, estos van siempre de un ion-list.  




Recuperando el valor donde se hizo click

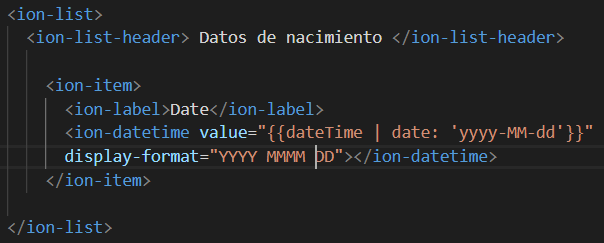




Ion Datetime

Se crea un componente para la pagina de date

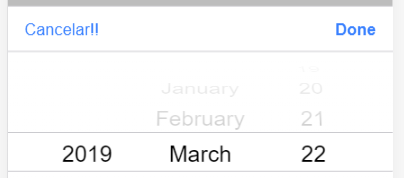
Se utiliza dentro de un ion-item





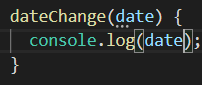
Cambiando el texto al cancelar y obteniendo el valor de la fecha seleccionada.





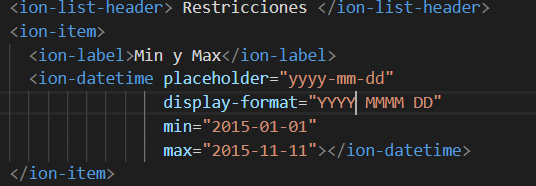
Se obtiene el valor con el ionchange y se pasa como evento como parámetro.





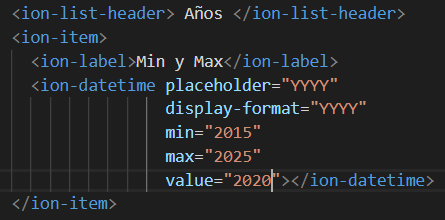
Datetime personalizado

Se personaliza para que solo se pueda seleccionar los rangos puestos en el min y max, esto no afecta a los días que tienen menos de 31 dias asi que es una validación a realizar.

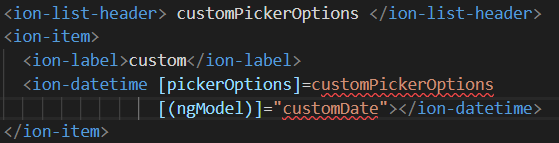


Listado por años

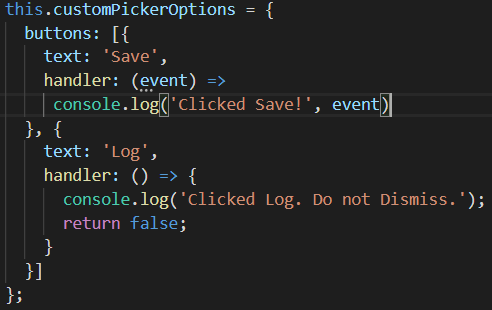
Se puede poner un rango de años y con el value se selecciona uno por defecto.



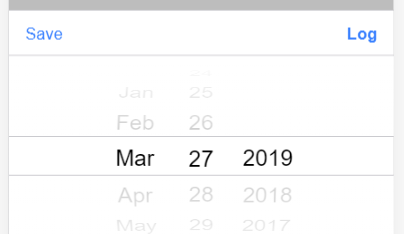
**Custom**



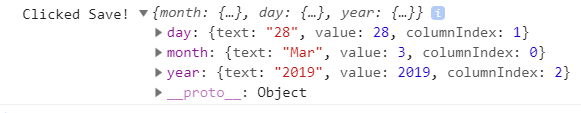
Se declaran las variables en la clase y se les asignan los valores en el ngOnInit



Al colocar el handler es la acción que se va a realizar al tocar el botón, y cuando se pone dentro un return false, el calendario no se cerrará.



Al seleccionar sabe este esta recibiendo la fecha en un objeto desglosado:



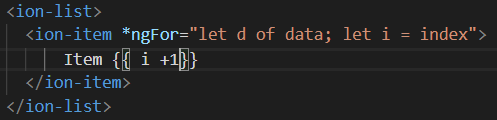
Ion-Fab

Fabs son elementos que pueden estar flotando y son seleccionables.

Se crea el componente y las configuraciones de siempre.

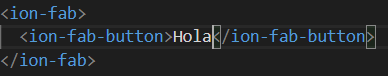
Creamos una lista de 100 elementos para simular el scroll.

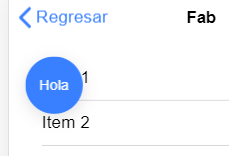




Utilizando el ion fab:

De esta forma el botón se quedara ahí aunque se haga scroll.



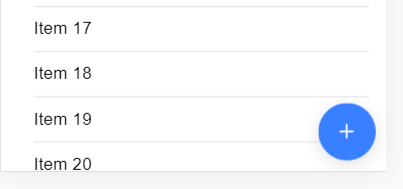


Para que se mueva con la pantalla utilizamos la propiedad slot=fixed. Dentro del ion-fab.

Ahora para cambiar la posición del fab podemos utilizar vertical y horizontal.

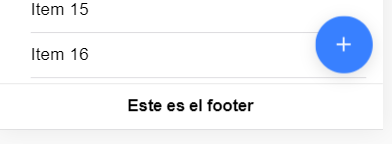
Con esta configuración, nos aparece en la parte de abajo, lado derecho.





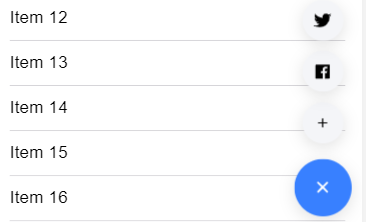
Si agregamos un footer

Si utilizamos un footer, ionic automáticamente posiciona el fab arriba de el footer.

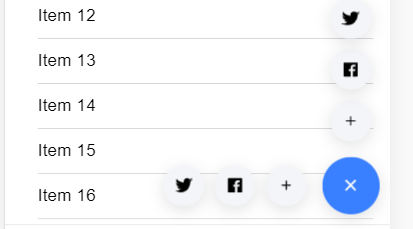


Para agregar submenús al fab se agregan dentro del ion-fab y se modifica la posición con la propiedad side:





Podemos tener hacia arroba y hacia la izquierda a la vez.

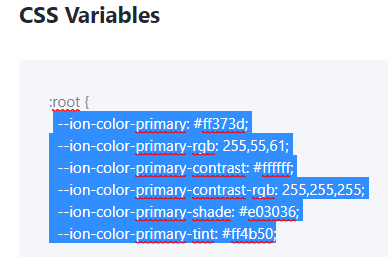


Creando variables de color

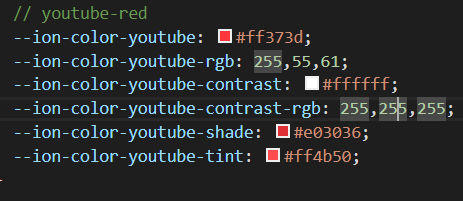
En el archivo variables.scss la carpeta theme se deben colocar las configuraciones.

Ionic ofrece en su pagina un generador de colores.

<https://ionicframework.com/docs/theming/color-generator/>



Con el código generado en la pagina podemos ponerlo en el variables y cambiarle el nombre al que deseemos usar.



Ahora en el item que desamos que utilice este color se lo agregamos  


Ahora debajo del root de variables hay que pegar el código:

.ion-color-youtube {  
 --ion-color-base: var(--ion-color-youtube) !important;  
 --ion-color-base-rgb: var(--ion-color-youtube-rgb) !important;

--ion-color-contrast: var(--ion-color-youtube-contrast) !important;

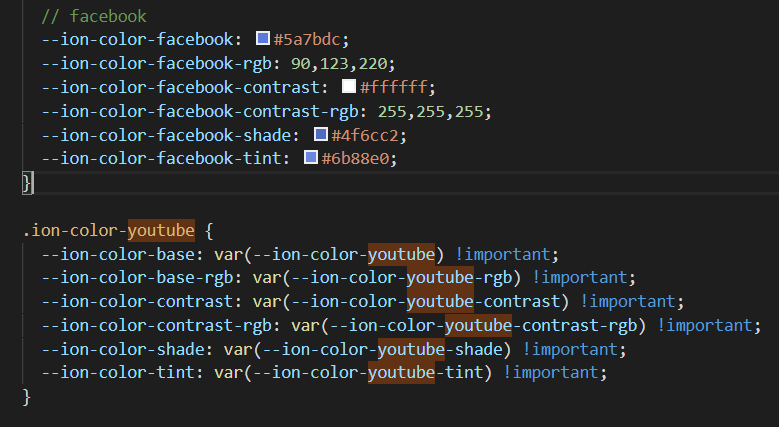
--ion-color-contrast-rgb: var(--ion-color-youtube-contrast-rgb) !important;

--ion-color-shade: var(--ion-color-youtube-shade) !important;

--ion-color-tint: var(--ion-color-youtube-tint) !important;

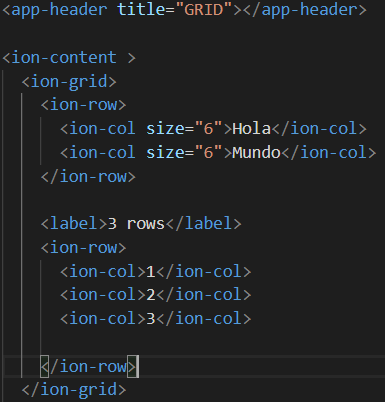
}

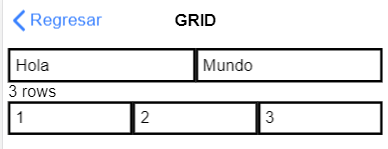




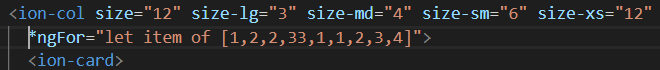
**Ion-grid**

Creando el componente de grid y agregándole el header como conocemos.





Podemos regular el tamaño del col utilizando la propiedad size





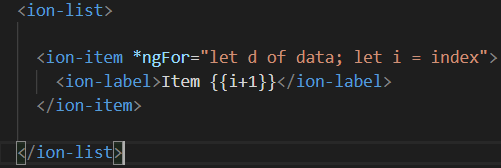
Infinite Scroll

Generamos la nueva pagina para infinite scroll.

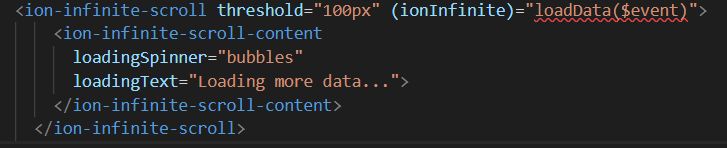
Creamos un arreglo genérico en el componente.ts



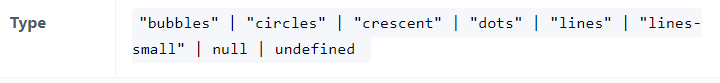
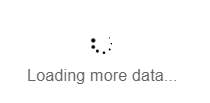
Y en el html iteramos el item con un for, dentro de un ion-list



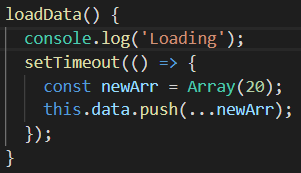
Luego copiamos el código provisto por ionic



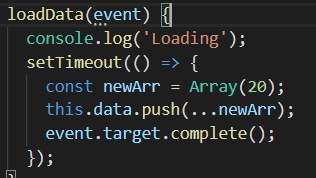
La propiedad teshold es la que dice que tan cerca del final es que empieza a cargar el infinite.  
Luego podemos personalizar el tipo de spinner con la propiedad loadingSpinner:

  
Y el loading text, el texto que aparece cuando esta cargando:  


Ahora dentro del método loadData vamos a simular una carga asíncrona de datos hacia el arreglo para que pueda cargar luego del infinite scroll.

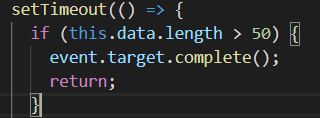


Si lo dejamos asi, el scroll no va a terminar de cargar aunque ya se muestre la información, es necesario poner que se ha completado luego de cargar los datos.

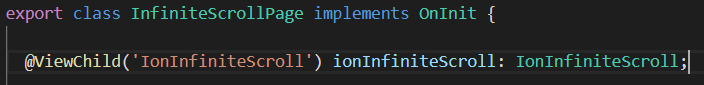


En el momento en que se acaban los registros a mostrar se debe para el infinite scroll.

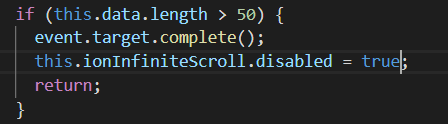
Dentro del setTimeot se valida el tamaño de los registros a mostrar, y hacemos complete el infinite scroll y retornamos.



Aun con esto el infinite scroll segira apareciendo al final de la pantallay dejara el espacioaunqe no cargara datos, es necesario acceder a la propiedad del html para poder detener este comportamiento.  
Desde el component.ts declaramos un ViewChild

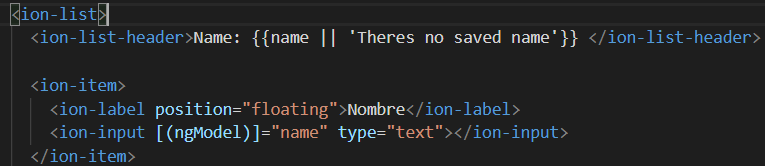


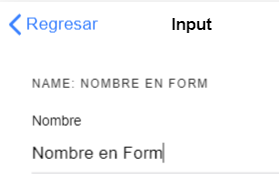
Y en la función accedemos a la propiedad disabled del infinite scroll declarado.



Ion Input

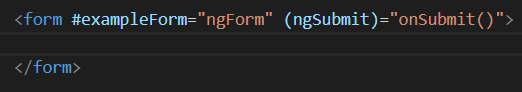
Creamos el componente

Dentro del html se crea una lista, con item dentro y luego el label con su input.  
Se puede bindear el valor del input con una variable declarada en el componente.:  


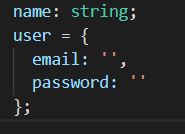


**Form**

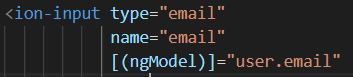
Se crea el formulario y se asocia a un ngForm con el #, luego se declara la acción que realizara al hacer submit.



Creamos un objeto en la clase el cual recibirá los datos del input

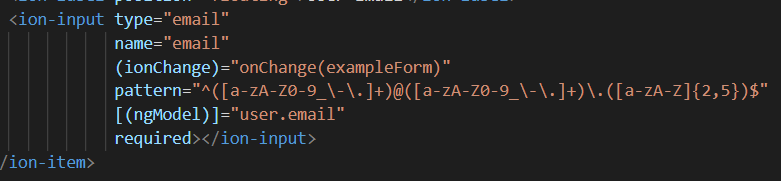


En el input declaramos el nombre, y bindeamos al objeto user creado en el componente.  
Es importante la propieda name.

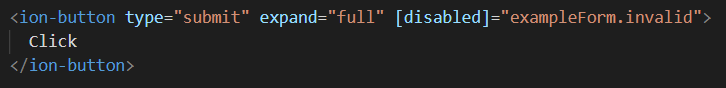


Al ser un email hay que validar su estructura con una expresión regular.  
Se puede poner como un pattern directamente en el html en el mismo input.

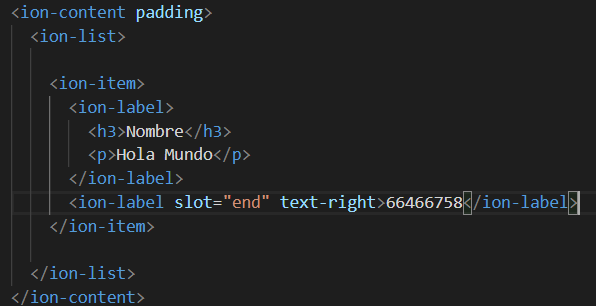


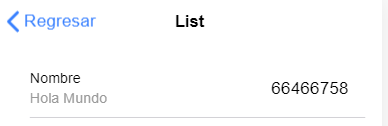


Y el botón también puede usar las validaciones del formulario



**Ion List**

Se crea la pagina de lista y se agrega al home.  
En el html creamos una lista normal como se han creado hasta el momento  




Ahora para mockear los valores, vamos a pegarnos a un servicio que provee la pagina jsonplaceholder.

<https://jsonplaceholder.typicode.com/users>

Para interactuar con el servicio es necesario crear un servicio en nuestro proyecto

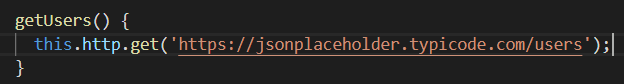


Ya que es necesario hacer peticiones http, Debemos importar en el app.module el HttpClientModule. (Talvez no aparezca en los imports automaticos pero esta es la ruta). Y se agrega a los imports.

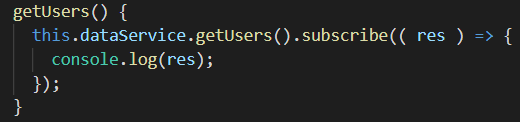


Y en el servicio importamos el HttpClient  


Y creamos la función que retorna un observable

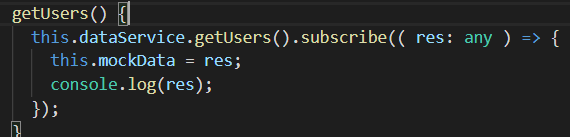


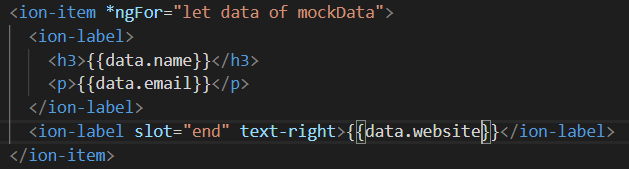
Ahora ya en nuestro componente podemos utilizar este servicio, primero importándolo y luego añadiéndolo en un método.



Esta información se debe almacenar en una variable del componente para luego mostrarla en la vista.

Esta es una forma suscribiéndose a la respuesta desde el componente:



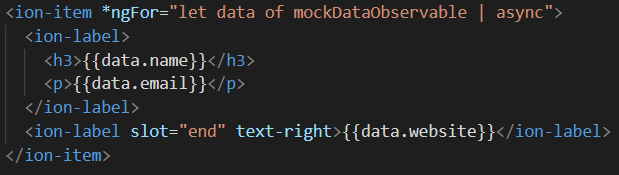


La otra forma desde la función solo asignar el retorno del observable a una variable del mismo tipo ya declarada.



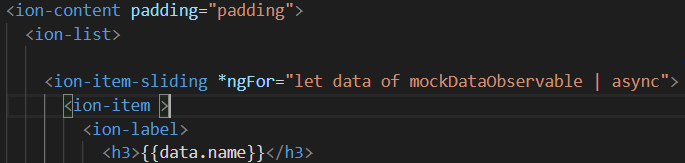


Y en la vista en el fo, pasar la data por el pipe “async”

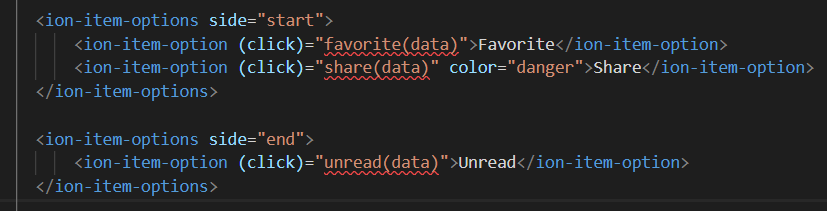


Añadiendo Sliding

Encerramos nuestro código de html en esta estructura, pasandole el comportamiento del ngFor.

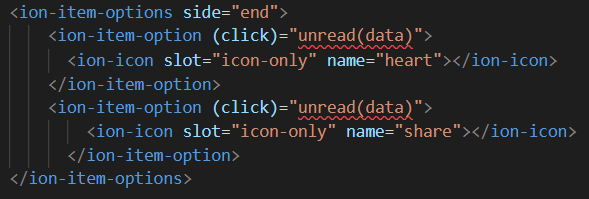


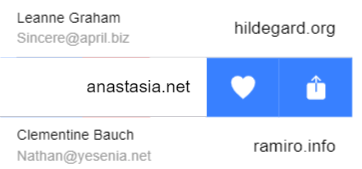
Y después del item antes de cerrar el item sliding pegamos las opciones que tendrá



Pueden haber varias opciones personalizándole la posición con el slot en el option.

Añadiendole la propiedad slot=”icon-only” se hace el icono mas grande y adaptado.



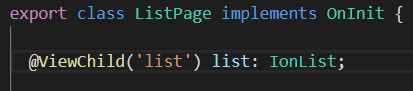


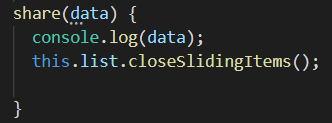
Al tocar un elemento se debe cerrar el slide.

Se necesita hacer una referencia al ion list, que permite cerrar lo que tenga abierto el item sliding.  
Se hace la referencia con “#”



Luego en el componente se escucha la referencia declarando el viewchild

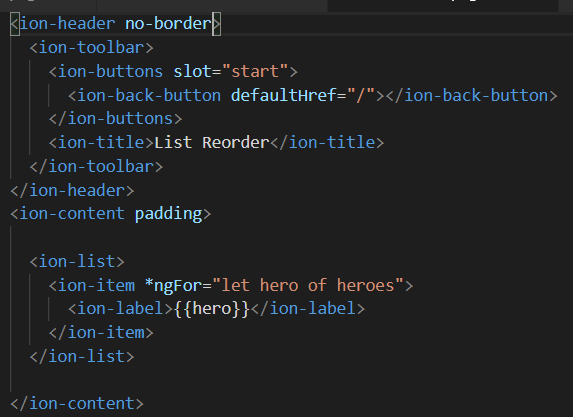


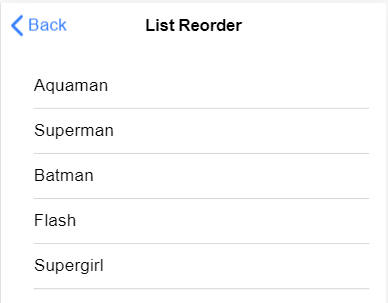


Ion reorder group

Creamos el componente, y en el ts hacemos un arreglo mockeado de superhéroes.

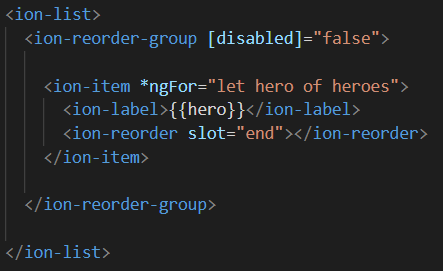






Para implementar el reorder-group, se crea el tag en el html, dentro del ion-list. Y dentro de este el item que se va a repetir y que puede ser reordenado. Este ion-reorder-group debe tener una directiva que le diga si esta habilitado o no.

Dentro del ion item se pone el ion-reorder, el cual tendrá el botón que permite hacer el reordenamiento.



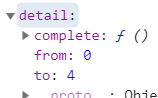
En este momento ya se pueden mover los elementos en la vista, pero al momento de soltarlos ocurre un error. Esto por que hay que llamar a un evento. Esto con la función ionItemReorder



Y en el component.ts escucharmos la función.

Si se imprime el resultado del event vemos una propiedad importante:

Detail, el cual tiene como información, la posición de la que venia, a la que llegó y el estado, completo:



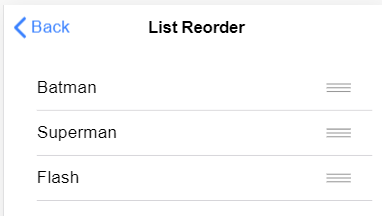
Para que no se quede colgada la lista hay que invocarla función complete del evento, luego de ser llamada el reorder group es ordenado según se encuentra en el arreglo. Por esto en este momento regresa a su posición inicial.

Luego extraemos el elemento a mover, con la función splice, que remueve n elementos del arreglo según los parámetros, y si se especifica la posición retorna el elemento a devolver en caso de ser mas de uno.  


Luego con la misma función podemos añadir un elemento al arreglo, especificándole en que posición de este queremos trabajar, seguido por los elementos a eliminar, en este caso ninguno, y luego el elemento a añadir.



Ya podemos reordenar desde la vista el arreglo.



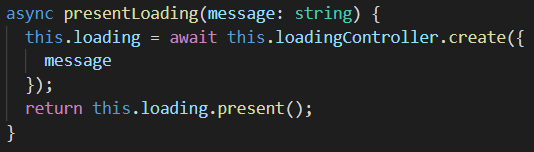
Ion-loading

Se crea el componente.

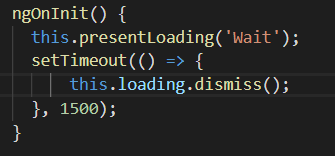
Se carga el loading controller dentro del constructor  


Dentro se crea una función asíncrona que mostrara el loading.

Se crea una variable loading tipo any que pueda almacenar el resultado de la funcion

}

Y ahora simulando la respuesta de un servicio, este loading se puede cerrar al momento de la respuesta

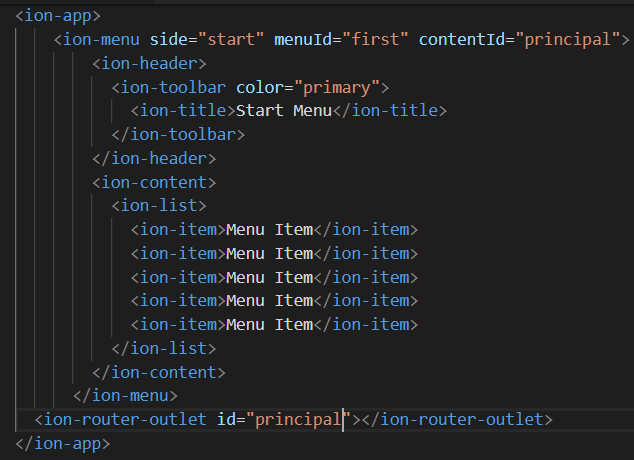


Ion menú

Para crear el menú desde el app.component.html.   
El menú esta sobre todas las paginas, por eso debe ir en esta parte de la aplicación

Pueden haber varios menú, estos se identifican con el menuId, pudiendo asi habilitar uno u otro.

Tambien es importante la propiedad contentId, el cual le dirá a la aplicación, la posición de este menú y a que otro componente va a mover al desplazarse por la pantalla.  
En este ejemplo, el content id hace referencia al id del routerOutlet que tiene el id “principal”.



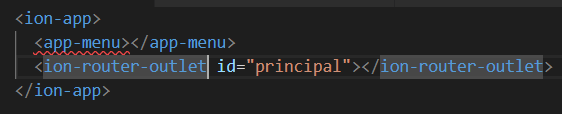
Importante la propiedad enable el cual permite habilitar el menú:  


Este código se puede componentizar para asi utilizarlo y solo utilizar su tag en el app.html y pasarle los parámetros necesarios.

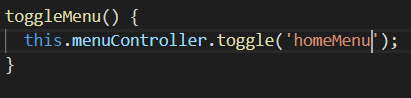
Se crea el componente menú dentro de components  

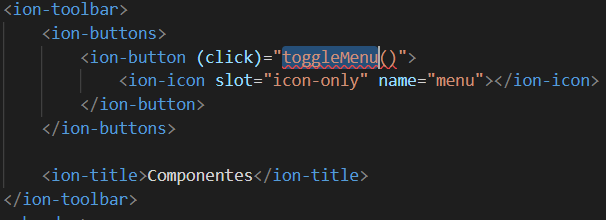

Luego pegamos el código del html del menú dentro.  
En el modulode componentes ponemos el MenuComponent dentro de declarations y export. Paras asi poder utilizarlo en otros componentes.

Y finalmente en el app.html ponemos su tag, app-menu.

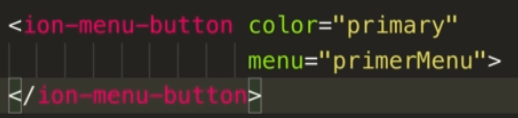


Ahora añadiendo un icono de hamburguesa para poder tocarlo y se abra el menú, desde el [home.component.html](http://home.component.html)



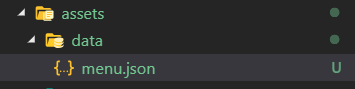


O simplemente en el home.html



Opciones del menú

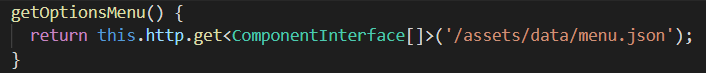
Crearemos un archivo json dentro de assets y una carpeta data, el cual tendrá el objeto del arreglo que tenemos la configuración de las rutas y los iconos.



Y cambiamos la estructura del objeto a comillas dobles:

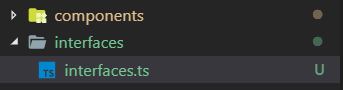


Ahora en el servicio se crea una función que devuelva esta configuración, especificando el tipo de dato que va a devolver esta peticion. En este caso un arreglo de la interfaz componente.

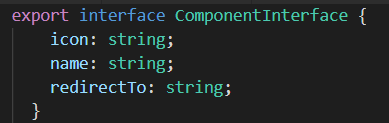


Ahora hay que optimizar las interfaces utilizadas,

Creamos una carpeta a nivel de services y components, llamada interfaces



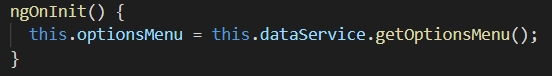
Y dentro pegamos la interfaz que teníamos definida para Component en home.



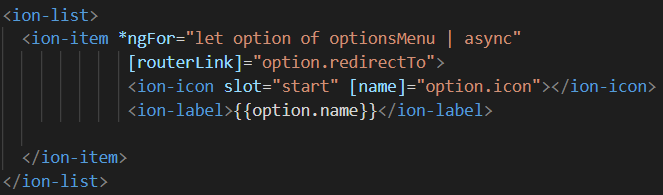
Ahora dentro del menú component improtamos la interfaz y declaramos una variable de este tipo.

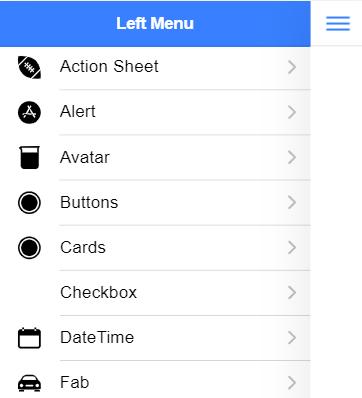
La cual va a recibir la respuesta del servicio del arreglo de opciones del menú. Y le asignamos el valor de la respuesta del servicio.



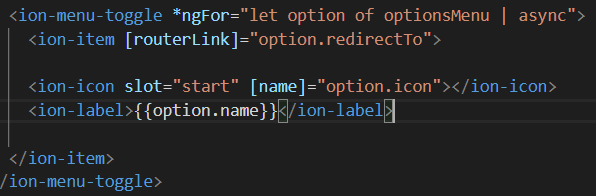


Luego debemos presentarlo en la lista del html

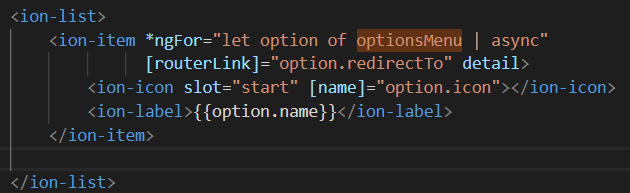




Para que el menú se cierre al momento de seleccionar una opción, es necesario incluir el tag ion-menu-toggle y dentro poner el ion-item. Importante el elemento a iterar va a ser el toggle.



Luego en la pagina de home repetimos la creación de la variable que alojara el arreglo de componentes y en la vista se muestran.



Modal

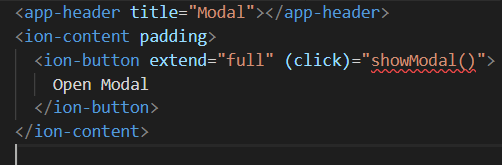
Es necesario generar 2 paginas, 1 la cual va a ser la pantalla que desplegará el modal y la otra el modal propiamente.



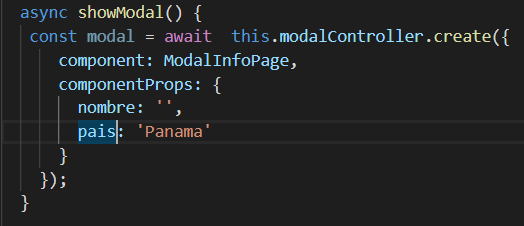


Eliminamos el modal-info del routing y se añade el objeto modal al arreglo de componentes para poder acceder a el.

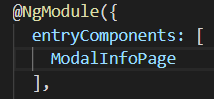
Agregamos el header al modal.

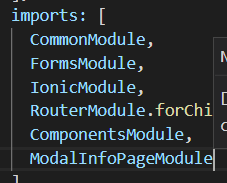


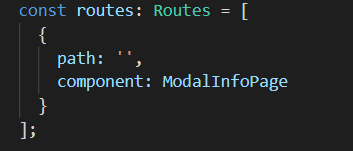
Y En el ts creamos la función que abrirá el modal.  
Esta función es asíncrona y dentro se declara una variable modal que va a escuchar una promesa.  
Se declara el modal controller que cree la modal con la siguiente estructura que va a permitir llamar al modal y pasarle propiedades:



Aun no se puede lanzar el ModalInfo ya que hay que agregarlo al modal.module.ts en **entry** **Components** para que no lance un error.

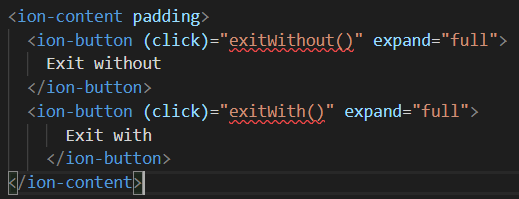


E importar el modalInfoPageModule, en imports.  


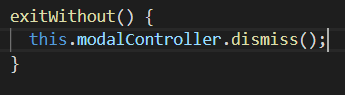
Al probar la aplicación al seleccionar modal, nos abre directamente el modal info page, esto por que en el modal info module existe una configuración de ruta creada.  
Eliminamos este arreglo. Y el forchild.  


Ahora desde el modal-info-page creamos un botón para poder cerrar el modal.

Al cerrar el modal podemos cerrarlo pasando argumentos, algún valor que hayamos obtenido desde el modal.

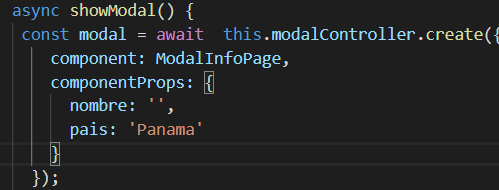
}

Sin argumentos

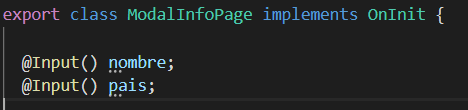


Enviandole argumentos desde el padre.

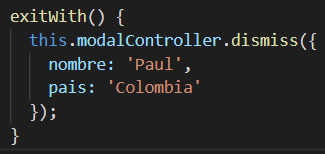
Recordando que se enviaron estos:



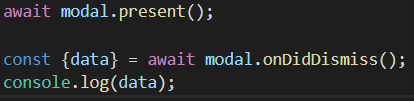
Desde el modal lo recibimos, siendo el mismo nombre con que declaramos la llave en el padre.



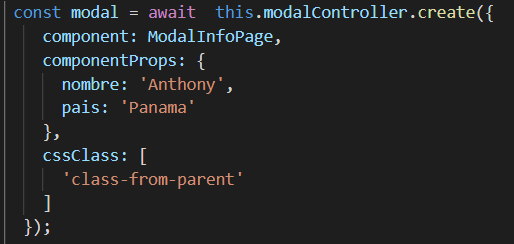
Y para salir con argumentos:

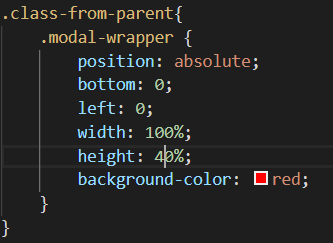


Y para recuperarlo en el padre:



Para modificar la altura del modal, es necesario hacerlo desde el app.scss o global.scss desde la clase asignada al declarar el modal y entrando a la propiedad modal-wrapper:





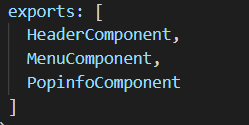
Ion pop over

Se crea el la pagina de popover  

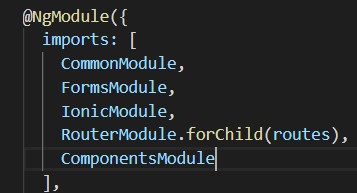

Y luego el **componente** que va a tener el contenido del popover.



Añadimos el popinfo component a los exports del module de componentes donde fue añadido:

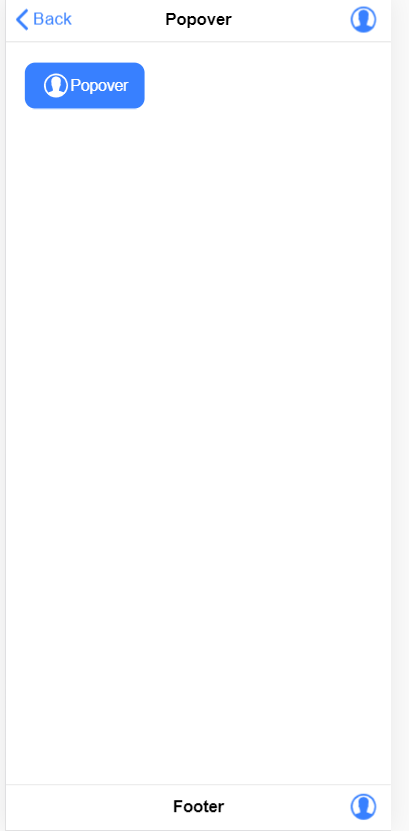


En el modulo de pages pop over añadimos el modulo de componentsModule donde fue añadido el PopoverComponent. El cual también comparte el header.



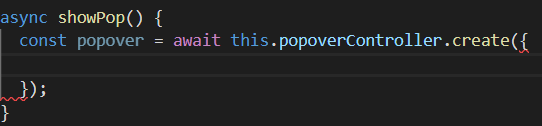
Y en el html del PopOver page



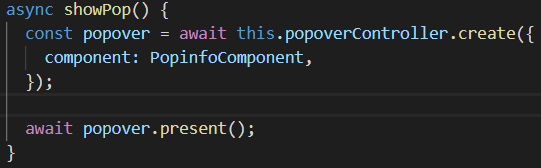


Agregamos el nombre de la función donde queremos llamarla en el html y en el componente desde el constructor importamos el controlador del popover y creamos la función.  

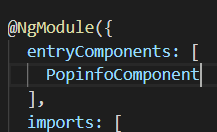

Declarando una variable para el créate del pop over la cual es declarada await, y para esto la función debe ser async:



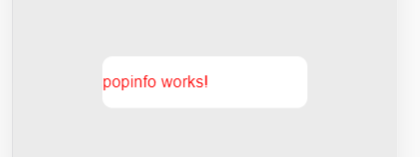
Dentro se declaran los parámetros, el principal es el component que es el que creamos el cual tiene la estructura del popover y hacemos el present.



Recordar agregar el PopinfoComponent dentro del modulo de popover page, en la parte de entryComponents. Esto por ser un elemento que se va a presentar dinámicamente en la vista.



Ahora nuestro pop over se muestra en la mitad de la pantalla con el contenido que se creó.

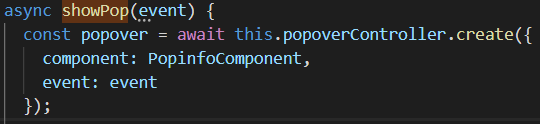


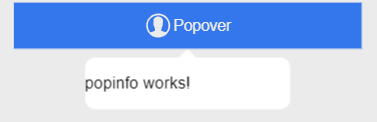
Configurando la posición del PopOver y añadiendo contenido.

Para arreglar el posicionamiento, es necesario pasar como parámetro el elemento donde se llamó.

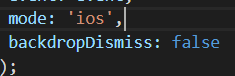


Y en el component.ts

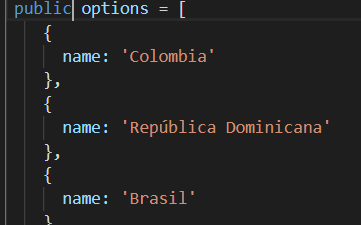
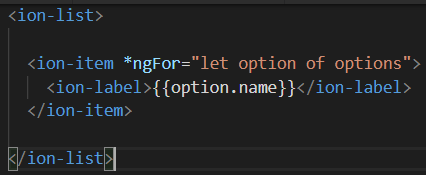


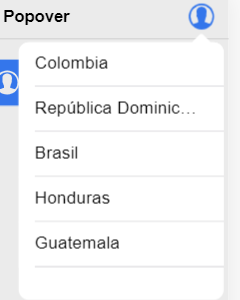


En este momento en ios se ve bien, pero en Android no aparece exactamente en el lugar ni con la flecha. Para homologar el comportamiento se agrega la propiedad mode. Adicionalmente si queremos bloquear otras acciones que no sean del popover esta la propiedad **backdropDismiss: false.**

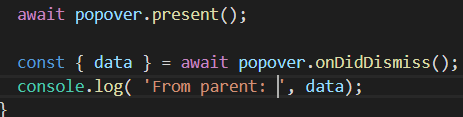


Ahora trabajamos dentro del componente del popinfo para mostrar opciones.

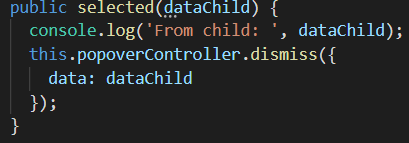


Para obtener datos desde el componente padre que llamo al pop over se utiliza una constante que escuche con un await el cierre del popover.



Puede ser onWillDismiss para obtener el dato antes.

Y desde el componente hijo se lo enviamos al hacer dismiss del mismo.



Progress bar

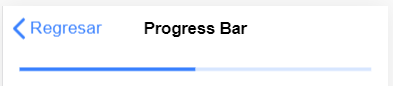
**Importante recordar que este valor va desde 0 a 1.**

Primero creamos una página.



Y en el html del progress bar  

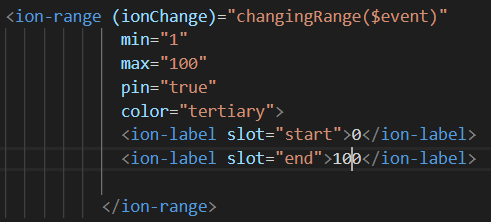

Lo cual se verá:

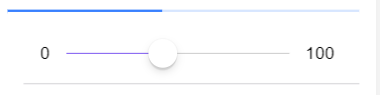


Ahora manejando esta barra con una variable desde el controlador:

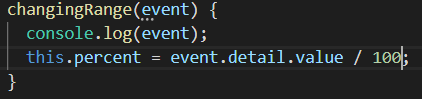


  
Este elemento se puede combinar con el ion-range





Y en el .ts

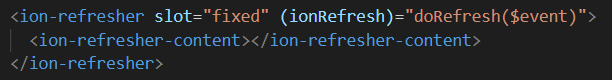


Ion Refresher

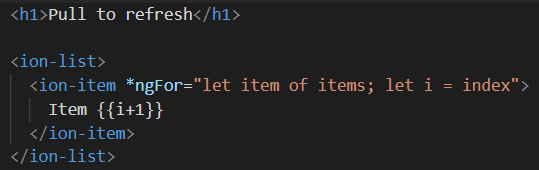


Y ahora agregarlo al objeto menú y luego en el modulo agregar el components module para poder usar el header.

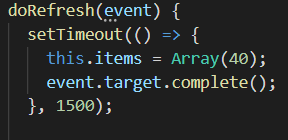
Se implementa el código que provee la documentación



Y se crea una lista de elementos de un arreglo que cargara luego de hacer el pull



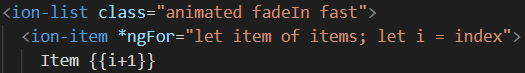
Simulando que el servicio carga en segundo y medio, se llena el arreglo con 40 elementos y se activa del evento recibido la función complete para que termine de cargar en la vista.



Para añadirle animación al cargar los datos podemos utilizar el anímate.css

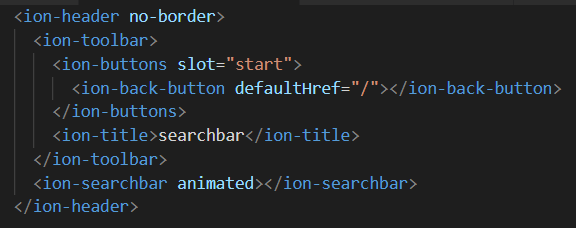
De ahí copiamos las clases .animated, .animated.fast y el fade in con sus keyframes

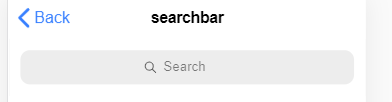
Y se pega en el global.scss y ya podemos utilizar la clase.



Searchbar

Creamos la pagina y utilizamos el searchbar dentro del header de la pagina.





El searchbar tiene el evento que escucha cuando cambia el texto dentro, que acciona una función.

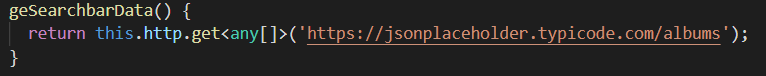


Ojo si se le aplica la propiedad debounce espera los segundos que se le parametricen, esto sirve si el texto a comparar es contra un servicio para poder hacerlo mas fluido.

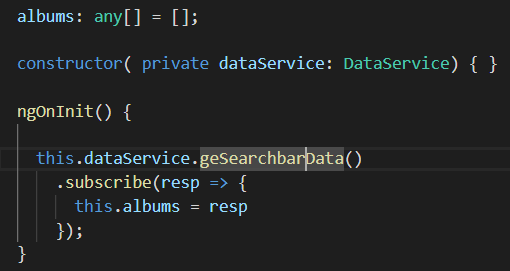
Para llenar el componente, se cargaran datos de un json que se pedirá por servicio.

<https://jsonplaceholder.typicode.com/albums>

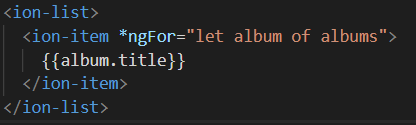
Se crea la función en la carpeta services identificando la respuesta como un arreglo de tipo any.

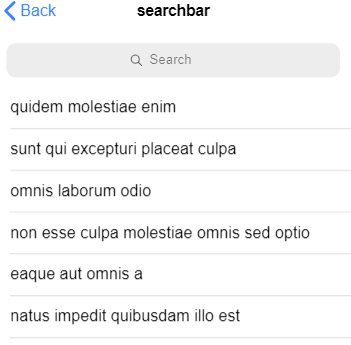


Y en el componente.:



Para ver esta respuesta en la vista





Para filtrar los elementos según la búsqueda se aplica un **PIPE**

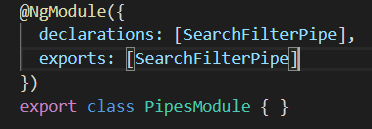
Para esto creamos un modulo dedicado a los pipes



Y ya creado el modulo de pipes creamos el pipe para el filtro

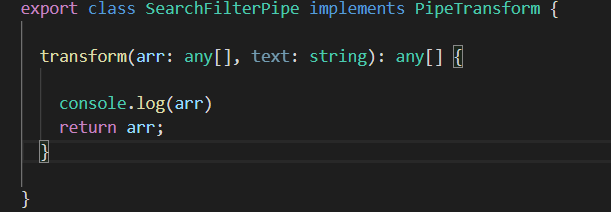


Como se manejarán los **pipes** por **modulo**, es necesario **añadirlo** a el **export** del modulo.



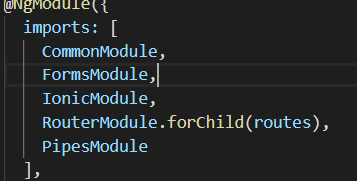
En el app.module es necesario declarar el nuevo modulo de pipes en los imports.

En el código del mismo pipe filter, creamos la estructura básica del pipe, que recibe un arreglo y una cadena la cual va a comparar la existencia en el arreglo y devolver el arreglo filtrado.

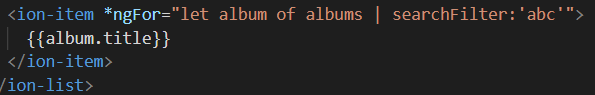


Probando esto en el componente searchbar

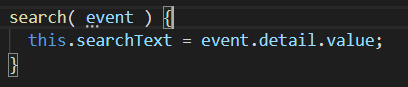
En el modulo de searchbar importamos el pipes module que contiene los pipes.



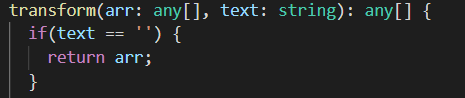
Y en el html por defecto siempre recibirá el elemento al que se le aplica el pipe, y se pueden mandar otros parametros mediante “:”.



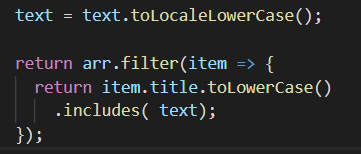
Ahora podemos pasarle como parámetro de filtro real, creando una variable en el componente searchbar y asignándole el valor que obtiene de el input de búsqueda.



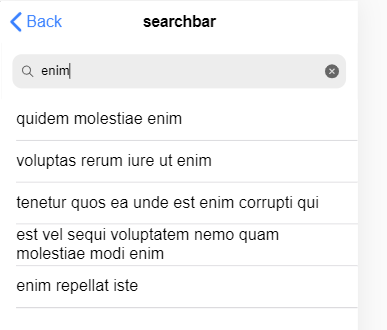
Dentro del pipe si el texto que se recibe es vacio se debe retornar el arreglo completo



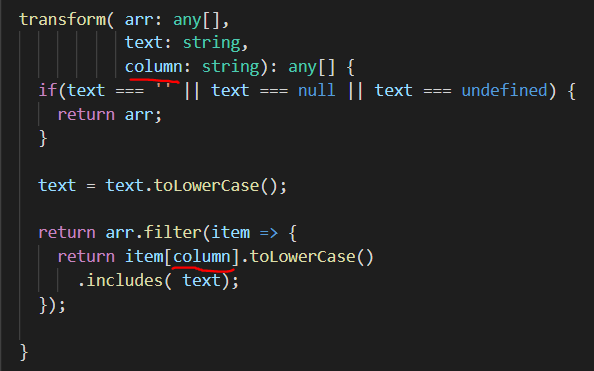
Añadimos la validación de filtrar el arreglo que retorne el filtro de cada elemento que contenga el texto que se está comparando.



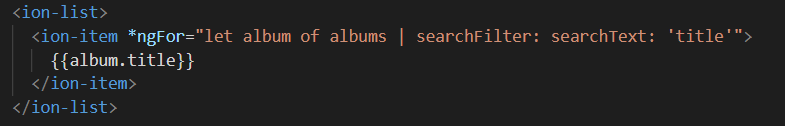
Y ahora el filtro funciona correctamente.



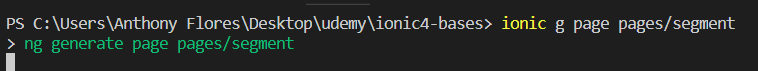
Podemos parametrizar más el pipe para que reciba otro parámetro y pueda filtrar por la propiedad que deseemos y este estructurado el objeto que estamos recibiendo dentro del arreglo.



Y mandamos un nuevo parámetro desde el html hacia el pipe

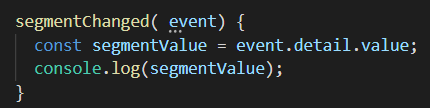


**Ion Segment**

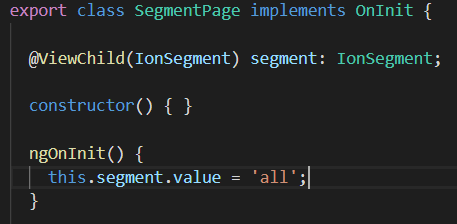




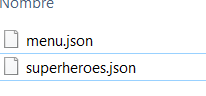




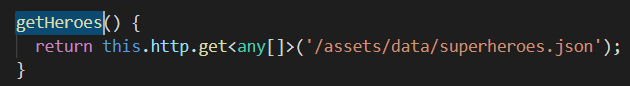
En este momento al cargar la pagina no viene ningún segmento seleccionado, para poder utilizar uno por defecto, lo referenciaremos desde el componente, con el ViewChild



Utilizando el json provisto en el curso, se copia en el assets, data. Es un arreglo de objetos de superhéroes.

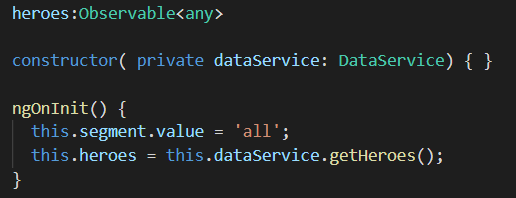


Ahora creamos un servicio que va a retornar este arreglo de objetos.

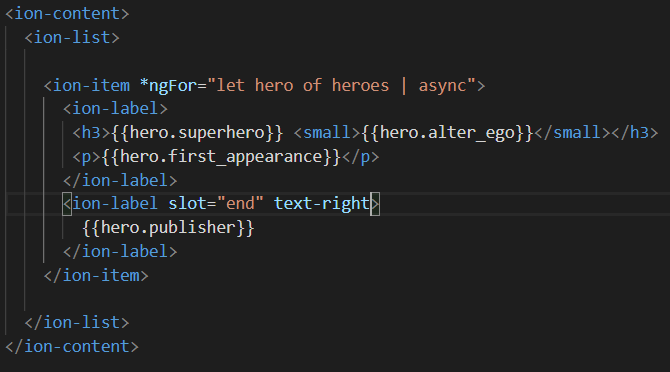


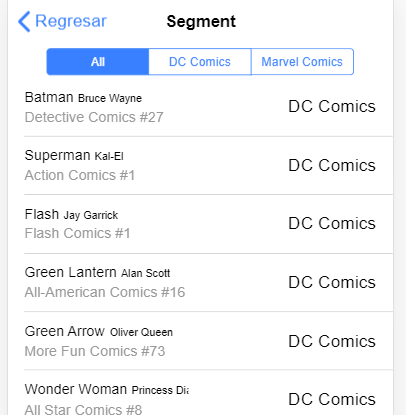
E importamos el servicio en la pagina de segment.

Creamos una variable observable que almacene el llamado al servicio para luego mostrarla en la vista



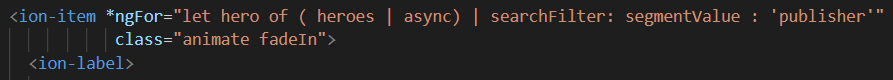
Importante añadir el pipe de async al ser un observable que estará disponible no necesariamente al mostrar la vista.





Ahora es necesario filtrar en este caso por la casa que lo publicó, utilizando el pipe creado anteriormente.

En el html, enviamos como parámetro, el segmento donde estamos parados como valor a filtrar en el arreglo, y como columna la propiedad que conocemos del objeto donde se encuentra la casa que publica al héroe.



Dentro del componente hacemos global el value del segmento para poder utilizarlo en el html

