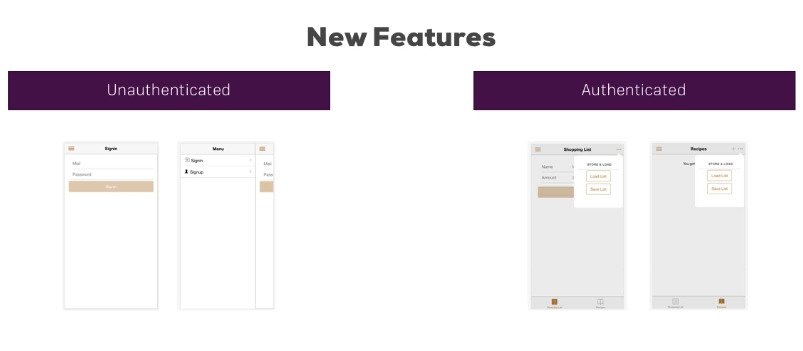
-Añadiendo un sideMenu donde elige si registarrse o iniciar sesión.

-Añadiendole un login a la aplicación de recetas,

-Guardando las recetas y lista de ingredientes en un servidor firebase.



Generar la pagina de signup y signin



Creando el sidemenu desde el app.html

En el app.html:

Se desea que al abrir alguna de las opciones que muestre el sidebar, se abra y desaparezca

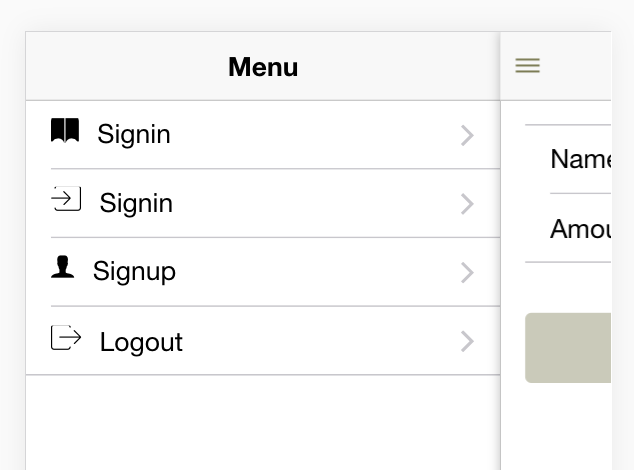
el menú de tabs, por lo que debe sobreescribir el rootPage.

Para esto directamente desde el app.html se agrega el ion-menu.

Añadiendo la propiedad con #nav , lo que hace una referencia local en el ion-nav, y unirlo,

o referir el menú a este tag con [ ].





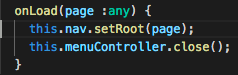
Para poder acceder a la propiedad root:

Se el utiliza el ViewChild y en sus parámetros se hace referencia a la que creamos

anteriormente en el dom:

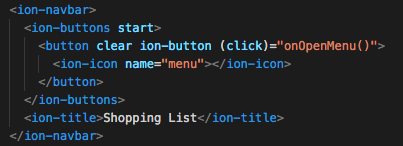


Para luego en la función redirigirlo según la pagina qu se seleccione, previamente declarado como una variable. Es necesario tener acceso al menuController para cerrar el sidemenu ya que al seleccionar una opción, este no cierra automáticamente.



Añadiendo un icono para saber que existe este side menú en las paginas:

En el navbar se agrega un ion-buttons, y dentro un botón que contiene la acción y el icono:



Y luego su respectiva función, que necesita implementar el menú controller desde el constructor



Creando la pagina de Signup y el formulario.

Se crea un formulario con los controles del template, haciendo énfasis en la contraseña utiliza la directiva de minlength =6, ya que firebase acepta solamente contraseñas con esa cantidad, esto puede variar según el backend



Añadiendole el controlador de la plantilla para acceder a los valores y para validar si es valido, luego llamando a una función si detecta el submit que pasa como parámetro el mismo formulario para captar los valores.

El botón de signin se activara solo si el formulario es valido:

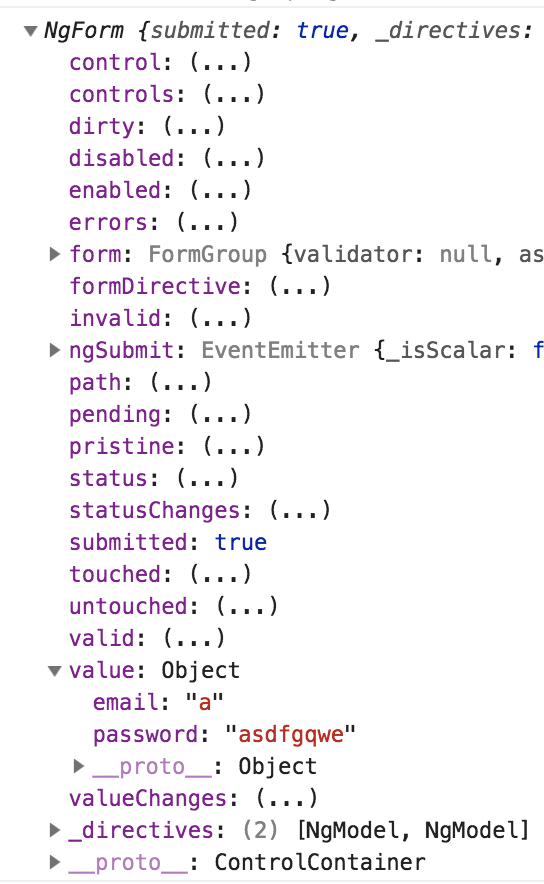




En la función recibimos los datos del formulario, en este caso se declara del tipo NgForm

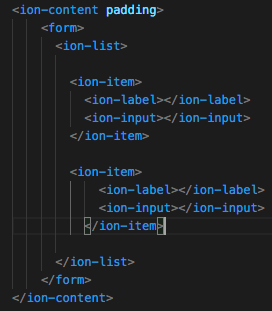


Lo que nos devuelve un objeto, y el campo name resulta relevante ya que asi tendrá el nombre de los datos ingresados en el objeto dentro de value:



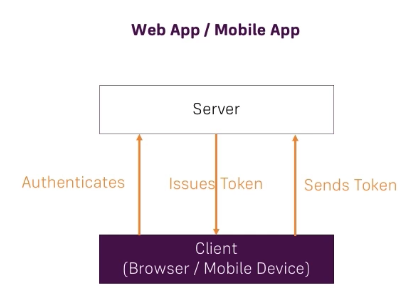
Creando la pagina signin

Siendo esta la estructura:

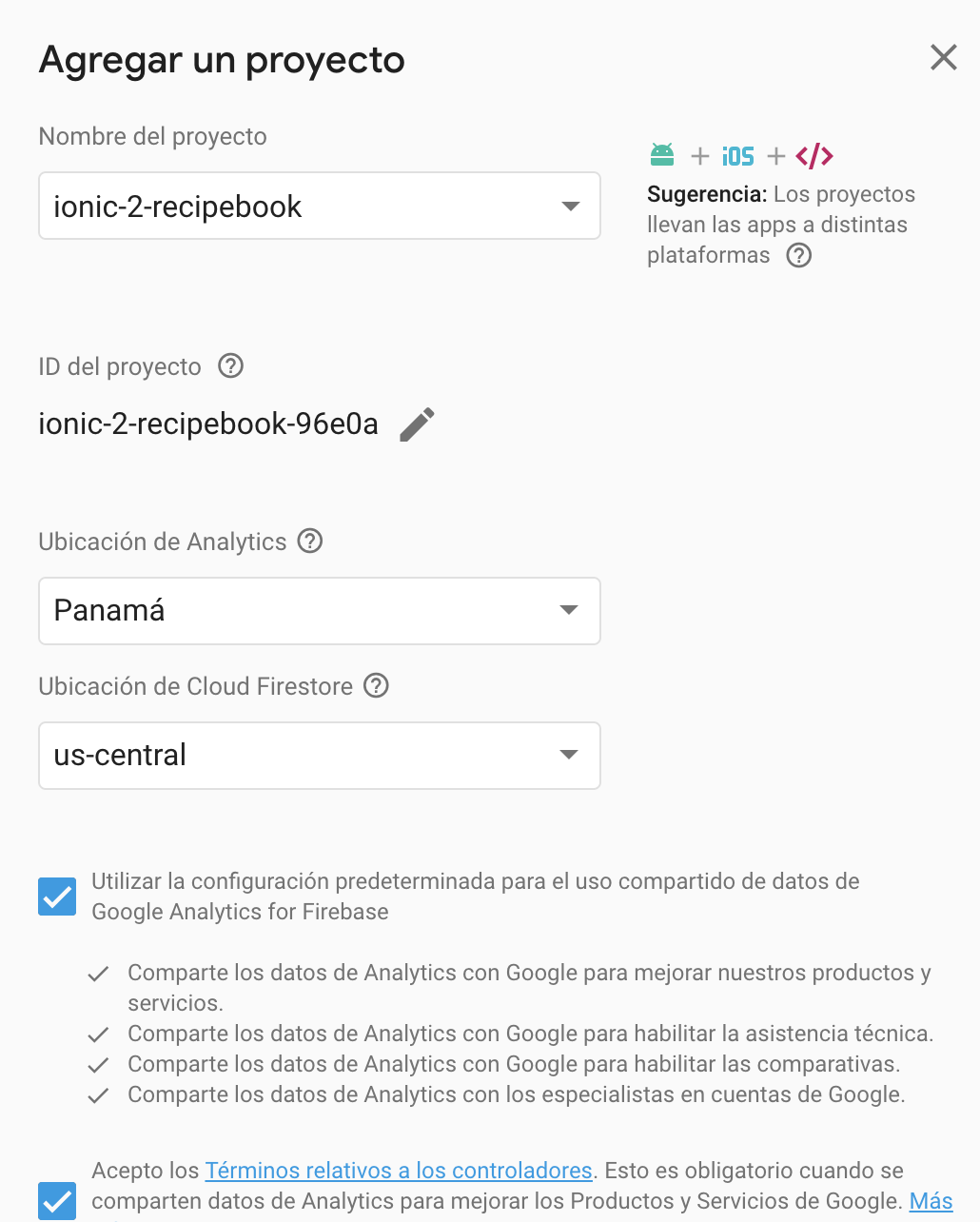


Como trabaja el servicio de autenticación:

Se en via la data que se escribe en el formulario hacia el servidor, y en este se validan los datos. En donde para validar durante la sesión iniciada se envía y recibe un token el cual será evaluado para certificar la validez de la sesión.

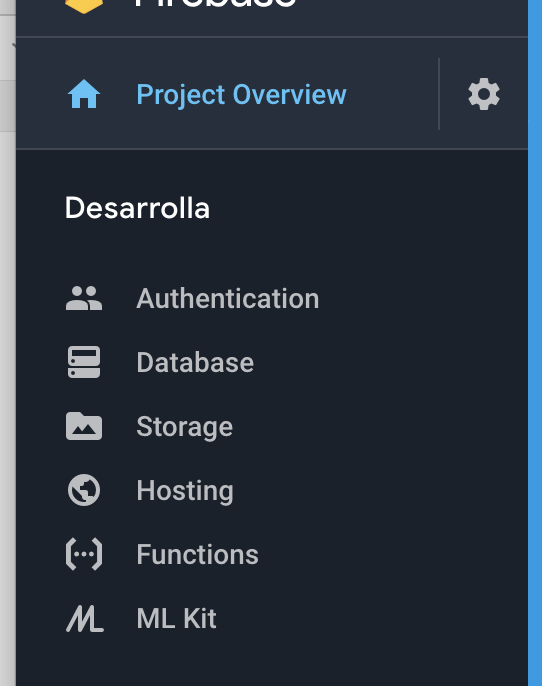


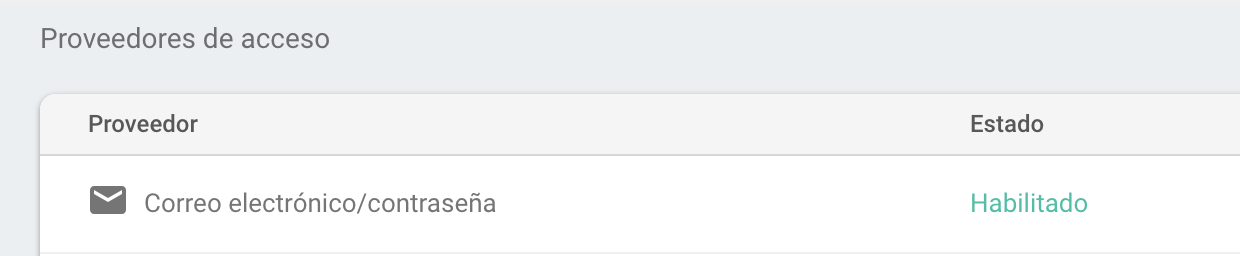
Setting up firebase:



en el panel izquierdo aparece la opción de autenticación:

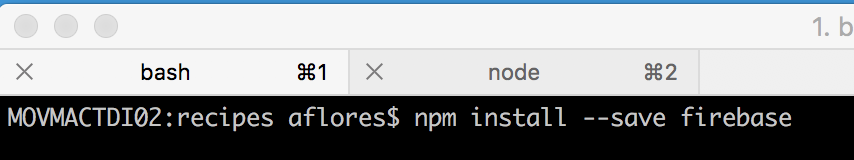
Es necesario tener en cuenta sobre instalar el sdk de firebase.



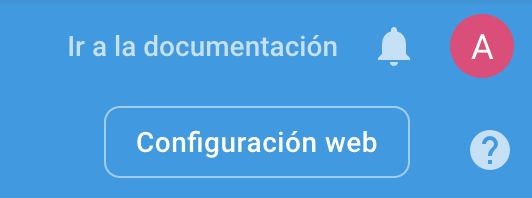


Añadiendo firebase al proyecto:

En la línea de comandos:



Una vez instalado hay que configurarlo, en la pagina en la parte derecha superior:



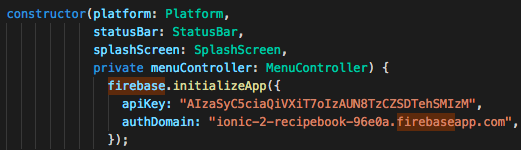
de todo el script que nos arroja copiar únicamente:



Ahora en el app.component:

Importar firebase y dentro del constructor inicializarlo con la información que nos mostrò la configuración de firebase.





IMPORTANTE, LUEGO DE QUE LA PLATAFORMA ESTE LISTA.

Ahora se crea un nuevo servicio auth.service.ts

Se importa firebase:



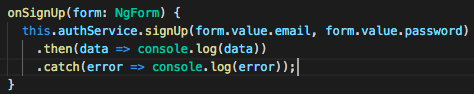
Y dentro los métodos necesarios:

El signUp para crear un nuevo usuario, el cual recibe usuario y contraseña, luego se accede a la función auth y luego createUserAndRetrieveDataWithEmailAndPassword.



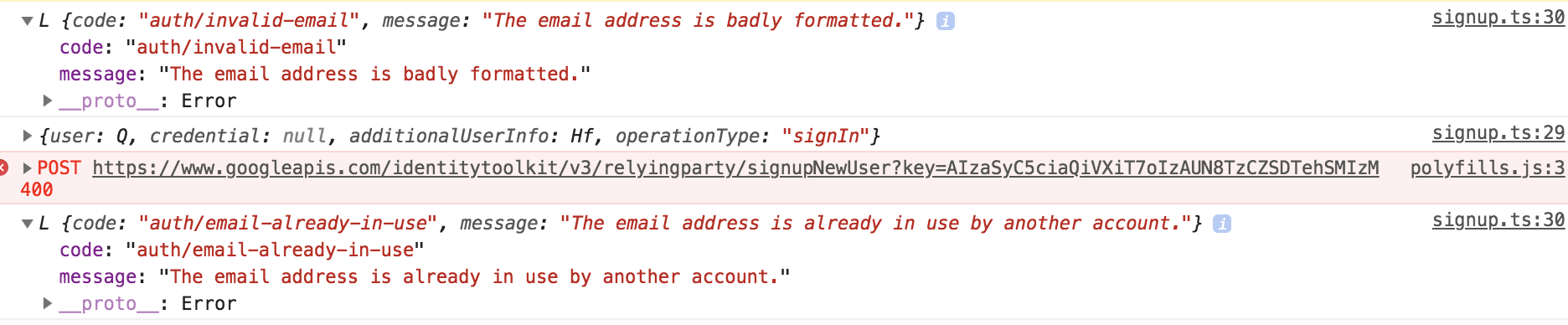
Ahora desde la pagina de registro (signup) se llama a esta función , ya que es una promesa se maneja de la siguiente forma,

El then , va a manejar cuando responda correctamente:

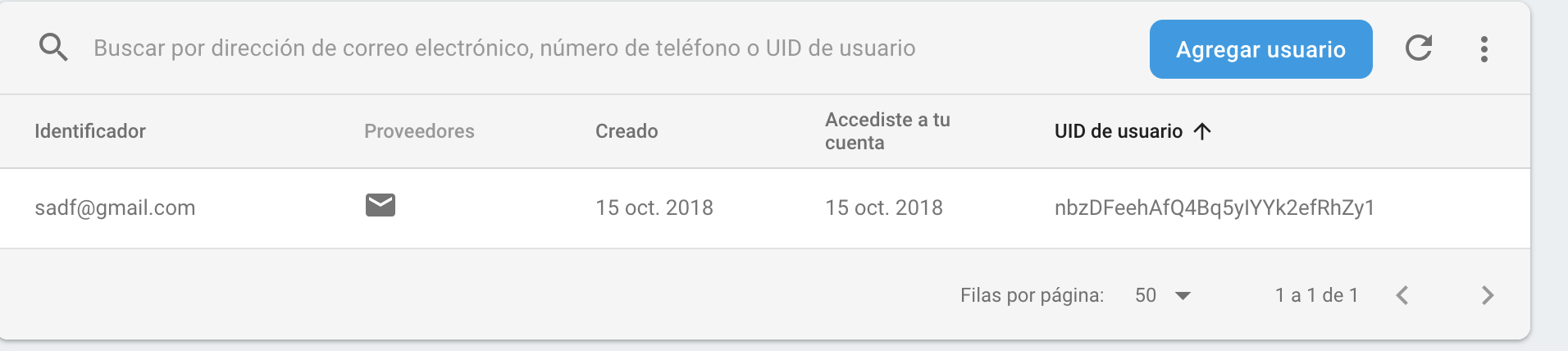


Al introducir la información en el formulario nos da ya validaciones,

1. En caso que el email sea invalido,
2. Correcto
3. Usuario ya registrado



y dentro de firebase



Añadiendo un spinner y mensaje de error:

Ionic cuenta con una librería especial que levanta un loading state.

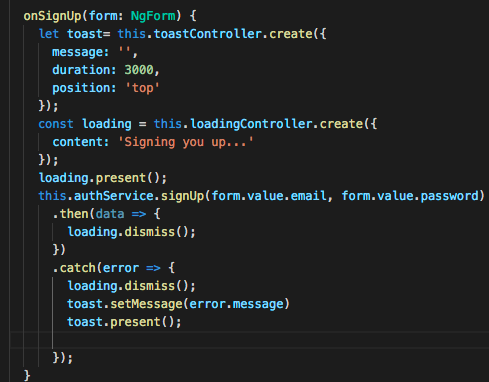
Se importa en el constructor y en el archivo:



Luego dentro de la función que estamos consultando el servicio, se crea una constante loading, que contiene el loading controller con la estructura del mensaje a mostrar.

Se accede a sus propiedades para mostrarlo (present() ) y para cerrarlo una vez retorne el valor ya sea correcto o un error.

Al igual que presentar un toast:

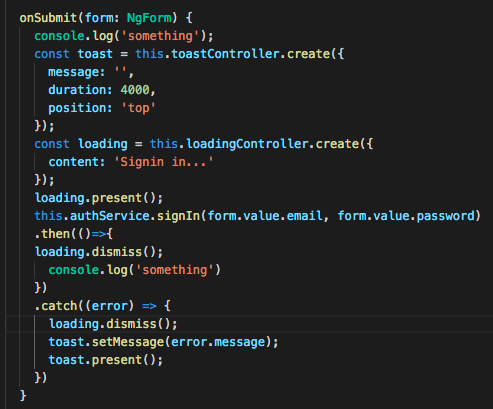


Signin in the app

El en AuthService implementar la función, la que nos retorna una promesa



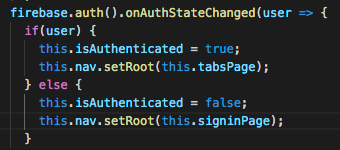
y en el signin component



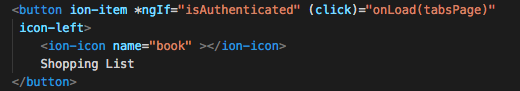
Managing UserState:

Para mostrar una u otra pantalla hay que validar desde el app.component, dentro del constructor:

Llamar a la función onAuthStateChanged, que avisará cuando el estado cambie si esta autenticado o no , recibiendo un objeto promesa.



Ahora una vez ya tenemos la información si el usuario esta autenticado, con la variable del componente isAuthenticated, podemos modificar el hmtl para que muestre las opciones según los provilegios deseados en el momento.



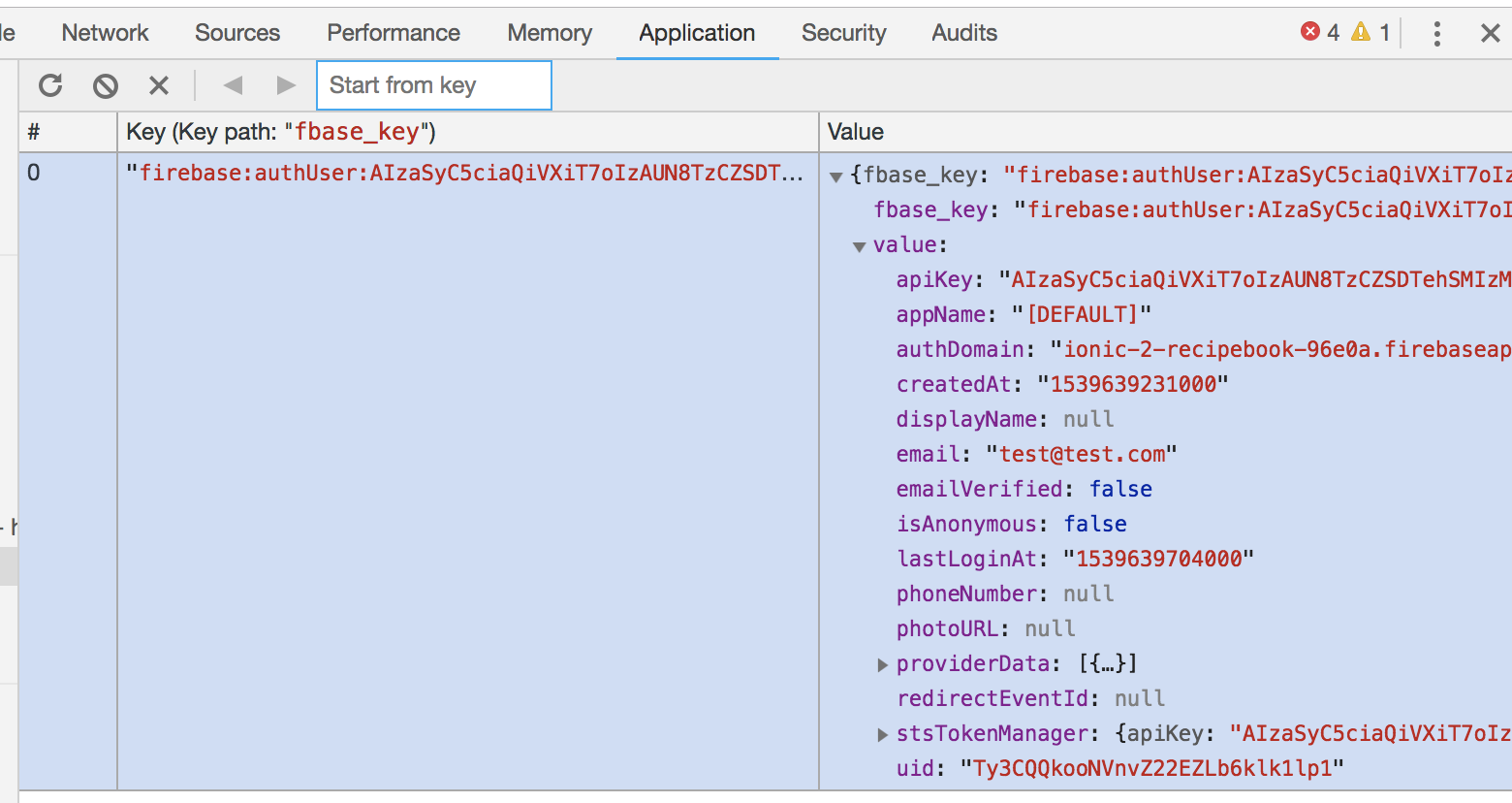
Añadiendo la función de log out

Desde el servicio



Como Firebase almacena el token:

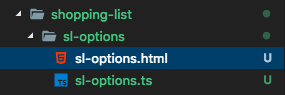
Al iniciar sesión se almacena dentro del local storage, exactamente en :

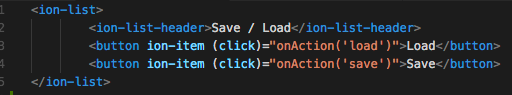


Añadiendo un popOver Component que desplegara las opciones para salvar o cargar.

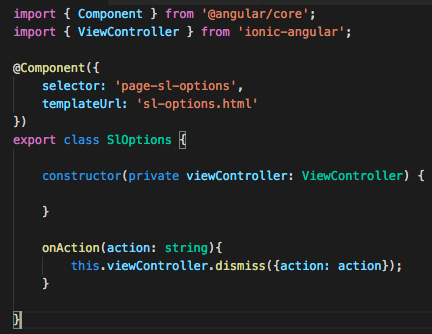
Se crea dentro de la carpeta shopping-list

Sl-options

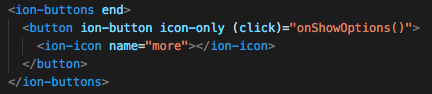




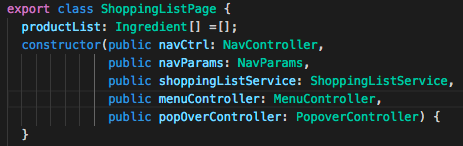
En el component, se crea el constuctor con controladores, :



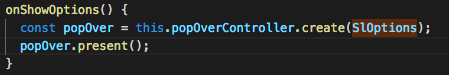
Ahora para mostrar la opción se crea el icono desde el shoppingList html



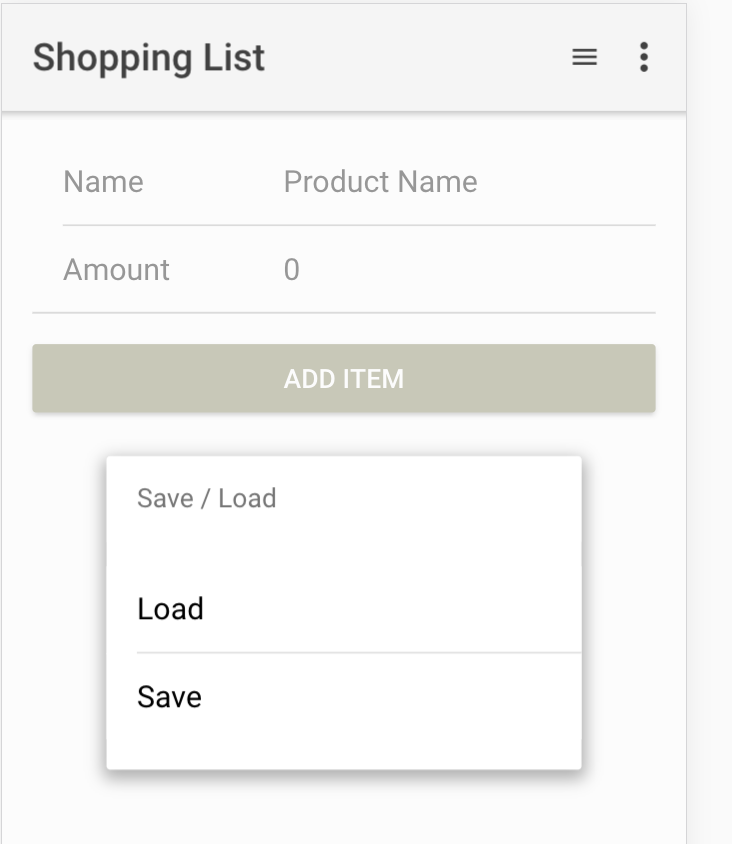
y luego en el component se agrega la función, y en el constructor se importa el controlador del pop over



Se crea la función en donde se crea una variable, y se le asigna el valor del controlador del popOver llamando a la pagina de su componente.



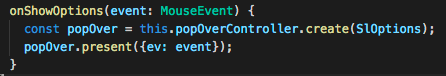
Al tenerlo asi se presenta el pop over en medio de la pantalla:

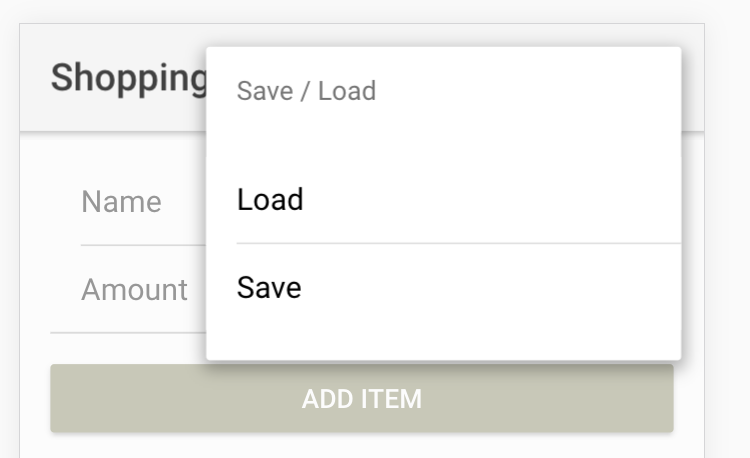


Para esto, hay que decirle al pop over desde donde esta siendo llamado, pasándole un event.



y recibiéndolo en el componente de shoppingList:





Utilizando el token para almacenar datos en firebase:

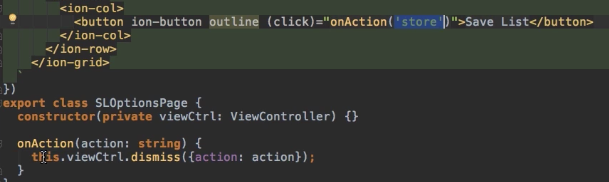
Para almacenar datos en firebase se necesita la información que esta en el token que se generó al momento de iniciar sesión.

Desde el servicio de auth.ts

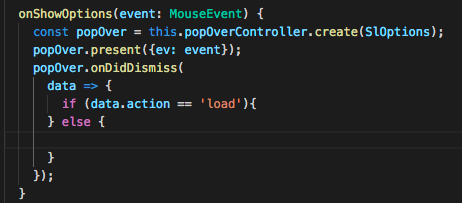
Accedemos a la propiedad auth().currentUser con firebase.



Al momento de elegir una opción en el pop over, este manda como parámetro una acción, la cual es recibida en el método y la pasa como un parámetro al momento de hacer dismiss.



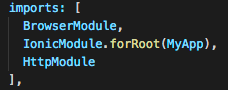
Ahora desde el shoppingList donde fue declarado el controlador del popover se accede a su propiedad de onDismiss, recibiendo el parámetro como una data y declarado el valor recibido desde el pop over como un objeto con llave, accedemos a su valor de la siguiente forma:



Luego se importa el servicio de authService para obtener la información del token del usuario, lo que devuelve una promesa, la cual se accede:

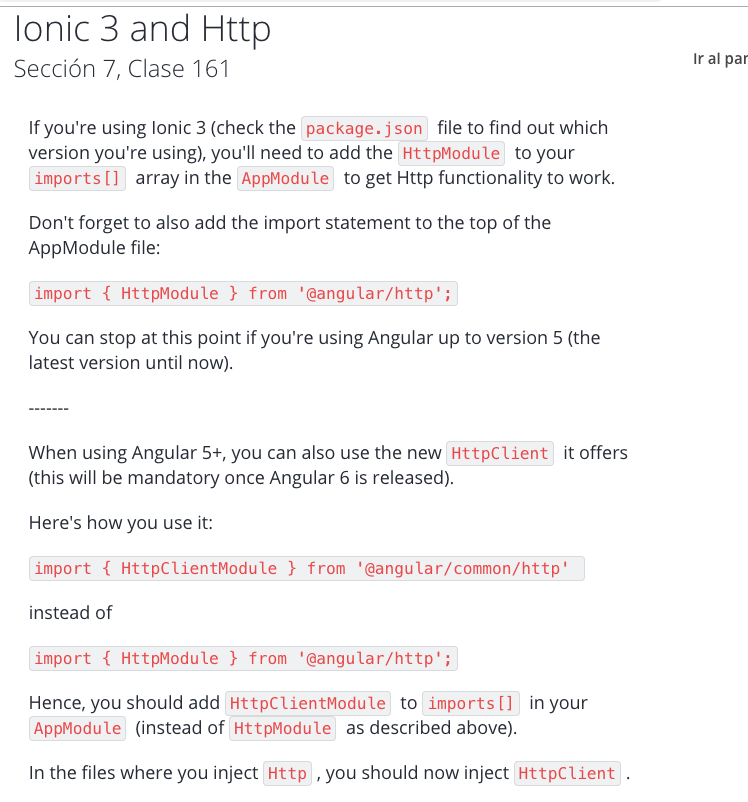
Desde el app module importar el HttpModule y agregarlo a los imports

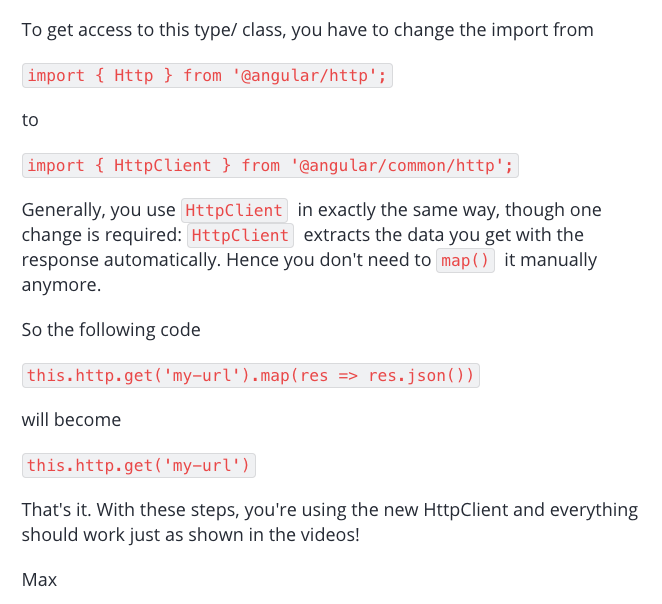




IMPORTANTE, SE UTILIZA AHORA EL HttpClientModule:

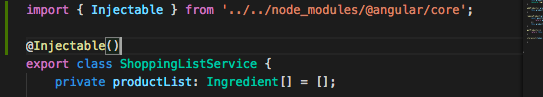






Ahora desde el shopping-list-service, se puede añadir la función storeList para guardar una lista el firebase.

Se necesita “INYECTAR” el servicio “http”, asi que es necesario implementar el @Injectable() en el servicio



e importar http en el servicio.

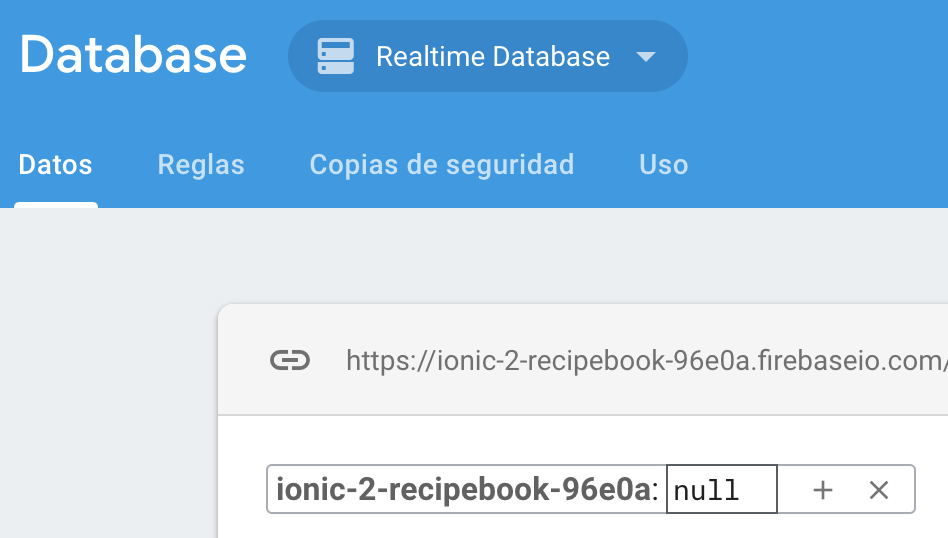


ahora se puede utilizar el http para agregar datos a firebase con la función put,

Se utiliza put ya que se va a sobreescribir la lista, de existir, para que tenga las anteriores y la nueva lista.

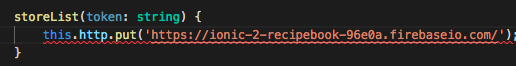
De utilizar post, firebase añadiría la lista a la lista anterior, teniendo una lista de listas.

En firebase podemos ver la dirección de la base de datos desde.



¡!VER ANOTACION MAS ABAJO SOBRE EL HTTP!

y esta es la dirección que se pone dentro del put



En vez de almacenar la data directamente en este url vamos a crear subfolders para el usuario activo, para que el usuario logeado tenga su propio “nodo” en la base de datos.

Para esto importamos el authService, donde tenemos el método para obtener los datos del usuario activo.

Lo obtenemos y se le asigna a una constante para luego completar el url que tendrá el subfolder, y luego agregándole como ultimo parámetro la lista de ingredientes:



YA NO SE MANEJA DE ESTA FORMA CON EL MAP

Como esta función retorna un observable, no se manejara directamente en esta función asi que se pone un return a la respuesta para manjearlo en el momento que es llamado con un subscribe.

Dentro de esta función la respuesta se mapea para que la respuesta se reciba en formato json. Utilizando el map (IMPORTANTE: para utilizar map es necesario importar el

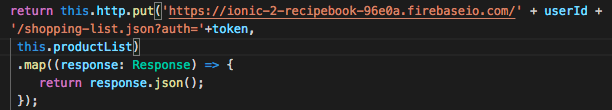




De la forma que esta escrita nuestra url, al acceder esta solicitud podría editar cualquier usuario al ni esta utilizando el token, osea el valor del auth de la sesión actual, para esto utilizamos el token y se modifica el path de la siguiente forma.

 <-

quedando la función:



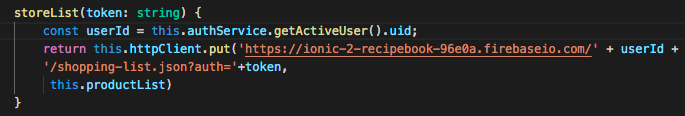
De esta forma solo si la sesión es valida podrá editar la información.

¡!! IMPORTANTE!!!

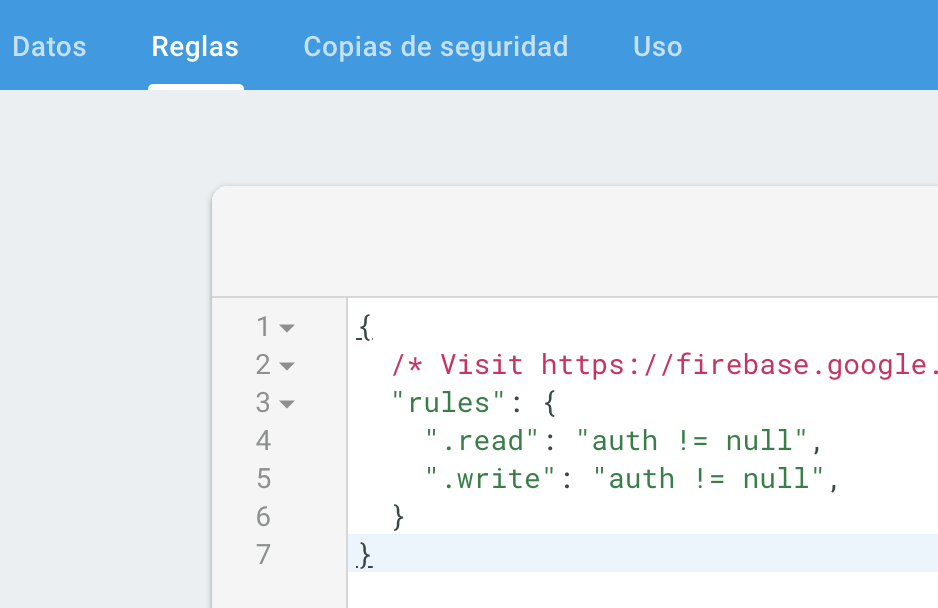
Se debe utilizar el



con este HttpClient ya no es necesario mapear la respuesta para un json ya que lo devuelve sin forzarlo.



y cambiar las reglas en firebase a :



Ahora enlazando esta función del servicio al shoppingComponent:

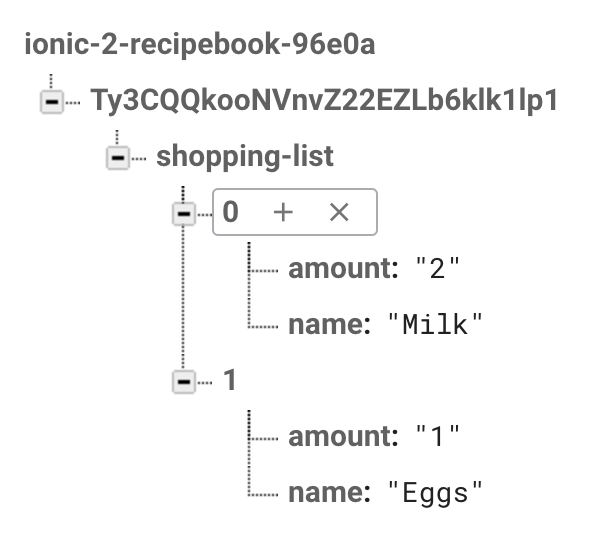
Se suscribe a la respuesta de la función.



Añadiendo un loding state mientras consulta el servicio:

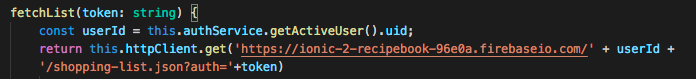


En firebase e agrega la información de los nodos de esta forma:

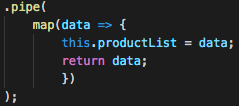


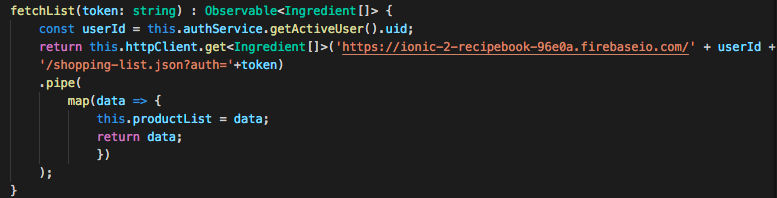
Para cargar la información que esta en firebase:

Utilizamos el método get, parecido al anterior con el url, userId, y el token. El cual mapea automáticamente y lo recibe como un objeto json.



Utilizando la función pipe y luego map, que funciona parecido a un observable, podemos utilizar los datos que retornan desde firebase automáticamente para cargar nuevamente el arreglo local de ingredientes.

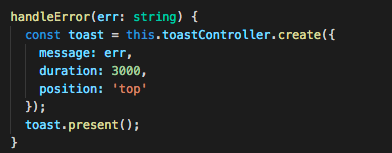




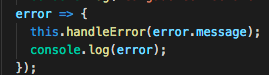
Al retornar un observable la función, desde la función load del shopping-list.ts, debemos subscribirnos a la respuesta.

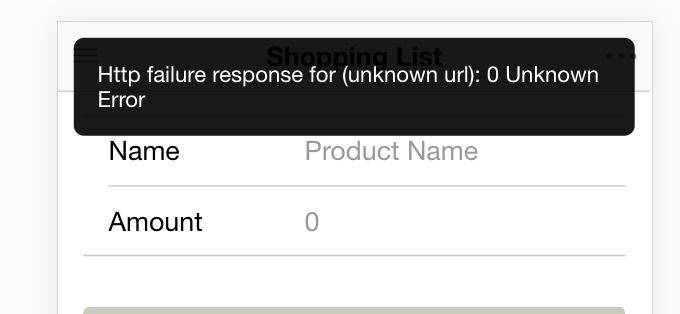


Manejando los errores, se crea una función para esto, que muestra un toast con el error recibido.



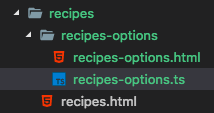
dentro del subscribe se llama a esta función cuando es necesario.





Repitiendo el procedimiento para recetas:

Creando las opciones de load y save en recetas:

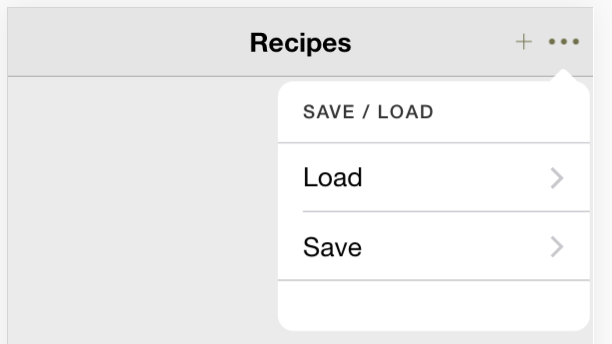


En el récipes html, poniendo el icono de more,



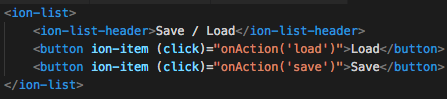
luego en el ts añadiendo la funcionalidad para que abra las opciones.

Añadiendo la pagina de opciones, al app module en declarations y entryComponents



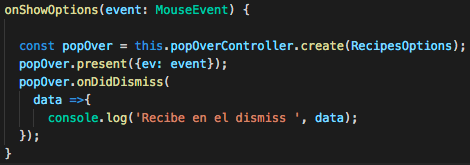
Recibiendo las acciones desde las opciones (PopOver)

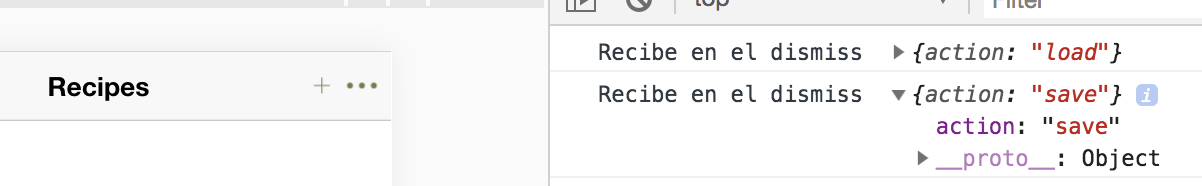
Desde el html se llama a la función y se recibe en el ts, el cual va a mandar la acción al componente principal de récipes, y lo recibirá con la función onDismiss.



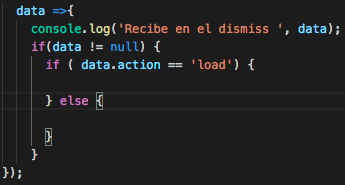


En el récipes.ts se recibe la información desde el onDismiss escuchando su respuesta.



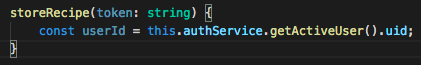


Se crea la lógica según el valor recibido en el objeto para realizar la acción.

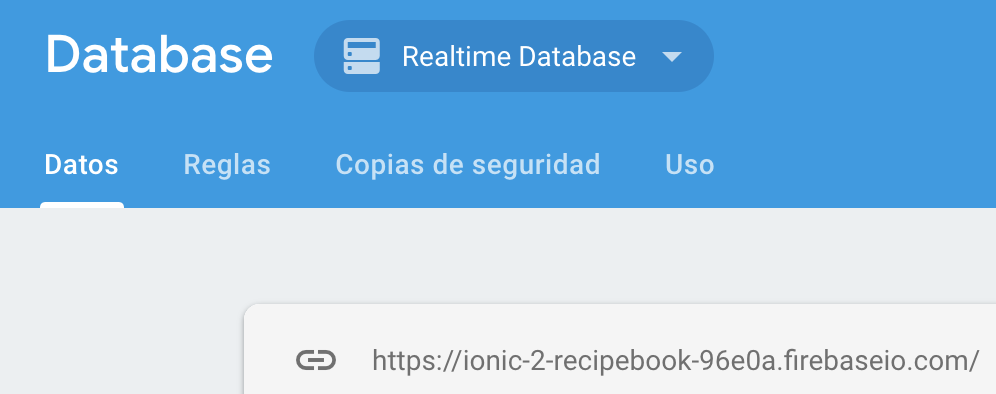


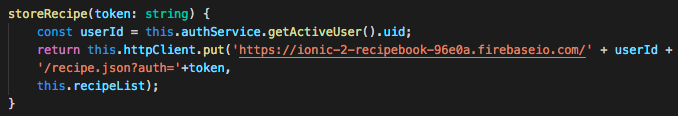
Se crean las funciones en el servicio de recetas recipe.service

Guardando la receta storeRecipe, donde se recibe como parámetro una cadena que contiene el token. Luego se busca el id del usuario con el authService



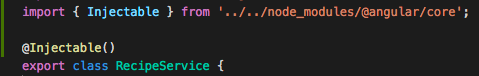
Se recupera la url del proyecto en firebase, adjuntándole el id del usuario y la extensión del json donde guardarlo con el token, y el segundo parámetro la lista local para almacenar.





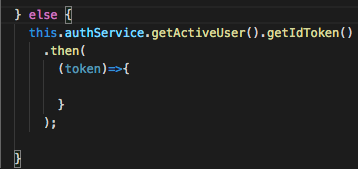
¡IMPORTANTE!

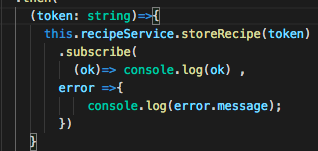
Declarar el servicio que hara la consulta a firebase como @injectable()



Ahora desde el récipes.ts se puede llamar al recipe.service para almacenar en el firebase.

utilizando el authservice para obtener el token y poder enviarlo a la función del servicio.



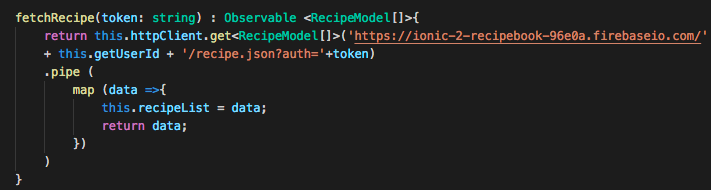




Cargando la información desde el servicio:

Se crea la función fetchRecipe, que recibe un token como parámetro, y retorna un valor observable de tipo RecipeModel arreglo.

Haciendo el llamado del httpClient get, se coloca el url del proyecto en firebase, con el usuario, y la data a cargar, con el token valido ya obtenido en los parámetros. Es necesario mapearlo para recibir el objeto correcto y asignarlo al valor local del arreglo de recetas.



¡IMPORTANTE!

Haciendo una función del userId

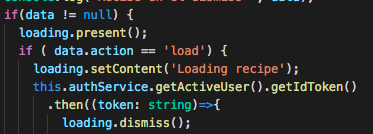


Ahora desde el recipe.ts se puede hacer el llamado a esta función del servicio.

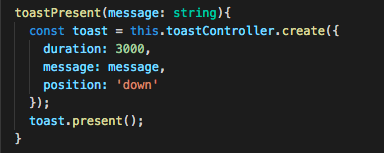


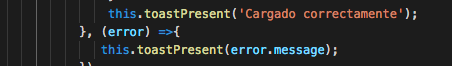
**Añadiendo loading**

****

****

Añadiendo toast con una función especifica, y luego es llamado según la acción.





BUGS

Cuando la receta no tiene ingredientes lanza un error.

Para esto en el momento de recibir las recetas, se recorre el arreglo recibido y verifica cada uno con la función hasOwnProperty y el nombre, y si no existe, se declara y se le asigna el valor de arreglo vacio.

