https://www.udemy.com/curso-de-angular-2-4-5-avanzado-mean-jwt/

Creando el proyecto:

Ng new angular-avanzado

En angular 7 para verificar la versión es  
ng –version  
Dependiendo de lo que este en el package.json, se puede cambiar el puerto default:

Y empezar el comando con:  
npm run start

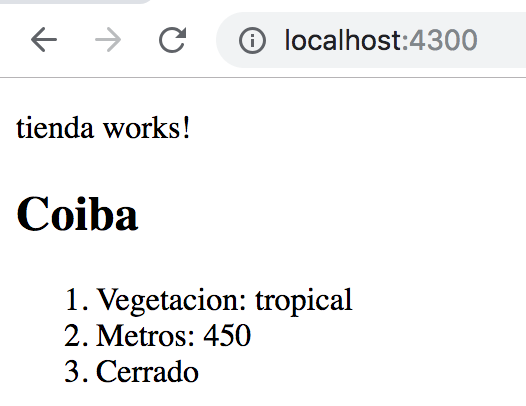


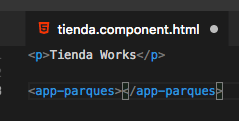
Creando componentes:

Se crea la carpeta componentes, y dentro los que usaremos, en este caso tienda.

Ng g c components/tienda

Parques





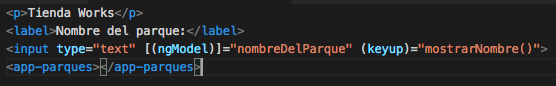


Comunicación entre componentes:

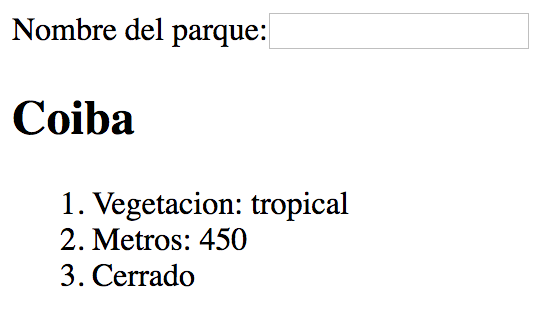
Input y Output

Pasando información desde el componente padre (tienda) Al componente hijo (parque).

Empezando se crea un input en tienda.component y se bindea el valor con un ngModel a una variable del componente tienda. (Importante importar FormsModule en el app.modules.).







Para recibir desde el componente hijo se utiliza el decorador Input() detrás de la variable



Una vez bindeado el valor podemos utilizar esta misma variable para asignarle el valor que tendrá, por ejemplo



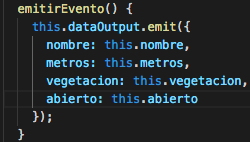


**Output**

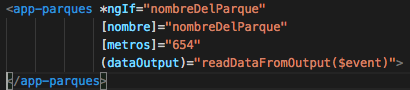
Para enviar datos al componente padre, hay que importar Output desde el componente hijo. Luego crear la variabe siendo una instancia del EventEmitter



Luego dentro de una función se puede declarar el dato de salida con la función emit.

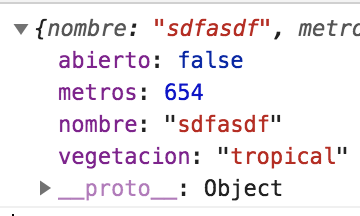


Desde el html del componente padre podemos utilizar el output con la variable declarada, luego llamando una función que recibe como parámetro un event, que realmente son los datos.



y finalmente en el component.ts llamamos a la función donde se le pasa la variable event y ya podemos utilizar el objeto que se recibió.





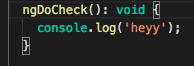
Hooks del ciclo de vida del componente:

Eventos que se lanzan dependiendo del estado del componente, o ciclo de vida.

OnChange, onInit, OnDestroy

Do Check

Se ejecuta luego del onInit, se ejecuta cada vez que pasa algo en nuestro html o componente.



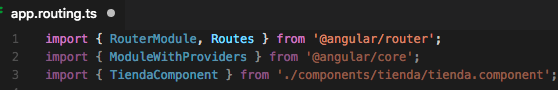
OnDestroy

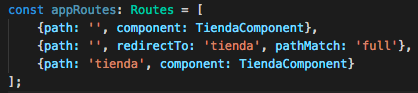
Se ejecuta cuando se destruye un componente. Por ejemplo cuando deja de mostrarse el componente hijo al no tener el nombre del parque en este ejemplo.

Configurando el sistema de rutas.

Se crea un archivo dentro de la carpeta app, app.routing.ts

Luego dentro se importa RouterModule, Routes, ModuleWithProviders y los componentes de la aplicación:

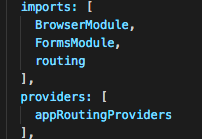


luego se crea una constante de rutas  


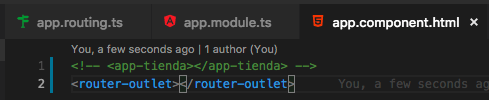
Luego exportando los provider de rutas y el modulo de rutas.:



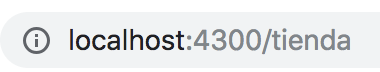
Luego desde el app.module importar estas variables en los imports y en providers:



Luego en el app.html mostramos el tag de router outlet, que mostrara el componente que este definido.



ahora carga por defecto esta ruta, pero si ponemos otra aparece un error



Para no tener error debemos configurar el router y asi cuando se ponga una ruta que no existe cargara por defecto la principal definida.

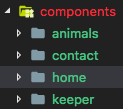


Routers y componentes:

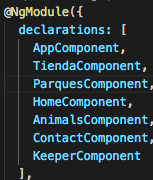
Creando componentes:

Home





Recordar implementarlos en el app.module en declarations

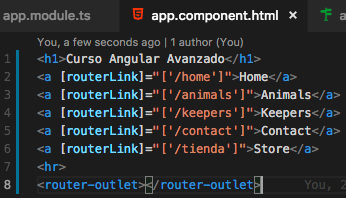


ahora como serán implementados en el nav es necesario incluirlos en el app.routing



Haciendo un menú de navegación

Desde el app componente creamos la navegación utilizando la propiedad routerLink

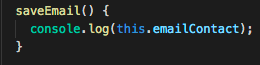




Utilizando localStorage y doCheck:

Nos permite guardar nuestros datos en la navegación, que persiste en la pagina durante la navegación de diferentes paginas.





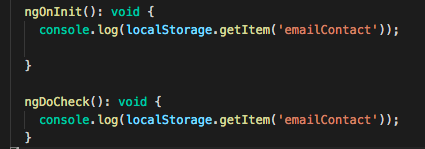
utilizando el localStorage



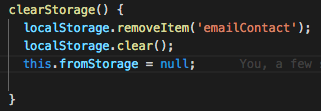
si luego de guardarlo se agrega una impresión del localStorage desde el app.Component

se va a imprimir en un ngOnInit y recuperara el valor puesto desde contact.

Para mantener esta información siempre actualizada se puede utilizar el DoCheck ya que cada vez que cambia algo en en nuestra pagina se ejecuta.



Eliminando un elemento del localStorage



Instalando bootstrap , jquery.

Desde el package.json

\_\_\_\_\_\_\_-

npm install jquery –sabe

Desde el angular.json incluir



Recordar cuando se va a incluir bootstrap debe ser luego de haber implementado jquery.

Despues de incluirlo se puede acceder a las propiedades de jquery importando

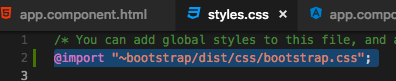
import \* as $ from 'jquery';

|  |
| --- |
| public ngOnInit() |
|  | { |
|  | $(document).ready(function(){ |
|  | $("button").click(function(){ |
|  | var div = $("div"); |
|  | div.animate({left: '100px'}, "slow"); |
|  | div.animate({fontSize: '5em'}, "slow"); |
|  | }); |
|  | }); |
|  | } |

BOOTSTRAP

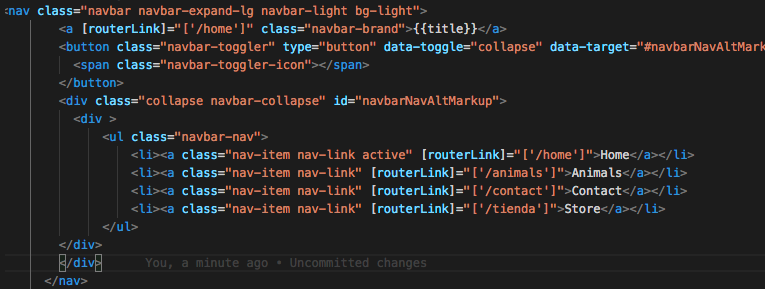
Npm install bootstrap

y desde el styles.css

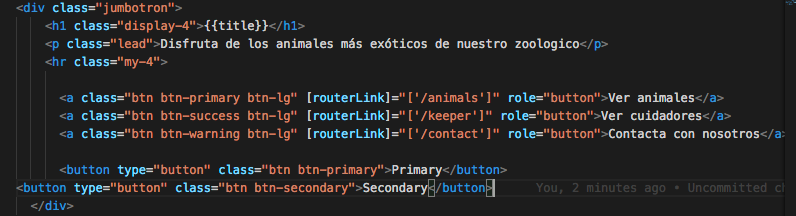


Luego como un plus podemos integrar una hoja de estilo global dentro del assets, styles.

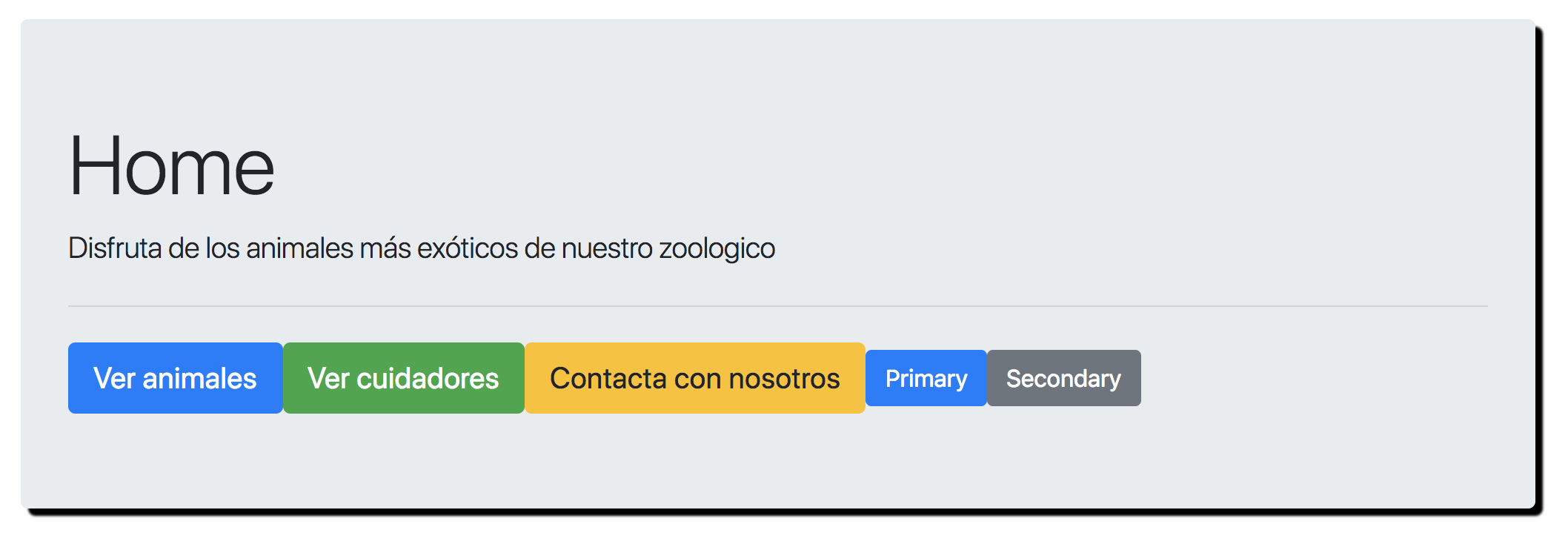
<link rel="stylesheet" type="text/css" href="assets/css/styles.css"

Maquetando el menu utilizando los routerLink 

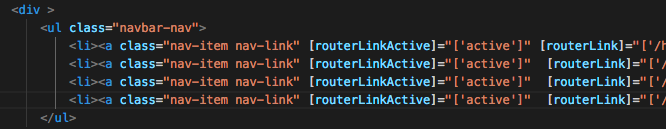
Bootstrap Jumbotron



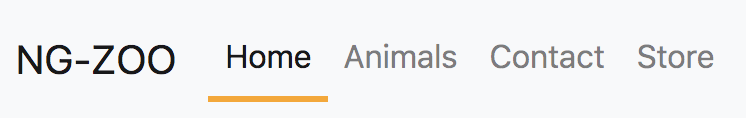




Marcar la ruta actual







JQuery

Para utilizar jquery dentro del componente lo importamos



luego en el html podemos declarar los elementos con id para luego poder utilizarlos



ya podemos utilizar el id para acceder al elemento mediante jquery ($),

en este caso por ejemplo se puede remover un atributo mediante la función removeAttr y luego realizar otra acción encadenada.



Incluyendo otra librería.

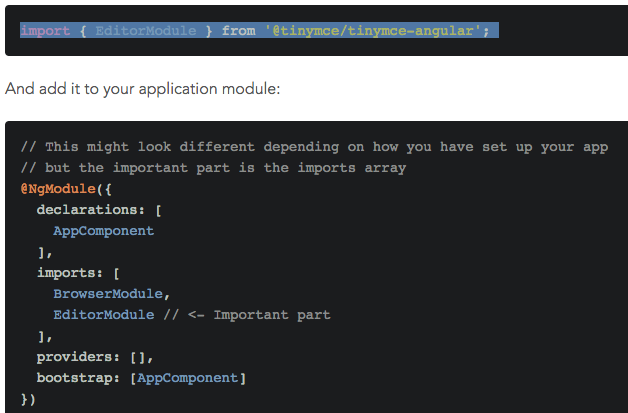
npm i jquery.dotdotdot

¿? No se pudo

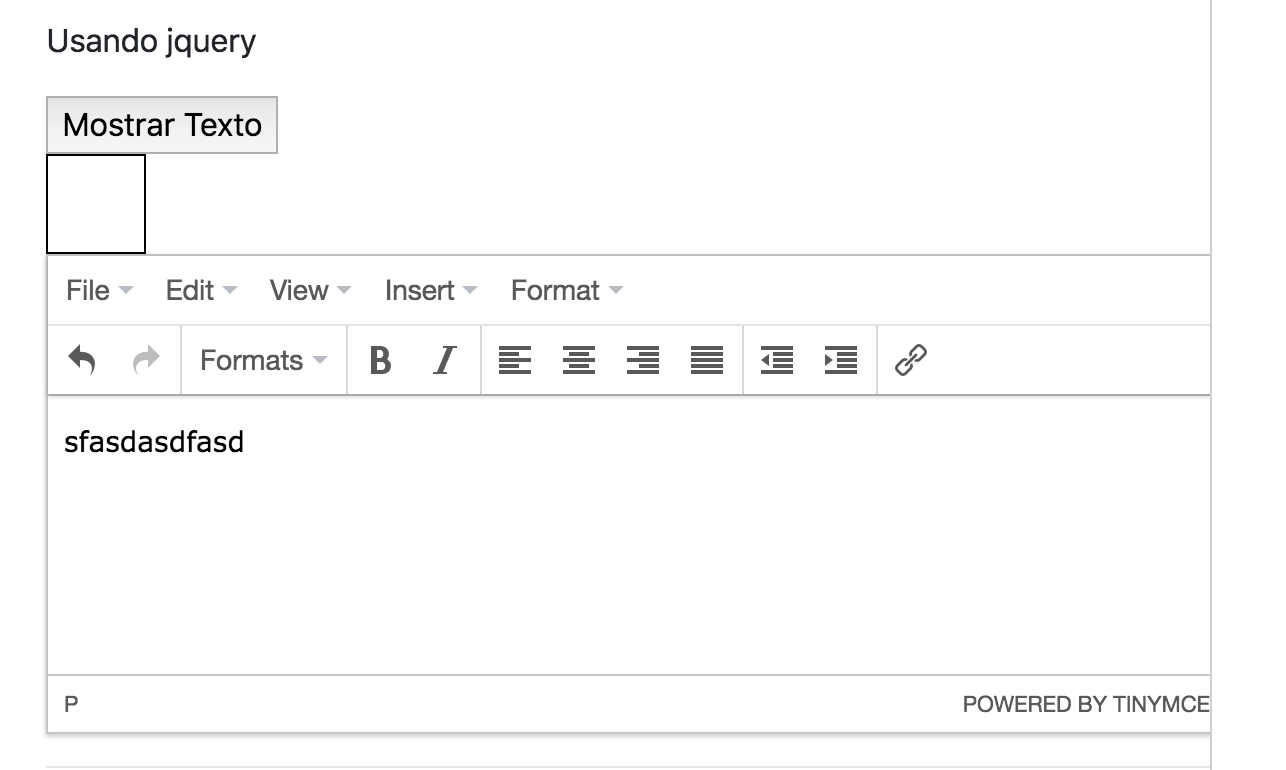
Instalando editor de texto enriquecido:

Tinymce









Componentizandolo:

Crear una carpeta simple-tiny, luego un archivo simple-tiny.component.ts

En el html se inserta el template de timy



y luego desde el componente padre se llama. En este caso tienda.component



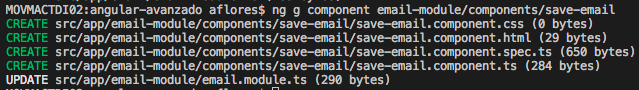
Explicación sobre scripts exteriores no sirve en el curso.

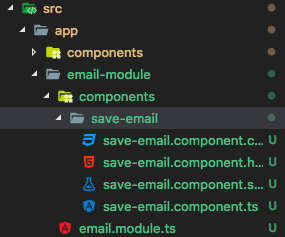
Modularizando webapps con Angular.

Creando una carpeta module puede tener dentro varios componentes, este modulo puede ser utilizado en el app y en otras apps si es estructurado de la forma correcta.

Creando el modulo en la ruta especifica dentro de su carpeta:

La carpeta email-module, dentro tiene sus componentes y modulo.

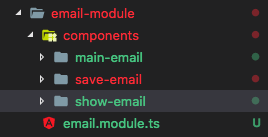




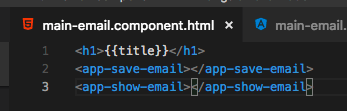
En el componente save-email se pasan las funciones existentes en contact



Igual para el show , luego se crea un componente principal en su propia carpeta.



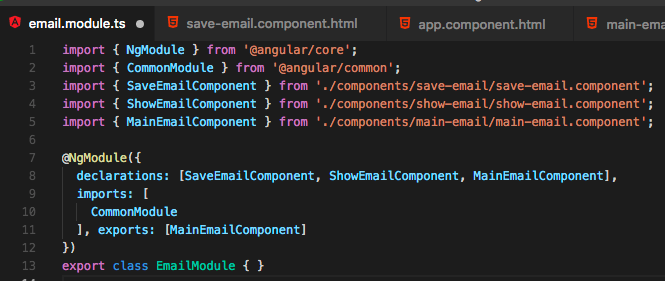
Donde el modulo principal tiene



Ahora el module principal debe contener las declaraciones de los componentes.

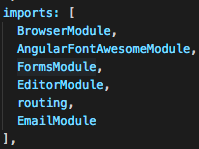


Importando el common module, declarando los componentes que contiene y declarando el modulo principal dentro del export.

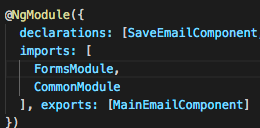


Para cargar el modulo en la aplicación principal hay que importarlo en el app.modules

Dentro de los imports



Y ya que dentro del save-email utilizamos el ngModel debemos importar el FormsModule dentro del modulo de nuestro “modulo”.



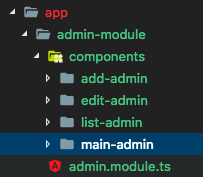
Al estar modularizado podemos utilizarla en varias partes de la aplicación utilizando el mismo tag de app-main-email

Creando un modulo avanzado con rutas:

Creando la estructura, una carpeta admin module: que dentro tiene sus componentes y su mismo module.ts



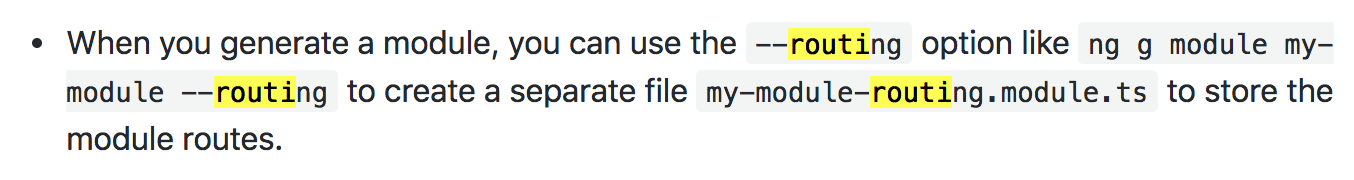
Creamos otros para: añadir, listar, editar:



Luego podemos añadir rutas hijas dentro del mismo modulo.

Dentro de la carpeta admin-module se crea el archivo admin-routing.module.ts

NOTA: Este pudo ser generado desde el cli al crear el modulo poniendo –routing

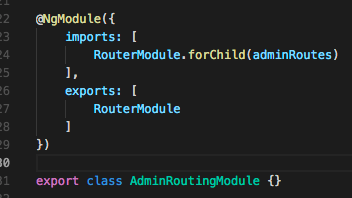


Siendo esta la configuración del archivo:

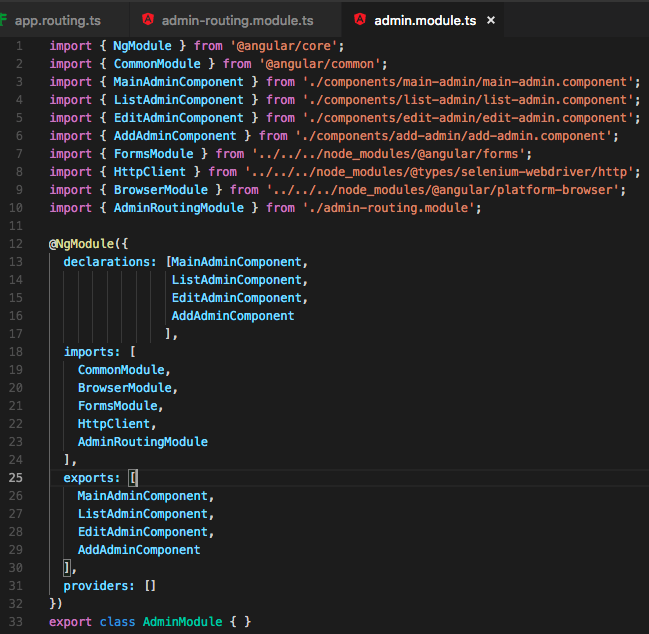
Un arreglo que contiene las rutas, donde hay una ruta principal la cual tiene hijos.  
Esto en el navegador se mostraría: /admin-panel/edit



Luego al final añadiendo el ngmodule pasándole el arreglo de rutas y exportando la clase de rutas

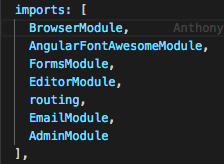


Utilizando el modulo de admin:



Para utilizarlo en la aplicación principal, es necesario importar esta configuración en app.module.

Y se importa en los imports del app.module:



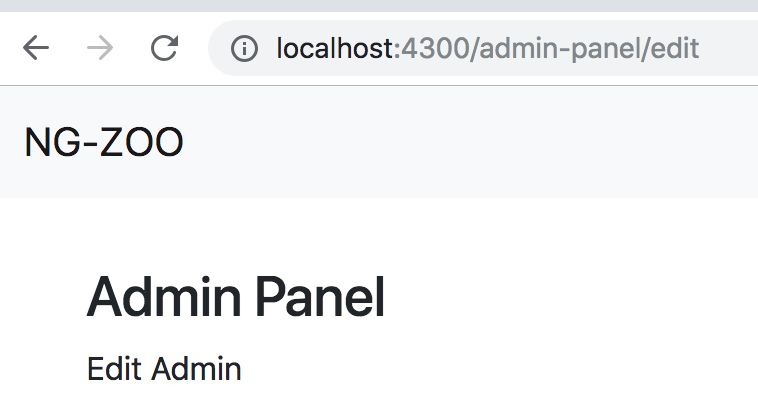
Modificando el routing para que el child route redirija al componente de list.



Haciendo esto aun no nos muestra el contendio de cada pantalla del admin.

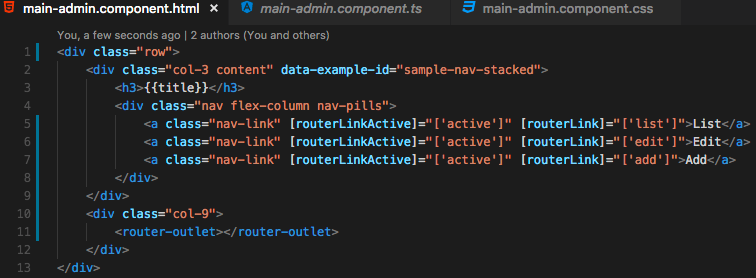
Hay que modificar el main de admin para utilizar el routerOutlet

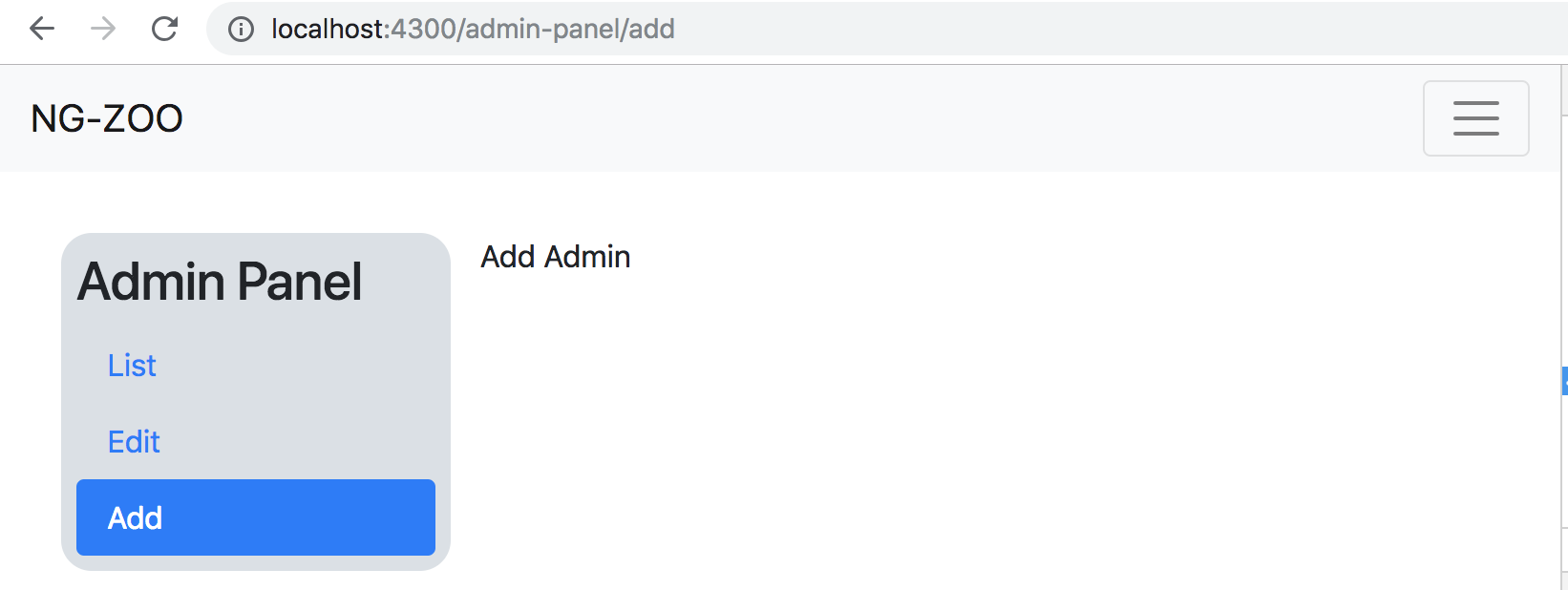




Recordar revisar el HTTP MODULE o lo que sirva en este momento

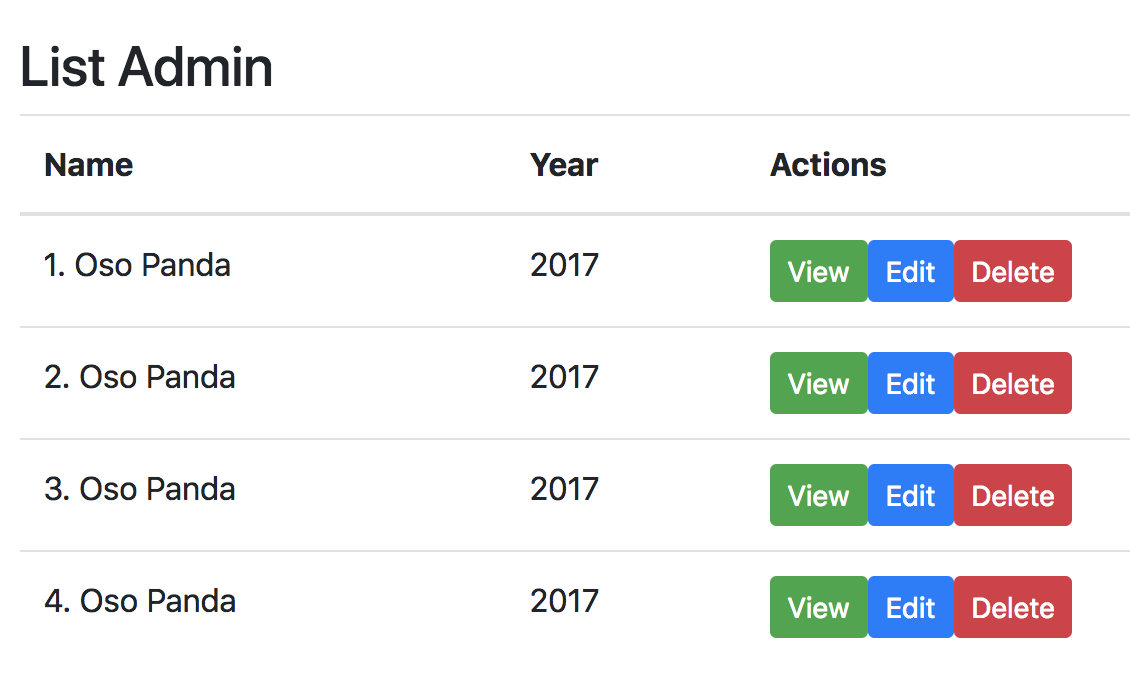
Maquetacion del modulo de panel de administración:



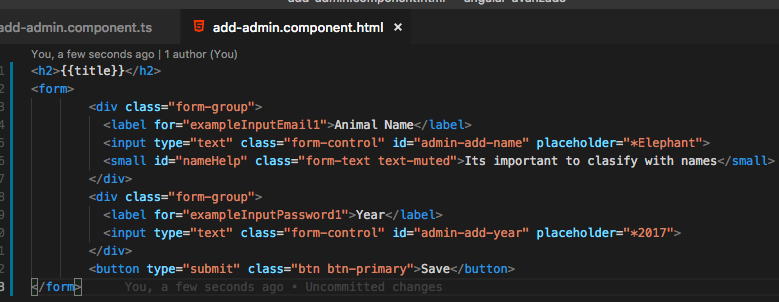


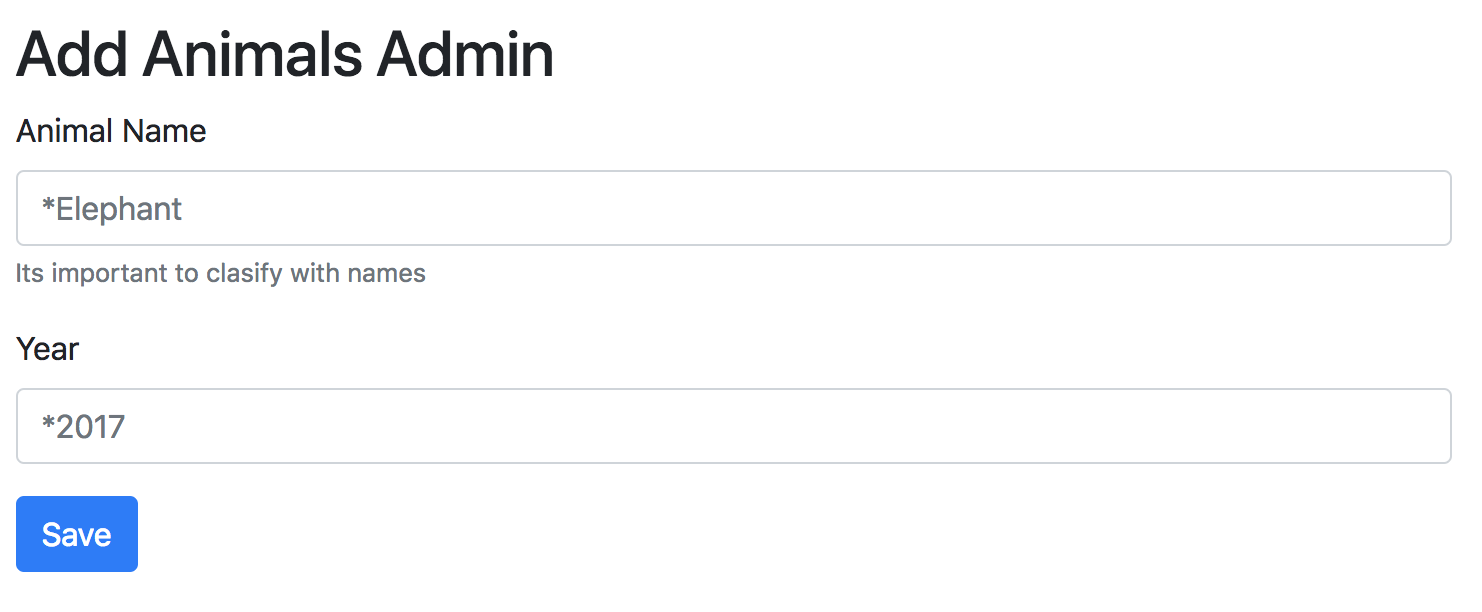
Ahora dentro de cada opción vamos a modificar el html.





Añadiendo un formulario para añadir imágenes en el add-admin:





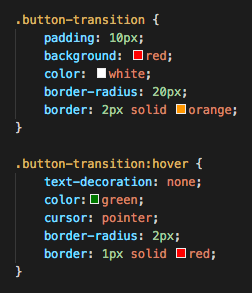
Animaciones CSS3

Transiciones:

Desde la pagina de store, donde hemos estado haciendo las pruebas se practicara.

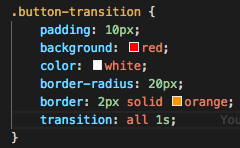


css



Añadiendo transiciones:

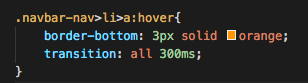
Se puede añadir la propiedad transition, y asignalo a todos los elementos o individual:



o

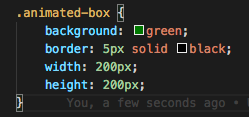


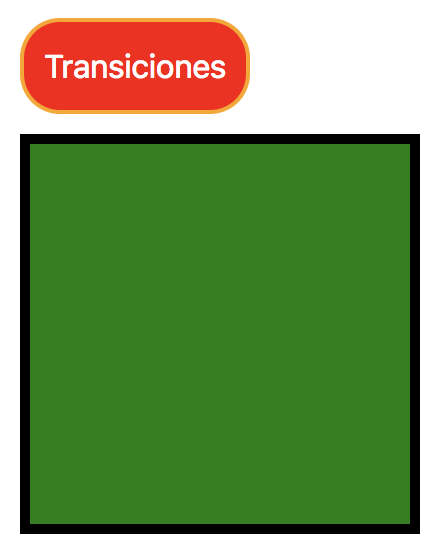
o específicamente a un elemento anidado.



Animaciones:

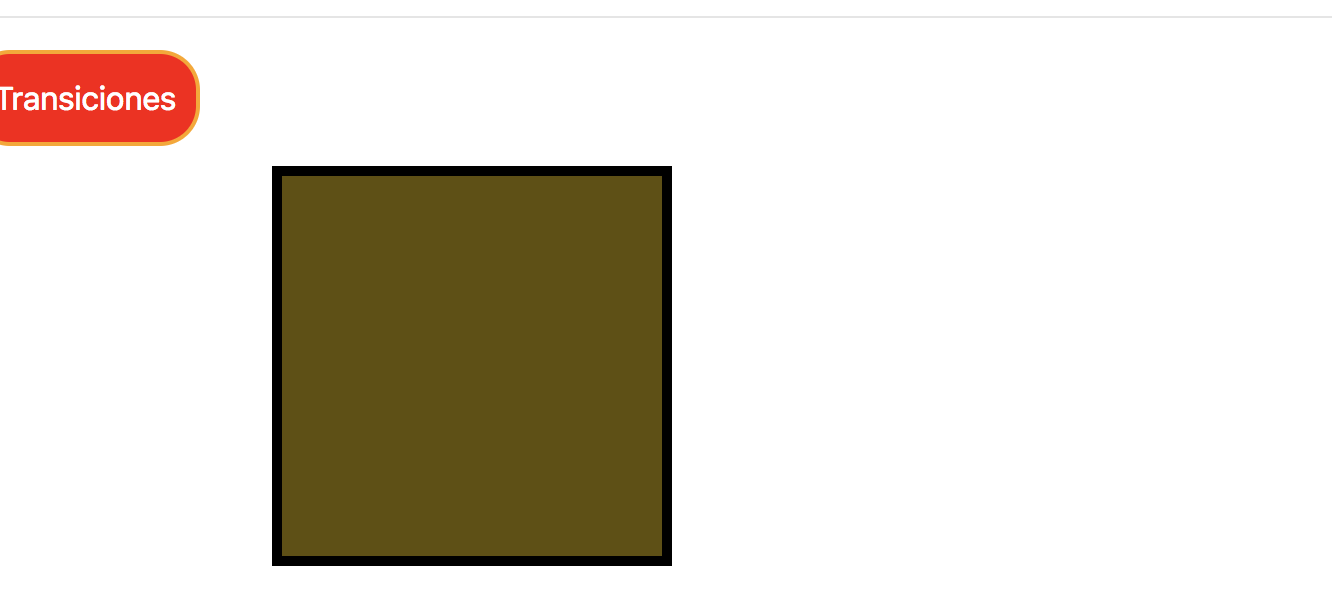
Con la propiedad animation se pueden utilizar keyframes



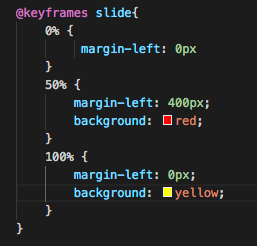


Añadiendo la clase animation:





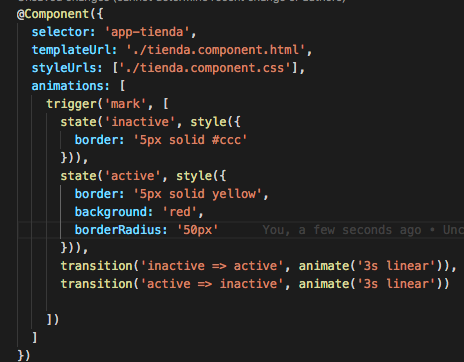
Existe otro tipo de keyframes por porcentajes:



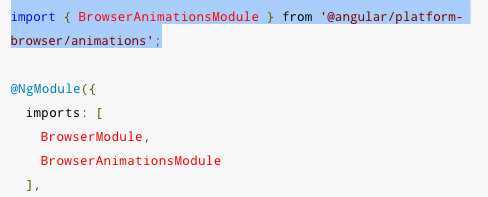
Animaciones con estados de angular:

Hay que importar





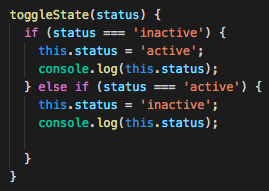
en el app.module



y para poder implementarlo en el html:



Haciendolo interactivo:



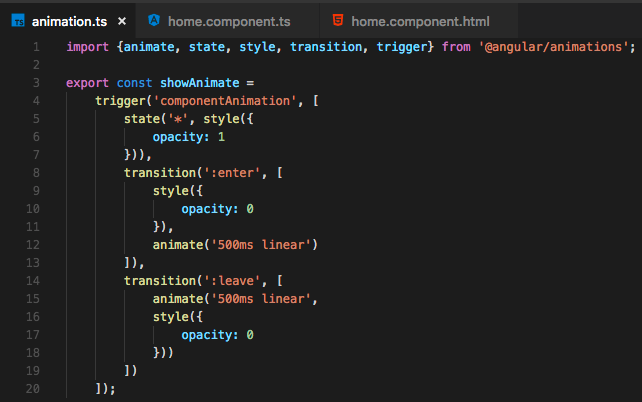


Animación entre componentes:

Se puede realizar animaciones entre rutas cuando entran o salen.

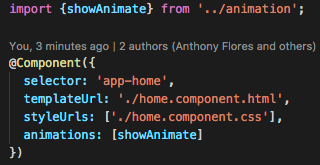
Para esto creamos un archivo especifico, dentro de components se crea el archivo animation.ts

Se realizan los imports necesarios, se crea la constante a exportar, se le asigna un valor de trigger a los cuales se les llamara con las diferentes transiciones.

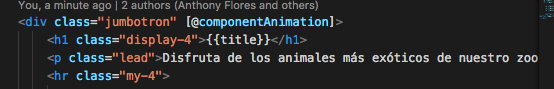


Para utilizar el animations.ts

Dentro del componente se impoerta la variable exportada y se añade a las declaraciones del componente:



Luego en el html se utiliza la directiva del trigger creado.

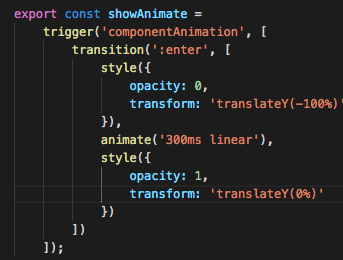


En el caso del leave no se presenta asi que se elimina el código:



Añadiendo la animación a los otros componentes se le agrega la directiva y se importa en el componente el showAnimate.

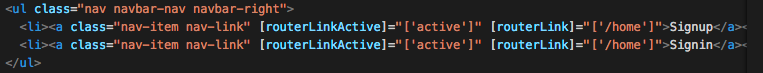
Se puede modificar la animación para que el componente aparezca desde arriba hacia abajo. Con el transform y translateY.



Login y registro desde el front

Regresando del proyecto del api backend

Añadiendo 2 elementos nuevos al menú, login y registro en la parte derecha



Se crea ahora el componente de login y de signup



Se agrega el routerlink en el html y el modulo

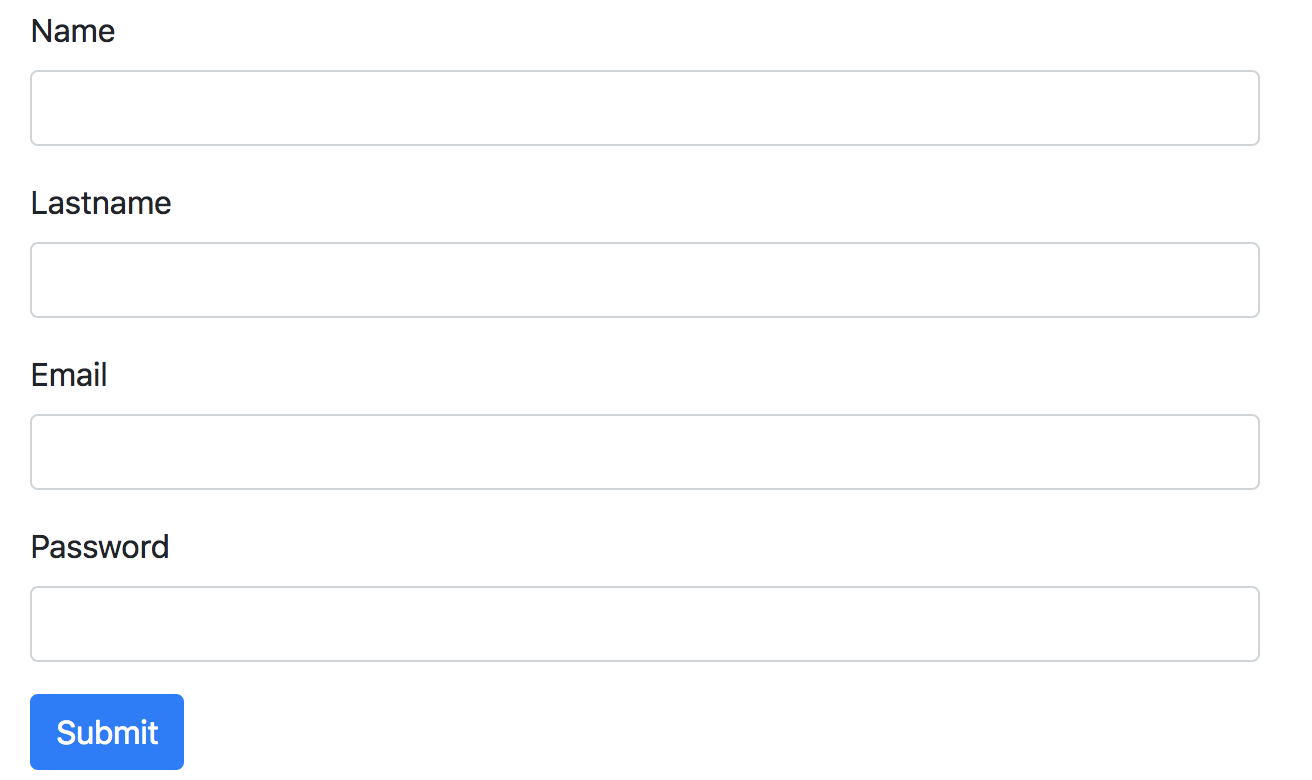


y en el routing



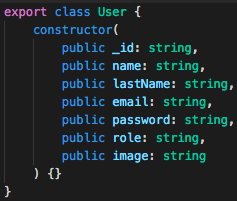
Creando el formulario de registro



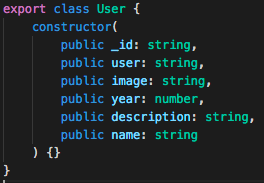


Creando los modelos

Dentro de la carpeta app se crea la carpeta models y dentro el modelo de usuario y luego de animal

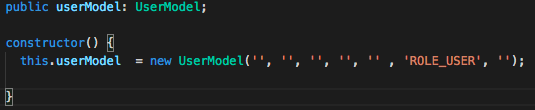


Verificar si tiene id o no



Recibiendo los datos en el componente desde el formulario del html

En el component.ts se crea una instancia vacia del modelo de usuario



Ahora desde el html para que angular pueda detectar el formulario debemos asignarle el tag de que es un form, y añandiendole el evento al sumbit

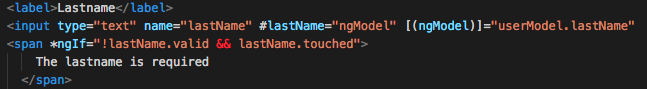


En cada input le vamos asignando un tag describiendo que es parte del ngmodel y luego asignándole directamente el valor con el databinding [()]

Y añadirle validaciones directamente con el modelo



Al igual con los otros campos



Haciendo el botón de submit disponible o no dependiendo de que sea válido el formulario completo

Ahora dentro del componente se crea la función onSubmit,

Recibir los datos





Creamos una nueva carpeta en la raíz de app services

Se crea un archivo global.ts dentro

Donde se guardara la configuración del url



Asegurarse ue el HttpModule esta importado en el app.module

Y el httpClientModule



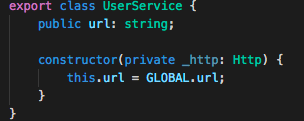


**Ahora se crea user.service.ts**

Se importan los modulos necesarios

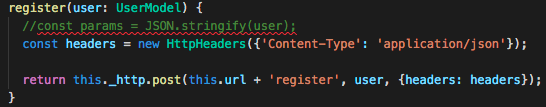


el rxjs,…map nis ayuda a obtener la respuesta del observable en formato Ajax



y en la función de registro del servicio obtenemos como parámetro el nuevo usuario a crear. Luego se implementa el header el cual tendrá permiso para acceder al backend, y finalmente se retorna la respuesta del servicio con el \_http que es instancia del HttpClient.

Pasándole como parámetro el url que viene de global + la función del servicio, el objeto (usuario) a ingresar , los headers.



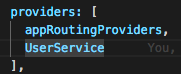
**Creando el método de registro en un componente para pegarle al servicio local**

Ahora desde el component.ts de register se importa GLOBAL y el servicio (global aun no necesario.)

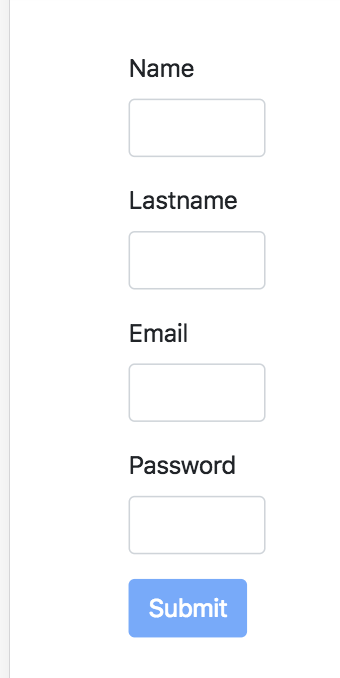




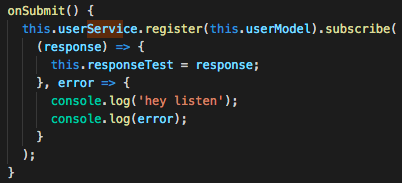
Recordar añadir el servicio desde el app.module providers



En el submit del formulario:

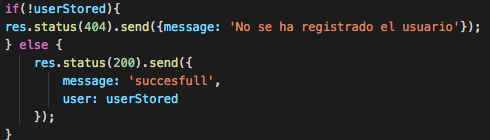


se implementa la funcionalidad del servicio haciendo un subscribe para obtener la respuesta o el error del servicio



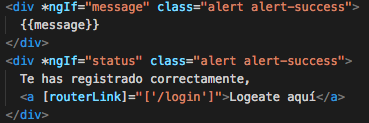
si imprimimos la respuesta del servicio



lo que coincide con la respuesta del servicio que desarrollamos en el backend:ç  


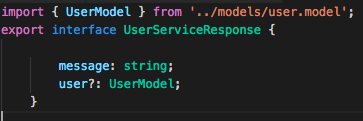
Luego podemos agregar una propiedad status, por si el servicio agrega correctamente al usuario poder direccionarlo a la pagina de login





**Controlando de una mejor forma la respuesta del servicio:**

Se crea una interfaz de respuesta del servicio:



luego se implementa en la respuesta del componente validando asi si el servicio responde un usuario al almacenarlo o solo un mensaje de error

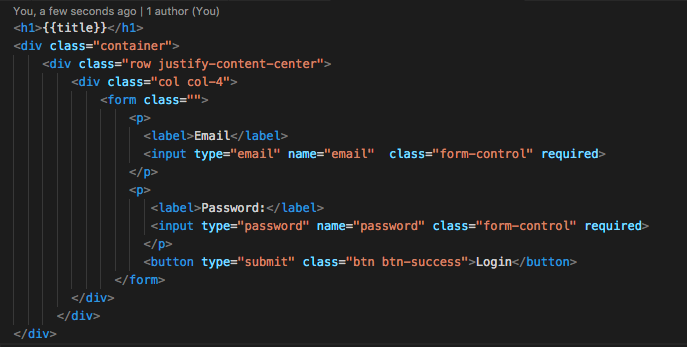


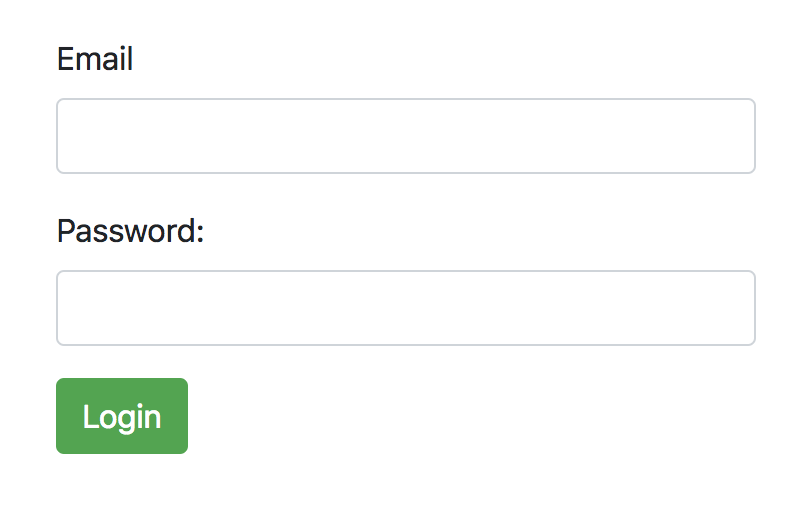
Se crea una variable local que contendrá el mensaje que tiene el servicio y solo se mostrara al recibirlo



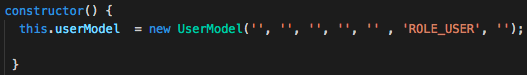


**Formulario de Login**





En el archivo component.ts se crea una instancia vacia del objeto user:



En el formulario se declara como un ngForm declarándole el nombre de acceso:



y en los inputs declarar su nombre y unirlo a la variable userModel con la directiva ngModel



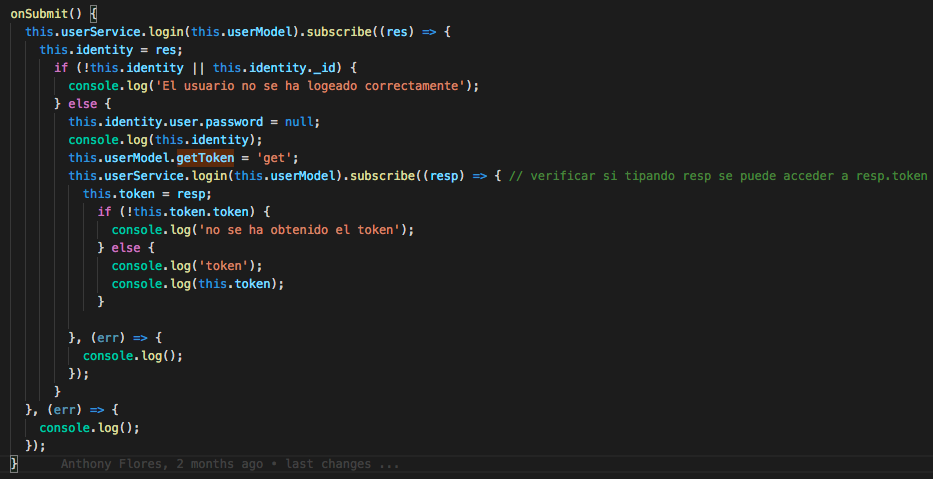
Añadiendo el mensaje de error si no se ha llenado o no es valido.

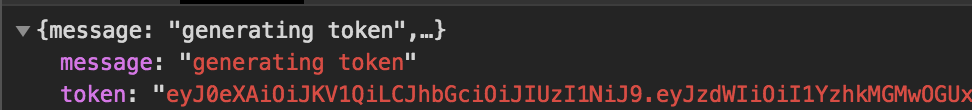
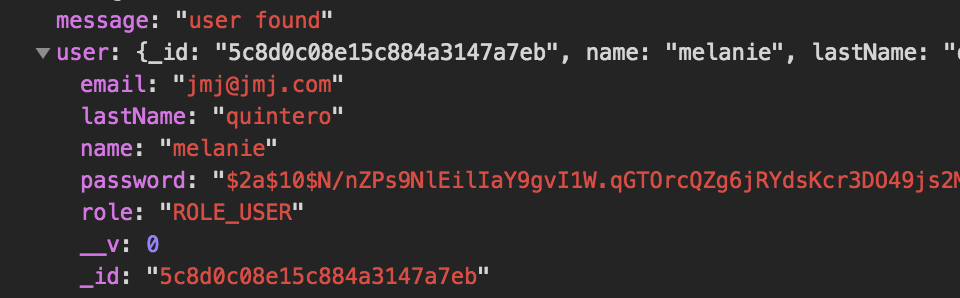


**Conectando el servicio con el front y back para login**

Es necesario obtener el token, que es como se valida el usuario dentro de su sesión.

Para obtener el token se modificó el objeto user añadiéndole la propiedad gettoken para poder obtenerlo según su valor.





**Local Storage para almacenar datos**

Una vez obtenida la información del usuario, es necesario almacenarlos para que persistan en la navegación de las paginas, y den el acceso a las paginas adecuadas.

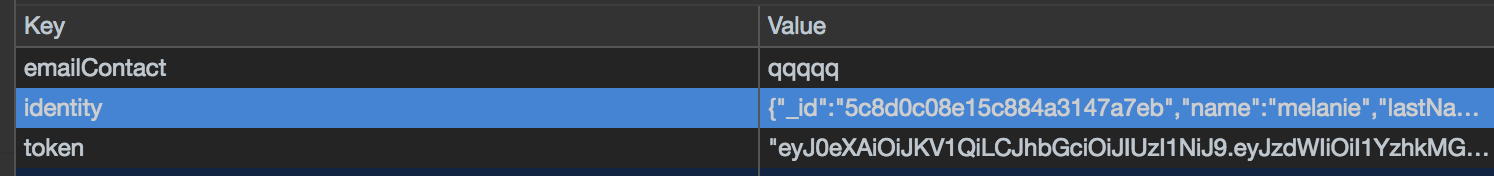




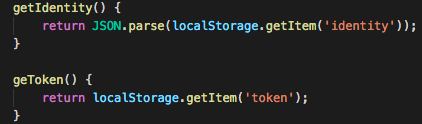
Es necesario hacer la conversión con el JSON.stringify o si no lo guarda como object object

Y para recuperar el valor:





La forma de obtener el identity y el token se puede mejorar, haciendo un método en el service que devuelva estos valores.



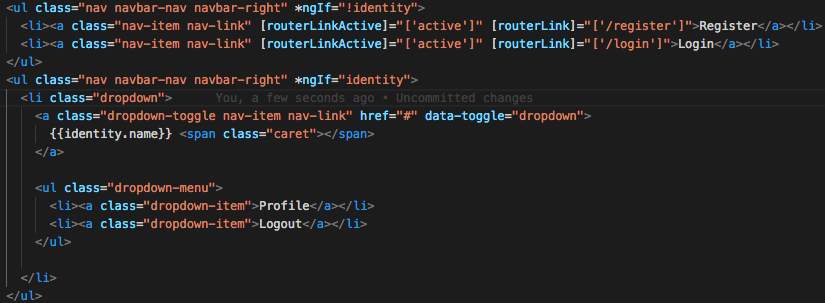


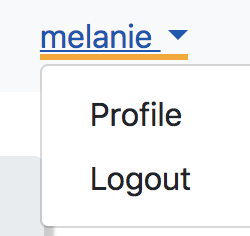
Ahora ya podemos validar desde cualquier lugar si el usuario esta logueado

En el app component creamos la variable identity consultando al servicio local y mostramos o no las opciones según el usuario logeado o no.



y desde el html se puede mostrar o no el menú:





Ya con estas validaciones se puede saber si el usuario esta logeado o no, después de logearse se le “redirecciona” hacia el home.



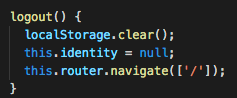
Al hacer login se redirige a la pagina home pero el header no cambia hasta presionar enter y recargar la pagina. Para hacer esto automático se debe agregar en el app .component.ts un ngDoCheck para detectar los cambios en la apliacion y verificar los datos de identity,



Saliendo de la sesión

Se limpian las variables del localstorage y también se vacial los objetos.

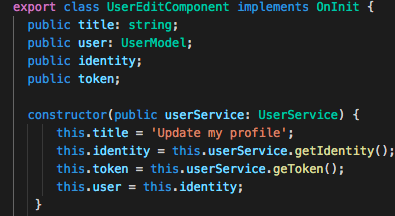
Luego se navega hacia la pagina de login.



**Creando pagina de mis datos**

Ng g c components/user-edit

Se crean las variables y se les asigna valor segun lo que se mostrará. Necesitamos los datos del usuario, identity, y el token.



y agregando la ruta desde el app.component.html

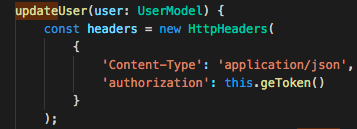


El cuerpo del formulario es parecido al de registro, solo que ahora van a estar llenos los campos con los datos del usuario. Recordar cambiar la validación del formulario por el nombre que se le dio en el tag form.



**Creando servicio para editar datos de usuario**

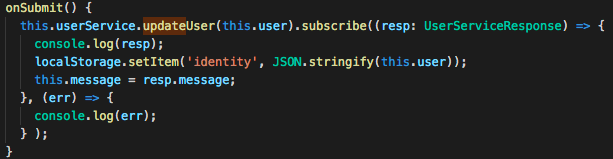
En este caso el servicio debe pasar la variable headers con el objeto authorization para poder modificar dentro de la base de datos con el backend.



y retornando la respuesta de nuestro api pasando como parámetro el url, el path definido en el api, con el url el id del usuario, el usuario como parámetro con los datos a actualizar y los headers.

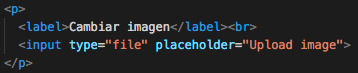


luego en el componente de user edit se utiliza esta función



**Subir imagen de usuario**

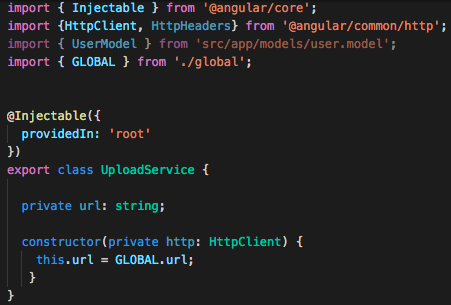
En el html

****

Para subir la imagen se va a crear un nuevo servicio

Ng g s services/upload

Y luego importando lo que necesitemos

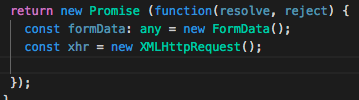


Ahora se crea un método que realiza una petición Ajax normal para bindear archivos.

Le pasamos los parámetros necesarios que puede ser la url en caso de que necesitemos apuntar a otra, parámetros, los archivos y demás.



Creamos una promsea dentro con las variables que van almacenar el archivo



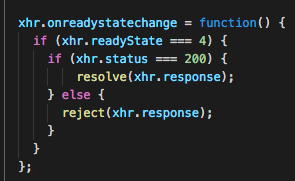
Luego se hace un ciclo for para recorrer el arreglo de file que recibimos en los parámetros



y dentro con el objeto formData le vamos añadiendo con el append la información del archivo a subir,



Creamos la petición con xhr (INVESTIGAR SOBRE ESTA PETICION)



y luego de configurarla hacemos la petición



hay que añadirle los parámetros y headers para finalmente enviarlo



Ya listo el servicio, vamos al componente y lo importamos para utilizarlo.

En el componente necesitamos un método que capture el fichero que fue introducido en el input.

Creamos un arreglo de tipo archivo



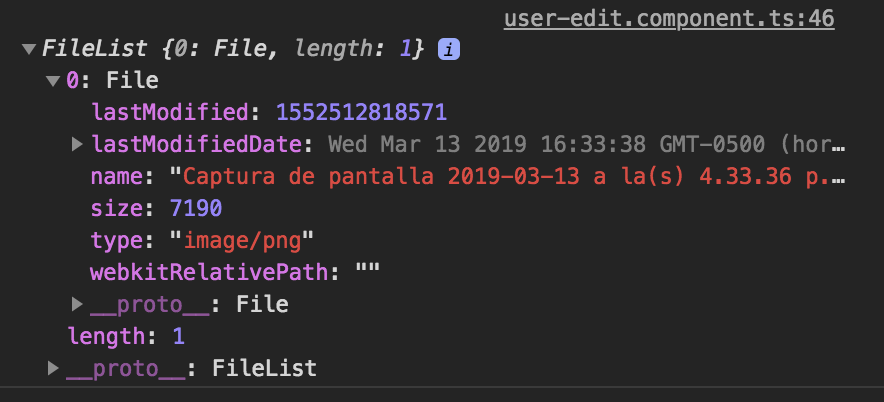
dentro de la función asignamos del parámetro accedemos a la propiedad target.files que contiene el archivo para almacenarlo.



Despues en el html en el input de archivo llamamos al método con un (change) que lo active

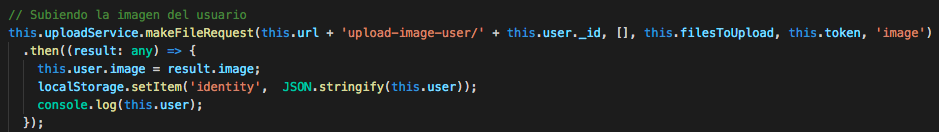


Haciendo una impresión del método

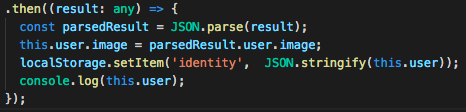


finalmente llamamos a la función creada pasándole los parámetros pedidos, donde el arreglo de strings se lo pasamos vacio.

De ser exitosa la llamada al servicio, asignamos el valor que responde el servicio en imagen, al objeto local de usuario y este lo actualizamos en el localStorage.



Importante!  
Al recibir respuesta de la promesa, esta llega como string, hay que convertirla a objeto con el JSON.parse quedando

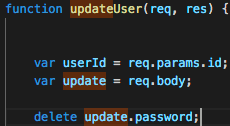


IMPORTANTE

Se borró la contraseña en la base de datos ya que cuando se le pasó al updateUser el parámetro user, se hizo con el identity local donde se había eliminado el password.

Para solucionarlo, en el backend debemos modificar el servicio de editar.

Eliminamos la propiedad password para que no sea modificada.



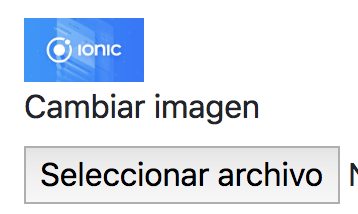
y para no ejecutar el update de imagen, condicionarla si la variable filesToUpload la cual tiene valor solo si se selecciona una imagen



Mostrando la imagen de usuario.

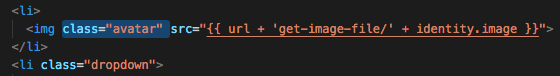
En el user-edit html vamos a mostrar la imagen de estar guardada.

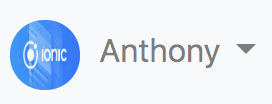




Mostrando la imagen en el navbar

Desde el app.html agregamos el código antes del dropdown y en el component.ts agregamos la variable url trayendo el valor de global.



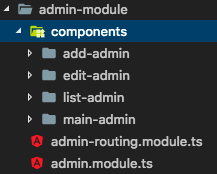


ahora al cambiar la imagen, esta cambia automáticamente en la vista.

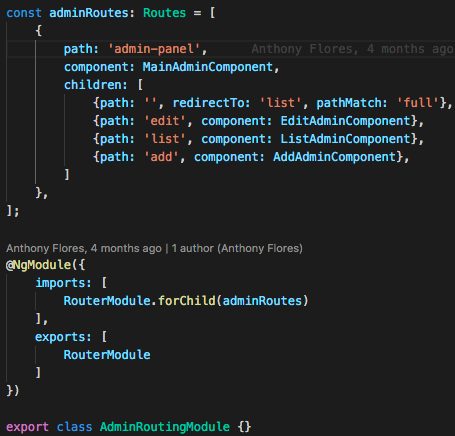
Mostrar panel de administración

En caso de que el usuario conectado tenga el rol de admin.

Recordando el modulo de administrador creado que tiene sus propios componentes.

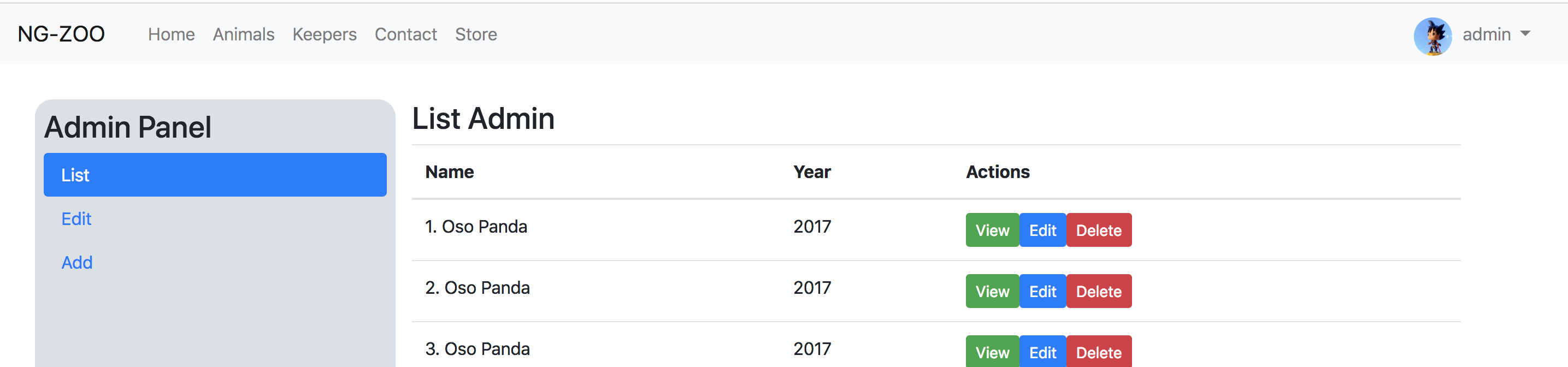


El archivo routing configurado en ese modulo:



Creamos la nueva opción con el routerLink apuntando al padre del modulo admin, el cual re direge al list :





“Guards” en el modo administrador

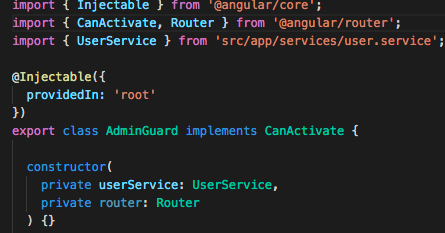
Los guards son pequeños servicios que permiten controlar si una ruta es visible para un usuario o no.

Se crea un archivo admin.guard.ts dentro de la carpeta services.

Se genera con   


El cual genera una plantilla, pero no es la que se usara en este ejemplo.

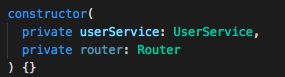
Al importar CanActivate es la propiedad que nos va a permitir controlar la seguridad de la ruta.



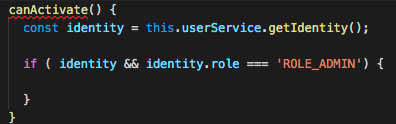
importamos también el userService



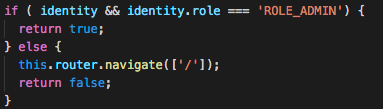
y asignamos los valores a las variables a utilizar



Se implementa el método canActivate y dentro se recupera el objecto y identity, y se comprueba su rol

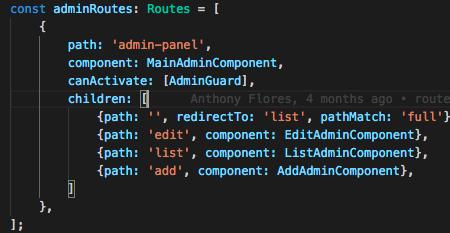


Este método de canActivate solo puede devolver un booleano ya que cuando sea llamado este va a preguntar si es o no admin.



Ahora se implementa dentro de el administrador de rutas del modulo de admin.



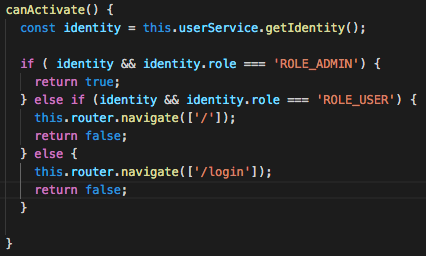


En este momento al intentar entrar al admin panel, desde las opciones del usuario o directamente del url no lo permite, manda al home.

Es necesario proveer el AdminGuard en el admin.module ya que es el modulo que lo estará utlizando para validar junto con el userservice.

.

Validando de una mejor forma para cuando ya sea que el usuario no tiene permisos o no este logueado lo redirija a home o al login respectivamente.



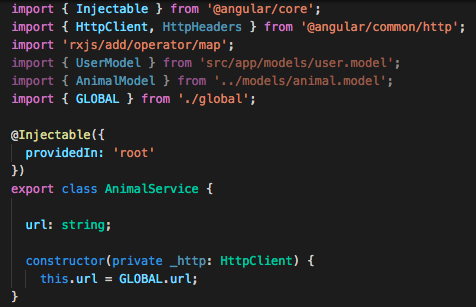
Como las otras rutas son rutas hijas, se encuentran protegidas por el guard provisto para el admin-panel

Componente de añadir animales

Se crea el servicio.



dentro importamos lo que necesitaremos

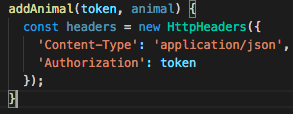


Para añadir un animal:

Según lo que tenemos en el routes del back



Es necesario enviarle un token y el animal

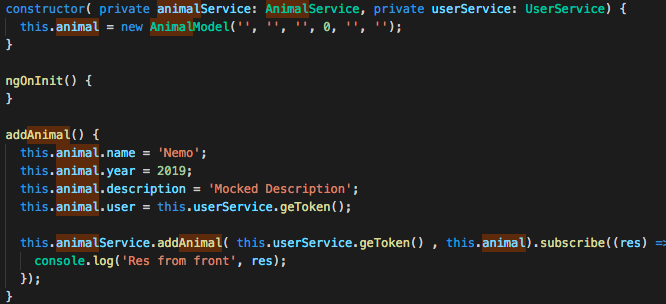


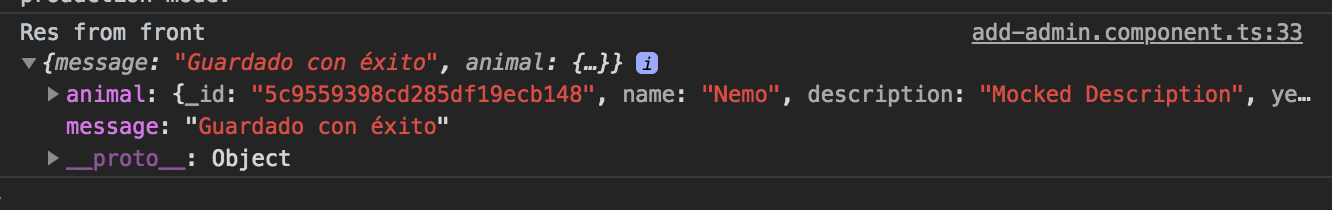
y retornamos la petición al servicio que es un observable



Desde el add-admin component utilizaremos este servicio asi que se hacen las importaciones necesarias.

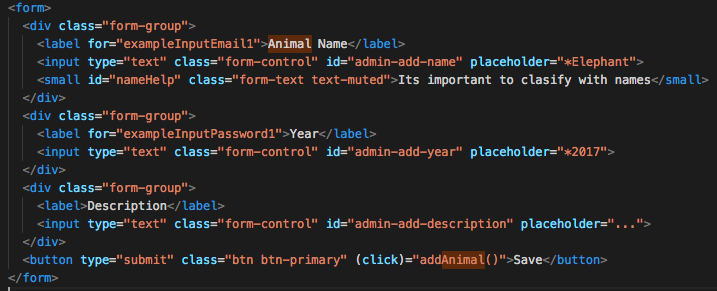
Probando el servicio mockeando las respuestas del formulario





Uniendo el formulario a la petición:

Actualmente el formulario no esta unido con el objeto que se pedirá



Se agregan los tags necesarios al formulario



y se bindean los valores de los inputs

Es importante añadirle a los inputs el

name, #name y ngModel



y hacer valido o no el botón dependiendo de la validez del formulario

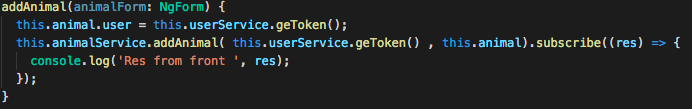


En el component.ts

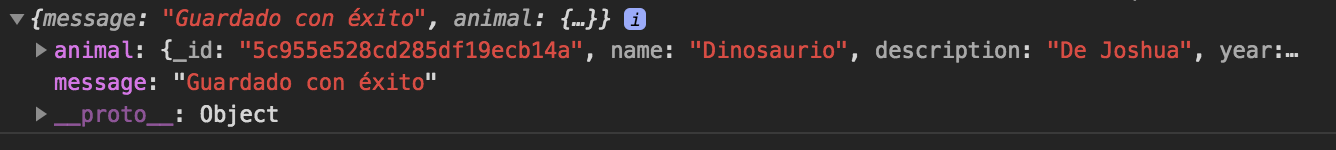


se recibe el formulario en la función para de resultar exitoso el llamado al servicio de añadirlo se puede resetear este.

Finalmente se envía le objeto animal armado, habiendo recuperado el token del usuario al que pertenece.

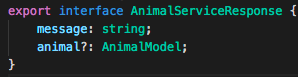


El servicio esta construido para que en la respuesta devuelva un objeto animal solo cuando este se ha añadido correctamente.

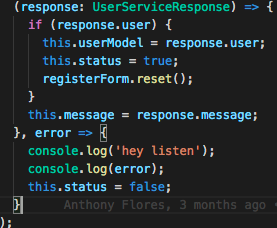


Por eso podemos hacer la comparación en base a ese valor para ver si fue exitoso el llamado.

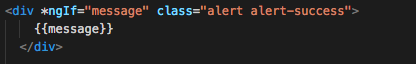
Creamos una interfaz de respuesta del servicio



Tipada la respuesta podemos validarla y responder en el html según el servicio.



Se añade el mensaje en la vista dependiendo del status que se genera luego de la respuesta.



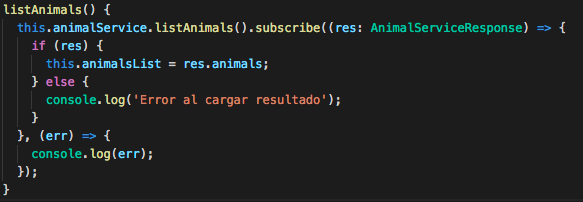


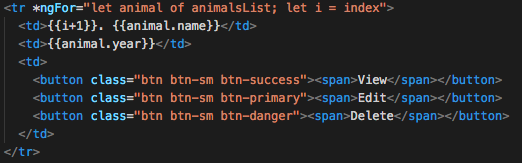
Se utiliza el mismo servicio usado en user para guardar la imagen del usuario , ya que este recibe como parámetro el url y el objeto a editar.



Podemos validar al tener una respuesta del servicio de agregar animal si este fue exitoso y redirigirlo directamente a la lista de animales.

Listando los animales







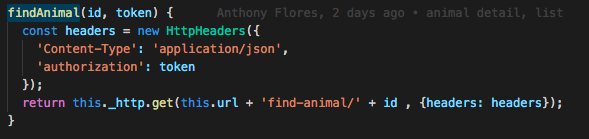
Detalle de animal

En la carpeta de componentes principales creamos el componente



Se agrega al app module.

Y se verifica su ruta en el routing  

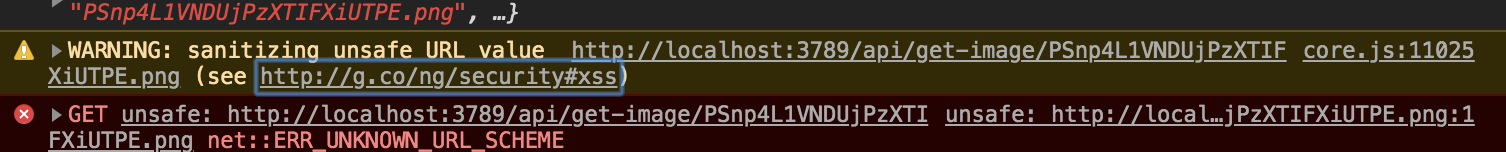

Se importan los controladores y servicios necesarios y luego se crea la función de traer obtener animal, la cual se hace en el service.:  


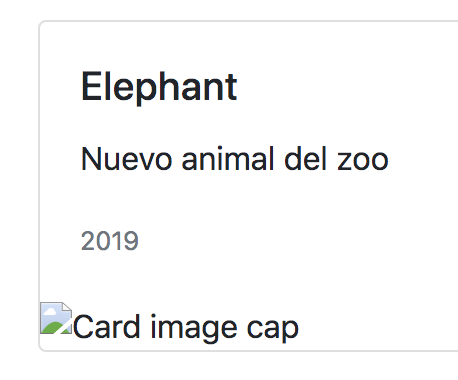
Y en el componente le pasamos los parámetros requeridos  


Importante, el id lo estamos recibiendo como parámetro asi que hay que enviarlo desde la pagina anterior List. Se envía como parámetro con el router link.  


En este caso se llama la imagen del html.



VERIFICAR TEMA DE TRAER IMÁGENES DE URL NO SEGURO, sanitizing etc  




Maquetando la pagina de detalle animal





**Editando un animal**

Creamos el servicio local que irá direccionado al api

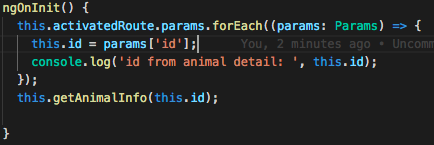
Creando primeramente los headers, y luego en la petición a retornar la dirección con el nombre de la función en el api y los parámetros requeridos.



Ejemplo de ruta en el servicio  


Ahora dentro del componente de editar:

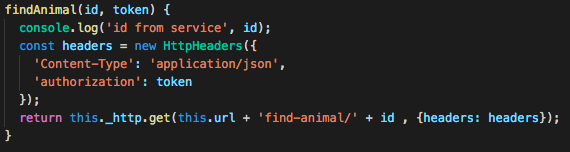
Al entrar obtenemos el id que nos viene por parámetros:



Con el id llamamos a la función que hara la petición al api que nos retorne la información del animal:



Que llama a nuestro back:

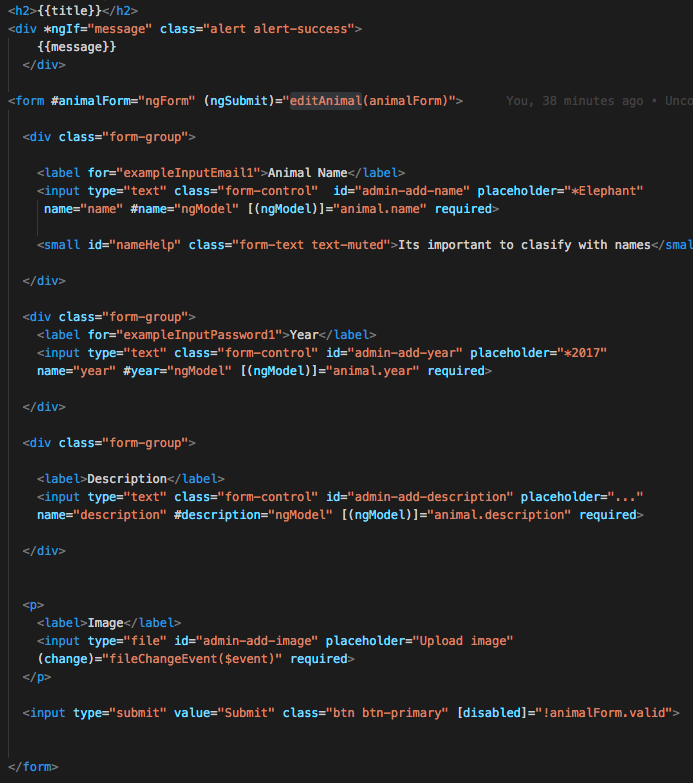


y este al servicio con la ruta find-animal en rutas:



que luego de pasar por el middleware de auth llama al controlador  

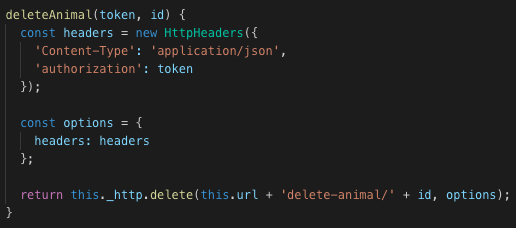

obteniendo asi en el front el objeto animal con el que se llenará automáticamente el formulario que esta bindeado con el objeto animal.



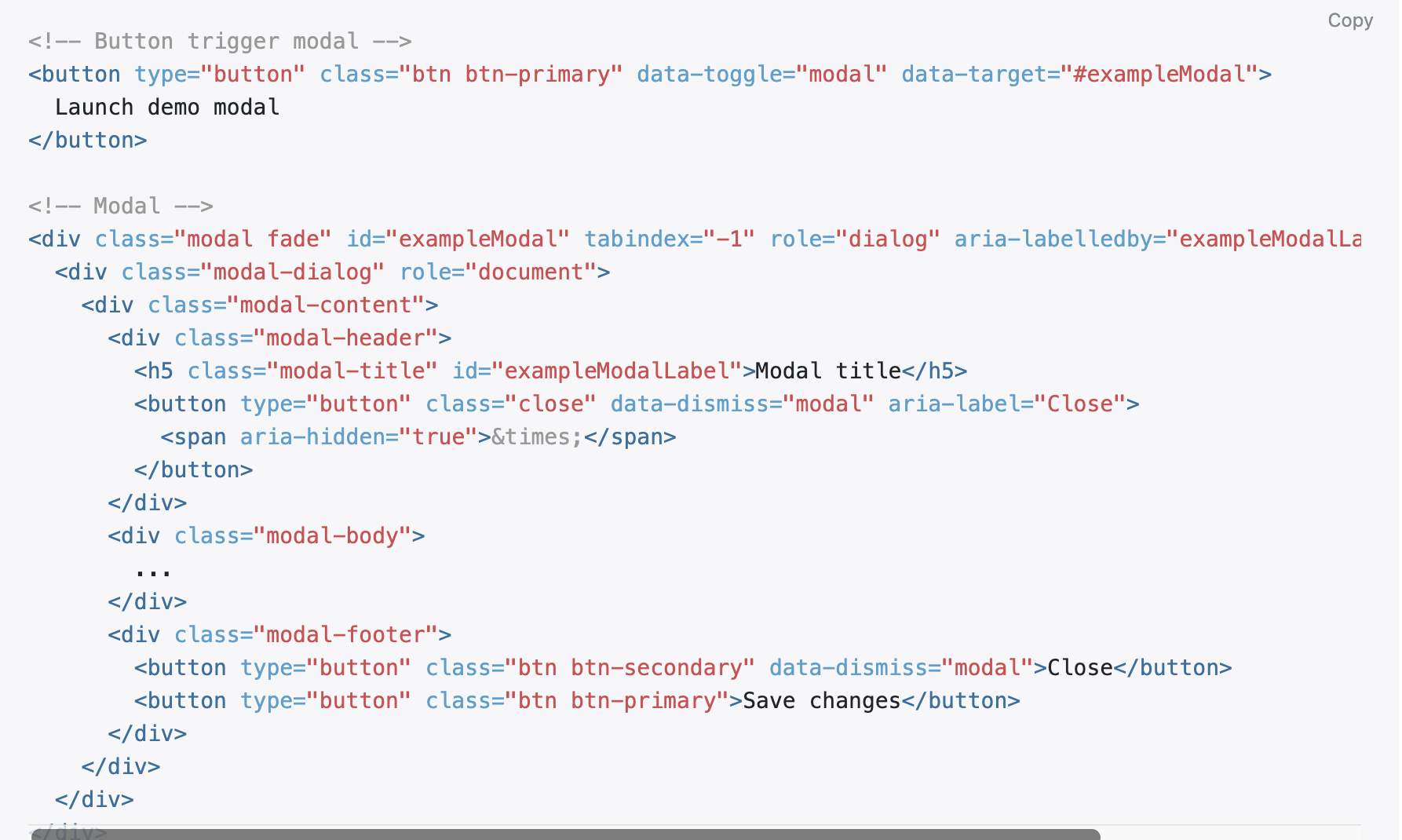
Una vez modificado el formulario llamamos a la función local  


Que llama a la función de nuestro servicio descrita en el principio de editar animal.

Boton PopUp y borrado de animales

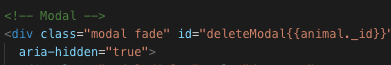
En el servicio local se crea el método de borrar  
En donde pasamos el parámetro headers de una forma diferente.  


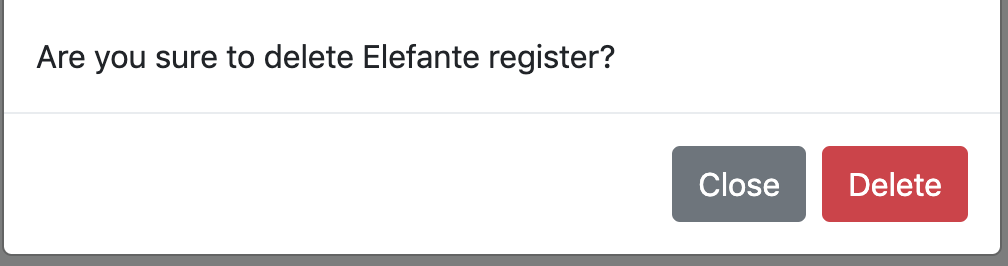
Luego en la vista se implementa un modal:

Se implementa desde la documentación de bootstrap  


Para hacerlo dinamico se pueden pasar valores utilizando un elemento “a” en vez de botón y su href:







Y ahora añadimos al component.ts la función borrar:

