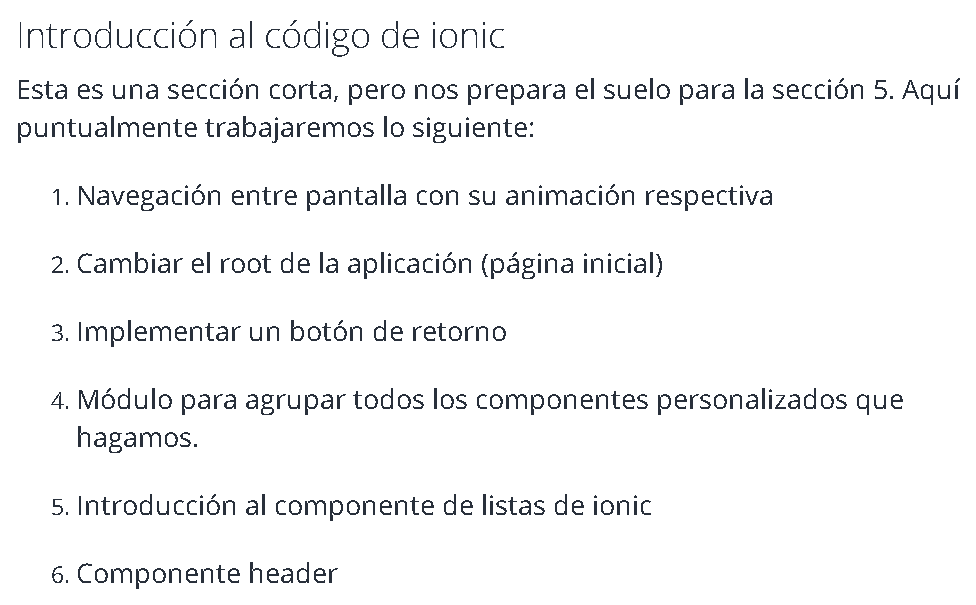
**Bases Ionic**

**Capitulo 1**



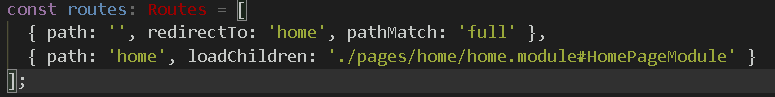
Para crear una aplicación



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.

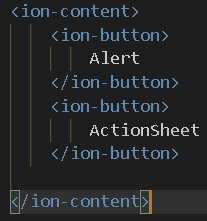


**Navegación entre paginas**

Creando paginas:

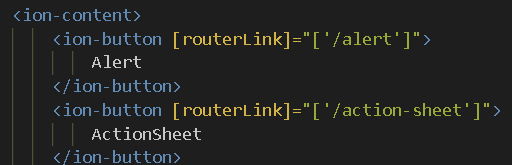
Se crean dentro de pages : alert y action sheet  


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



Para añadirle la navegación

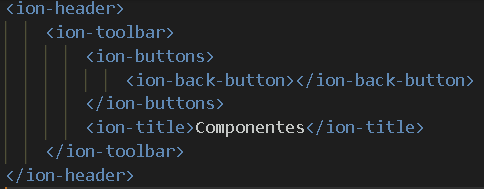
Se puede manejar de igual forma como en angular con el routerlink y el nombre ya agregado a la lista del modulo de routing.



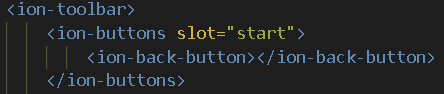
**BackButton**

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**

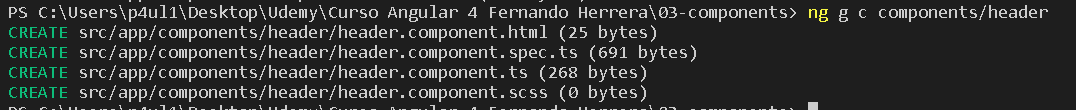
Dentro de app se crea una carpeta de components y dentro un modulo que va a manejar todos los componentes de la aplicación. Esto para no duplicar código por ejemplo con el header y solo hacer el llamado a este con su tag y personalizando un titulo.

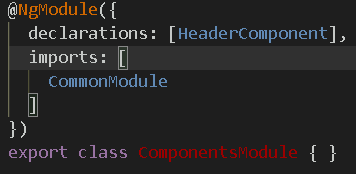


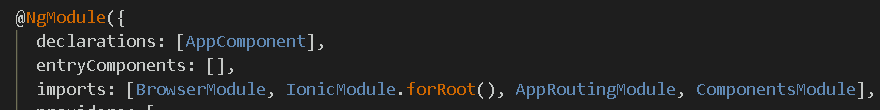
Para luego crear el componente del header



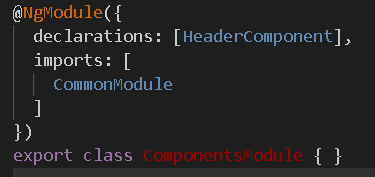
Verificar si no se modificó el module de componentes, ya que este componente debe estar declarado ahí para poder ser utilizado. Si no está, agregarlo manualmente.





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  

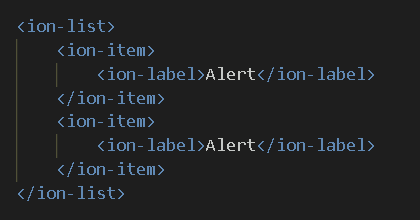

También se le puede añadir la propiedad al header de no border para que no tenga borde de sombra.





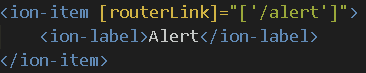
**ION-LIST**

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”.



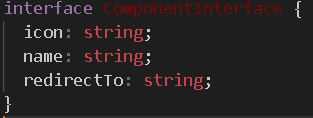




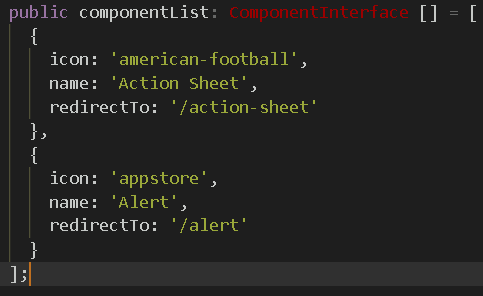
**Automatizando la lista**

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)

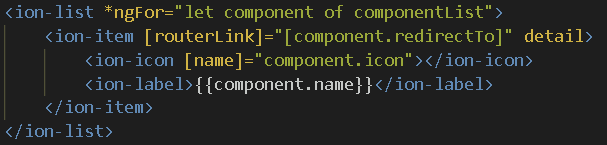
Para esto creamos una interfaz dentro del [home.page.ts](http://home.page.ts) que va a ser de tipo componente.

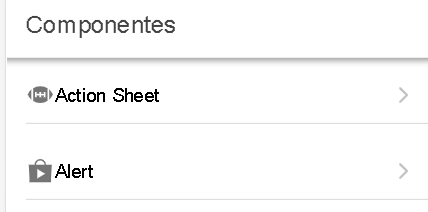


Y dentro de la clase del home creamos el arreglo de este tipo con la informacion de los componentes creados.



Ahora dentro del html podemos iterar en base a este arreglo



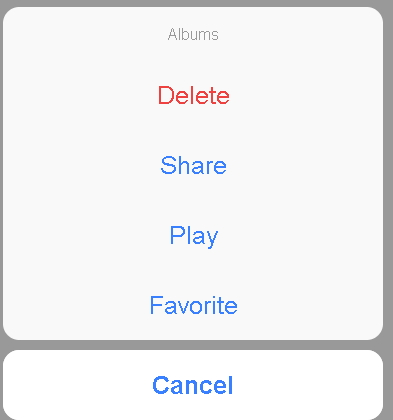


Arreglando los iconos, podemos definir su posición con la propiedad que usamos anteriormente “slot”.



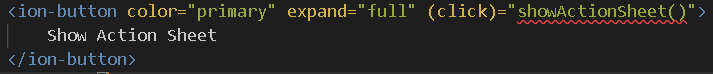


**Ion-action-sheet**



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

La propiedad expand se refiere al tamaño del botón.





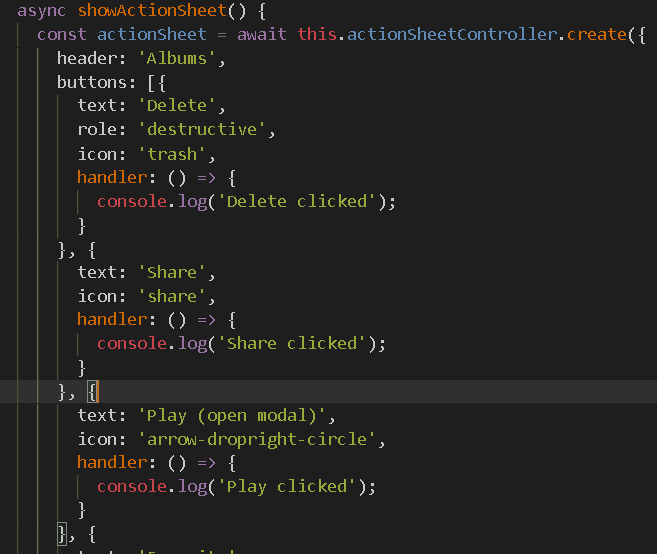
Creamos la función en el componente, esta empezará con el tipado **async** ya que es una función que regresa una promesa:



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.



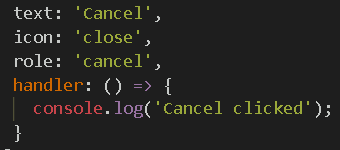
Añadiéndole luego el contenido del action sheet:



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.

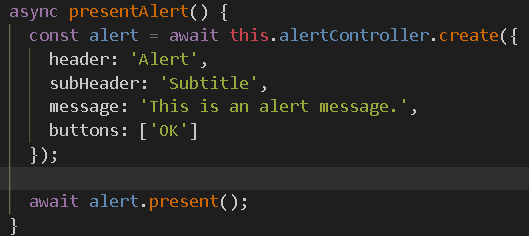


Se puede forzar también a que no se pueda tocar fuera del action-sheet para cerrarlo, o sea que es obligatorio seleccionar uno. Esto, agregándole la propiedad backDropDismiss: false.



Alert

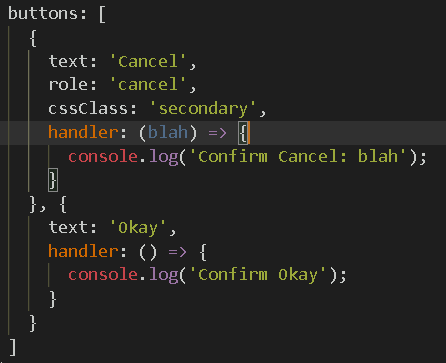
Importando el AlertController en el constructor, y luego creamos la función que llamara al alert

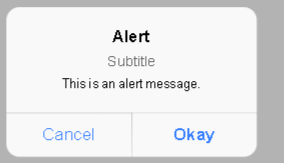


Esta función se llama desde el html creado para el alert

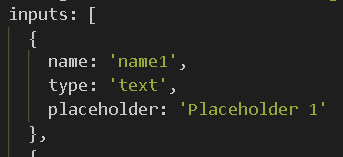


Añadiendole el arreglo de botones con sus acciones

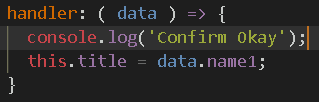




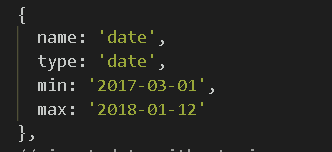
Podemos también tener inputs dentro del alert, para esto se declara un arreglo de inputs a los cuales para acceder a sus valores se podrá mediante el nombre que se le asigne.



Dentro del botón de confirmar:



O con fechas



**Avatars**

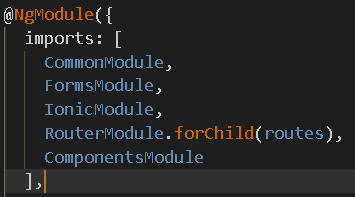
Creamos la pagina para avatar



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.



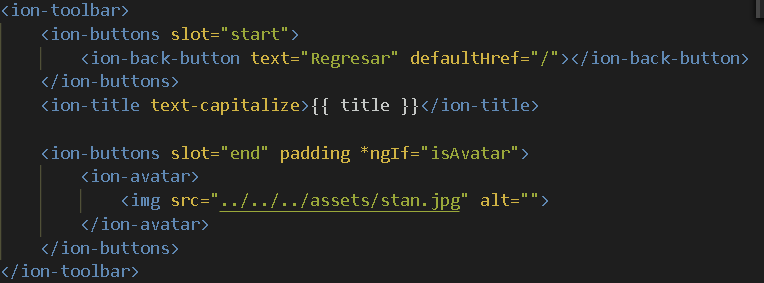
Recordando añadir al modulo de avatar el Components module, el cual tiene declarado el HeaderComponent, para poder utilizar el app-header en el html.



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:



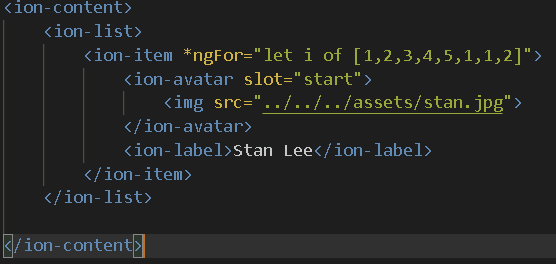
Con el código del avatar dependiendo de la variable que lo activa, y esta esta declarada como un @Input

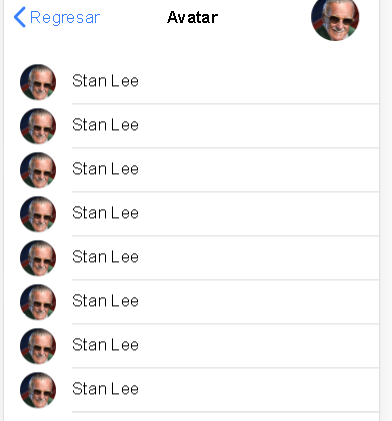


Luego en el componente que lo este llamando se le pasa el valor, en este caso desde el avatar component.



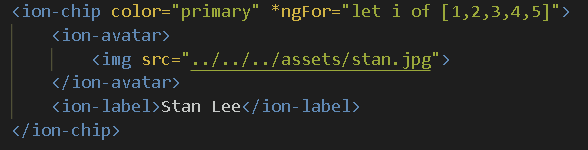
**Lista con avatars**





**Chips**





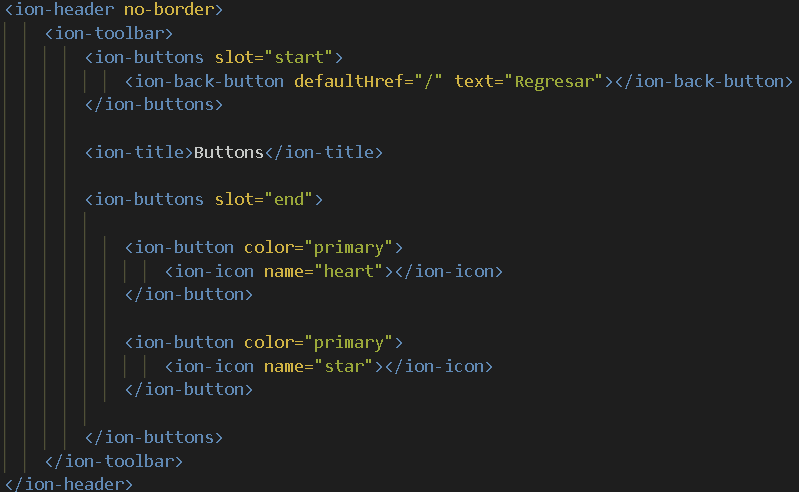
**Ion-button**

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.

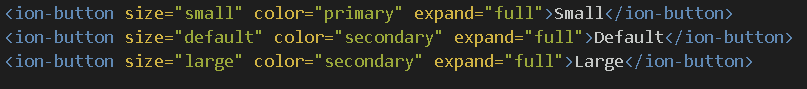
Dentro del header declaramos un grupo de botones que serán visualizados en el

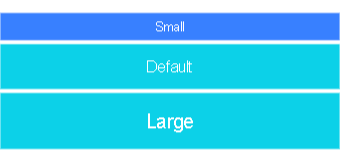




Ahora dentro del content

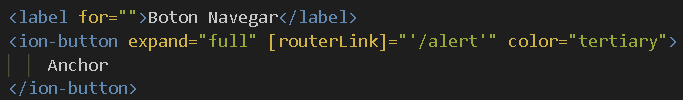
Los botones normales





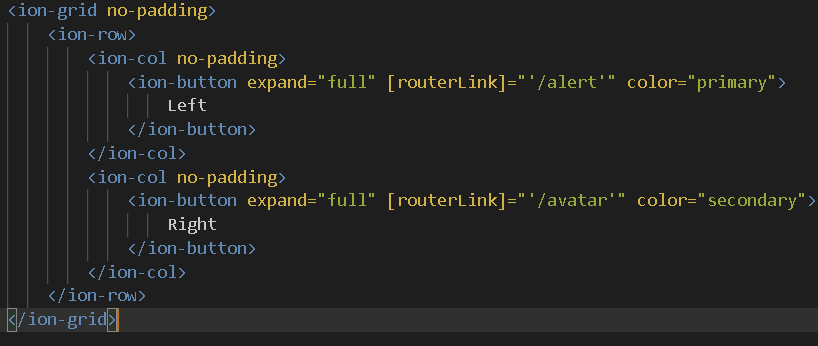
**Boton para navegar**

Utilizamos el routerlink



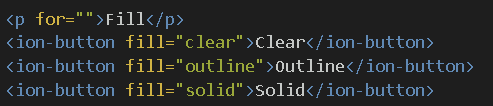


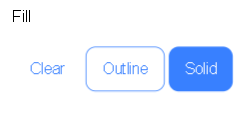
Botones en la misma línea (ROW + COL)





Fill

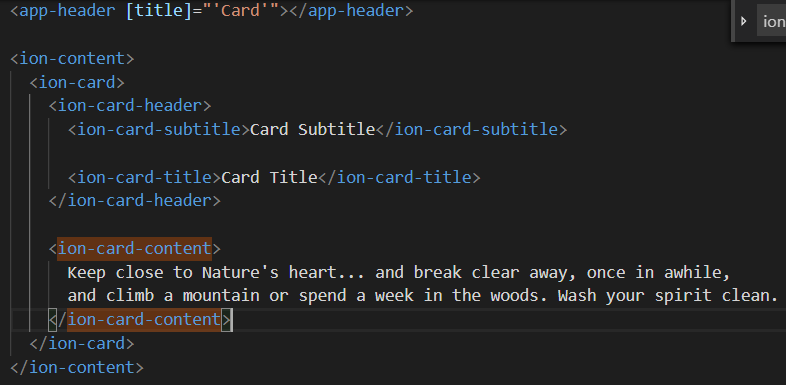


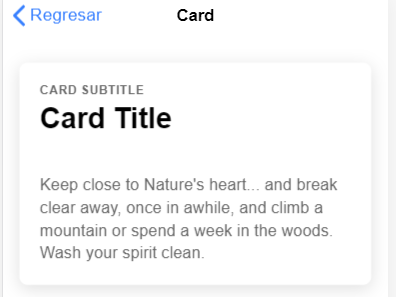


**Cards**

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.

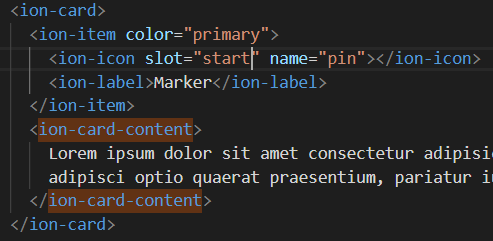


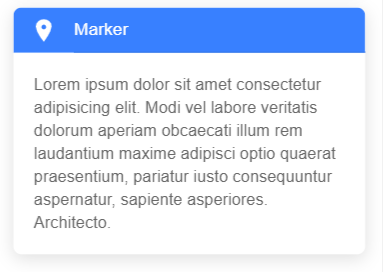


Sabemos que estos elementos se visualizan de formas diferentes en andriod y ios, podemos unificar su comportamiento con la propiedad **mode.** Pudiendo poner md para Android y ios para Ios.

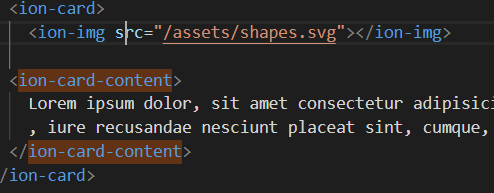


Otro ejemplo de card





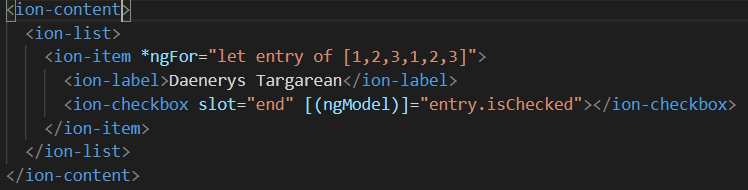
Ejemplo 3

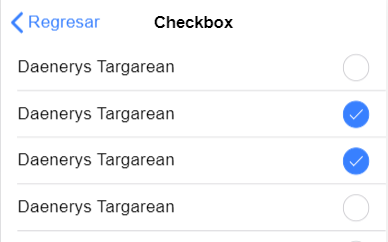




Ion-checkbox

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

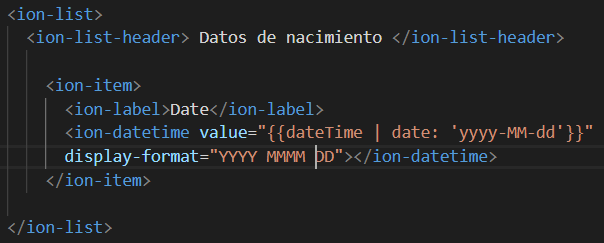




Ion Datetime

Se crea un componente para la pagina de date

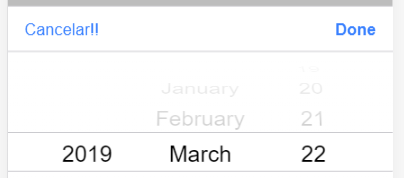
Se utiliza dentro de un ion-item





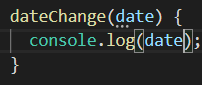
Cambiando el texto al cancelar y obteniendo el valor de la fecha seleccionada.





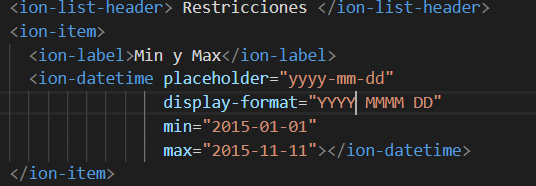
Se obtiene el valor con el ionchange y se pasa como evento como parámetro.





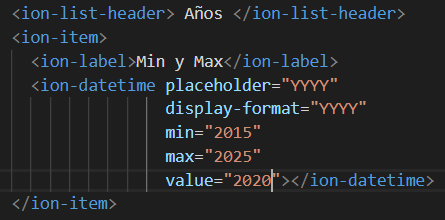
Datetime personalizado

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.

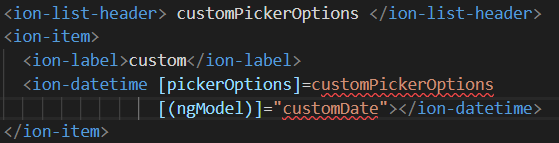


Listado por años

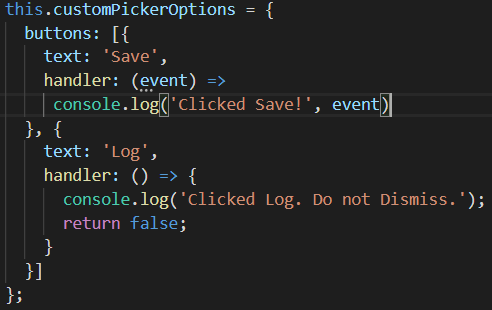
Se puede poner un rango de años y con el value se selecciona uno por defecto.



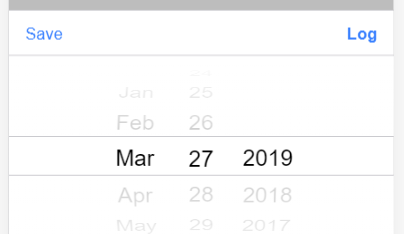
**Custom**



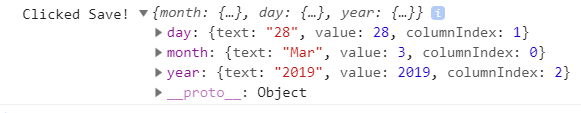
Se declaran las variables en la clase y se les asignan los valores en el ngOnInit



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:



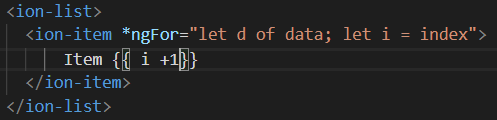
Ion-Fab

Fabs son elementos que pueden estar flotando y son seleccionables.

Se crea el componente y las configuraciones de siempre.

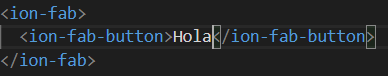
Creamos una lista de 100 elementos para simular el scroll.

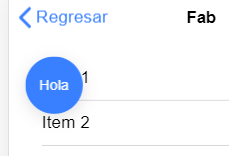




Utilizando el ion fab:

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



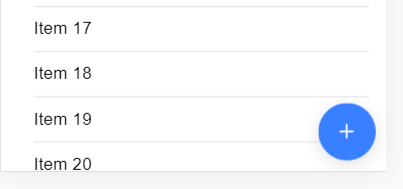


Para que se mueva con la pantalla utilizamos la propiedad slot=fixed. Dentro del ion-fab.

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

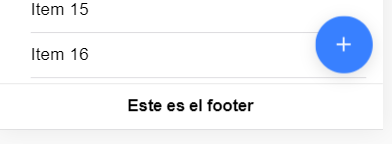
Con esta configuración, nos aparece en la parte de abajo, lado derecho.





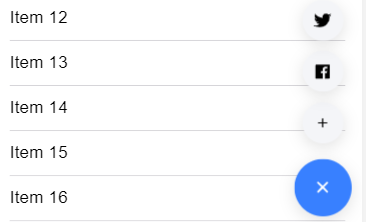
Si agregamos un footer

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

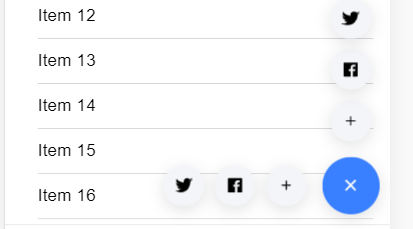


Para agregar submenús al fab se agregan dentro del ion-fab y se modifica la posición con la propiedad side:





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

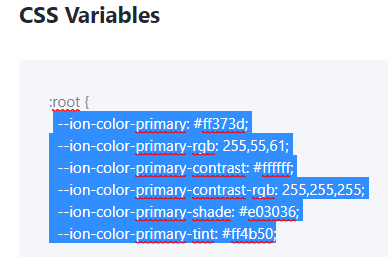


Creando variables de color

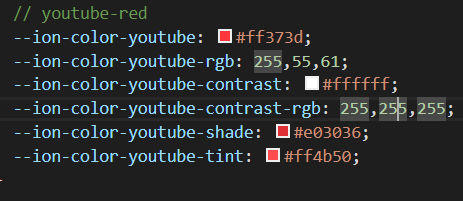
En el archivo variables.scss la carpeta theme se deben colocar las configuraciones.

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;

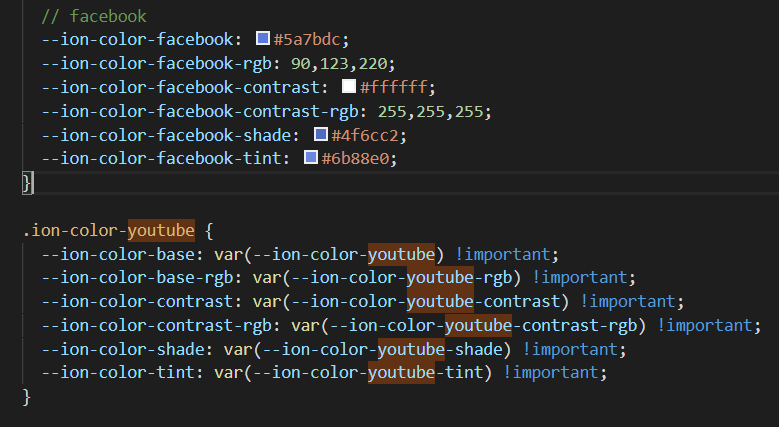
--ion-color-contrast-rgb: var(--ion-color-youtube-contrast-rgb) !important;

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

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

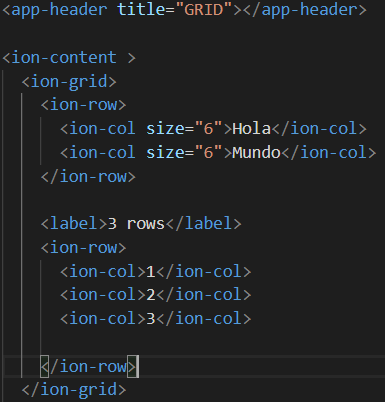
}

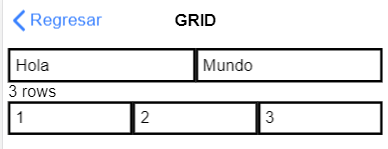




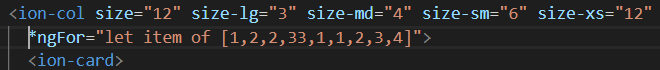
**Ion-grid**

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





Podemos regular el tamaño del col utilizando la propiedad size





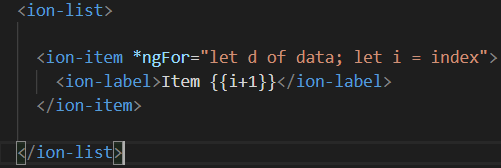
Infinite Scroll

Generamos la nueva pagina para infinite scroll.

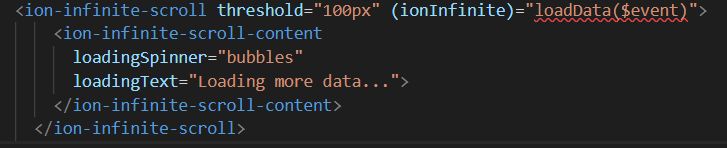
Creamos un arreglo genérico en el componente.ts



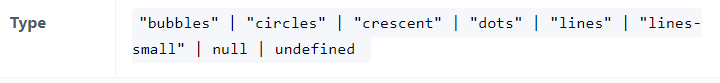
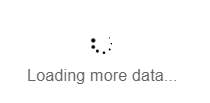
Y en el html iteramos el item con un for, dentro de un ion-list



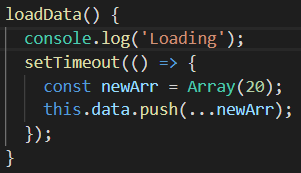
Luego copiamos el código provisto por ionic



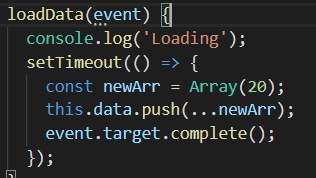
La propiedad teshold es la que dice que tan cerca del final es que empieza a cargar el infinite.  
Luego podemos personalizar el tipo de spinner con la propiedad loadingSpinner:

  
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.

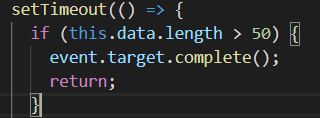


Si lo dejamos asi, el scroll no va a terminar de cargar aunque ya se muestre la información, es necesario poner que se ha completado luego de cargar los datos.

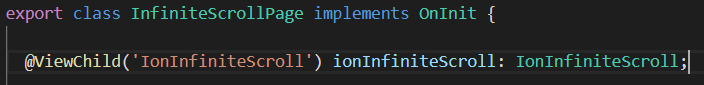


En el momento en que se acaban los registros a mostrar se debe para el infinite scroll.

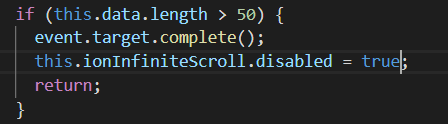
Dentro del setTimeot se valida el tamaño de los registros a mostrar, y hacemos complete el infinite scroll y retornamos.



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

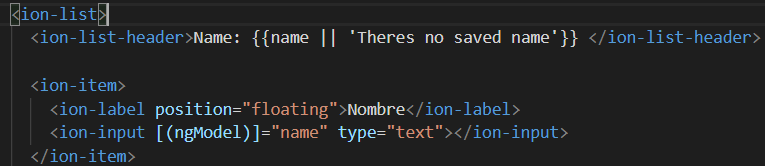


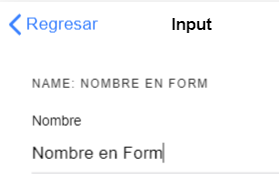
Y en la función accedemos a la propiedad disabled del infinite scroll declarado.



Ion Input

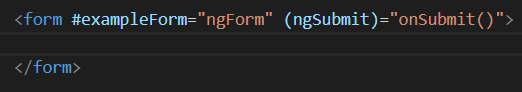
Creamos el componente

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.:  


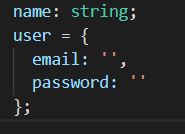


**Form**

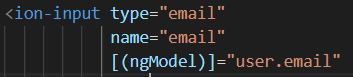
Se crea el formulario y se asocia a un ngForm con el #, luego se declara la acción que realizara al hacer submit.



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

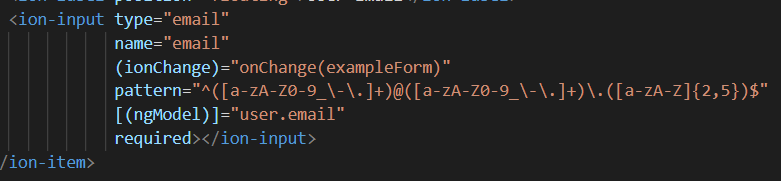


En el input declaramos el nombre, y bindeamos al objeto user creado en el componente.  
Es importante la propieda name.

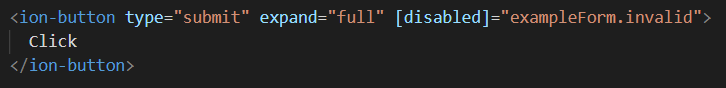


Al ser un email hay que validar su estructura con una expresión regular.  
Se puede poner como un pattern directamente en el html en el mismo input.





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  
